Ian Stroud · Hildegarde Nagy

Solid Modelling and CAD Systems

How to Survive a CAD System

Springer

Solid Modelling and CAD Systems

Ian Stroud · Hildegarde Nagy

Solid Modelling and CAD Systems

How to Survive a CAD System

Springer

Dr. Ian Stroud
Ecole Polytechnique Federale de Lausanne
STI-IPR-LICP
Station 9, Batiment ME
ME-AI393
1015 Lausanne
Switzerland
e-mail: ian.stroud@epfl.ch

ISBN 978-0-85729-258-2 e-ISBN 978-0-85729-259-9

DOI 10.1007/978-0-85729-259-9

Springer London Dordrecht Heidelberg New York

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

The use of registered names, trademarks, etc., in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant laws and regulations and therefore free for general use.

The publisher makes no representation, express or implied, with regard to the accuracy of the information contained in this book and cannot accept any legal responsibility or liability for any errors or omissions that may be made.

Cover design: eStudio Calamar S.L.

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

This book is dedicated to our families

Preface

Unless you are Homer, the sensible course to take in any narrative, whether it be fiction or somewhat factual, is to start at the beginning, proceed through the middle and finish at the end. This book is not like that. CAD is something of a Gordian knot in which it is difficult to find an end from which to start. The purpose of the book is not to criticise, but to explain how CAD systems work, relate this to theoretical aspects, explain shortcomings and limitations. The book is not supposed to be about any one particular system, though some are mentioned. Many CAD systems have similar functionality and, wherever possible, this functionality will be explained rather than particular implementations. This is because CAD systems are dynamic and, hopefully, will continue to change as new requirements and ideas appear. The techniques behind, though, are more stable and, in order to understand what a CAD system is doing, it is helpful to understand what is going on behind the interface.

CAD systems are a combination of knowledge from, at least, the three domains: computer science, mathematics and engineering. They are complicated and hence it is difficult to be an expert in all three domains. For this reason it is common that software experts implement the software rather than the engineers for whom the systems are intended. This means that there can be differences between the way that a system works and what the user would expect.

The intended audience of this book is:

- CAD users—Both students and professional users who would like to understand more about the CAD tool that they are using. The examples and exercises are intended to give an insight into what is going on behind the functions so that it is easier to avoid problems, but also to understand those problems which do occur.
- CAD teachers—People who would like to teach CAD at a higher level than just which buttons to push to get a result. The exercises and examples in the book are intended as a supplement to normal teaching with example exercises.
- CAD system developers—The book tries to present some of the 'other side' of CAD, a little of where the current system philosophy is wrong or where it could be improved.

The way this book is organised is that the first chapter is intended to demonstrate some simple object creation sequences and to pose some of the questions that a user might put. These questions are a sort of introduction to the material in some of the other chapters.

Chapter 2 explains some of the history, about different solid modelling methods, before describing Boundary Representation (B-rep), the current main method, in more detail. CAD systems worked with wire-frame modelling and then constructive solid geometry (CSG) methods before settling for B-rep.

Chapter 3 describes two-dimensional definition methods and Chap. 4 describes the main model creation operations that are used to create models. Along with the operations are brief descriptions of algorithms and special cases. There are suggestions for various experiments to illustrate some limitations and explain what is happening. Chapter 5 describes, briefly, another important area, the geometry of parts. Chapter 6 defines non-manifold models and examples of their use in a hybrid modelling environment. Chapter 7 describes graphics input and output in CAD. Together this group of chapters describes the core modelling functions of CAD systems.

The next chapter, Chap. 8, describes information in models. This can be thought of as an add-on to the basic model for improving the level of communication between application areas. Chapter 9 describes current methods for exchanging model data.

Chapter 10 describes some aspects of the complex topic of features. This is a rather subjective topic and there is a wide variety of perception about what constitutes a feature. The chapter presents some of the history and some of the uses of features.

Chapter 11 presents some of the gaps in CAD. In general CAD systems are often used as modelling systems to create detailed geometric models of products whereas a lot of design precedes this phase. The chapter presents a little research work in this area as an illustration of how CAD systems might (and should) develop to provide a uniform design environment.

Chapter 12 presents some aspects of command files and their modern use as history records in CAD is presented. The chapter also describes aspects of object parametrisation. Often, products are collections of single models connected in some way. Chapter 13 concerns the way that these collections, or assemblies, are built up and connected. Chapter 14 describes different aspects of CAD in communities. This concerns how to work with other people in CAD as well as analysis methods to aid the designer communicate with other partners in the production chain. Finally, Chap. 15 outlines some possible projects. These projects were used in teaching at the EPFL to reinforce different aspects of CAD.

Good luck.

Lausanne, April 2008

Ian Stroud
Hildegarde Nagy

Acknowledgments

I would like to thank, again, all present and past colleagues for their support and help throughout the years.

Many thanks to Claudia Rainfurth and Meinolf Gröpper of VDMA for arranging the documentation of VDA-FS.

The comments made in Sect. 14.1.1 are not only from personal observations but also long discussions with many other people. While I would like to mention especially Friedrich Glantschnig, Peter Müller and Professor Suk-Hwan Suh, there are many other people from the European STEP-NC project, the ISO TC184 SC1/WG7 and ISO TC184 SC4/T12 communities who are too numerous to mention. While the section is a personal presentation, it has been formed based on their input and I would like to thank all concerned.

I would also like to thank Professor Paul Xirouchakis and all colleagues at LICP of the EPFL for their support.

Contents

Chapter 1
Case Studies

This chapter is about the end of the story. The purpose is to explain a little bit about how CAD systems are supposed to be used so that the details in subsequent chapters can be set into context.

The first examples here take a known example and try to explain how to model it using a CAD system. This should not be confused with design. This is modelling. Design is a wider task, the aim here is to use examples to explain the functionality of a CAD system so that it is easier to apply it to design problems. A little about design is given in Chap. 11 where early phase design tools and methods are mentioned.

The solid images for these examples have been created using CATIA version 5.

1.1 Linear Extrusion Piece

The first piece is a simple linear extrusion part, shown in Fig. 1.1.

A drawing with the dimensions is shown in Fig. 1.2. This part is made by defining a set of shapes and extruding them linearly as additions or subtractions from the shape.

1.1.1 Modelling Method

The following building method is arbitrary. The aim of this, and later exercises is to introduce some basic working methods. The instructions here are not related to any particular system, hence no icon pictures are included. Instead, the reader should match the functionality to that of the CAD system available. When working with systems it is, in my view, better to think in terms of the functionality rather than the icon because this helps to switch between systems.

I. Stroud and H. Nagy, *Solid Modelling and CAD Systems*,
DOI: 10.1007/978-0-85729-259-9_1, © Springer-Verlag London Limited 2011

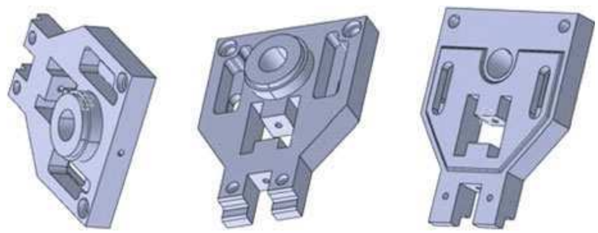


Fig. 1.1 Simple linear extrusional part

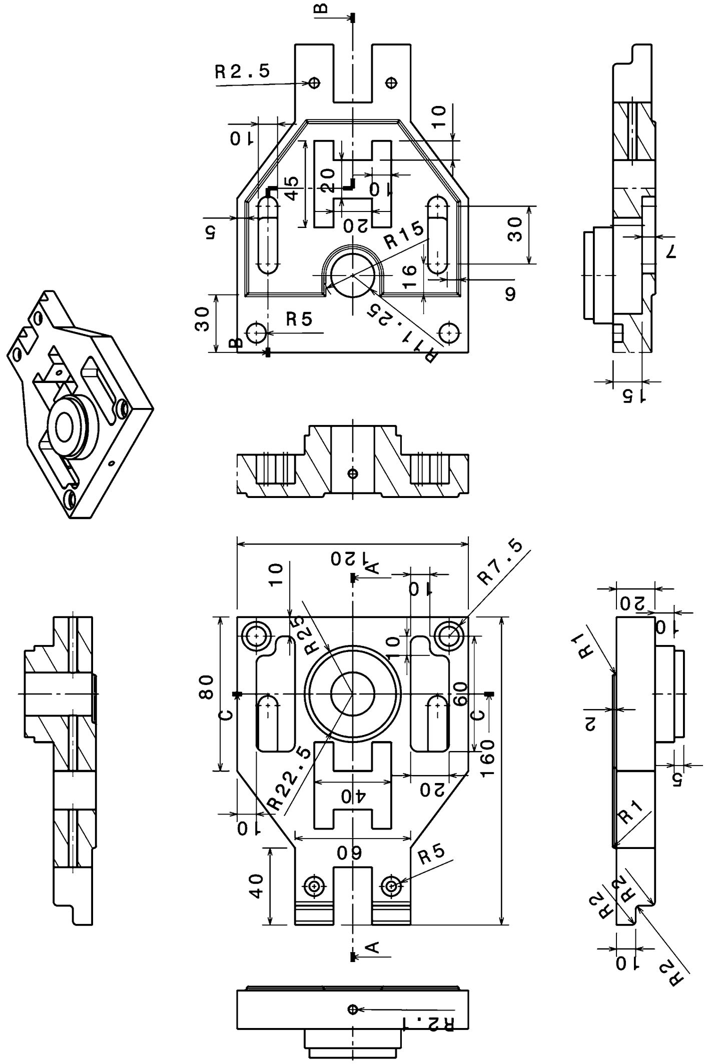


Fig. 1.2 Simple linear extrusional part drawing

The first part in these exercises is one which has the basic shape made by linear extrusion. The part shape and dimensions are arbitrary and any resemblance to an industrial part in existence now or in the past is purely accidental.

1.1.1.1 Step 1: The Basic Shape

Create a base with shape as shown in Fig. 1.3. Extrude this 20 units to create the basic object, shown in Fig. 1.4. Extrusion is explained in Sect. 4.2.

The general idea with most current CAD systems is to create a base shape and then add elements until the final shape has been created. This creates the design incrementally. The alternative might be to create sets of elements, features (described in Chap. 10), connecting to known parts and then fill in the gaps between these. This type of method is not currently supported, hence the need to create a basic shape and then add or subtract elements.

The basic shape is made as a simple, connected figure consisting of lines, circles and maybe other curves. The methods for this are explained in Sect. 3.2. The positions of these are usually, nowadays, controlled by what are called "constraints". These constraints may be relational, such as perpendicularity, parallelism or tangency, say, and distances represented by dimensions.

The actual method of creating shapes varies a little. Rather than try to explain existing systems, which are subject to variation, the intention is to try to explain the techniques behind what the systems do. The techniques tend to be more stable than the interfaces so it seems preferable to understanding the techniques rather than a particular system. Also, CAD system vendors usually provide good explanation material for learning their systems which helps match knowledge of a technique to the system interface.

Two variants can be termed the absolute position variant and the relative position variant. With the absolute variant the positions of the points and lines are given in global coordinates relative to an origin. With the relative position variant, the positions of points are given relative to each other, so that edge lengths are correct, but that the position in space is not fixed. Usually systems have a mixture of these, absolute positions can be defined but relative lengths tend to be preferred.

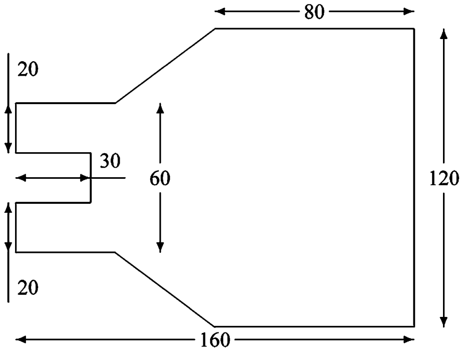


Fig. 1.3 Linear extrusion example, basic shape

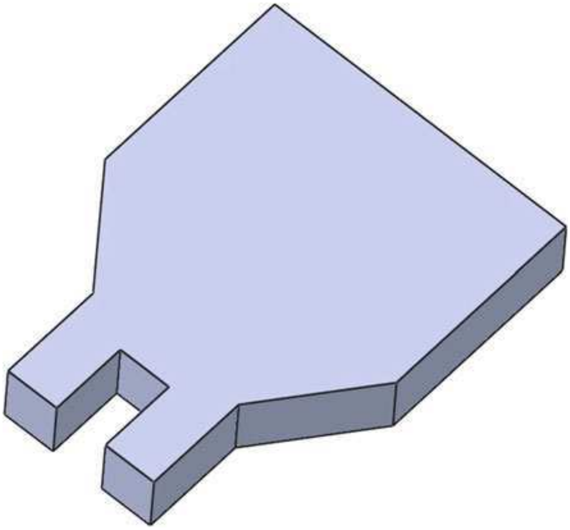


Fig. 1.4 Base part

In absolute positions the shape in Fig. 1.3 might be defined as a polygon with the coordinate point sequence:

(80,−60) (80,60) (0,60) (−40,30) (−80,30) (−80,10) (−50,10) (−50,−10) (−80,−10) (−80,−30) (−40,−30) (0,−60) (80,−60)

This produces a figure centred about the global origin. Note that the coordinates are defined in two dimensions because shapes are created in planes, hence the third coordinate is defined from the plane, and the coordinates of the points are relative to the projection of the global origin onto this plane.

With the relative position method, the right-hand edge would be defined as having a length of 120, the overall width of the piece as 160, etc. as in the sketch in Fig. 1.3. The relative method has an obvious connection with drafting and is easy to use, provided the rough shape is sufficiently close to the final shape. (If it is not then there is a risk that the figure will be deformed when giving the correct dimensions).

Although the relative position method seems clearer and easier to use, it is useful to have the global position known. With the pure relative method, you might have the coordinates:

(83.3333,−67.652) (83.3333,52.348) (3.3333,52.348) (−36.6667,22.348) (−76.6667,22.348) (−76.6667,2.348) (−46.6667,2.348) (−46.6667,−17.652) (−76.6667,−17.652) (−76.6667,−37.652) (−36.6667,−37.652) (3.3333, −67.652) (80,−67.652)

to take a wild example. In addition, the figure could be rotated. This should not be a problem for a user because the local shape is more important than its global position. However, in some cases it may be necessary to combine extruded shapes using Boolean operations (see Sect. 4.1) If this is necessary, then it may be easier to do if the objects have known positions. Note, also, that some systems provide a grid so that it is easier to create a nearly correct shape before adding the relevant constraints and dimensions to the shape. See Sect. 3.2 for more details of these.

1.1.1.2 Step 2: The Circular Extrusions

Draw a circle radius 25 on the top face (Fig. 1.5) and extrude it 10 units (Fig. 1.6 top). Draw a second circle, radius 22.5 and concentric on the top face of this circular extrusion and extrude this upwards 5 units (Fig. 1.6 bottom). Create a circular hole, radius 11.25, concentric with the two extrusions, and it extrude it downwards to the end of the object to create a hole (Fig. 1.7).

Note the lines on the cylindrical faces of the extrusions. The presence, or not, of side edges round a cylinder has varied over the years. See the glossary under the headings "cylinder representation" and "fake edges", and Sect. 2.9, for more explanation.

This is now an incremental extrusion, which is described in Sect. 4.2. What this means in effect is that the shape, in this case a circle, is extruded to produce a separate object which is then combined with the original body using a Boolean operation. The use of Boolean operations has benefits for ensuring the correctness of an object. The two positive extrusions create cylinders which are then "added" to the base object. To create the hole the extrusion creates a cylinder and then this cylinder is subtracted from the object. This is explained in more detail in Sect. 4.2.

Note that some systems use feature names for operations. I have seen this in CATIA v5, for example, and for me it runs contrary to feature theory. For linear extrusions there is an operation called "pocket" in CATIA v5 which is a linear extrusion and a subtract operation. There is another operation called "Groove" which is a circular extrusion and a subtract operation. There is also an operation called "Hole" for creating round holes by creating a volumetric model and then subtracting this. There are two main reasons why I am critical. The first is that the "feature" which results is not necessarily a pocket, groove or hole. The result depends on the geometry of the extrusion shape and its position with respect to the object. The cylindrical hole could be created as a "pocket", "groove" or hole. The second reason for my criticism is that the feature is not maintained during

Fig. 1.5 Linear extrusion example, circular extrusion

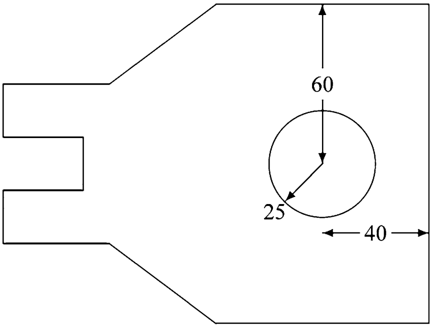
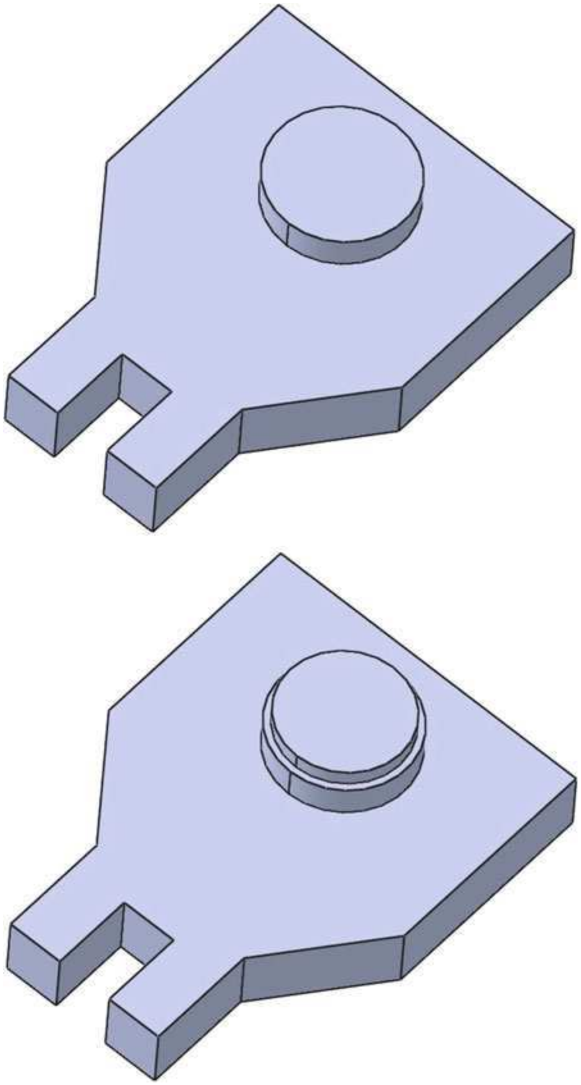


Fig. 1.6 Circular extrusions



modelling, which is a second criterion for having a feature-based modeller. More of this later in Sect. 4.2 and Chap. 10.

1.1.1.3 Step 3: Complex Pockets on Top Surface

Make a shape like that shown in Fig. 1.8 on the top surface of the original extrusion. The roundings have radius 4. Extrude it downwards, or cut it out to a depth of 15. Make a mirror image shape on the other side of the object. The positions of these are shown in the original drawing. Note that some CAD

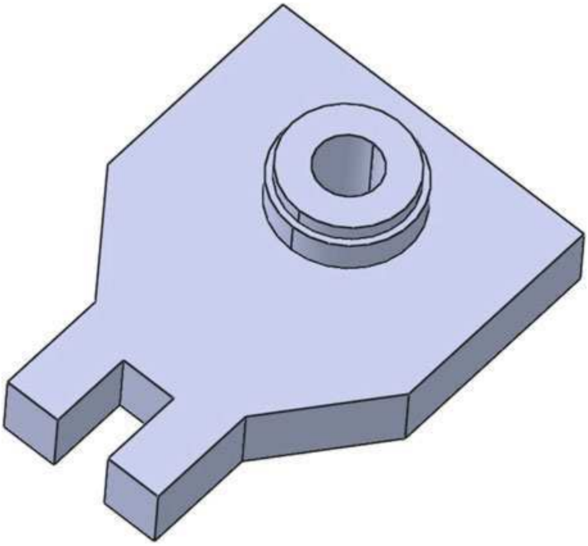


Fig. 1.7 Cylindrical hole through extrusions

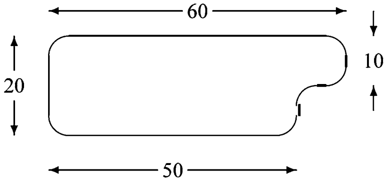


Fig. 1.8 Linear extrusion example, Rounded pocket shape

systems let you perform a symmetry operation on an operation, so that it is easier to produce the second cut-out.

As with the cylinders, creating the pockets is done by creating a base shape, extruding this to create a volume and finally subtracting the volume from the solid. Although this may seem complicated, the benefit is that if the extrusion cuts through part of the object then a hole will be created. This happens in step 5. The result of creating the two pockets is shown in Fig. 1.9.

1.1.1.4 Step 4: Small Extrusion on Underside

This is a simple extrusion but with a complex shape with roundings, or blends on the edges. The extrusion shape partly follows the original contour at a distance of five from the edges of the original contour. This is shown in Fig. 1.10.

The result of the extrusion is shown in Fig. 1.11. As always, this is done by creating an extruded volume and then adding this to the basic figure. As with the other extrusions, though, this is hidden behind the interface. The interface may indicate only a positive or negative extrude, how the operation is implemented is something else.

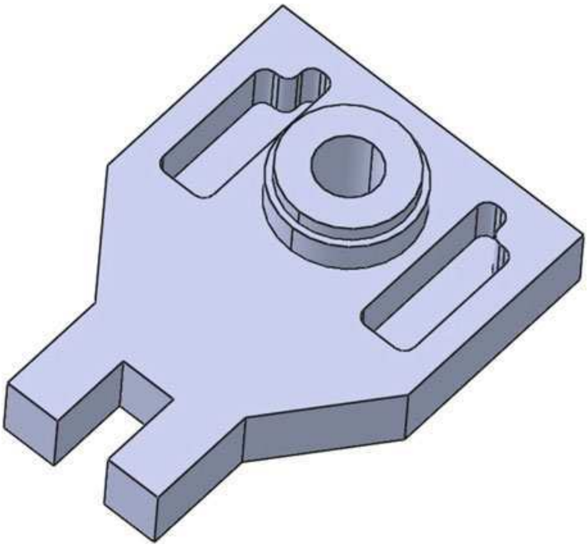


Fig. 1.9 Complex rounded pockets

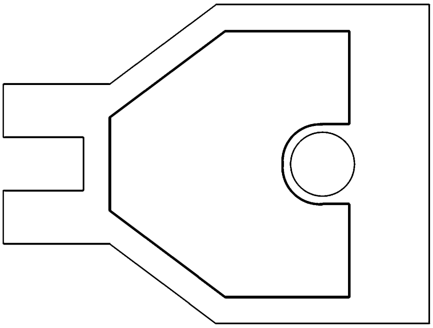


Fig. 1.10 Bottom extrusion shape

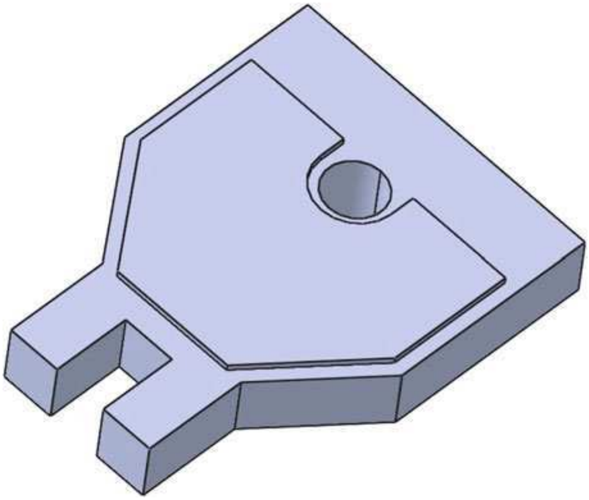


Fig. 1.11 Bottom extrusion

The edges on the bottom of the extrusion and the edges where the extrusion is attached to the basic shape are rounded, radius 1. This should make the side faces disappear as the blends meet in the middle of each of the thin faces round the extrusion. See Fig. 1.12. More on blending in Sect. 4.8.

1.1.1.5 Step 5: Slots on Bottom Face

Once the extrusion on the underside has been made it is possible to create two simple pockets, depth 7. The size and placement of these is shown in Fig. 1.13.

By now you may be getting tired of all this extrusion, but these two slots show what happens when one extruded shape meets another one, Fig. 1.14.

The advantage of using a Boolean operation is that when the second extrusion is carried out the portion of the pocket bottoms where the two pockets touch is removed. If a Boolean operation were not used you would have two back-to-back faces with zero thickness between them. If you managed to produce an object like this then, when it is manufactured, you would probably have a ragged hole shape

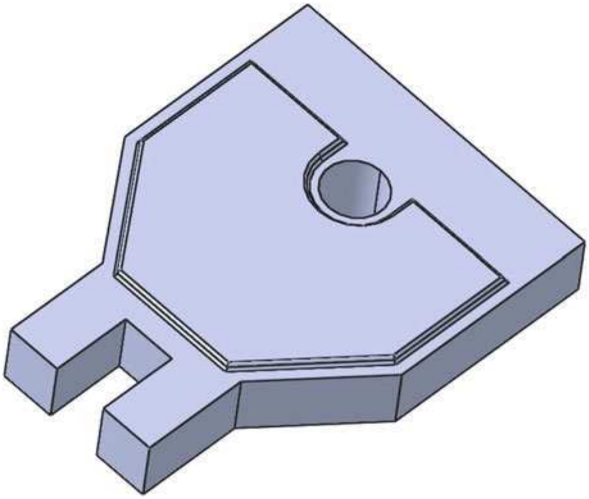


Fig. 1.12 Blended edges on bottom extrusion

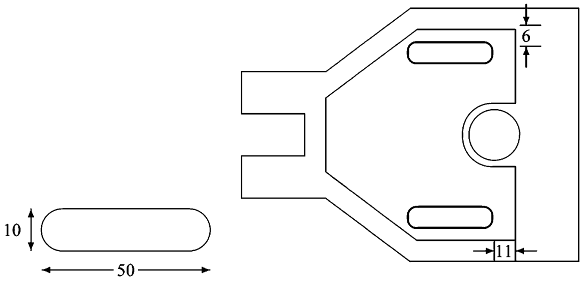
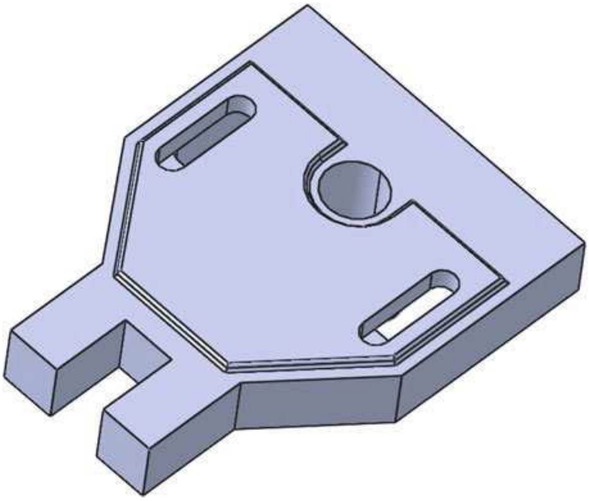


Fig. 1.13 Linear extrusion example, bottom pocket shape and placement

Fig. 1.14 Pockets on bottom



or shapes as the tool partially breaks through. More on other effects of collisions during extrusions in Sect. 4.2.

1.1.1.6 Step 6: Front Cutout

This is a simple rectangular cutout with blended edges, which is shown in Fig. 1.15. This should be more or less routine, a rectangular shape is drawn on one of the planar faces at the left of the object and then this is extruded and subtracted from the object.

The edges in the step are then blended with a radius of 2 units.

Note, though, the feature result. This is a cutout, as with the pockets earlier, but it is not a pocket. Which is what the comment in step 2 was about.

1.1.1.7 Step 7: Fixation Holes

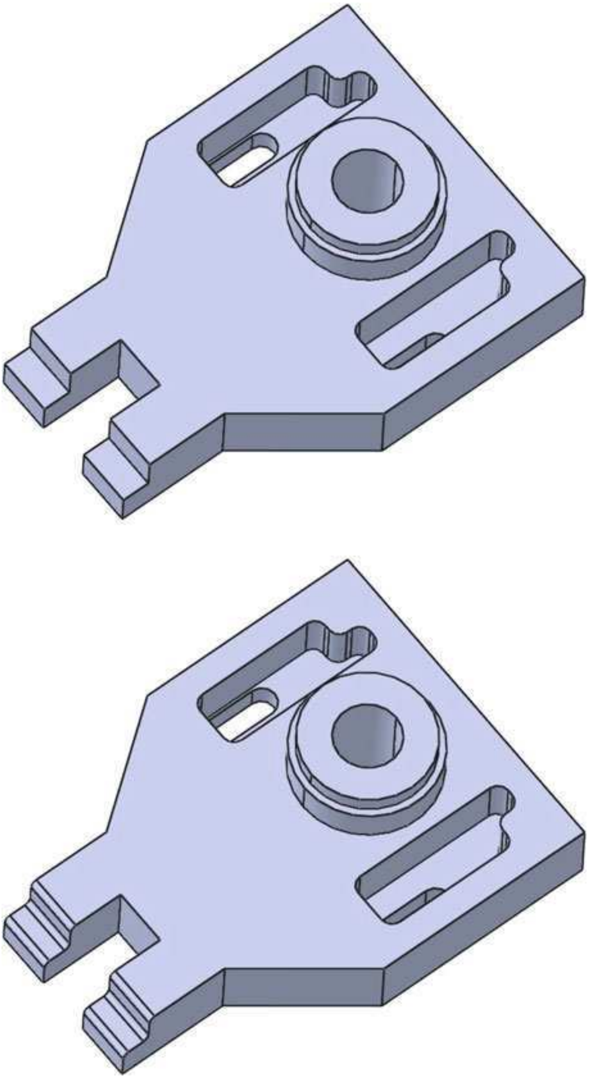
There are four fixation holes in the object, two in the prongs and two in the wider part, as shown in Fig. 1.16.

There are different ways to produce this result depending on what the system allows. If the system has a special hole-making operation then this can be done in one step. If not then it might be done in various ways.

Consider the stepped hole in Fig. 1.17.

With linear extrusions this can be done as shown in Fig. 1.18. A circle of the desired radius for the top part is created centred at the position for the hole. This is extruded downwards to create a circular pocket. A second hole of smaller radius is inscribed on the bottom of this pocket and this second circle is extruded downwards to a specified depth or through the object.

Fig. 1.15 Front rectangular cutout



Another method is to create a planar section with the shape of the hole profile. This is then rotated about the long vertical axis with option cutout (or similar) so that the resulting volume is subtracted from the basic shape, as shown in Fig. 1.19.

A third method, similar to the second method, is to create a profile and rotate it to create a separate object. This object can then be moved to the correct position and subtracted, illustrated in Fig. 1.20. An advantage of this method is that the object can be copied and arranged into a pattern to create multiple holes with the same operation. This is explained in Chap. 4.

Fig. 1.16 Fixation holes

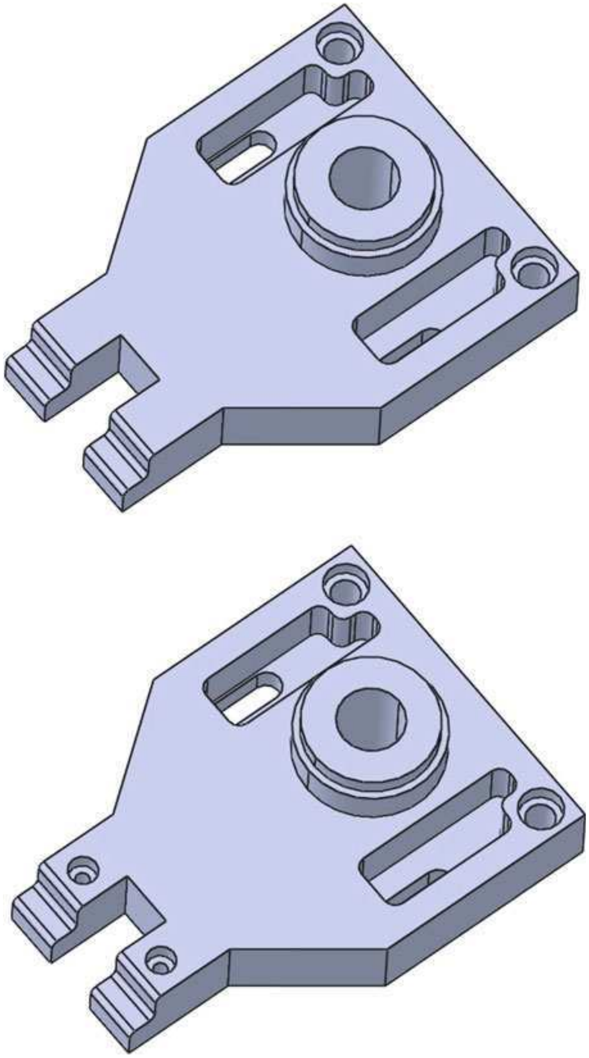
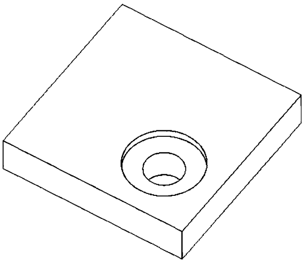


Fig. 1.17 Stepped hole



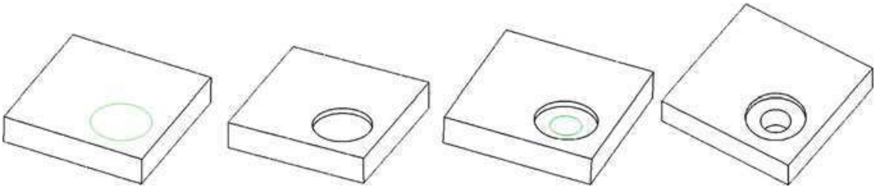


Fig. 1.18 Making a stepped hole with linear extrusions

Fig. 1.19 Making a stepped hole with circular extrusions

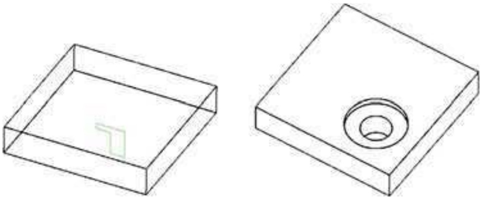


Fig. 1.20 Making a stepped hole with a hole object

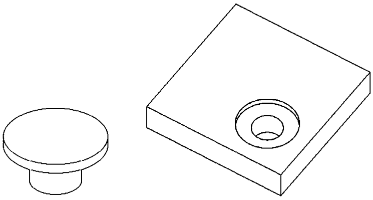
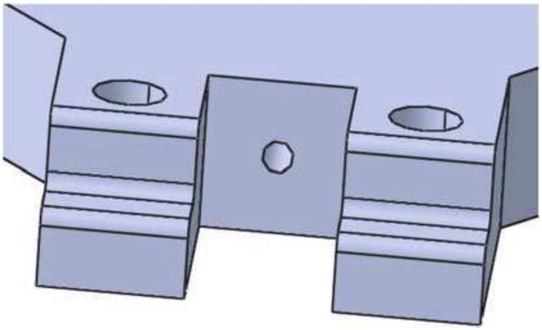


Fig. 1.21 Longitudinal hole



1.1.1.8 Step 8: Longitudinal Hole

This is a small hole through the centre of the object, as shown in Fig. 1.21. This can be created by inscribing a circle of the correct diameter, here 4, and this circle is extruded through the object with a cut-out. In the object in this exercise, which was made using CATIA v5, there is an option to add a thread to a hole. Adding a thread is another interesting variant. Geometrically, a thread is complicated, but creating a threaded hole is a relatively straightforward manufacturing operation. There was a lot of discussion about threads in the early days of modelling because it is more efficient to simply attach an information entity to the hole with the thread details. This is easier for the modelling system and also easier for downstream applications to identify the thread information rather than trying to reconstruct it from complex geometry. See Chap. 8.

Look closely at the curved edges in the model. Sometimes these appear facetted rather than perfectly curved, but this is only a graphics effect. This will be explained elsewhere, in Sect. 7.2.1. In the model the hole is exact, but if you look closely at Fig. 1.21 you can see this effect.

1.1.1.9 Step 9: H-shaped Cutout

This is a simple cutout extended through the piece, and is shown in Fig. 1.22.

Note how the long hole created in the previous step is cut by the extrusion. This is another example of the benefit of using Booleans in conjunction with modelling operations.

1.1.2 Questions

1.1.2.1 Why Not Make Just a Half Piece and Reflect It?

The use of symmetry presupposes that it is known beforehand that the shape is symmetric. Sometimes it is possible to determine this and so only half the piece need be designed. See Fig. 1.23. It is also possible that part of the object is

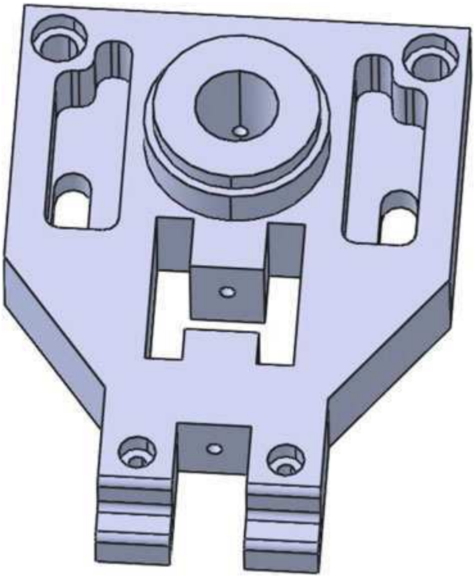


Fig. 1.22 H-shaped cutout

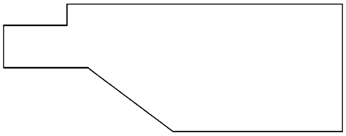


Fig. 1.23 Linear extrusion example, basic half shape

designed using symmetry, and then this symmetric object is used as a base part. In this example it seems a little ridiculous to create the circular extrusions in step 2 as half shapes. This could be moved to after the symmetry operation.

This means that only one complex pocket on the top face, half the extrusion on the bottom face, one simple pocket on the bottom face and one each of the two different sized fixation holes created. The matching elements are created with the symmetry operation.

1.1.2.2 What Shape Is the Base Shape?

There are many options about how to decide on the base shape. As well as that shown in step 1, the half shape for symmetry is a second possibility. Yet another possibility is that shown in Fig. 1.24. Here the prongs are created in a subsequent shape by creating a rectangular shape and then extruding it through the object with a cutout operation.

It is sometimes easiest to create simple shapes and then to add extra features. This is connected with the design process. It also makes it easier to parametrise part shapes, as discussed in Chap. 12.

1.1.2.3 How to Create Blended Shapes

Should you create rounded shapes and then extrude them or extrude square shapes and then add blends? The complex pockets could have been made by extruding shapes with sharp corners, as shown in Fig. 1.25, extruding this to create a pocket and then rounding off the appropriate edges.

This is a question related to the reasons for which the shape has rounded elements.

This is something else that is connected with design. If the desired shape has rounded elements then it is more logical to create a two-dimensional shape with

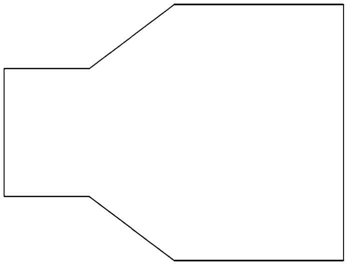


Fig. 1.24 Linear extrusion example, simple basic shape

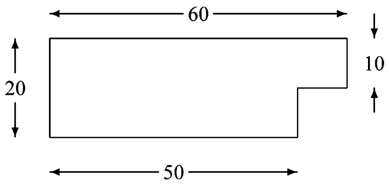


Fig. 1.25 Linear extrusion example, Simplified step 3 pocket shape

rounded elements and extrude this. If the rounded elements are there for functional or manufacturing reasons, say, then it seems logical to round the edges with a blend function. Design is not simply a matter of producing the right shape, the design method should also be logical. For someone else using the design, a manufacturer or another designer, say, it is not always easy to understand why the elements of the design are present. If the design is to be modified later, using the history tree (Chap. 12), then this needs to be logically structured to make the modification easy. Ideally there should be yet another support structure which describes the design decisions used to arrive at the final shape. This is not the same as the history tree of the object, which is the translation of the designers ideas into the modelling tools of the CAD system. Without this design decision tree the history tree should be made as logical as possible.

1.1.2.4 How Do You Choose Limits for an Extrusion?

Should this be a depth, to next, to last or what? Actually this will be dealt with in more detail in Chap. 4. You may have noticed these options for extrusion operations. The classical parameter is a depth value, a real number, but the other options, such as "to last", "to next" and so on, offer the possibility to set this limit dynamically. This is also important for parametric parts, which are described in Sect. 12.3.

1.1.2.5 How and Where Do You Make It?

This is now a difficult question because a manufacturing expert will decide on the exact method. The question, though, concerns modelling modifications to adapt a shape for manufacturing. It is possible to imagine that a design has two main shape models and may have intermediate shape models in between. The first shape model is the ideal shape, the shape that the designer would like. The final shape is the shape that will be produced corresponding to the shape of the physical piece. There may be other intermediate shapes. For example, if the part is to be made by casting then the may be a model with some features removed, draft angles added and blends. This would be used to make the mould cavity. The question is where and how to adapt the ideal shape to the practical, realised shape. If the designer introduces changes, such as blends or draft angles, then it may be difficult for the manufacturing expert to know what is there for which reason. In some cases, for particular firms who have a fixed manufacturing philosophy, this may be possible.

For other firms, there may be a wider choice. This problem has been addressed by so-called “concurrent engineering”, where a group of experts collaborates to produce a design. However, what happens after ten years if a spare part is to be made, maybe on the other side of the world? The manufacturing method for a one-off spare part is usually different from that chosen for mass production. If the changes made for a manufacturing are mixed with the model it may be difficult to disentangle them later. This is a general problem. What would be useful is a design decision history description, but this does not yet exist. The construction history of the CAD system is simply a record of which operations were used to create a design, not why they were used. For this reason it is useful to structure the modelling sequence logically.

Note that using a CAD system effectively is not just about creating a shape but also about communication between different partners in the production chain.

1.2 Rotational Piece

The next exercise concerns a rotational part, shown in Figs. 1.26 and 1.27. As with the linear extruded part, this is made by creating a basic shape and then adding elements incrementally.

1.2.1 Modelling Method

The part is basically rotational, so the base shape is made by creating a profile and extruding it round an axis.

1.2.1.1 Step 1: The Basic Shape

The initial profile is shown in Fig. 1.28, and the result of rotating this around an axis is shown in Fig. 1.29.

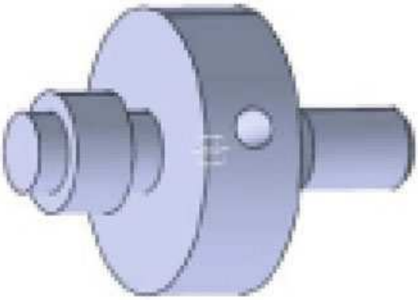


Fig. 1.26 Simple rotational part

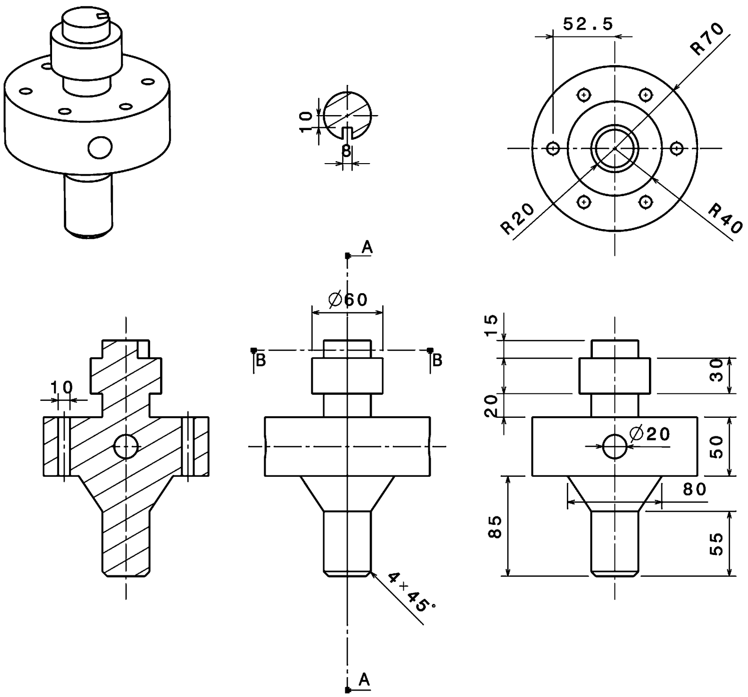
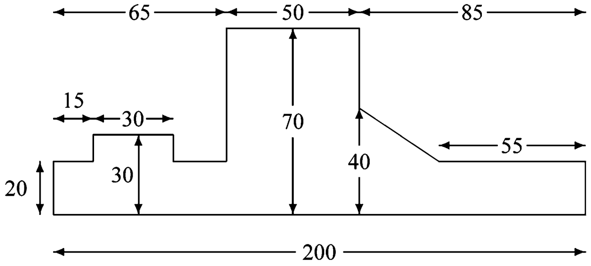


Fig. 1.27 Simple rotational part drawing

Fig. 1.28 Example 2, Basic shape



Note the same comments made in step 1 of the linear extrusion exercise apply here as well. There are two philosophies, one to create shapes in absolute positions, the other to make the shape elements relative and ignore the exact positions. Here it is slightly preferable to have the object in a known position because of the hole making operation in step 3.

There are several variants that can be made with the same profile shape. Several different shapes can be made by choosing the parameters differently. Choosing the same axis, labelled "A" in Fig. 1.30, but rotating through 270° only, you get the variant at the top left of Fig. 1.31. Choosing edge "B" as the axis gives the shape at the top right of Fig. 1.31. Choosing edge "C" as the axis gives the shape at the

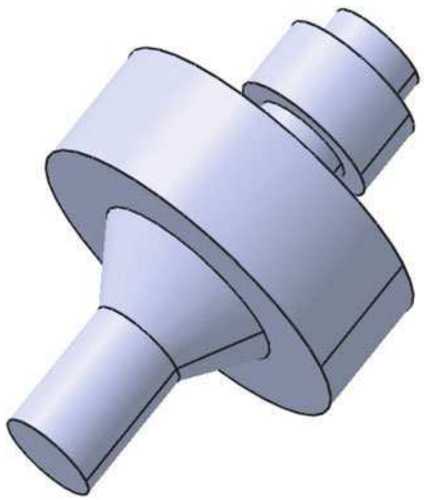


Fig. 1.29 Basic shape for rotational part

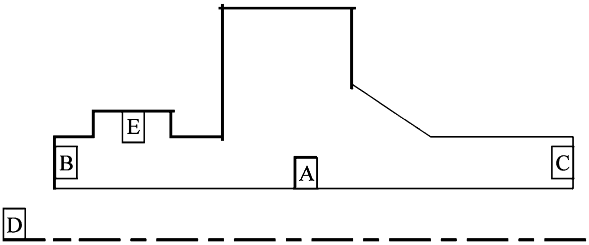


Fig. 1.30 Example 2, basic shape

bottom left of Fig. 1.31. If an axis is used which is not an edge of the profile, as axis "D" gives a through hole along the axis, as at the bottom right of Fig. 1.31.

What happens if you choose edge "E" as the rotation axis? In this case the system usually complains and does nothing. This is because the axis cuts through the object and rotating the profile about this axis will create a problem object. If the rotation is less than 180° then the edge used as the axis will become a so-called "non-manifold" edge, an infinitely thin neck between two solid portions. This topic is explained further in Chap. 6.

1.2.1.2 Step 2: Adding a Chamfer

The next step is to chamfer one edge to give the object shown in Fig. 1.32.

This is done in one step by selecting an edge and specifying a depth. The chamfer operation, described in Sect. 4.7, is a classic example of what were called "local operations", another example is the blending operation. Local operations were operations that took an isolated part of the object and made a specific change. Sometimes such changes are difficult to create in another way. This is something

Fig. 1.31 Rotational variants

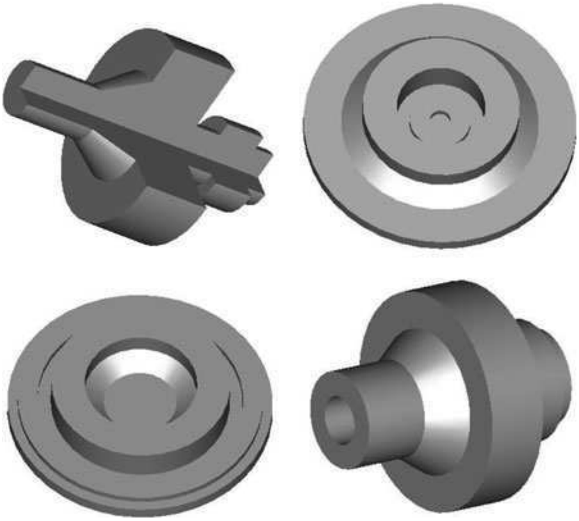
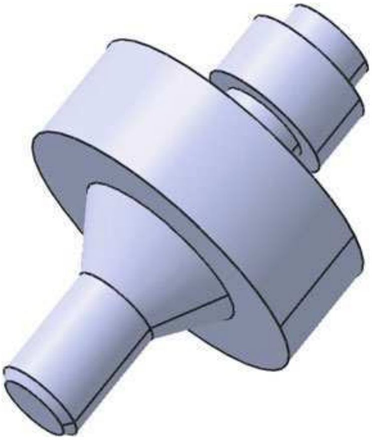


Fig. 1.32 Basic shape with chamfer



that provided an advantage to Boundary Representation, or B-rep, (Sect. 2.7) over Constructive Solid Geometry, CSG, (Sect. 2.6).

1.2.1.3 Step 3: Adding the Hole Perpendicular to the Rotation Axis

A hole is made through the object, giving the object shown in Fig. 1.33. The perpendicular hole is interesting because you need to describe shapes on planar surfaces and, in this case, the logical surface is cylindrical.

Although it is possible to inscribe a shape on a curved space by creating the shape in parameter space, this is probably not useful in general. A better technique

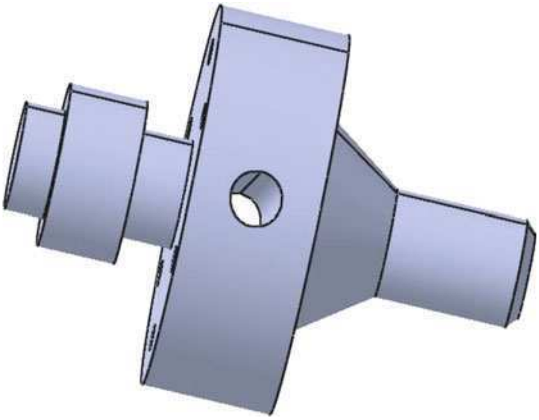


Fig. 1.33 Hole perpendicular to the rotation axis

is to create the shape on a plane and then project it onto the curved surface or, the solution usual in CAD systems, to create the shape on a plane, extrude this shape to create a volume and then subtract the volume to create a hole.

If the profile shape from step 1 was created at an exact position then it is possible to use the same plane to create a circle in the right place and then extrude this, with a cutout operation, to create half the hole. If the CAD system allows symmetric extrusion then this can be done in one step. If not, it may be necessary to extrude the bottom of the hole in a second step to cut out the second half of the hole. Yet another possibility is to create a plane tangent to the cylinder and use this to define the circle to be extruded through the object to create the hole.

1.2.1.4 Step 4: Adding a Hole Pattern

A common concept in CAD is the notion of a pattern, rectangular or circular. In this case the pattern is circular and might correspond to a set of fixation holes. The result is shown in Fig. 1.34.

Patterns are usually created in current systems by copying solids and transforming the copies into regular patterns. In some systems this is also done on operations which produce a solid as a partial result by copying and transforming the intermediate solid.

In this case the holes are simple cylindrical holes, unlike those in step 7 of the linear extrusion exercise. These holes could be done using a hole-making operation, if one exists, or by making one hole-shape (a cylinder in this case) and making a pattern of this, or as separate extrusions. This depends on the possibilities offered by the CAD system. The least desirable is the last one, to make the holes separately using extrusions because the holes then appear in the model as unrelated shapes.

Note the positioning of the pattern, refer to Fig. 1.27 if necessary. For this object the position of the hole created in step 3 is somewhat arbitrary because the

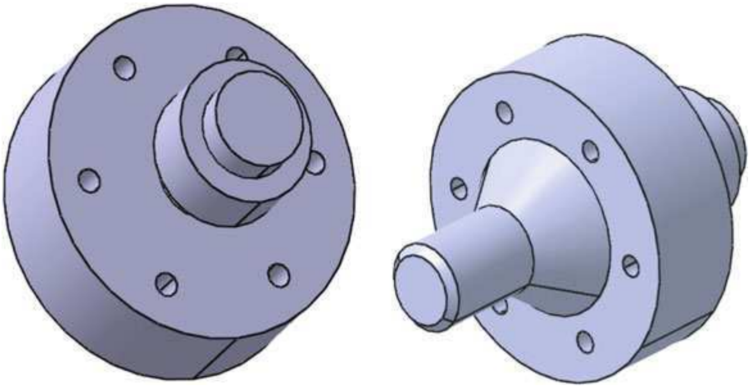


Fig. 1.34 Adding the hole pattern

basic part is rotationally symmetric. The hole pattern should be positioned relative to this through hole because the through hole provides an “anchor” for the angle of the pattern about the axis of rotation.

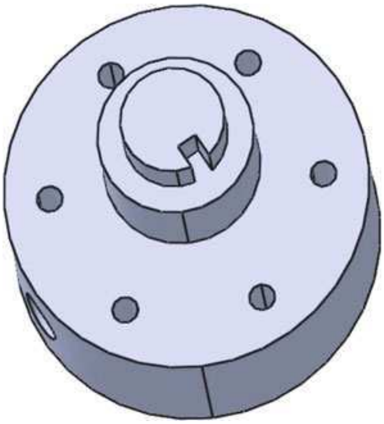
1.2.1.5 Step 5: Adding the Keyway

The keyway, shown in Fig. 1.35, is a model part for fixing a part to an axis to avoid rotation.

This can be done by inscribing a rectangle on the end face and then extruding it downwards with a cutout operation, Fig. 1.36.

The width of the rectangle is 8 and it is placed ten units from the axis of rotation. It is perpendicular to the through hole.

Fig. 1.35 Adding a keyway



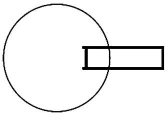


Fig. 1.36 Example 2, Basic shape

1.2.2 Questions

1.2.2.1 Why Use a Chamfer, Why Not Incorporate the Angle in the Basic Extrusion Shape?

The answer to this question is also related to the comments about creating models in a logical manner in the linear extrusion exercise. If this angled edge is part of the basic design then it should be part of the profile, if it is there for functional reasons then it should be added as a chamfer. Chamfers are added to aid insertion of pieces or to remove sharp edges, for example.

This also illustrates one of the difficulties of creating exercises with a known solution. Because you, the reader, can see that there is a chamfer then it is possible to consider adding it as part of the profile. Modelling a known shape is not design and in this exercise the chamfer was considered as a separate element. There is also another problem in communication, that of why the chamfer is there. If it is there to avoid a sharp edge, for example, then it may be necessary to add if the part is made by turning from a cylindrical stock. If the part is made by casting, say, then the edge which has been chamfered may be blended for the moulded part. This blend would be adequate for removing a sharp edge, but if this reason is not evident for a manufacturer then an extra manufacturing operation may be planned to create the chamfer from the blended part. This increases cost. Communicating design decisions is not easy but it is helped by having a logical modelling sequence rather than just creating the right shape.

1.2.2.2 Why Is There No "Make Keyway" Operation in a CAD System?

Maybe when you read this, or during the life of this book there will be a keyway operation, but at the moment there isn't, so this step can be used to make a small point.

The existence or not of an operation depends on several factors. Many operations present in modern systems have a long history going back to the start of solid modelling. Extrusions, circular and linear, were in Braid's original system [1]. Boolean operations, at least in a simplified form, were there too, and these have long been a central part of solid modelling. Chamfering appeared at the end of the 1970s as did drafting, tweaking, symmetry and other experimental operations. Other operations, such as shelling and sheet modelling techniques appeared during the first half of the 1980s. Sometimes these have found their way into commercial software, sometimes not. Development of an operation, though, costs time and money. If this development is not likely to show a commercial return then there is less likelihood that it will appear. In contrast, if a strong industry, such as the aeronautics or automobile industry ask for something then it may well appear because of the size of those market sectors. An operation such as the keyway making operation can be done relatively simply by other means, so is less interesting. However, the need to provide a modelling history closer to a design decision tree demands such operations. This topic will be discussed further in Sect. 4.17.

1.2.2.3 How Do You Decide the Angle of the Keyway Round the Axis?

This question is about the use of multiple objects in design. Some systems now impose a limitation that only one object can be designed per file. This seems to be an arbitrary restriction which is imposed as part of the system philosophy rather than for technical reasons. This can be a problem because it is normal that parts are designed in an environment with other parts. The reason for the one-object-per-file restriction seems to be that it is sort-of logical. CATIA v5 has the one-part-per-file, but you can design in a product environment to have other pieces around it. IDEAS ms9, at least, allowed multiple bodies in a file. See Sect. 1.8.

The point is, and why the question was there, that the keyway is dependent on two different parts, the rotational axis of this exercise and the part to which the axis is connected. If the second part has to be oriented specifically, say with respect to the hole created in step 3, then there has to be an agreement between the two parts. The keyway is, therefore, a feature between two parts, which is not something that is generally considered when talking about features. Multi-body features such as the keyway and bolt holes were considered by Csabai [2] and will be mentioned further in Chap. 11.

1.3 Hollowed Lofted Part

The third solid creation exercise concerns lofting. Lofting is a technique which comes from the ship-building and aircraft industries. Apparently, the term comes from the English word “loft” meaning the space at the top of a house, This was the only space big enough to create the large scale drawings needed for the ship or aircraft. The example, shown in Fig. 1.37 is not intended to be anything in particular, just an arbitrary shape.

The lofting technique creates a shape “interpolating” a series of sections. The notion of interpolation and lofting are described in detail in Sect. 4.14.

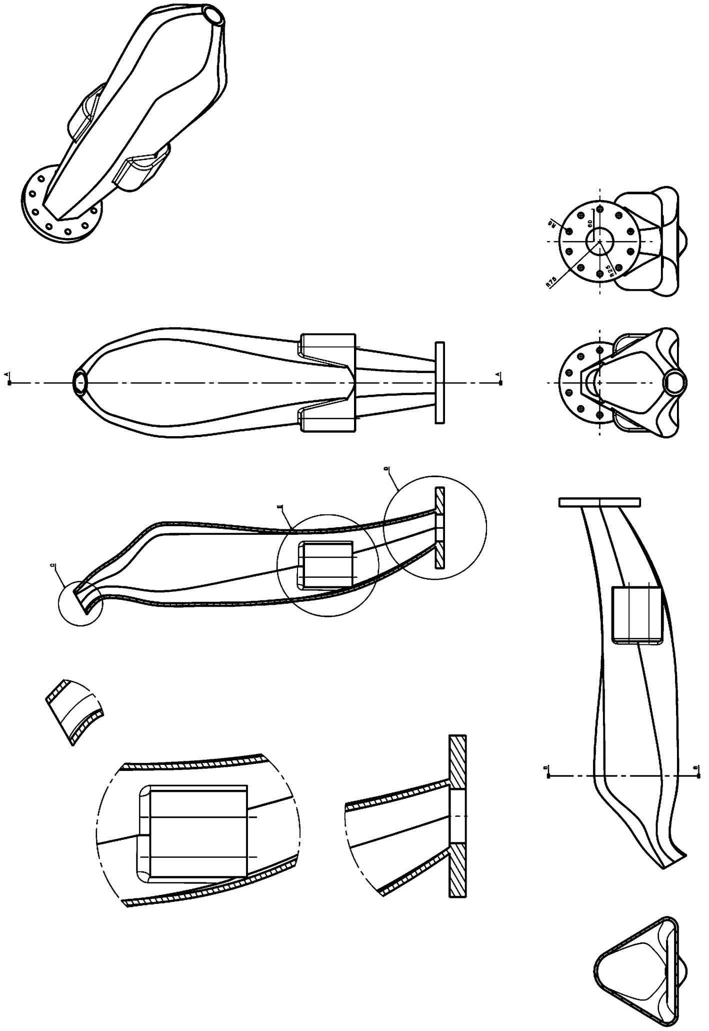


Fig. 1.37 Simple hollow lofted part drawing

1.3.1 Modelling Method

As with the other examples, the modelling method consists of creating a basic shape and then adding elements incrementally. In this case, though, the geometry is more complex. The idea of the exercise is not to create an exact geometry but to illustrate lofting as a method for creating a base shape and one or two other facilities. If you want, vary the sections from those described, keeping in mind that they should all have six edges. Later on this will be varied, but for this initial exercise this is a suggestion.

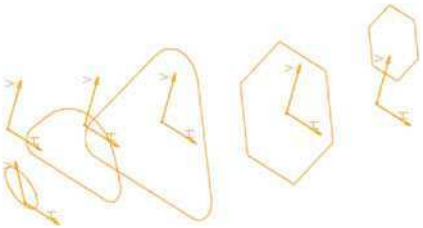
1.3.1.1 Step 1: Creating the Sections

It is easiest to perform the lofting operation if the sections have the same topology. However, this is not strictly necessary. Sections could be considered as a circuit. The start point is considered as 0 and 1, and the other vertices have values between 0 and 1 depending on the length of the section. This is probably unclear now, it will be discussed later in Sect. 4.14. What it means is that it is necessary for the system to be able to match points on the sections in order to create the geometry.

The sections are shown in Figs. 1.38 and 1.39. These sections lie on different planes. The first is on the plane z = −500, the second on the plane z = −250, the third on the plane z = 0, the fourth on the plane z = 100 and the fifth on a rotated plane (the plane z = 200 rotated 45° about the line y = 0, z = 200.

If the system allows sketching on arbitrary planes then the first step is to create the section planes as free-standing geometric entities which are then referenced for sketching entities. If the system has only one sketching plane then this is used to create the sections which are then translated to the correct positions.

Fig. 1.38 Lofted part sections



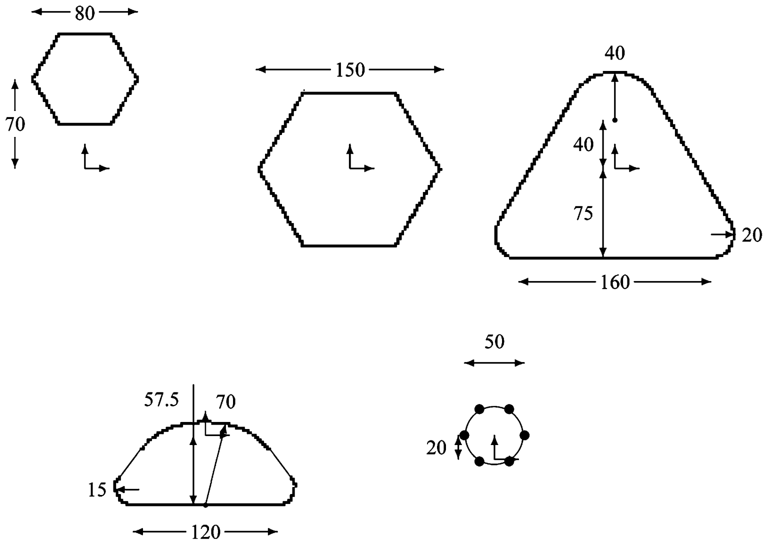


Fig. 1.39 Lofting example sections (not to scale)

The shape and the positioning of these sections is not actually critical for the exercise because the exercise is there to demonstrate a technique. The reference point for the sections is the point X = 0, Y = 0 on each of the sections.

1.3.1.2 Step 2: Creating the Loft Between the Sections

The lofted shape is shown in Fig. 1.40. Be careful about the start points and directions for the sections. There are various ways of showing this when selecting the profiles, but there should be a start point and a direction. This establishes the relationship between matching points on the sections. The curves passing through, or interpolating, these points and the curves of the sections form a grid bounding the surfaces of the lofted part.

Figure 1.41 shows some simple sections. In this figure section 1 starts at p1, and continues through p2, p4, p3 and back to p1. Section 2 has the order p5, p6, p8, p7, p5. Section 3 as the order p9, p10, p11, p12, p9. Section 4 has the order p13, p14, p16, p14, p13. As shown in the figure, the points are matched in sets: (p1, p5, p9, p13), (p2, p6, p10, p14), (p4, p8, p12, p16) and (p3, p7, p11, p15). Interpolating these

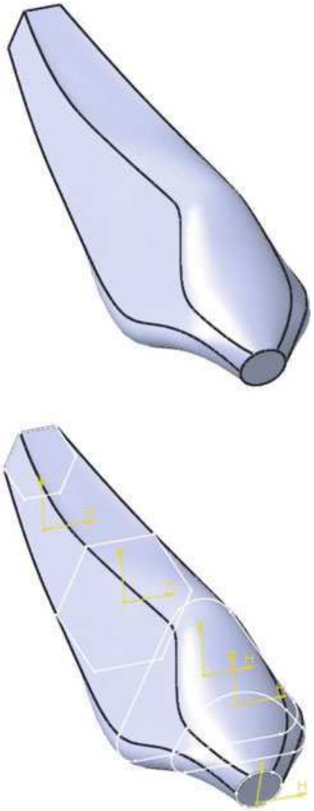


Fig. 1.40 Lofted part basic shape

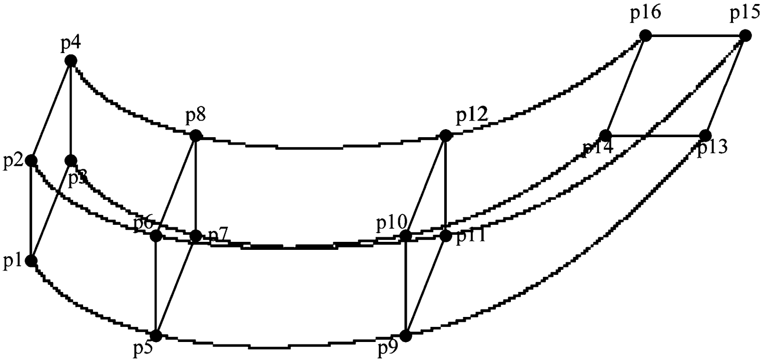


Fig. 1.41 Simple lofting sections

points gives the side curves for the surfaces. This may be one curve or several curves, depending on the system. If there are several curves, though, you have more options for shape control. One option is to control the magnitude of the tangent vectors across the sections. Another option is to let the curves "break" at the sections, meaning that the interpolation is done in sections so that the curves are not necessarily curvature or tangent continuous at the sections. Yet another option is to make the interpolation curves normal to the start and end section planes. The way this is done is explained in more detail in Sects. 5.5 and 5.6.

If the start points are not aligned then the curves interpolating, or passing through, the points becomes twisted. Imagine that point p6 in the second section is taken as that start point. The point sets would become: (p1, p6, p9, p13), (p2, p8, p10, p14), (p4, p7, p12, p16) and (p3, p5, p11, p15). This is why it is necessary to take care in specifying the start points for the sections.

If the section orientation is not respected then you get a different effect. If, say, the second section is reversed but with the same original start points you would get the point sets: (p1, p5, p9, p13), (p2, p7, p10, p14), (p4, p8, p12, p16) and (p3, p5, p11, p15).

There are many variations. Unfortunately the system cannot tell what you want automatically. The technique of lofting is useful because it is general, but because it is general all (or almost all) shapes that can be made are permitted.

Once the curves have been interpolated then it is possible to define a set of surfaces bounding the new volume. The first surface, for example, is bounded by the curves from p1 to p5, p5 to p6, p2 to p6 and from p1 to p2. This is explained in Sect. 4.14 and Chap. 5 in more detail.

1.3.1.3 Step 3: Creating the Thickened Part

Figure 1.42 shows the result of this step. This is a part which also has to be hollowed but is different from the lofting. The aim of this step is to show how extrusion is a compound operation involving a simple fextrusion and a Boolean operation to join the new extrusion and the lofted part.

The shape itself is a simple rectangle with rounded corners. It is sketched on the plane of the second section and extruded 100 units in the direction (0,0,1).

1.3.1.4 Step 4: Blending the Thickened Part

The thickened part can be blended to merge it into the lofted shape, as shown in Fig. 1.43. This is another illustration of the use of blending. There can be some difficulty in determining the blend radius. For the example a blend radius of 5 was used on some of the edges of the extrusion from step 3.

Done at this stage the blended edges are included in the hollowed part, they become part of the basic object shape.

1.3.1.5 Step 5: Hollowing the Object

This part creates a thin-walled part from a solid, Fig. 1.44. The basic operation is relatively simple and is described in Sect. 4.9. The object to be hollowed is copied,

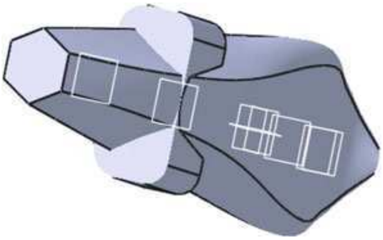


Fig. 1.42 Lofted part thickened part

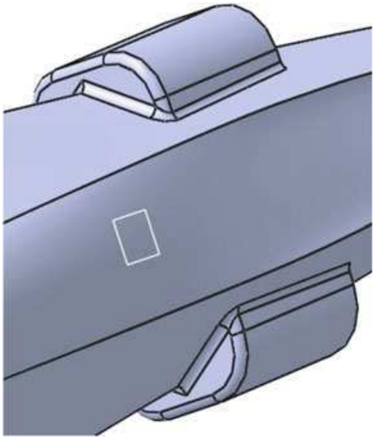


Fig. 1.43 Lofted part: blended thickened part

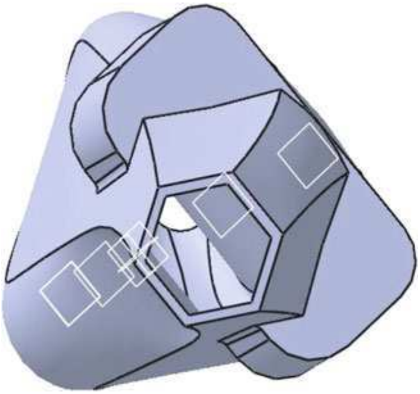


Fig. 1.44 Lofted part: shelling

the copy negated and then the negated copy is put back into the original object as a cavity. The original object and the negated copy are then offset, according to the user's wishes, to create the final part. It is usual to allow some faces of the object to be removed at this stage and possibly to allow different thicknesses. In this step the wall thickness is 5 towards the interior and the front and back faces of the object are removed.

1.3.1.6 Step 6: Extruding the Flange

Figure 1.45. This involves inscribing and extruding a circular shape on the back face of the object. The shape is a circle, radius 75 mm, extruded 15 mm.

1.3.1.7 Step 7: Creating the Hole Pattern

Figure 1.46. This is usually done in two parts. The first part creates one hole, or one hole shape, as a prototype and the second part makes the pattern from the prototype.

If the system allows creation of a hole then the hole can be created directly. At the time of writing this operation may well be done by creating a hole-shaped solid and then subtracting this from the solid. Creating a pattern of a hole involves copying the solid, making a pattern of this and then subtracting all the copies from the object.

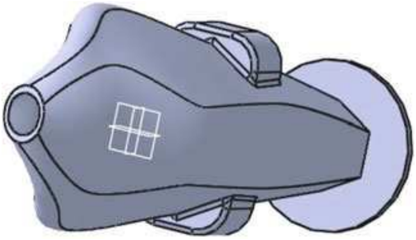


Fig. 1.45 Lofted part: flange shape

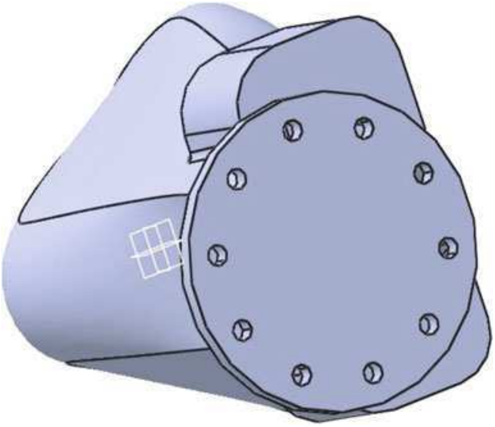


Fig. 1.46 Lofted part: flange hole pattern

Another hole-making method requires you to make the hole shaped solid yourself, make a pattern of the solid and then subtract this explicitly from the original object. These two are very similar, the hole-making operation step just hides the mechanism.

1.3.1.8 Step 8: Piercing the Flange

Figure 1.47. This is just to allow access to the interior of the object. A circle, radius 25 mm, is created in the middle of the flange and then extruded 15 mm towards the interior to pierce the flange.

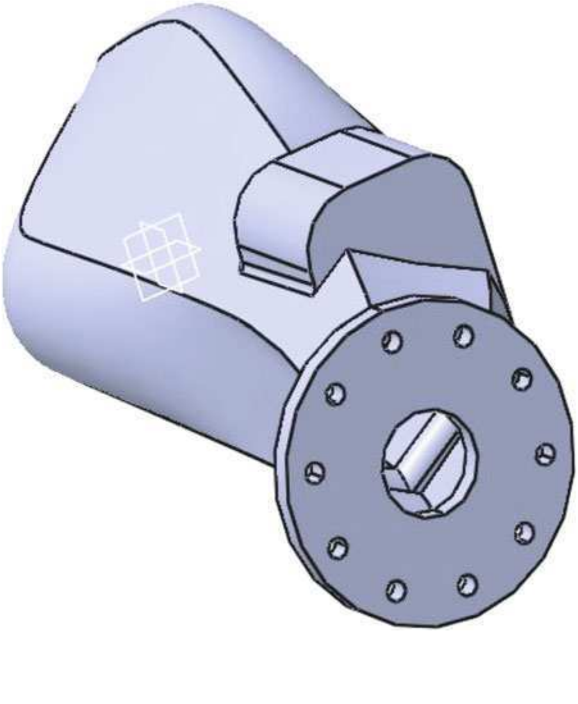


Fig. 1.47 Lofted part: flange piercing

1.3.2 Questions

1.3.2.1 What Happens If the Start Point and/or Section Directions Do Not Match?

This is one of the key considerations in using lofting. Look at the objects in Fig. 1.48. This object is created by interpolating square cross-sections. The cross-sections are simply translated. The "obvious" loft would simply match each edge to the translated edge. However, the twisted loft is also a valid and useful application of lofting. The system could, perhaps, propose start points in a more intelligent manner by trying to match points, but in the end it is the user has to decide what is required.

1.3.2.2 What Happens If the Sections Have Different Topology?

This can be handled but the question is whether or not it is what you want. A nice demonstration in some systems is to produce a lofted shape between a square and a circle. These have, of course, different topology, so the question is how to match points. One way of doing this is to treat each section as a paramatrised path with the start corresponding to parameter value 0 and the end point corresponding to parameter value 1. If the section is closed then these are the same point, but every other vertex on a section has a single parameter position. To match vertices, then, you compare the parameter values. If the parameter values of vertices in successive match then the vertices are used. If a vertex cannot be matched then a point at the same parameter position on the next section is calculated as an intermediate point (Fig. 1.49).

1.3.2.3 Why Can't the CAD System Judge If the Lofting Result Is Correct or Not?

This is very difficult. The software cannot really tell you if there is an error unless the error is really obvious. However, if lofting creates a surface which is

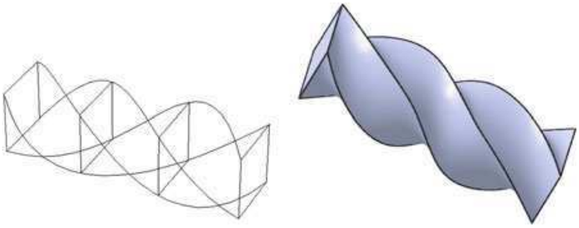


Fig. 1.48 Twisted lofts

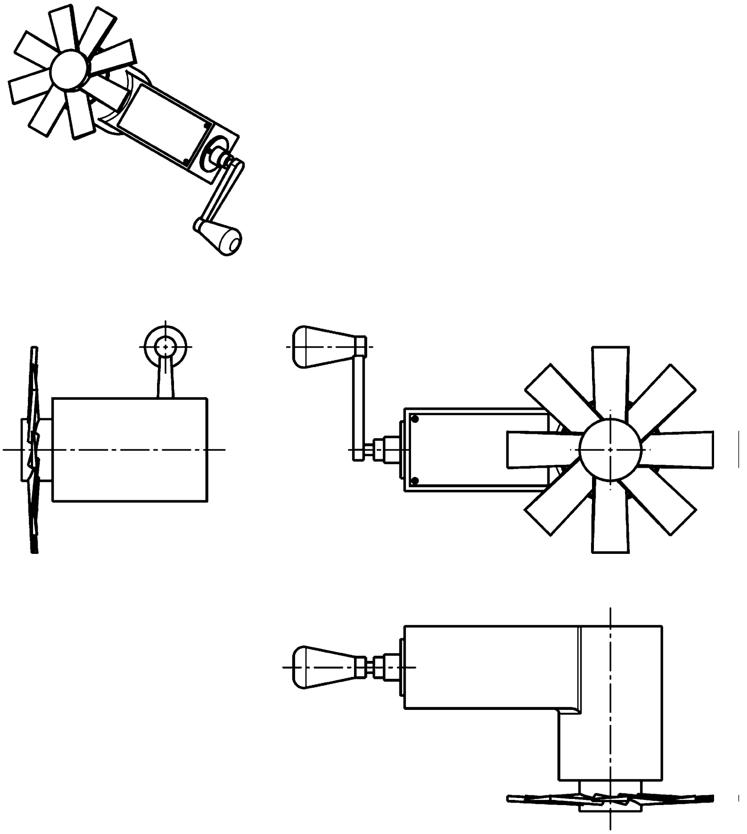


Fig. 1.49 Simple assembly

degenerate or self-intersecting then this is not necessarily wrong unless the degeneracy or self-intersection occurs within a face. If the lofting is a solid creation tool then this can be checked, but not if lofting is used to create a complex surface.

1.4 Creating Assemblies

The previous exercises were concerned with single parts. As indicated in the rotational part exercise, though, products are often compound models. The set of single parts is called, here, an assembly although it may have different names in different systems.

1.4.1 Creating the Parts

To illustrate an assembly the exercise uses a simple structure with six separate parts, shown in Fig. 1.49. These are shown separately in Figs. 1.50, 1.51, 1.52, 1.53, 1.54 and 1.55.

First, make the objects shown in the drawings.

1.4.2 Creating the Assembly Structure

The next step is to import these objects into an assembly structure. Determining an assembly structure is not always straightforward. However, it is important to understand the mechanism used to represent assemblies in order to use them correctly. This is described in Sect. 2.10 and, in more detail, in Chap. 13. Here, though, all the parts are unique and appear only once in the assembly so they may be imported simply one after the other, as shown in Fig. 1.56.

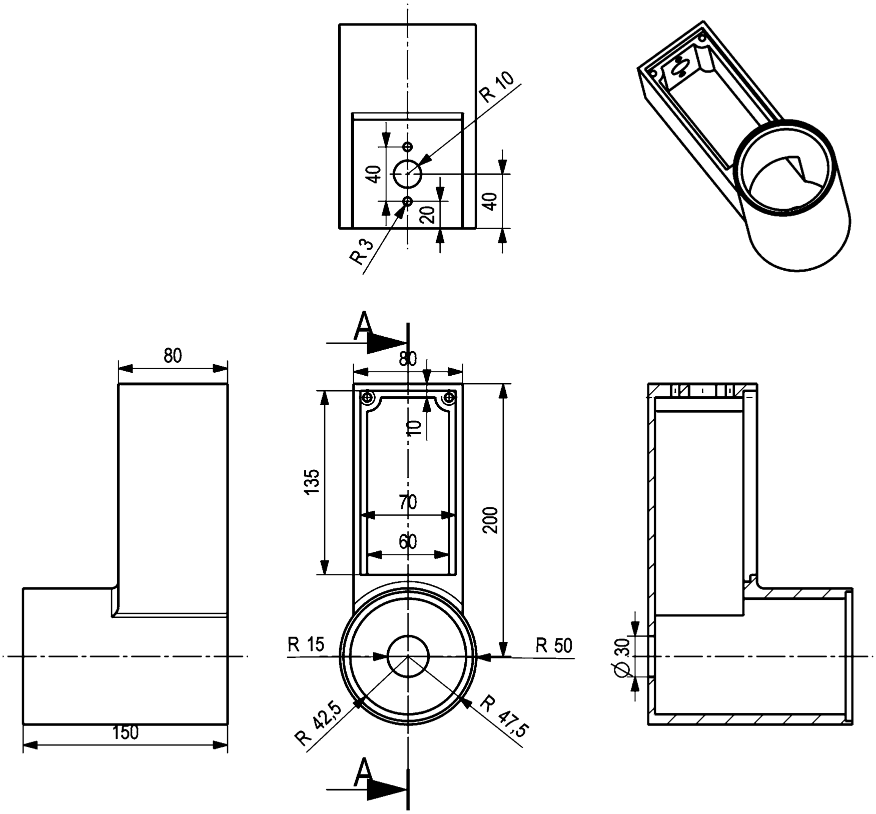


Fig. 1.50 Simple assembly base object

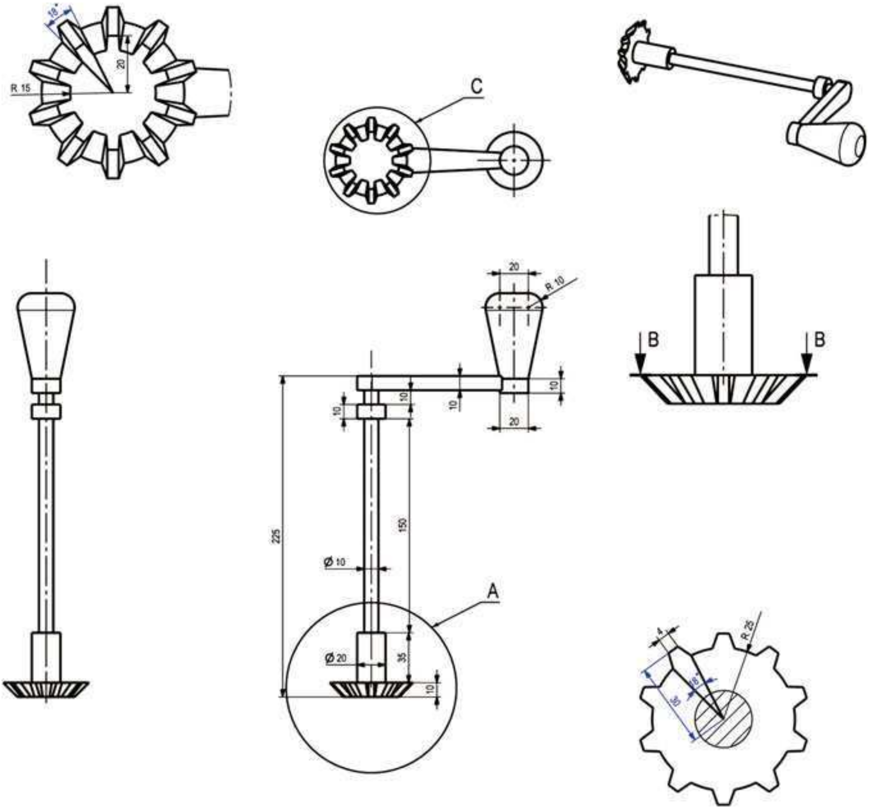


Fig. 1.51 Simple assembly winder

The items in the assembly are termed “instances” and the assembly is a tree structure of these. The instances refer to an object model or a sub-assembly for the part geometry. They also have an associated placement, or transformation, information entity. Such transformation entities could contain a variety of information such as scaling or symmetry, but should only contain translation or rotation information, as explained later. The transformation information describes how the object which is instanced is moved from the position in which it was defined to the position in the assembly. So, if an object, say a bolt, is used ten times in an assembly then there is one bolt model and ten instances which refer to the single model. This is illustrated in Fig. 1.57. Each of these ten instances has a separate transformation matrix so that, when drawing the assembly for example, it appears that there are ten bolts. If the assembly is set up correctly then a Bill Of Materials (BOM) can be calculated directly from the assembly. Each object model in the assembly is one element of the BOM, counting the instances referring to the models gives the element count.

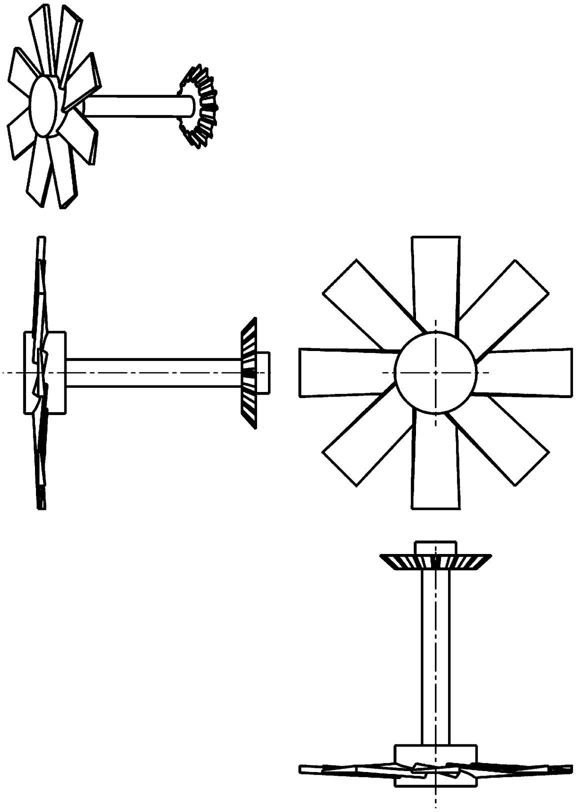


Fig. 1.52 Simple assembly rotor

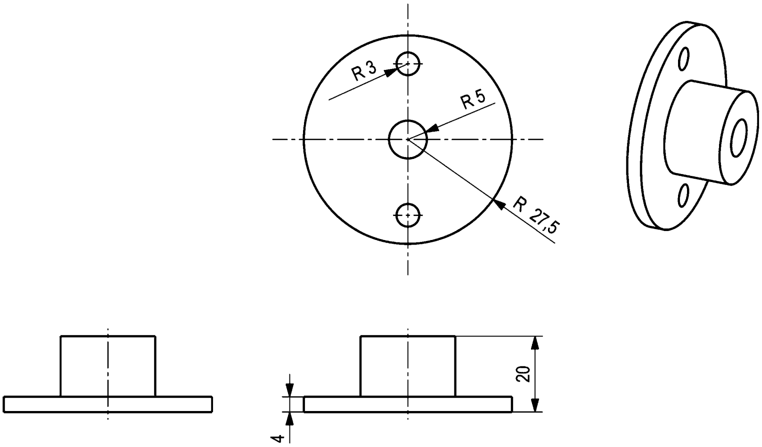


Fig. 1.53 Simple assembly ferrule

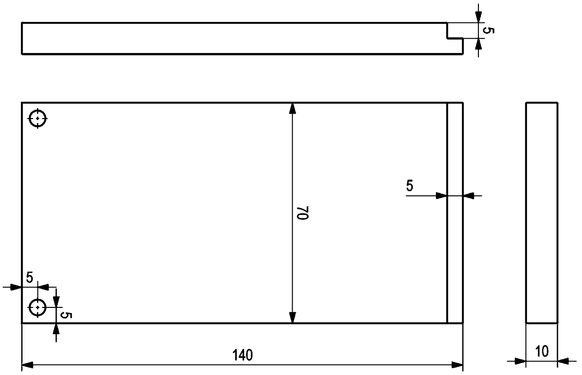


Fig. 1.54 Simple assembly rectangular cover

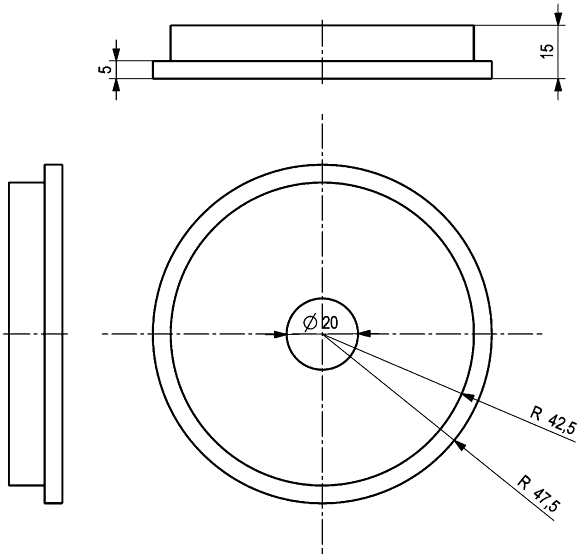


Fig. 1.55 Simple assembly circular cover

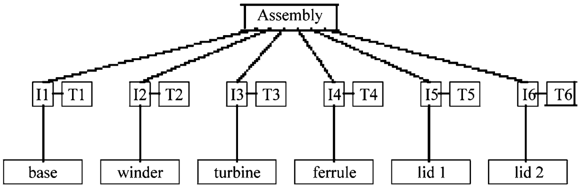


Fig. 1.56 Simple assembly structure elements

The transformation information can be set up explicitly, by saying how to translate and rotate a part model into place in the assembly. This gives a correct static picture of the assembly. However, if a part in the assembly is moved then it is necessary to reposition all other related parts separately. In modern CAD systems the user sets up relationships, or constraints, between elements and the corresponding transformation information is calculated from these relationships.

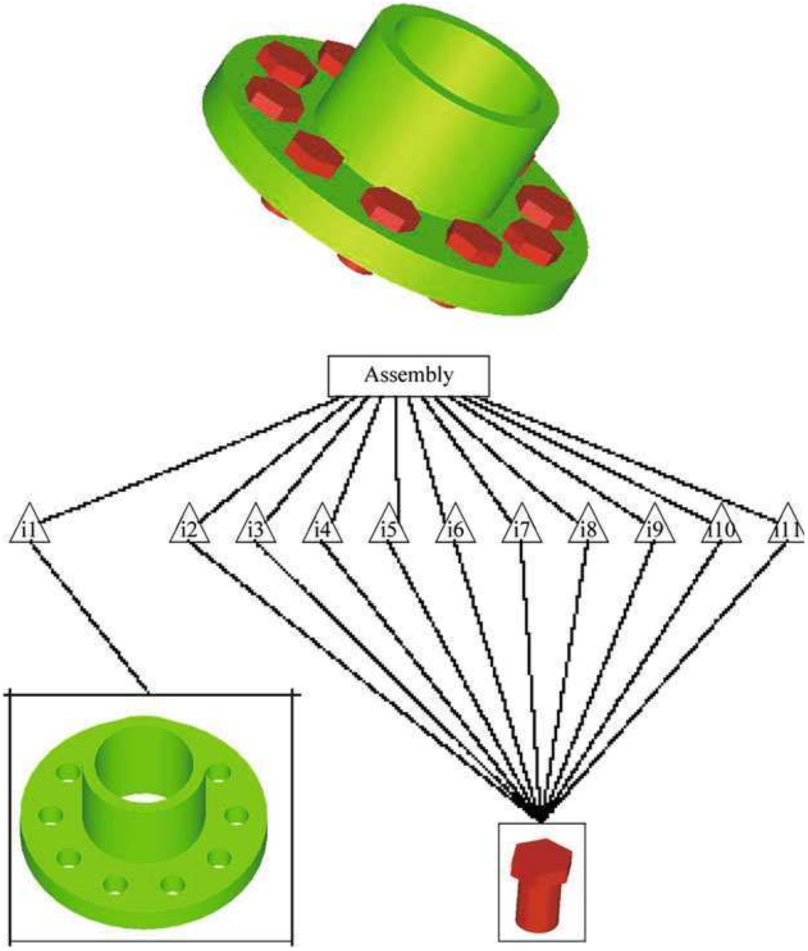


Fig. 1.57 Simple instance structure

If an item is moved then the transformations of related elements are automatically recalculated. The relationships for the simple assembly structure are described in the next section.

1.4.3 Constraining the Elements

Constraints remove degrees of freedom between two objects. Objects have six degrees of freedom: three translations and three rotations. Constraints remove different degrees of freedom between the objects. The constraint combinations are not always the same as the physical assembly methods used.

The first constraint is to fix the base part of the assembly. This means that all degrees of freedom are removed and that other elements will be moved to this.

Other constraints relate planes, lines or points and these are described in Sect. 13.4. There are six of these:

1. Plane–plane
2. Plane–line
3. Plane–point
4. Line–line
5. Line–point
6. Point–point

Note that there are directions for lines and planes. A plane–plane constraint can have the plane normals aligned or in opposite directions. It is common to need to align planes with opposed normals because this corresponds to one object face is in contact with the face of another object, but aligned normal constraints are sometimes useful, as well. Lines, too, can have the same direction or reversed direction.

Another common constraint is to align a cylindrical shape with a cylindrical hole. Note that this is a variant of the line-line constraint because the axis of the cylinder is aligned with the axis of the hole. It would be harder to align a square peg in a round hole, for example, because there is no convenient central line for the square peg.

So, take the flange and align the plane of the back face with the end plane of the base object, Fig. 1.58. To do this it is usual to select a planar constraint function and then the two faces. Planes can be thought of as being defined by a point and a normal vector. The planar constraint function therefore translates object 2 so that the point of plane 2 lies in plane 1. Object 2 is then rotated so that the normal of plane 2 is parallel to the normal of plane 1. There are two solutions to this, one solution has the two face normals aligned, the other parallel but in opposite directions. Some systems distinguish between these two. If the normals are parallel but in opposite directions then the two objects are in contact. This is quite a common case so is sometimes preferred in CAD systems.

Note also that the position of object 2 with respect to object 1 is not constrained. Sometimes the centre of one face is placed at the centre of the second face. The constraint itself just removes the freedom of rotation about the x and y axes and translation along z (if the plane normals are considered as the z axes). Translation in the x and y directions and rotation about the z axis are not fixed. This is done by subsequent constraints.

The first constraint can be a natural choice, but subsequent constraints are not always obvious, depending on how the constraint solver works. One choice might be to take one of the small holes in the flange and align it with one of the small holes in the back face of the base object. Aligning two cylindrical holes is usually done by aligning the axes of the two holes. Naturally you would then repeat the process for the other hole to fix the flange completely.

One problem is that this will probably not work. It says “probably” there because commercial software makes advances all the time, and this may change.

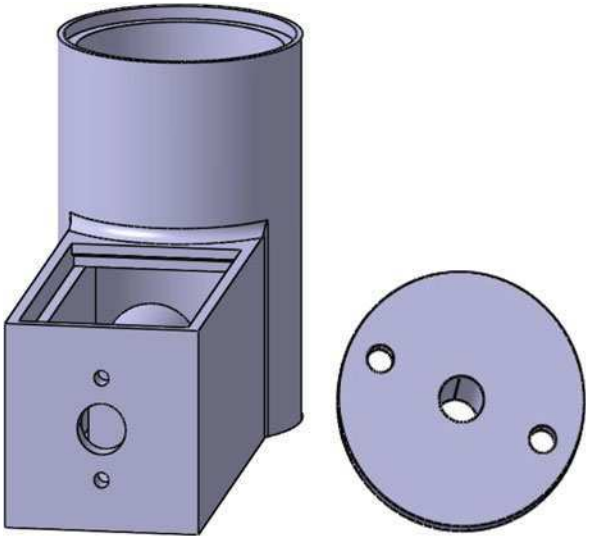


Fig. 1.58 Constraint between base and ferrule

What is wrong is that the same constraints are specified several times. If, having created the planar constraint, you specify that the axis of one of the small holes in the flange is to be aligned with the axis of a small hole in the base then you are over-constraining the objects. Assuming that the lines are specified each by a point and a direction then the line-line constraint means translating object 2 so that the point of line 2 is on line 1 and then fixing the lines to be parallel. Moving the points is all right, this just removes the x- and y- translations that were left as degrees of freedom after the first constraint was applied. However, constraining the axes to be parallel means that you are again removing the rotations about the x- and y- axes that were already removed when the first constraint was established. Your CAD system may or may not permit this. What you really should do is to specify that a point on line 2 should be on line 1 and forget about the alignment of axes. However, this is not particularly natural for a user.

The real problems come, though, if you try and align the other hole. Since the flange and the base object are both rigid objects what you are implying is a constraint between the relative positions of the holes in each object, not between the objects themselves. If, for example, the hole axes in the flange are further apart than the axes of the holes in the base object then there is no way that the constraints on the two holes can be satisfied at the same time. In one system the two holes could be constrained by saying that the relative directions of the axes in the two objects should be parallel. This is not particularly natural.

Consider again what these subsequent constraints mean. I chose them above because they correspond roughly to what you would do when physically assembling the parts, screwing a bolt into one of the holes and then another into the second hole. However, this is not the way to consider the constraint process. What you really want is that the central hole of the flange should be aligned with the large hole in the base. The two smaller holes in the flange and in the base should be toleranced with respect to the central holes to achieve this effect. The small holes are what Csabai [2] terms “connection features”. In his terms you would want to establish a “rigid” or fixed relationship between the two parts and the holes are just there as a mechanism to do this. However, the constraint mechanism in CAD systems has not really been developed with this in mind. It is possible to structure the product information in a more logical manner than now, but this is a subject for research. At the time of writing the onus is on the user to get it right, so you have to think about how to do things. It is not just a case of getting a result, this is an implicit means of communication. A personal point of view is that the constraints above should be done by a planar face constraint, as above, then aligning the large hole in the ferrule with the corresponding hole in the base. This still leaves one degree of freedom, rotation about the z-axis. I would make the line joining the centre points of the small holes in the ferrule parallel to the line joining the two smaller holes in the base, but this is rather unsatisfactory.

Next take the winder and set the constraints. Again, it would be natural to align the shaft with the ferrule and with the internal structure of the base. This, though, creates a conflict. Consider Fig. 1.59.

The boxes represent elements in the model. The dotted lines represent rigid relations, because the the elements are in the same object and therefore have fixed relative positions. The solid lines represent constraints between different objects. The lines marked with an “R” represent rotational relationships fixing movement in the XY plane and allowing rotation about the Z-axis. These, together with the fixed relation marked “F” form a loop, or cycle, in the constraint graph. Such loops are potential sources of conflict, say if the base hole and the base internal hole are not aligned, in which case it is impossible to satisfy all the constraints. Such loops may cause an error message saying that the object is “over-constrained”. What you are actually implying is that there should be a positional tolerance between the internal hole and the hole in the end face of the base object. This is something else that needs to be developed further, that the functional

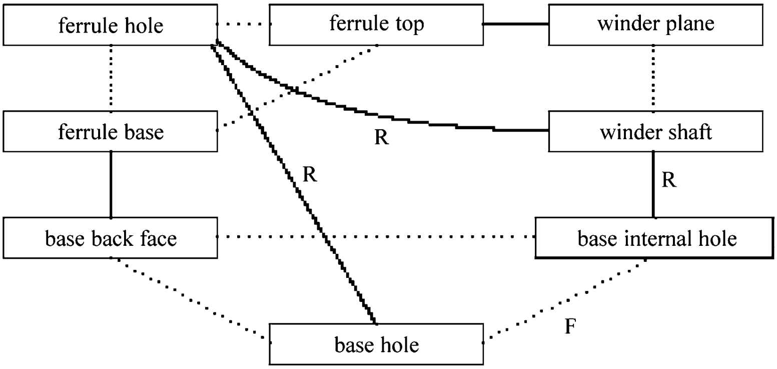


Fig. 1.59 Constraint structure

tolerance requirements are introduced in a logical way for manufacturing. For current purposes, though, it is better not to specify the coaxial constraint between the ferrule and the winder, just a co-planar constraint between the top face of the ferrule and the appropriate handle face.

The cover of the base can be inserted by having a co-planar constraint between the bottom face of the cover and the appropriate face of the base and then setting up constraints between the matching holes, say. It is, perhaps, worth mentioning that with a similar assembly task in an EPFL exercise, a student set a constraint of coincidence for two point pairs expecting the cover to be fixed. This is illustrated in Fig. 1.60. The idea was to constrain P_3 to be at the same place as P_1 and P_4 to be at the same place as P_2. Ignoring the fact that this creates an over-constrained constraint set, even if the constraint solver of your CAD system allows this, the two blocks are not fixed, there is a rotational degree of freedom about the axis P_1P_2. In the physical world this would have been fine, in the CAD world there is a problem. The reason for mentioning this is to illustrate the difference between the user logic and the CAD system logic. The CAD system cannot take into account the physical properties that disallow this rotation, because the constraint tool is mathematical. This means that it is necessary to think more about the way of establishing constraints. Hopefully the user interface will improve, but at the time of writing, constraint setting is not always easy or logical from the point of view of the user.

The circular cover can be constrained using a planar face constraint and a coaxial constraint. However, this still allows one degree of freedom, the cover can rotate about the axis. If the cover is to be screwed into the base then there may be no physical model elements which can be used to fix the two objects.

Now constrain the rotor. This involves a constraint of coaxiality and a planar constraint. The planar constraint is from the bottom of the part holding the rotor blades and the coaxiality constraint can be between the axis of the rotor and the

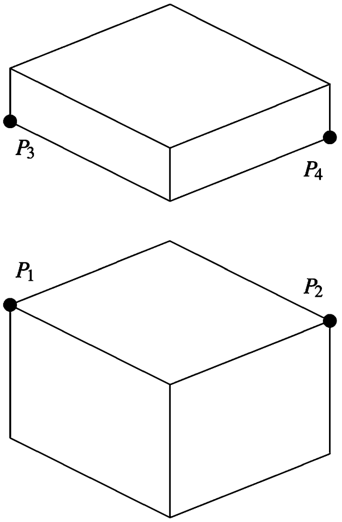


Fig. 1.60 Point–point fixations

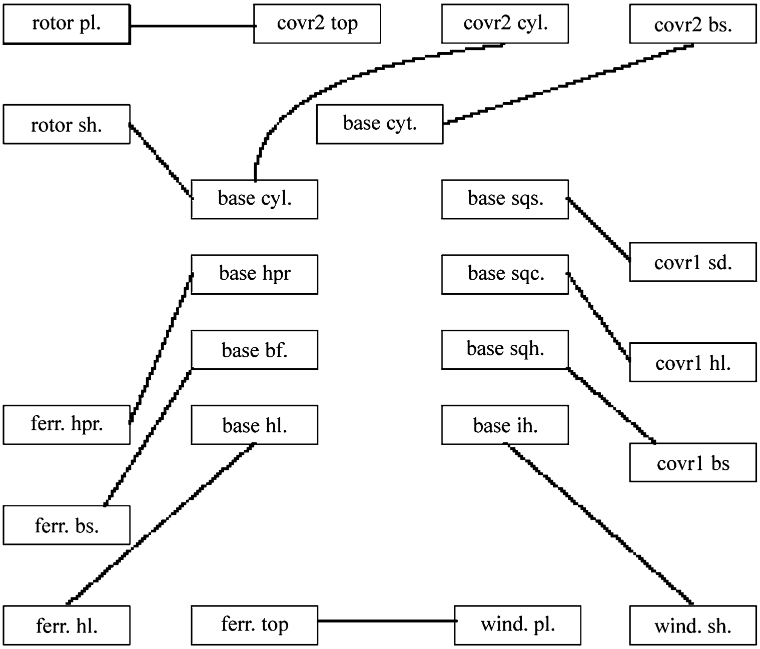


Fig. 1.61 Final constraint structure

hole in the circular cover. This is about the same as the other constraints, this leaves one degree of freedom for rotation about the rotor axis.

What about the gear wheels? It would be good to be able to impose some sort of relationship between them. This, however, may be a separate package for kinematic simulation. In any case, it is difficult to set up a set of constraints for the gear wheels and is much easier just to have a rotation ratio relation between the two shafts.

The final constraint graph, without the implicit fixed relations between elements in the same object, is shown in Fig. 1.61.

1.4.4 Questions

1.4.4.1 Can You Assemble Parts Which Physically Do Not Fit?

This is why the handle and gear were made into one part in the assembly. If you think about the assembly that you have just made, it is a really stupid assembly. This is not due entirely to inattention on my part, it is deliberately bad. The reason for this is to emphasise the need for the user to understand and think about what is being done. It is not enough to assume that if the CAD system does it, it will be possible to do it physically.

It is obvious that the winder cannot be inserted. The ferrule cannot be put on the winder, the winder cannot be inserted into the base. So, why didn't the system tell you? Actually, to analyse this properly is complex. Constraints are usually used for linking objects in an assembly. Constraints use points and lines, so it is possible to constrain a diameter 20 pin in a diameter 10 hole, for example. Checking this would require a global test which is expensive.

This is a variant on the Boolean operations (see Chap. 4) and involves performing an intersection operation on each pair of objects in the assembly. If the result is non-empty and a volume, not just surface contact, then there is potentially a problem. This is neither sufficient nor necessarily correct, though. In some assemblies there are parts which deform when assembled. These might be thin parts or parts where the material has elastic properties, such as rubber. Boundary representation models are implicitly rigid, so the above test would indicate a collision where there is, in fact, no problem. It would be necessary for the user to overrule the automatic error. For the assembly in the example all the parts just touch, there are no volumetric overlaps, unless of course the two gears are not positioned correctly. However, the winder cannot be inserted into the base. This is a very complex geometric and volumetric reasoning task which you perform much better than a computer. Even that is not enough because you could actually manufacture this assembly if you really wanted to. With techniques such as rapid prototyping you can build pre-assembled parts, although this is research rather than practice.

Don't throw up your hands in horror. What it boils down to is that you have to know what you are doing when creating an assembly, which is why you have this exercise.

There is a technique, called design-for-assembly, or DFA, which seeks to analyse products in terms of their assemblability. This, though, works at a different level from the geometry.

1.4.4.2 Do the Constraints Imposed Correspond to the Assembly Constraints of the Real Objects?

This is another question which you need to understand. Assembly constraints do not always correspond to real assembly operations. This was why the point was made about trying to use realistic constraints whenever possible. You should not just try and get a result, but try and decide logical connections between objects.

1.4.4.3 Why Did You Design the Housing and Covers Separately?

Because you had to, but should this be so? There are a lot of possibilities for extra operations in CAD systems which are not there because they have not been thought of, or because they are too expensive to develop, or because they would only be used by a small minority, or several other reasons. It is worth noting shaping operations that you need frequently and asking the CAD supplier if they exist or can be added. For some things it may be sufficient to develop a command macro, for others it might provide an interesting new operation. Being able to analyse the utility of a CAD system and specify new operations is a skill that is lacking. In general CAD systems are developed by computer specialists, because of their complexity, while the people who know what is needed are the system users. Being able to communicate user needs to system developers is an important skill to develop. In order to do this it is useful to understand the basic system elements so as to be able to formulate requests or suggestions in terms that a system developer can try. This, however, is not enough.

The example of the covers mentioned earlier are almost, but not quite, Boolean operations. A lidding operation might be done as:

- Lid shape creation
- Shape projection
- Side wall creation
- Object rearrangement
- Object separation

It is important to identify the cases where the operation is supposed to work. It might be only on planar faces or planar and cylindrical faces, or any surface. If the operation is to work only on planar faces then the face plane can be used as the sketch plane, if not, then it may be necessary to specify a projection direction as well. The lid cutout also needs to be decided, if it is in one direction or if it is done in the direction of the surface normal. For the purposes of this example, it is assumed here that any surface will be cut and the surface normal will be used as the cut direction. It is also assumed that the lid will be in one piece and not multiple pieces.

The lid shape creation process involves sketching a two dimensional shape and assigning edge profiles to the edges. The default profile is a straight profile, but a lipped profile, as in the example of the base, should be possible as well. Curved profiles might also be desirable. If the target face, which would be an input parameter, is planar then the sketch plane can be the same as the plane of the face. If the face is non-planar then it is necessary to give a point on the surface and the sketch plane would be the tangent plane at this point.

The projection procedure is described in Stroud [3]. If the projection goes outside the face then those faces should not be perpendicular to the projection direction, nor undercut the give face.

The side walls of the projected shape are ruled surfaces, these are bounded by edges which are the projections of the corner vertices. The side surfaces need to be intersected with the faces of the object to find the internal boundaries of the lid and the curves which limit the side faces.

The external and internal boundaries need to be duplicated and pairs of side walls created. These are then rejoined to create physically separate objects which are then separated into two physical objects.

That, roughly, would be an operation specification for a lidding operation. That should be enough for a system developer to produce something, although testing to produce a stable commercial version would be necessary.

Being able to specify new operations is important for the general development of CAD systems. Many of the operations appeared more than twenty years ago, and several important ones more than thirty years ago. A brief list is given at the start of Chap. 4. During the early research into solid modelling it became clear that there were many operations that could be included in Boundary Representation modelling. Introducing new operations into CAD systems helps to keep the techniques dynamic, so it would be useful to have active research in this area.

1.4.4.4 Why Was the Gear Pair Created as Two Separate Objects?

This is related to the previous question. It is preferable to maintain catalogue of standard parts such as gears and just insert these into a design. The winder and rotors were deliberately made as one piece with the gears. However, it is better to make the gears as separate objects to be attached to the winder shaft and rotor shaft. This would also improve the assemblability of the whole mechanism.

However, assuming that there are no standard gears available, it would be preferable to design the two gears at the same time rather than as separate objects in one operation. In general, though, CAD systems do not let you work with two objects at the same time in this way. What they give you is tools for creating one object and let you do the coordination. In the case in question, it is possible to design one tooth on each wheel and repeat these as circular patterns to get the complete gears. It is better to use standard gears or a complete gear design package if possible.

1.4.4.5 What About Tolerances?

A CAD system is capable of creating exact geometry with an accuracy of 10^{-6} which is much more accurate than manufacturing processes. The assembly has no tolerances assigned, that is something which the user has to think of.

Some comments about tolerances were made during the description of setting constraints. In general, though, the question of tolerances is not easy to resolve. For a start, there are different types of tolerance. There are functional tolerances and manufacturing tolerances, for example. The functional tolerances are the tolerances allowable so that the product will work. These, of course, affect the manufacturing tolerances and the assembly tolerances.

Then there is the "philosophy" of the system. In the early days of CAD there were drafting systems which were sort of electronic drawing boards. These changed the tools without changing the methods of design. Three dimensional wireframe, surface modelling and solid modelling changed design into a three dimensional task. However, for a long time now, annotated drawings have continued to be the main method of communicating shape information between several areas of production. In other words, the old methods have still not been fully replaced. Information such as tolerance information is still expected to be communicated on two dimensional drawings and not via three dimensional models. Then there is the question about what tolerances are there for. The idea behind tolerances seems to be to communicate functionality indirectly, but why not communicate this information directly? There is scope for rethinking the whole process, deciding how and what to communicate. This requires a global overview and research, though, while CAD systems need to keep their perspectives to a commercially viable level.

1.5 Determining a Modelling Sequence

To start with, it is necessary to be clear that modelling an existing object is not the same as design. This makes it difficult to set exercises because, having the final result, it is easier to decide how to use a CAD system. The exercises earlier in the chapter were intended to familiarise you, the reader, with the modelling tools. This section tries to suggest how you can model a known solid.

It is very often convenient to start modelling with an extrusion operation, linear or circular. If you want to start modelling a piece, just look at it and try and work out what the core extruded shape is and what can be added on or subtracted from it.

It is then usual to use extrusion, or sweeping operations ("extrusion" and "sweeping" are used here to mean the same thing) up and down to create various shape elements. It is best to distinguish between the basic shape and practical shape elements. The basic shape may be that needed to solve a design problem, the practical shape elements may things such as chamfers, blends, draft angles, fixation holes, keyways, and so on. The reason for distinguishing these is to try and produce a logical sequence that can be parametrised, modified by you, or modified by someone else.

The sequence itself is now usually recorded as part of the CAD system. In the early days of solid modelling systems command files were used to create objects. CAD systems themselves often created log files, lists of commands used by users to create designs. As memory restrictions have eased it has become possible to record these in memory and allow editing. The commands include the parameters used to make the object parts, changing the parameters causes a rebuild from that point to create a new object. This is described in Chap. 12.

Some operations which might be classified as "general" are: Boolean operations, extrusions (circular, linear, along a path or curve), symmetry (or reflect or mirror), shelling.

Special operations might include: chamfering, blending (or filleting), drafting, hole-making, etc. As well as having a shape changing effect, these have a semantic content which may be relevant. The operations mentioned above might be considered as finishing operations, operations to make adjustments to the design shape for some reason. Chamfering may be used to taper off sharp edges for assembly. Another reason is to smooth off sharp edges which might otherwise cause injuries during handling. Blends may be added to avoid sharp edges in moulds or to indicate that material may be left during milling for example. Draft angles are for creating sloping sides, very often for mould making, an example of an operation closely related to a specific manufacturing method. Hole making may add bolt holes and fixing elements.

Information comes last. Information is volatile because it may be made invalid during modelling if added too early. For example, if two parts having different material types are added together, which material should be left on the result object? There is no information mathematics corresponding to the mathematics of geometry and topology to handle conflicts. The fewer modelling operations you do after setting information elements the better.

Once again, it is necessary to stress that this is not design. The point of doing exercises with CAD systems is to familiarise you with shape creation tools so that, if you know what you want, you can find and use the tools needed to create it. Sometimes it may be useful to remodel existing parts, especially if they come from data exchange files with no structure. However, design is more difficult to teach because you know the "answer" in advance if you get the final shape, but you need to know how to use a CAD system before you can use it for design.

1.6 Design Exercises

The following exercises are intended to show a little of the difference between design and modelling mentioned in the last section.

1.6.1 A Bracket

Imagine that you have to design a bracket. It should support a round shape with a diameter of 200 mm. It should have three round fixation holes arranged in a triangular pattern.

Ideally you would want to be able to create individual shape elements and then join them together in a natural way. To some extent you can do this by creating geometric elements and then linking them, but the tools for doing this

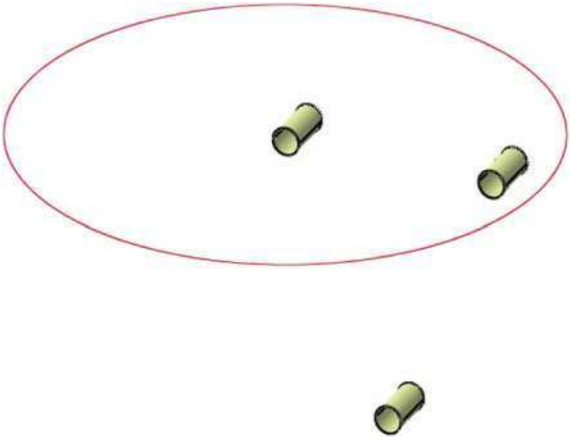


Fig. 1.62 Design elements for a bracket

are not perfect. The holes, for example, should be defined as three free-standing holes before the material surrounding them has been created. A method or doing this was invented in an old modelling system called UNI-BLOCK by Katainen [4], but this system used a different type of modelling than is used in CAD today. It could be done without much difficult as a special operation, but the basic idea of CAD systems is that you start with a solid and add the holes afterwards. There is a fair amount of scope for research into building up designs from partial models.

Figure 1.62 shows the design elements that you would expect as a start.

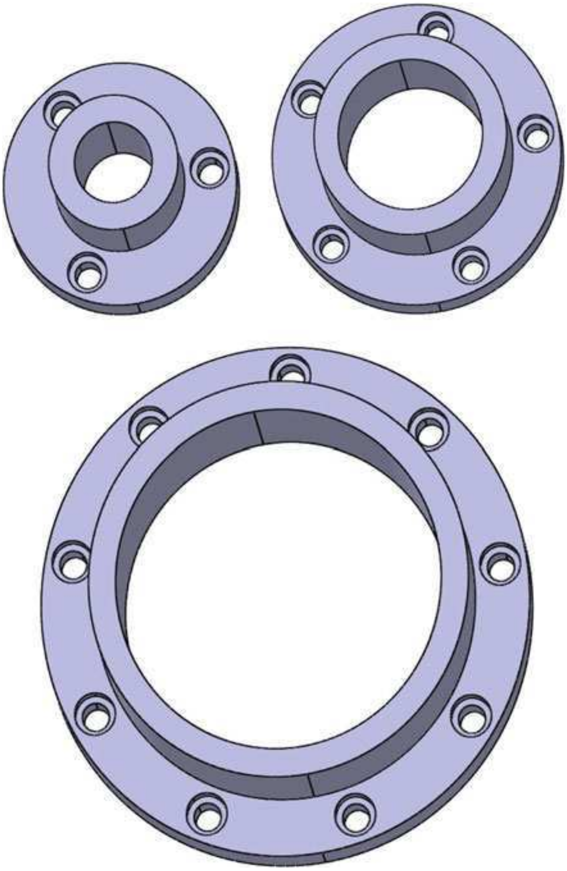
1.6.2 A Flange

For the flange you have a different set of requirements. This description is based partly on an early exercise by Helldén to create a flange. His requirements were that the number of holes round the flange should depend on the diameter of the main hole. In other words, it is necessary to build rules into a design. Some CAD systems allow this facility and it is worth getting to know. It will be discussed in Chap. 12.

Figure 1.63 shows elements in what might be a *family* of parts. The members of such families are not just simply scaled versions of each other. The shapes of the elements may be slightly different, as with a set of spanners, an early subject of research into part families by Braid and Hillyard.

In Fig. 1.64 "r_1" is set to be "$R + 30$", for example. A similar relation is set up between "R" and the radius of the tube part of the flange, for example: "$r_2 = R + 10$". The radius of the hole pattern is $R + 20$, with the radius of the holes less than 10, in the flanges shown in Fig. 1.63, the holes have radius 5

Fig. 1.63 Flange entities



and the counter-sinks radius 7.5. A more complicated relation needs to be made between the radius of the hole pattern and the number of holes. For example:

if $(R<25)$ *then* $n = 3$ *else if* $(R<45)$ *then* $n = 5$ *else* $n = 9$ where n is the number of holes in the pattern.

Note that, as well as the change in the number of holes, the flange outer diameter is always 30 more than the radius of the large hole and the tube wall thickness is always 10. If simple scaling were used then the flange outer diameter and the tube wall thickness would vary.

It is not always easy to see these relationships before a design has been completed. If there are known formulae for elements then it may be possible to set these up in advance. If not, then, after the first design is complete, it may be desirable to go back and redo the design with relational dimensions set up. See Chap. 12 for more examples.

1.6.3 A Car

This is a little ridiculous to do as a whole exercise, but it can be useful to show a little of a design process as an illustration of the importance of early phase design and communication. The notions here are from the work of Attila Csabai [2] and will be elaborated further in Chap. 11. The following is only meant as a simple introduction.

One classic description of cars was the "three box design", where one box was the engine, one the passenger compartment and one the luggage compartment (or "boot" or "trunk", depending on which English you use). Figure 1.65 shows a hypothetical example.

Note the overlaps between the three boxes. In most product designs the product elements do not fit exactly into boxes, so bounding boxes overlap. These overlap areas are not errors, according to Csabai, they are where designers need to negotiate over the use of the common volume.

With this method of working, a chief designer starts the solution by dividing the product into functional elements of the product being designed. In Fig. 1.66 the solution to the design problem consists of three main functional units, named D1,

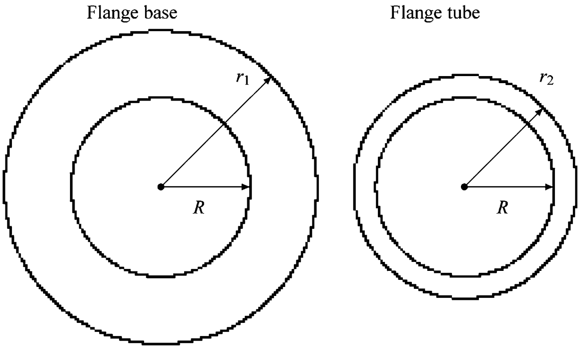


Fig. 1.64 Dimension relations

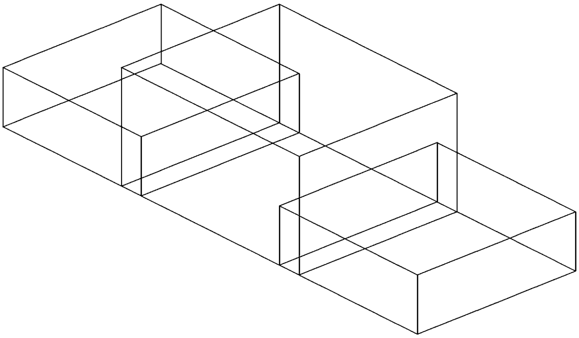


Fig. 1.65 Three box car design

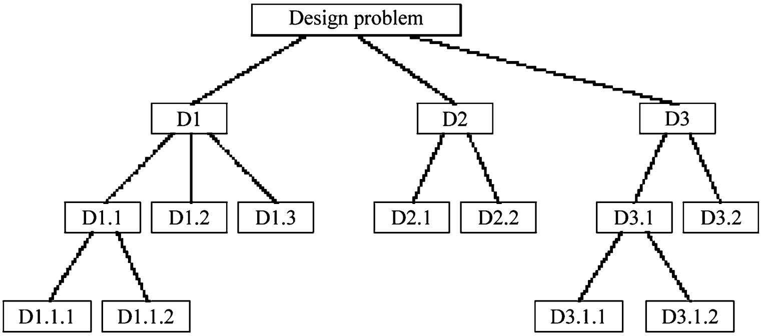


Fig. 1.66 Design problem subdivision

D2, D3 in the figure. D1 is subdivided into three new elements, D1.1, D1.2 and D1.3, and D1.1 further into elements D1.1.1 and D1.1.2. D2 is broken into two elements D2.1 and D2.2. D3 is broken into D3.1 and D3.2, and D3.1 into D3.1.1 and D3.1.2.

The elements to be designed are: D1.1.1, D1.1.2, D1.2, D1.3, D2.1, D2.2, D3.1.1, D3.1.2, D3.2. In Csabai's method each subdivision of the design problem (called a "design space") is represented by a simple geometric form, rectangular block or cylinder, for example. Connections are established between the elements, and the design spaces and connection positions form a geometric framework within which a designer can work. Of course, the number of elements in the subdivisions and the number of levels will vary. The important thing is to subdivide the problem into subproblems until the subproblems can be dealt with. The whole system also provides a framework within which the design solutions can be constructed and understood.

In the three box solution for the car, for example, D1 might be the motor box, D2 the passenger compartment and D3 the luggage compartment. D1.1 might be the wheels, D1.2 the motor and D1.3 the lighting system. D2.1 might be the gearbox and drive system, while D2.2 the seats. D3.1 might be the rear wheel system and D3.2 the luggage space. Obviously there are more elements, but I do not want to deal with a complicated example such as a car in detail here.

One obvious connection is between the motor and the gearbox. Other connections might be fixation or attachment points. Many possibilities exist. These connections have to be established jointly, or by a superior and then provide the design framework around which a common interface can be coordinated. If an assigned space is insufficient then the design space structure provides a framework for establishing negotiations between designers.

Facilities for creating these types of models, layouts in Csabai's terminology, are not currently integrated into the majority of CAD systems. Nevertheless, this way of working during the initial phases of design provides a logical, structured way of integrating designs and design teams. This is why Chap. 11 will deal with the topic in more detail.

1.7 CAD System Structure

A block diagram of the elements of a CAD system is shown in Fig. 1.67.

At the heart of the CAD system is a modelling kernel which creates, modifies and maintains the shapes defined by the system. This modelling kernel is accessed via a user interface which is nowadays part of an integrated graphics environment. The models are displayed via graphics systems which can provide visual feedback to the designer. CAD systems often contain various application systems which use the models created for other purposes. Database systems, in the simplest form just a discfile, are used to record models between CAD sessions.

1.7.1 Geometric Kernels

It has become common practice for CAD software firms to use common modules for the geometric modelling part. These are called geometric “kernels”. At the time of writing there are two notable ones in Boundary Representation, called Parasolid and ACIS, and one Constructive Solid Geometry kernel called SvLis. Boundary representation and Constructive Solid Geometry (CSG) are described in Chap. 2.

As shown in Fig. 1.67, the geometric kernel sits at the heart of the CAD system. It provides the datastructures, basic routines and high-level routines commonly used in solid modelling. The geometric kernel used to be developed by CAD software developers, but developing and maintaining a full system is costly so several software suppliers have opted to use well developed software modules and to concentrate on interfaces. This means that there is some standardisation between the functionality of different systems.

1.7.2 The CAD User Interface

There are lots of different CAD interfaces so it makes no sense to try to describe just one. What is possible is to describe some features of modern interfaces in terms of

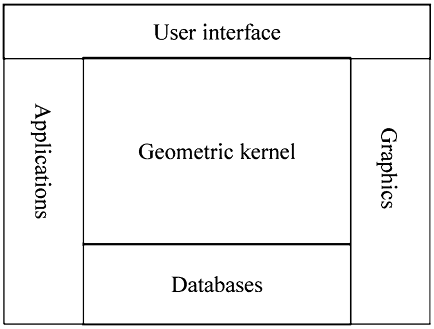


Fig. 1.67 The main elements of a CAD system

past interfaces and to describe what the interface is doing. Figure 1.68 is intended to convey something of the feeling of using software with a user interface.

Computers and graphics have both gone through a period of evolution. Graphics used to be done by plotters with "move-to x,y" and "draw-to x,y" type commands. In early modelling systems commands were given as text commands. Developments in interactive graphics have meant that CAD systems have become interactive with rapid feedback.

In early interactive graphics the graphics screen was often a separate animal onto which graphics information in two- or three-dimensions was loaded. This meant that there was sometimes a separate text screen and keyboard in addition to the graphics screen. Another implication of this early separation was that there were various peripheral devices around to supplement the systems. For example, there was a magnetic pad in front of the user to which there was attached a pen-like device. The pen was moved across the pad and the pad sensed its position. Pressing the tip of the pen caused a signal to be sent to the computer to initiate an action. Paper templates could be put over the pad with menu commands written on them so that the user had a command menu. Light pens were also used to address graphics screens directly, although they were heavy and proved tiring to use for long periods. Pads and light pens have now been superseded by the mouse and on-screen menu commands. Another input device was a "button box" with a number of physical buttons linked to the software. The software established a link to the buttons so that when one was pushed it initiated a software action. At least one graphics system had a set of knobs which could be used to drive graphics functions. Each slight turn of the knob caused an interrupt. This was used by Wingård and Palm (The GPM ASEA robot simulation. Private communication (1983)) to move different elements of an ASEA robot (see Chap. 13).

At the time of writing there is a sort of standard workstation configuration of keyboard, mouse and screen. Some of the early distinctions are still more-or-less

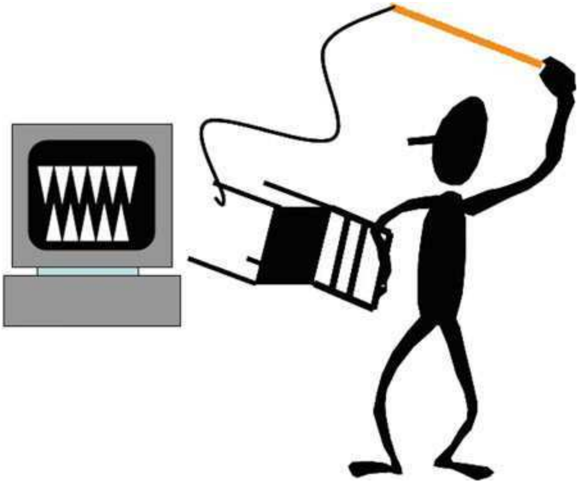


Fig. 1.68 Telling the CAD system what to do

there. One CAD system used button-like backgrounds to icons, possibly because an earlier version of the system used physical buttons. It is usual to divide the screen into functional windows, one is a graphics screen and there may be separate text window or windows. The functions are kept separate. Some systems separate the history tree and put it in a separate window. And so on.

The decisions about what to put where are part of the basic design of the system and give it its look. There are many books on graphics and techniques which describe the techniques in detail. There is also extensive literature on Graphics User Interfaces (GUIs) and Computer and Human Interaction (CHI).

There is a difference, as well, in the range of applications supported in CAD/CAM systems. The larger systems cram everything into the same environment as different modules. This means that communication between the modules is simple, but the modules themselves may not be as advanced as dedicated application code on the market. This means, though, that the system is layered. It can be expected that there is a design module for object creation, an analysis module for finite element analysis for example, a CAM module and so on. Which ones are necessary, though, depends on the user and so cannot be generalised.

In general, putting lots of elements onto the screen gives a cluttered appearance and makes it difficult to find the functions needed. One solution is to give the modelling system a “Windows” appearance. Another solution is to put functions and function groups onto the screen as icons, as mentioned earlier. Since there are many possible functions there is a tendency to have to group them into functional groups. You might have all extrusions (linear and circular) in one group, Booleans in another group, chamfer and blend (or fillet) in a third group. Learning where these are is a matter of practice. Learning which functions are available is also a matter of practice, which is another reason why CAD exercises are useful. If the CAD supplier has some predefined exercises then it is worth following some of these to get to know the functionality and placement of the functions. Since the layouts of individual systems may be considered as commercial property, no examples are shown here.

The user interface reflects the functionality of the geometric kernel. There are a number of “standard” operations which are likely to appear. Examples are extrusion (linear and circular), Boolean operations, draft angles, chamfer, blend (or fillet), thickening sheets, hollowing out solids, symmetry. In addition, customers may ask CAD suppliers for special operations which are then developed by the CAD firms. These are what you see in the user interface.

There are other ways of accessing CAD systems using programming techniques, through so-called *application programming interfaces*, or APIs. At the user level there is usually some kind of macro-language facility to allow users to program common sequences of operations, or to build parts. These call routines in the API using a command language. There may also be a programming interface where a developer can access data structures and basic functions of the modelling system. This programming interface is unlikely to be useful to the majority of CAD users because it requires specialist modelling knowledge and programming skills.

Current CAD systems use interactive graphics windows for communication with the modeller. Briefly, every point on the screen has a graphics address. When a mouse, or other pointing device is used this causes a cursor to be moved in the screen. Activating a mouse key, say, causes the cursor position to be passed to the software. If the position is over a command window then a command is activated. If the cursor position is over a model window then this needs to be interpreted as a model element.

1.7.3 Graphics

There are two types of graphics commonly used in CAD systems: *dynamic graphics* and *static graphics*. The dynamic graphics interface provides the user with quick feedback about the shape of the part. Static graphics is used for producing engineering drawings. Although both may use similar techniques they are used for different purposes so are described separately here.

owadays, dynamic graphics uses a standard form, such as triangles, to communicate shape information from the CAD system to a graphics card which handles the graphics screen. The triangle data resides in a local memory and is pushed through a series of transformation matrices which reposition it, clip it, flatten it and eventually the triangles are used to colour different parts of the graphics screen. The triangle information stays the same, apparent rotation, translation, scaling, etc. of the screen is performed by changing the transformation matrices.

Chapter 7 deals with the way that graphics works in more detail.

1.7.4 Applications

Applications add richness to the software environment, turning it from a strictly CAD (Computer Aided Design) tool into a CAE (Computer Aided Engineering) tool. There are advantages and disadvantages of doing this. One advantage is that the user has the same tool for everything, with the same style and similar interfaces. If changes have to be made for, say, manufacturing, it is easy to switch back to the design module and change the design. The disadvantage is that the application modules may not be the most advanced in any particular domain because the software developer does not have enough resources to keep every module up to date.

The alternative to using complex, multi-module software is to use dedicated software and use data exchange to switch between them. This can make changes less easy and there is information loss, usually, in using data exchange. However, the dedicated software may be easier to use and better than a module in a large system. Also, an advantage for smaller companies is that the cost of a few smaller software modules may be less than one integrated system.

1.7.5 Discfiles and Databases

One of the fundamental necessities for a CAD/CAE system is to be able to save a model for future use. This is described later in Chap. 9. Briefly, though, software can output two types of data, proprietary discfiles or standard discfiles. Proprietary discfiles are usually smaller than standard discfiles and so are commonly used. The proprietary discfile is generally a textual version of the memory datastructure.

1.7.6 Standalone Systems and Multi-User Work

Design is often a multi-person activity, but CAD systems are typically single-user systems. This leads to a fragmentation of effort because design information can only be communicated when one designer saves work, essentially posting it to a common area. This may involve data exchange between different systems, which is often accompanied by data loss. Even worse is where the communication happens by static means in terms of drawings. Then, human interpretation is needed to ensure the communication.

The isolation of CAD systems is a weakness in the design process. Where the user is someone responsible for all or part of the chain then this problem does not exist, but this is not usually the case. Two methods exist for integrating CAD processes, the first a methodology, the second a technical solution.

1.7.6.1 Concurrent Engineering

Concurrent engineering is a methodology, or technique for sharing technical knowledge from different experts during the design process. The idea is that the designer considers one set of tasks, the solution of the design problem, without necessarily considering all production aspects. The designer would create a design and then negotiate with a manufacturing expert in order to optimise manufacturing, an assembly expert to check assembly, and so on. The aim is, of course, to reduce the direct costs of, say manufacture or assembly labour costs, and to reduce the costs involved in redesign to correct mistakes. Figure 1.69 illustrates this.

The experts talk to each other using normal communications means and communicate using a common database. There are, of course, problems of version control, to avoid more than one person changing the model at any one time, but these can be solved as with any database system. A more difficult problem is that the experts can only see the design when it is released by the current user. This makes the process “reactive” and not “proactive”. The reactive element is that the designer makes a shape which is then analysed and corrected. In a proactive system the designer would be helped to make an optimised shape. Nevertheless, this methodology is well-known and used. Section 14.2 discusses this topic in more detail.

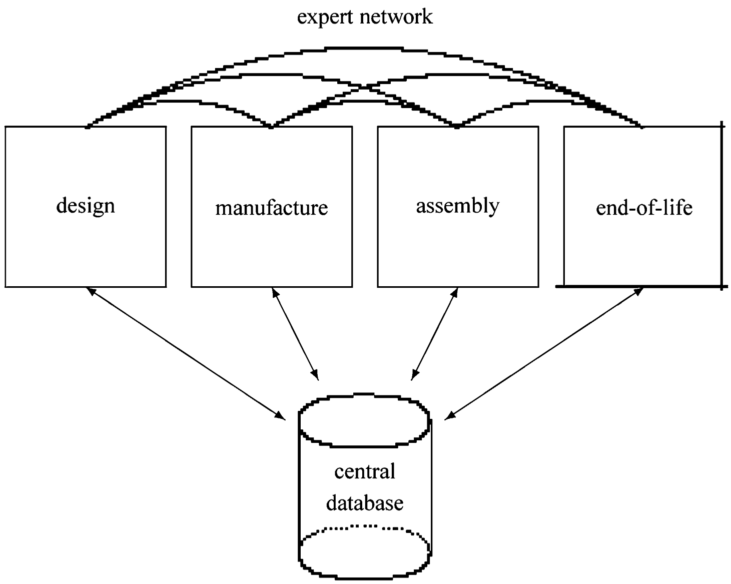


Fig. 1.69 Concurrent engineering

1.7.6.2 Multi-User Systems

An alternative to the reactive method is to have collaboration using multi-user systems. With a multi-user system the different users can see and interact with each other's models during the design process. Multi-user systems exist for some interactive tasks, but they do not seem to have been developed much for CAD. A multi-user layout design system was developed by Attila Csabai [2], but this field remains open.

Two methods of creating multi-user CAD systems are illustrated in Figs. 1.70 and 1.71.

In Fig. 1.70 there is a central computer which handles all the modelling. The CAD systems function as graphical interfaces, sending commands to the central unit which creates and manipulates the models. The commands are created from menus and communicated in the internal language used to store construction history. The central unit sends back the graphical information as a graphical model for local manipulation.

In the second type of multi-user organisation, shown in Fig. 1.71, the CAD systems do the local processing and communicate the same commands to other connected CAD systems which reproduce all models locally. This can be done using the DJINN standard [5] developed by the United Kingdom Geometric Modelling Society. See Sect. 14.2.4.

The first type of method is similar to the old mainframe computer with terminals. The second type of organisation is more appropriate for distributed computing.

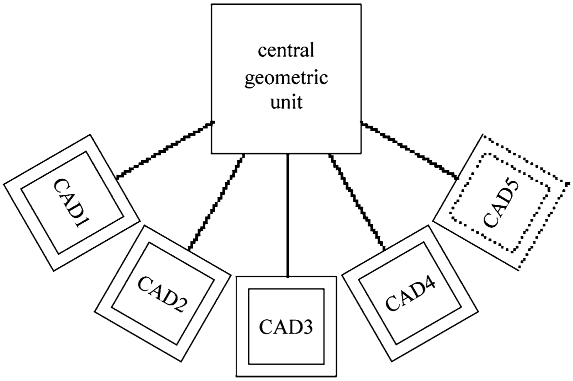


Fig. 1.70 Multi-user system organisation I

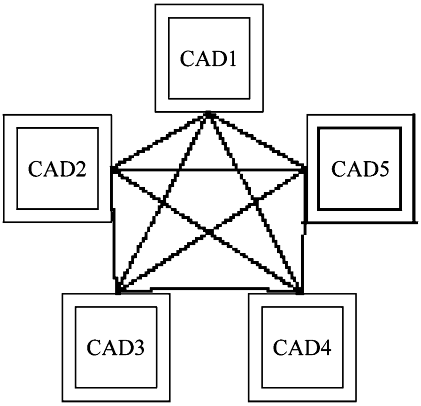


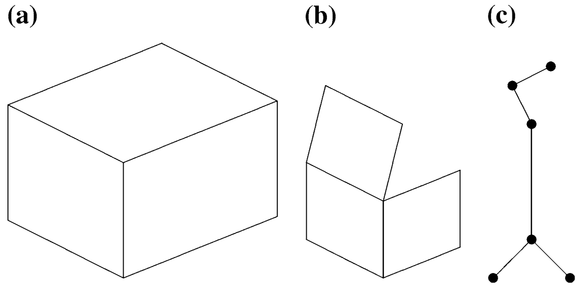
Fig. 1.71 Multi-user system organisation II

Neither of these exists at present, as far as I know, but could do in the future. There is a tendency towards collaborative systems in the research environment.

1.8 The Modelling Environment

An important question in the design of a CAD system is whether to have single model or to allow multi-model environments. Some modern systems limit the user to designing a part as an isolated object and then putting products together as assemblies in a different environment. This is an artificial restriction and sometimes the systems themselves allow the user to get round these restrictions. A personal opinion is that the single model environment is too artificial and that it has not been thought out sufficiently. There is a common problem, which will be described more in Chap. 4, where "single" models may actually be composed of several separate pieces without the user being informed. Another problem is when the system allows the introduction of separate solids. This can be useful for modelling different object parts which are then combined using Boolean

Fig. 1.72 Solids, sheets and wires



operations. However, if there is no obligation to join the parts, then the rule enforcing one model per file seems to be broken. This is not solved by modelling in an assembly environment, as some systems propose. An assembly is a definite structure with certain properties and strategies for handling its components (instances). It is more natural to let users have multiple object environments and let them build assemblies from these. The single-model-per-file problem can be solved by associating file definitions with the objects.

Another difference between systems is how they integrate solids and partial models. Figure 1.72 shows some examples of different model types.

Solids, in Fig. 1.72a are models of normal objects. Sheet objects, in Fig. 1.72b, can either be surface models or compound models intended as idealisations, described in more detail in Chap. 5. Wire models, in Fig. 1.72c, may be used for various purposes, especially as early sketch elements.

If your CAD system allows you to mix these freely then you run the risk of having objects which you think are solid but which are only partially defined. The other option employed by CAD systems is to have solids in one module and sheets, or surfaces, and wires in another module. This means that it is necessary to go through a conversion step in order to make a partial model into a solid. Having the models in separate modules means that using them is somewhat clumsy, but the benefit is, or should be, consistency. An option is to colour the object types differently in order to be able to check visually what you are working with.

1.9 Chapter Summary

This chapter is intended as an introduction to CAD systems where the reader can match modelling intent to the modelling functionality of the CAD system used. The chapter is also meant as an introduction to the topics handled in more detail in later chapters. The questions after the exercises are intended to stimulate the reader and indicate questions that he or she should be asking when using a system. In general, when an error occurs, it is not necessarily so that the user has done something wrong. Sometimes there is a difference in perception about the way that operations should work from the user and implementer viewpoints. This book is intended to show the implementer viewpoint and this chapter is the introduction to that.

1.10 User Exercises

1.10.1 The MBB Gehäuse Rohteil

This object, shown in Figs. 1.73 and 1.74 was used as a test part for comparing modelling systems in 1979 and again in 1983. The tests were organised by Computer Aided Manufacturing-International (CAM-I). Try making

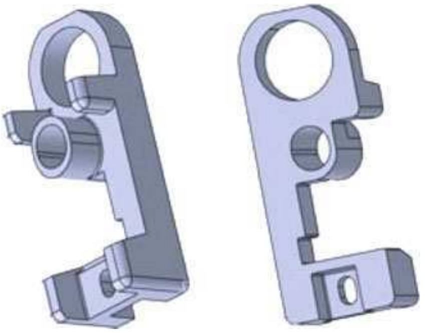


Fig. 1.73 MBB Gehäuse Rohteil

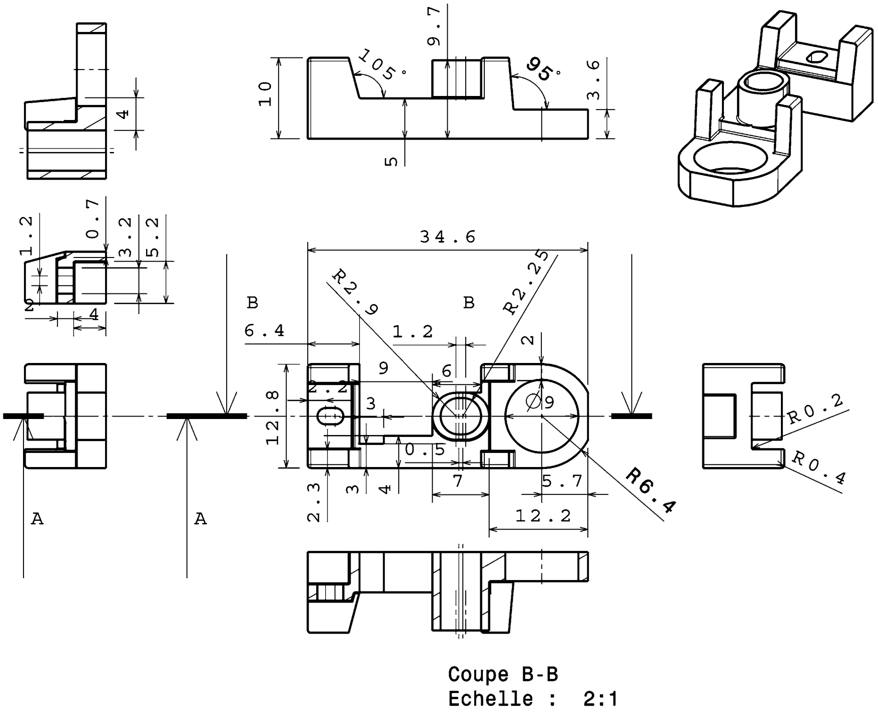


Fig. 1.74 MBB Gehäuse Rohteil drawing

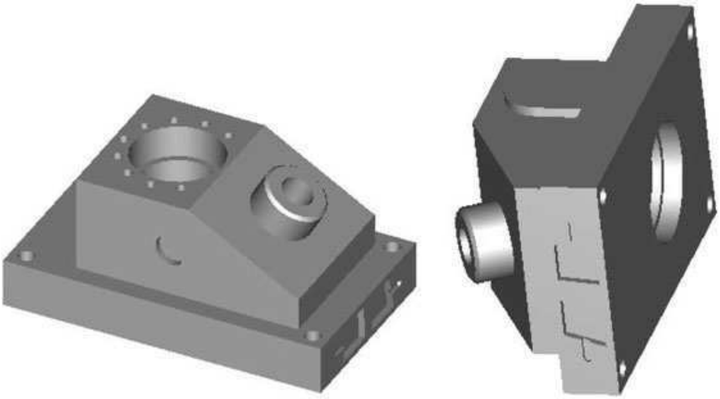


Fig. 1.75 ANC101 object

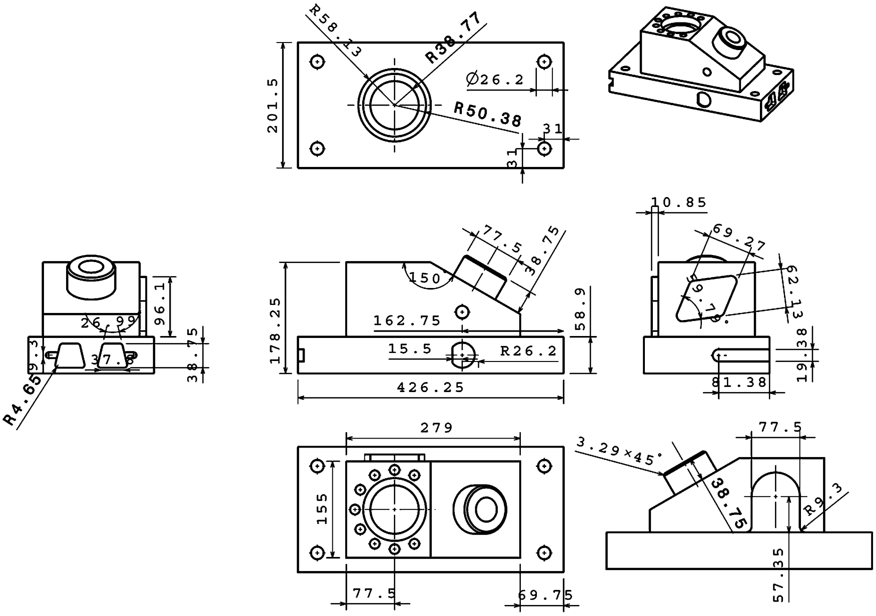


Fig. 1.76 ANC101 drawing

this object, deciding yourself on the steps needed. The aim is to avoid a detailed set of instructions so as to give you more freedom on deciding how to do things.

1.10.2 The ANC-101 Object

Try a second example. The second object, shown in Figs. 1.75 and 1.76 is also a piece from CAM-I, called the ANC-101 object. ANC stands for Advanced Numerical Control and was the name of one of the CAM-I discussion groups.

1.10.3 Linear Extrusion Part Using Symmetry

This exercise is intended to show the different methods of working for a complete part and a symmetric part. The intention is to make part of the part and then use symmetry to complete it, as mentioned during the questions. Think about the part in exercise 1 and how it could be recreated by using symmetry. In effect, model half the part and then reflect it to create the full part. However, think about which elements you should include in this. Where is the basic object symmetrical? Which are the elements that should be added after the symmetry?

References

1. Braid, I.C.: Designing with volumes. Ph.D. Dissertation, Cantab Press, Cambridge Computer Laboratory, University of Cambridge, Cambridge (1974)
2. Csabai, A.: Layout modelling and evaluation methods and tools. Ph.D. Dissertation, EPFL, STI-IPR-LICP, Switzerland (2003)
3. Stroud, I.A.: Boundary Representation Modelling Techniques. Springer, Heidelberg (2006)
4. Katainen, A.: Monadic set operations. CAD J **14**(6), 351–354 (1982)
5. Djinn—A Geometric Interface for Solid Modelling. Specification and Report, first published 2000 and available online from the Geometric Modelling Society. The "Geometric Modelling Society" home page is currently at http://www.bath.ac.uk/ênsab/GMS/gms.html or can be found by a search engine. A link to DJINN documentation is included

Chapter 2
How Objects Are Modelled

Various methods have been developed for representing shapes. CAD systems have included several of these techniques at different times in their development. A short historical perspective is given in the first section of this chapter. At present the main technique used is called “Boundary Representation”, a solid modelling technique which is described later in this chapter.

2.1 History

The history of CAD is quite long, but you won’t find it described here. The purpose of this section is not to mention particular systems, however important, but to introduce the techniques. Figure 2.1 shows the main time line of modelling development.

Early modelling systems were capable of creating wireframe, or line, drawings of shapes. However, while pictures may communicate information between people they are not enough for computer applications. Both the car industry and the aircraft industry needed to create and manufacture complex shapes. Surface modelling techniques and systems were developed in order to facilitate toolpath generation for machining. It was also realised by various people that solids could also be modelled and these techniques arrived during the 1970s. In the beginning there were several different techniques, but during the 1980s two became predominant and eventually only one technique became used for the majority of CAD/CAM applications. Although these techniques were largely ignored by the CAD developers at the outset, they became accessible and are now common. An overview of various techniques is given below.

I. Stroud and H. Nagy, *Solid Modelling and CAD Systems*,
DOI: 10.1007/978-0-85729-259-9_2, © Springer-Verlag London Limited 2011

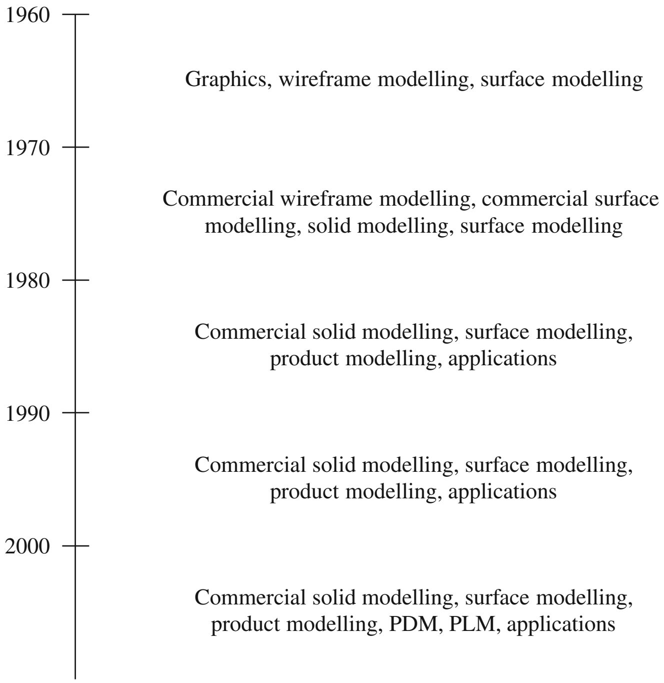
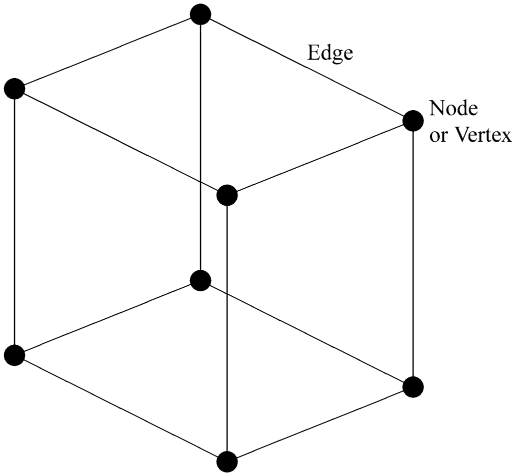

1960
Graphics, wireframe modelling, surface modelling
1970
Commercial wireframe modelling, commercial surface modelling, solid modelling, surface modelling
1980
Commercial solid modelling, surface modelling, product modelling, applications
1990
Commercial solid modelling, surface modelling, product modelling, applications
2000
Commercial solid modelling, surface modelling, product modelling, PDM, PLM, applications

Fig. 2.1 Modelling and CAD timeline

Fig. 2.2 Simple 3D wireframe model



2.2 3D Wireframe Modelling

Wire frame models consist of nodes, or vertices, and links between them, called edges, as shown for a cube in Fig. 2.2.

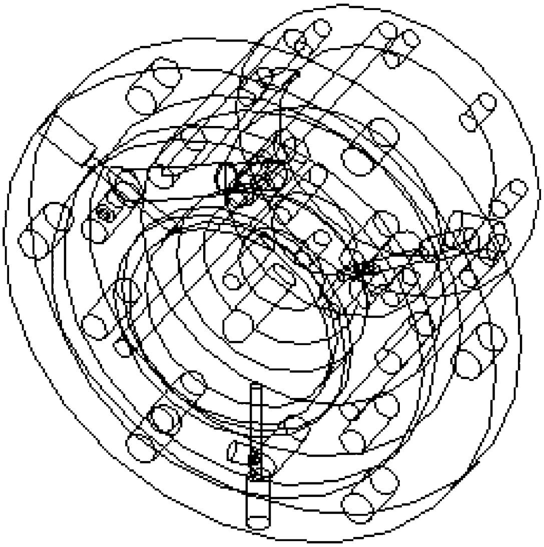


Fig. 2.3 Complex wireframe object

This is enough to produce pictures but several straightforward and useful operations cannot be performed without surface information. Hidden line removal, for example, for realistic image creation is impossible unless you have surface information. Figure 2.3 shows a complex object in wireframe mode where it is difficult to see the object represented in the figure. More importantly, automatic toolpath generation for machining cannot be done without surfaces. This particular shortcoming prompted development of surface modelling systems, described later.

Another problem is that it is possible to create objects which it is impossible to realise. See for example Fig. 2.4 which shows a well-known optical illusion which can be made with just point and lines.

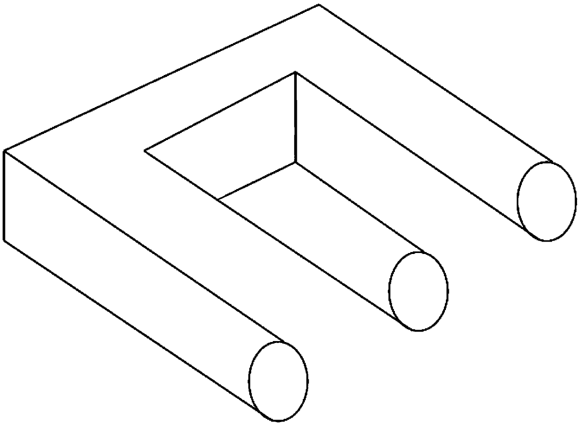


Fig. 2.4 Impossible object

Wireframe modelling has a place, though, in modelling as a support for other operations, as described in Chap. 4. It is important, though, to keep wireframe modelling as a support, or to keep it simple. It is possible, with a lot of effort, to create a complex model, but it becomes harder as the complexity increases.

2.3 Drafting Systems

Drafting systems were essentially electronic drawing boards. They could be used to create planar line drawings and made copying and amendment very easier. However, while a step forward they preserved the problems of classic design techniques, the possibility of creating incorrect drawings and that multiple views had to be created separately.

Some of the operations in solid modelling, notably linear and circular extrusion, seem to have their roots in 2D drafting. However, there are many benefits from using 3D models so drafting systems will not be dealt with here.

2.4 Surface Modelling

Surface modelling was a step forward in that these systems can represent the space between the edges in a wireframe model. Geometric modelling as a subject has a life separate from solid modelling and continues as an important subject for research.

Surface modelling systems model portions of an object with complex surfaces. Objects such as car bodies, aircraft and ships use complex surfaces of this type. With some products it is not necessary to have a solid model behind and a surface modelling system is sufficient. For other uses surface modelling systems have been used to create objects which appear closed, although the surface patches are not joined, simply placed adjacent in space. Such a lack of connection can cause problems because there is no information to check consistent orientation and connection. There are also problems which are inherent in the mathematics of surfaces because surface patches have four sides, usually, though three-sided patch mathematics has also been formulated. Real objects often disobey this requirement leading to mismatches between patch boundaries.

However, the usefulness of surface modelling means that the techniques are often incorporated into solid modelling systems. Perhaps the first such modelling system was the GPM volume module by Kjellberg et al. [1]. The use of surfaces and other complex geometry in CAD systems is something which will be described in more detail in Chap. 5.

2.5 Solid Modelling

This book is concerned with the application of solid modelling techniques in CAD systems. Solid modelling began in different places sometime at the beginning of the 1970s. In Cambridge Ian Braid, in the BUILD series of systems, developed the boundary representation technique. Work was also done on this by Kimura in the GEOMAP system. Also in Japan, in Okino, the half-space technique was being worked on which emerged as the TIPS system. Similar ideas were being worked on in Rochester by Voelcker and Requicha, which resulted in the important PADL system using the CSG technique. Generalised sweeping was also used as a solid modelling technique, representing objects as two-dimensional forms and extrusion definitions. Cellular modelling, both with uniform cells and adaptive cells, the so-called octree technique were examined. A good presentation of these is given by Jared and Dodsworth [2] and more explanation than here is given in Stroud [3].

As far as CAD systems go, it is important to know that Constructive Solid Geometry, or CSG, systems were used initially while now Boundary Representation has taken over. A simple illustration to show the way in which these two methods work is shown in Fig. 2.5. On the left you have the CSG representation. The object at the bottom is modelled as a tree structure in which a rectangular block and a cylinder are first "added" and then a cylinder is subtracted to form the hole. To the right you have the Boundary Representation version. The solid at the top is modelled using a connected set of faces, each of which is a bounded surface portion. The faces are shown in exploded form at the bottom right of the figure.

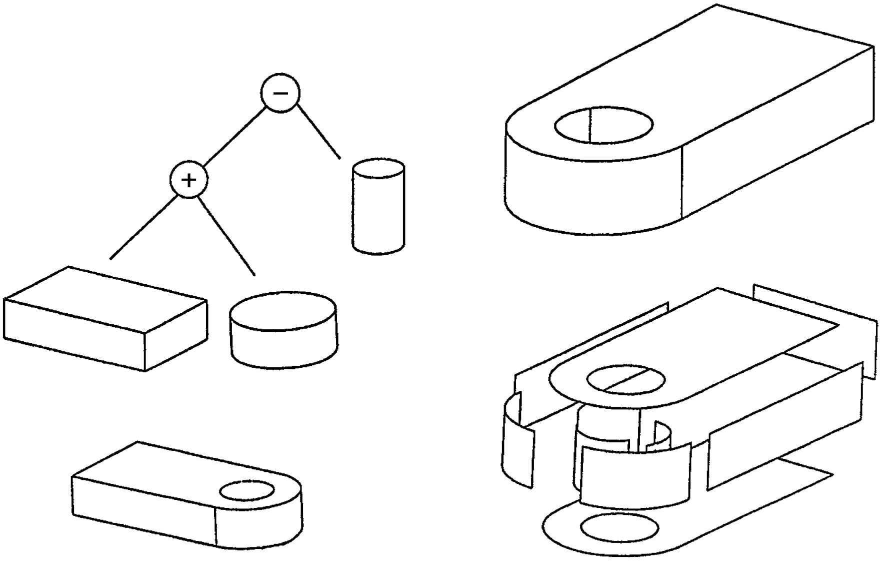


Fig. 2.5 Representing solids with CSG and Brep

To summarise, CSG models the part as a tree structure where the leaves are positioned primitive objects and the intermediate nodes are Boolean operations to combine them. Boundary Representation models the "skin", the interface between solid and non-solid.

2.6 Constructive Solid Geometry

As stated above, constructive solid geometry, or CSG, systems represent an object by combining a set of primitive objects. See Okino et al. [4], Requicha and Voelcker [5] or the interesting monadic variant, UNIBLOCK, by Katainen [6]. Each of these primitives is made of a set of half spaces defining point sets. In two dimensions, with lines instead of planes, this is illustrated in Fig. 2.6.

The central portion, the square, can be defined with the equation:

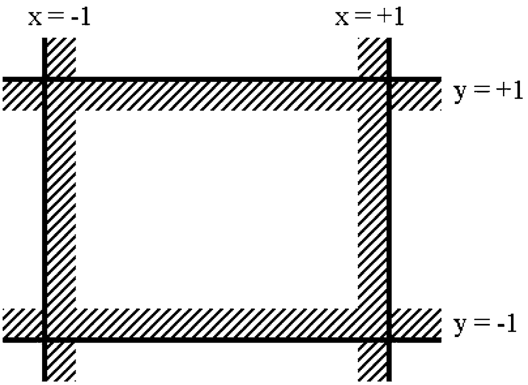
$$(x \geq -1) \wedge (x \leq +1) \wedge (y \geq -1) \wedge (y \leq +1)$$

Similar sets of relations can be established to define a set of three-dimensional primitive shapes. A set of normal primitives might be:

- Rectangular block
- Wedge
- Cylinder
- Cone
- Sphere
- Torus

When building a model, these primitives are created and positioned and then they are combined by applying Boolean operations. These Boolean operations are different to those described in Sect. 4.1. With CSG the Boolean operations are logical combinations of type AND, OR, INTERSECT. Consider the two objects in Fig. 2.7.

Fig. 2.6 Simple object with half space representation



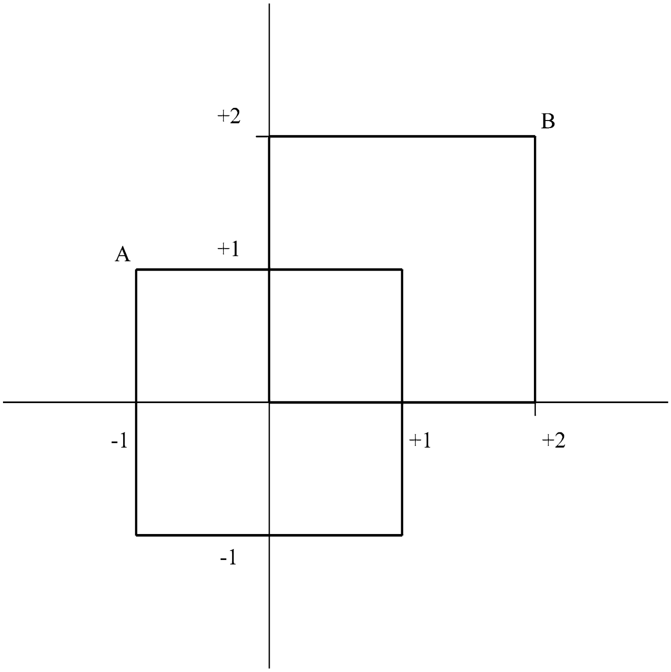


Fig. 2.7 Addition of two squares

These are two squares, one from -1 to 1 in the x and y directions, the other from 0 to 2 in the x and y directions. The combination of the two can be represented simply by the following expression:

$$\begin{aligned}A \vee B = ((x \geq -1) \wedge (x \leq +1) \wedge (y \geq -1) \wedge (y \leq +1)) \vee ((x \geq 0)\\ \wedge (x \leq +2) \wedge (y \geq 0) \wedge (y \leq +2))\end{aligned}$$

The resulting area is shown in Fig. 2.8.

Subtraction can be done in a similar way. The relationship is essentially everything in A which is not in B, or $A \wedge (\sim B)$ or:

$$\begin{aligned}A \wedge (\sim B) = ((x \geq -1) \wedge (x \leq +1) \wedge (y \geq -1) \wedge (y \leq +1)) \wedge ((x \leq 0)\\ \vee (x \geq +2) \vee (y \leq 0) \vee (y \geq +2))\end{aligned}$$

The result is shown in Fig. 2.9.

For the intersection the corresponding equation is:

$$\begin{aligned}A \wedge B = ((x \geq -1) \wedge (x \leq +1) \wedge (y \geq -1) \wedge (y \leq +1)) \wedge ((x \geq 0)\\ \wedge (x \leq +2) \wedge (y \geq 0) \wedge (y \leq +2))\end{aligned}$$

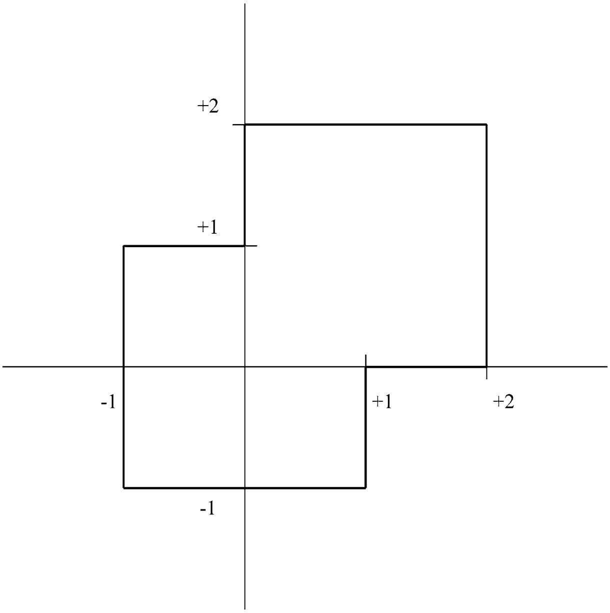


Fig. 2.8 Addition of two squares

which can be simplified to the equation:

$$A \wedge B = (x \geq 0) \wedge (x \leq +1) \wedge (y \geq 0) \wedge (y \leq +1)$$

The result is shown in Fig. 2.10.

There is, though, a slight complication in that with some operations you can get hanging geometry. If, from the result in Fig. 2.9, you subtract the block defined as:

$$(x \geq -1) \wedge (x \leq 0) \wedge (y \geq -1) \wedge (y \leq +1)$$

you get a hanging edge with the naive scheme mentioned above, as shown in Fig. 2.11.

To get round this problem the CSG researchers developed the notion of "interior" and "closure". See Tilove and Requicha [7]. Since this book is not about CSG techniques, rather than describe this in detail it is more important to see how this works in practice.

To create a 3D model, a number of 3D primitives are created and combined in the same way as outlined above. Consider the object in Fig. 2.12. This might be modelled by the set of primitives and operations shown in Fig. 2.13.

Another alternative is shown in Fig. 2.14.

Which to choose depends on the way that a user mentally decomposes the shape. As can be seen from the figures, though, while the end result is the same,

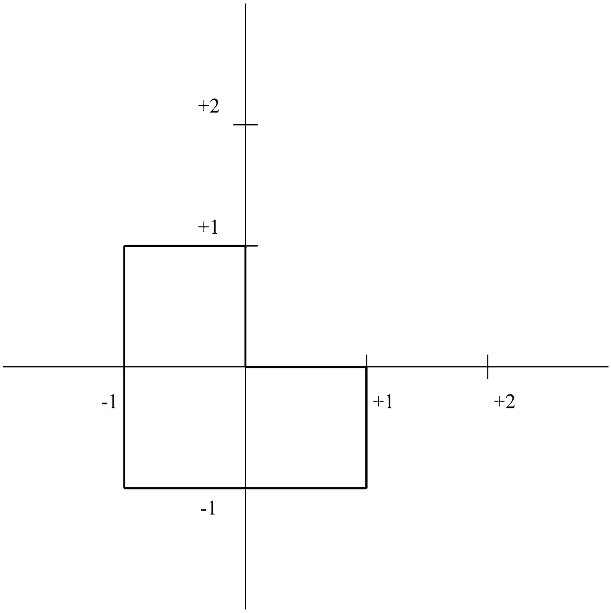


Fig. 2.9 Subtraction of one square from another

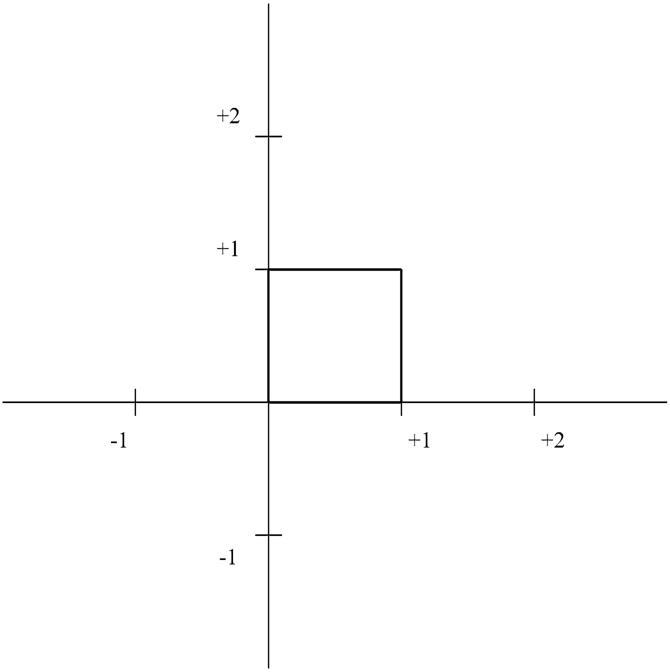


Fig. 2.10 Intersection of two squares

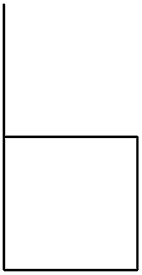


Fig. 2.11 Result with hanging edge

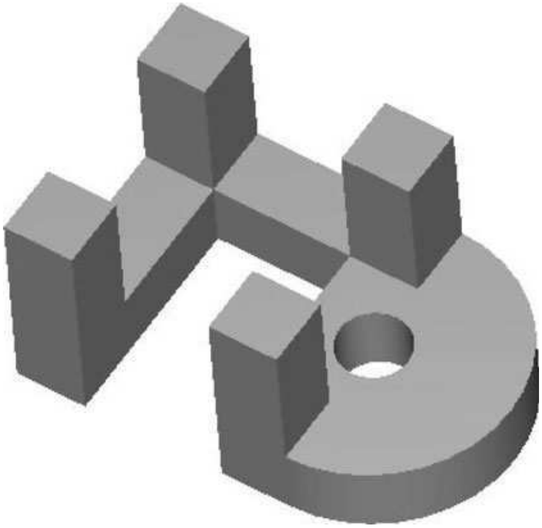


Fig. 2.12 Simple object

the intermediate steps and the resulting tree structure are quite different. This has been cited by several people as a problem with CSG because it means that shapes cannot be compared by simply comparing the tree structure.

Another drawback of CSG modelling is that it is not always convenient to formulate a design operation in terms of Boolean operations. Consider a chamfer operation such as that shown in Fig. 2.15. This is conveniently defined in a CAD system by picking the edge and giving an appropriate chamfer depth value.

In a CSG system how would you do that? For a start, it is necessary to note with the edge is convex or concave. For a convex edge this means subtracting a wedge of the appropriate size. For the concave edge in the figure this means adding the wedge. In the case in the figure, the faces where the edge ends are perpendicular to the edge. If they were not then it would be necessary to preshape the wedge. The basic size of the wedge is, of course, computed from the length of the edge.

What is more awkward is that the edge doesn't exist.

In the strict CSG sense the object on the left of Fig. 2.15 would probably be made by subtracting one block from another. The concave edges in the interior of

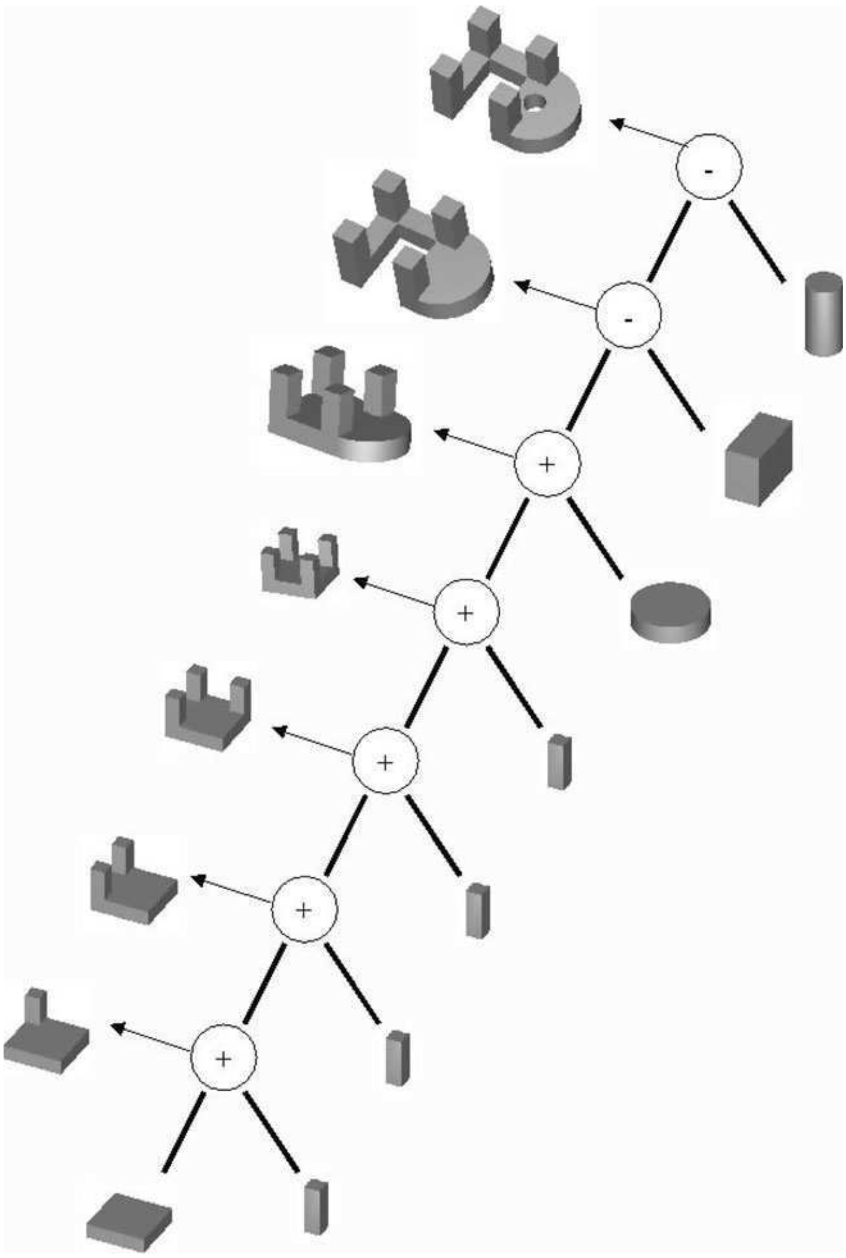


Fig. 2.13 First CSG decomposition of simple object

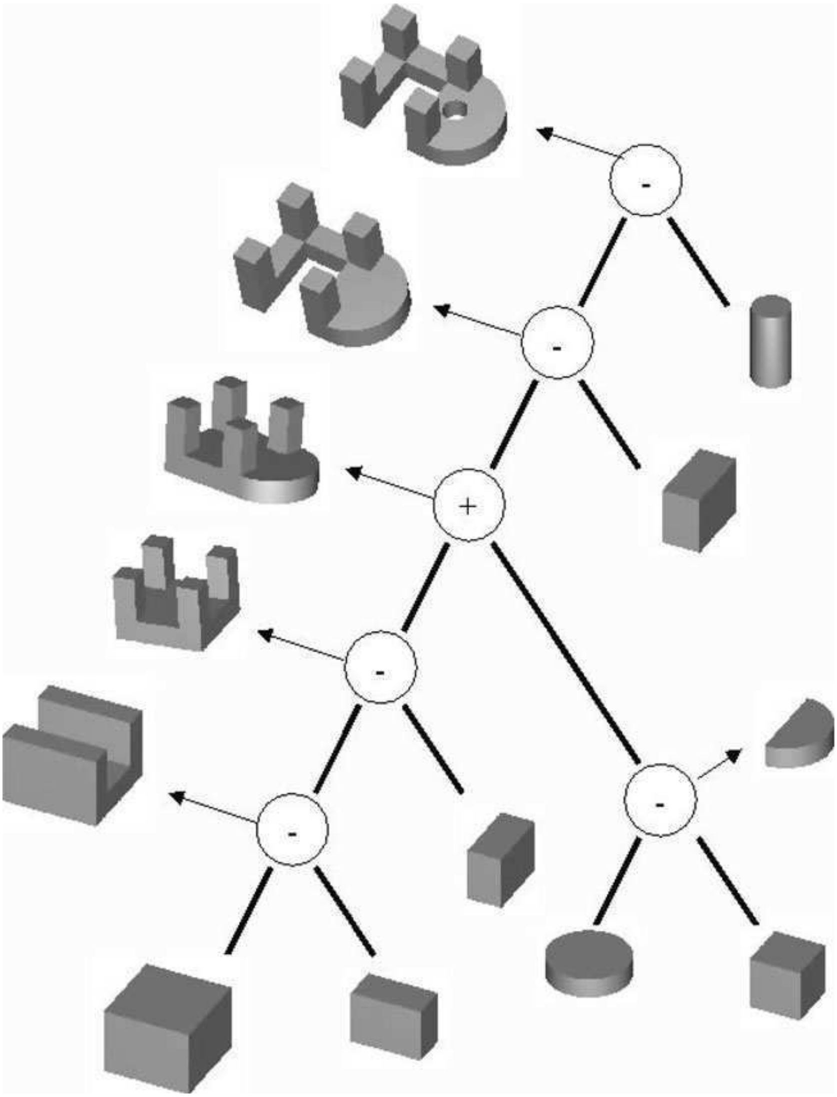
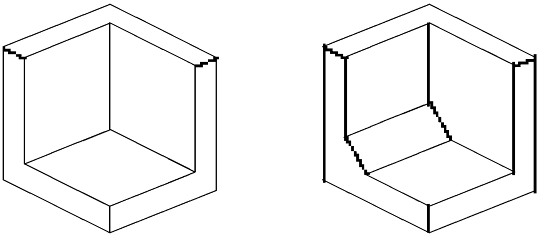


Fig. 2.14 Second CSG decomposition of simple object

Fig. 2.15 Chamfering an edge in an object



the object appear as a bi-product of the definition. They aren't actually explicitly defined. In Okino's system, graphical images were produced by slicing the result object was sliced with planes and the resulting intersection lines drawn to show the result. The "edge" is then just a visual element in the image. In the PADL system of Requicha and Voelcker the strict CSG approach was eventually supplemented with an explicit Boundary Representation model for graphics purposes. This means that there is an explicit edge in the graphics model, but this is not strictly part of the CSG model. In the CSG philosophy you would have to define the wedge explicitly, shape its ends, if necessary, and determine whether to add it or subtract it from your model. This is possible for modelling a known part but makes design harder.

Another type of problem comes when a shape is to be extruded. This implies that the 2D shape has to be decomposed into squares, circles, triangles, etc. Each of these basic shapes corresponds to a primitive. The primitives are then added together to produce the extruded shape as a volume. Extrusion is much more straightforward in boundary representation modelling.

One of the obvious differences between CSG and Boundary Representation modelling was the richness of the Boundary Representation modelling set. A whole range of operations, linear and circular extrusions, Boolean operations, chamfers, tweaks, etc. were defined compared to the Boolean operations of CSG. It is perhaps this richness together with the flexibility of Boundary Representation for degenerate and special objects that led to the current domination of boundary representation.

2.7 Boundary Representation

The currently widely used technique for solid representation in CAD/CAM is the boundary representation technique. These techniques are described in detail in [3]. The purpose here is to put these into a context so that it is possible to understand the implementation of CAD/CAM systems, how they work and why they work that way. Some of the basic concepts are reproduced here to help explain these details.

Boundary representation models of solids have two basic parts: the topology and the geometry. These are kept separate for practical reasons. The topology defines the structure of the model, the geometry its shape, Fig. 2.16.

The main elements of the topology of a model are faces, edges and vertices. There are other elements which are there for practical or efficiency reasons, Fig. 2.17. The loop, for example, is necessary to represent multi-connected faces, but it is accessed via the face which it bounds.

The topology provides links to other elements and so makes the structure connected. For many operations you refer to a topological element of the model. Blending, for example, might take a face, edge or vertex as input, depending on the implementation. If it takes a face then the usual logic is that all edges around the face will be blended. Similarly, if a vertex is given then all edges at the vertex will be blended, usually with the vertex as well.

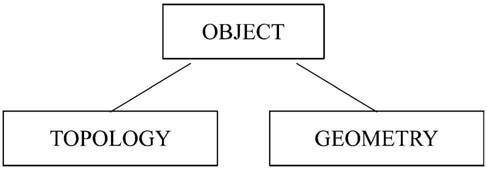


Fig. 2.16 Basic data structure subdivision

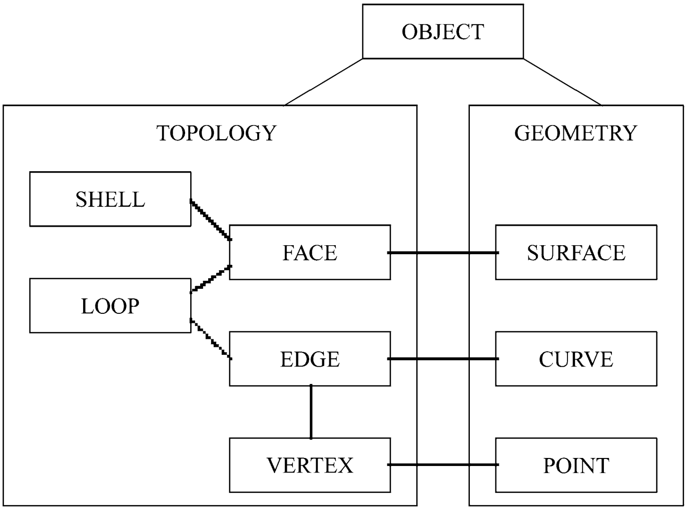


Fig. 2.17 Basic elements of the data structure

Each face is part of a surface. Each edge is a portion of a curve and a vertex lies at a point in space. This relation is illustrated in Fig. 2.18.

A full, single model data structure and definitions might be as shown in Fig. 2.19.

The data structure definitions for this structure are shown below. It is not really important to memorise these because this structure is not unique and is just a simplified example structure. About the only place where you need be concerned with these is for data exchange. They are shown here to explain how the models work. Each entity is a block of computer memory consisting of a consecutive set of memory words. The definitions are:

```
class body
int number;
shell *pshell;
edge *pedge;
vertex *pvert;
body *next;
```

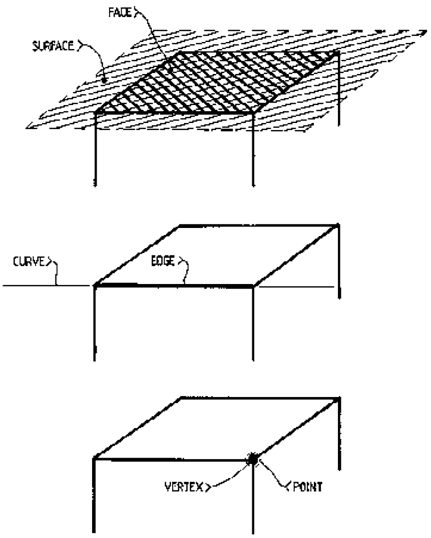


Fig. 2.18 Faces-surfaces, edges-curves, vertices-points

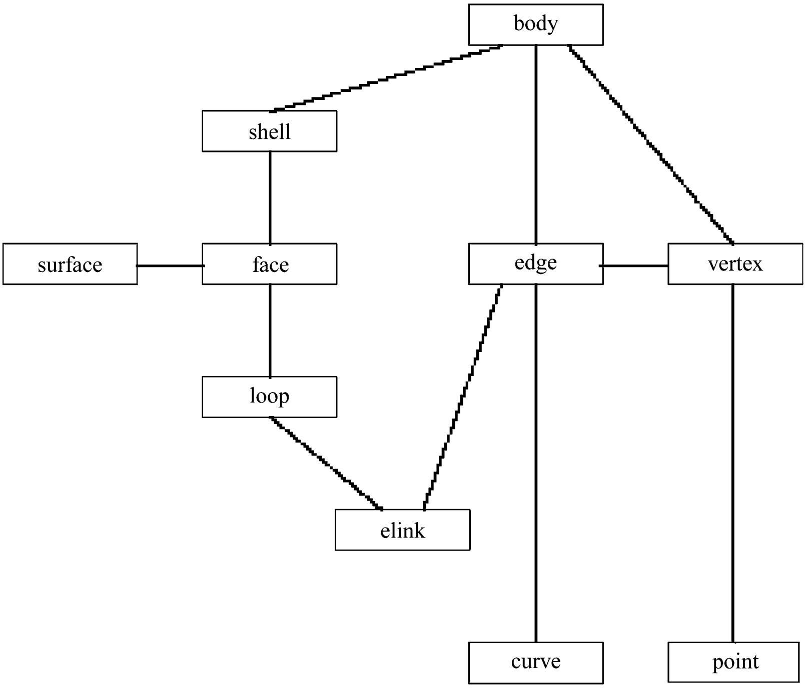


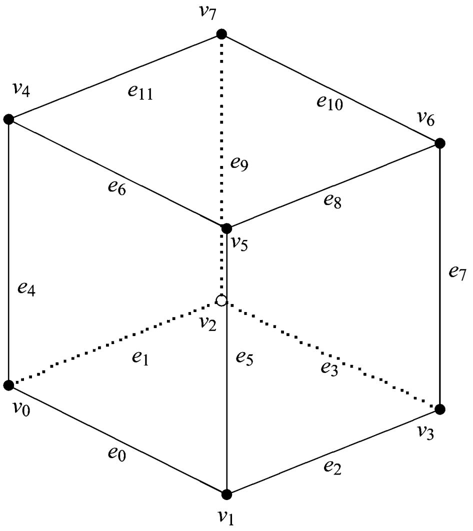
Fig. 2.19 Simple boundary representation data structure

```
class shell
int number;
face *pface;
shell *next;
body *pbody;

class face
int number;
shell *pshell;
loop *ploop;
surface *surf;
face *next;

class loop
int number;
elink *eref;
loop *next;
face *pface

class elink
int number;
elink *cclink;
elink *cwlink;
loop *ploop;
edge *pedge;

class edge
int number;
elink *rlink;
elink *llink;
curve *pcurve;
vertex *svert;
vertex *evert;
edge *next;

class vertex
int number;
union(loop,edge) *pref;
point *pos;
vertex *next
```

As an example of how an object is modelled, consider an object such as a cube (Fig. 2.20).

In memory, if you create such an object, you might have a structure like that shown in Fig. 2.21, with the circular nodes representing entities and the lines representing pointers. The **elink**s are shown as small circles without numbers, the right elinks on the top row, the left ones on the lower row.

Fig. 2.20 A simple cube



If you were forced to build the datastructure manually in a CAD tool you would probably not use a CAD system. You, or anyone, would probably also make many mistakes because it is just too complicated. Frankly, the datastructure diagram in Fig. 2.21 is a mess, but it illustrates graphically what is in memory. Building this up is hidden by operations and suboperations so that you can work logically. [One thing you should note, though, is that the boundary representation is a very localised representation, which has advantages and disadvantages for CAD]. The real data structure of a solid modeller is fundamental to a CAD system, but it is not directly of interest to a user. To explain this, you might consider that the structure of a modelling system is a little like an onion. This is illustrated in Fig. 2.22.

At the heart of the modelling system are the data structures, of course, but these are hidden behind a set of interrogation and manipulation routines. This is done so that the data structure can be modified without disturbing the whole modeller. The geometry, also, is hidden. When working with algorithms the method is to work with the categories curve and surface rather than with geometric types such as straight line, circle, sphere, cone, etc. This makes it possible to change the geometric set, if necessary. For the user the representation should be completely transparent. It should not be necessary to know whether geometry is represented by a general type, such as NURBS, or by explicit types such as planes, straight lines, cylinders, etc.

On the next level up are simple traversal routines, for finding neighbouring elements in the datastructure or finding sets of connected entities. Also, there is an important class of operation, called the Euler operators, which are used to create complex operations. There are also more complex utilities, such as copying and transforming objects.

Complex operations, like Boolean operations or extrusions, are built on top of the simple utility routines to make them more stable if changes are made to the

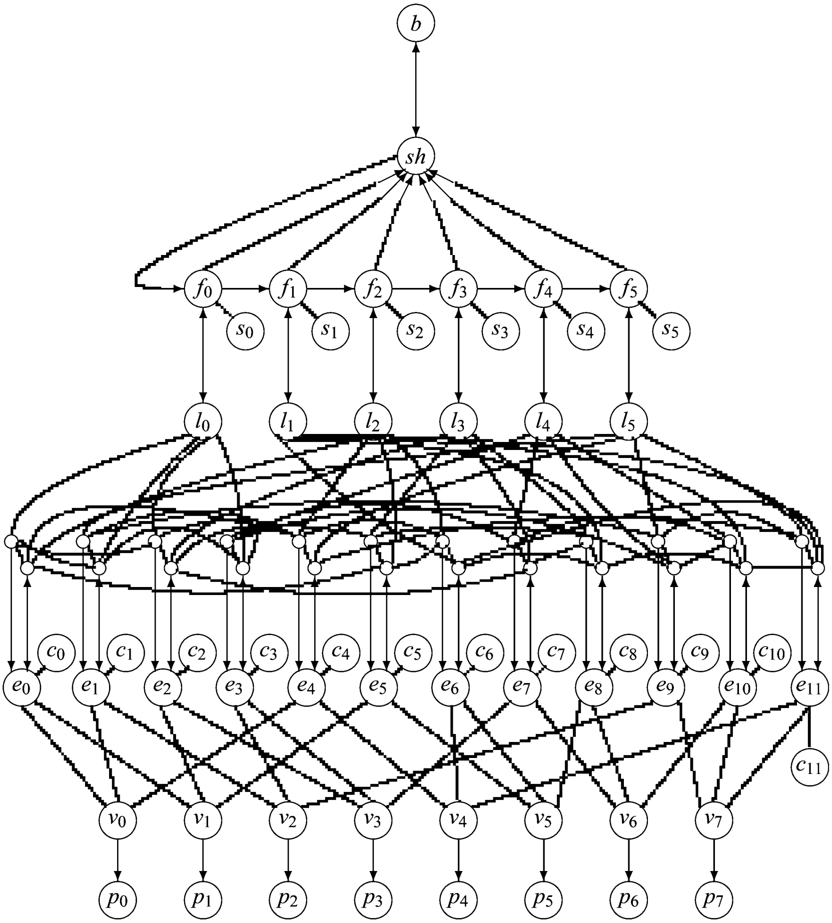


Fig. 2.21 Simple cube topological datastructure

basic datastructures. Applications can be built on top of the high level routines. The boundaries of these may be a bit fuzzy but this has been a general strategy since it was established in the BUILD system.

2.7.1 Finding Entities from Each Other

The datastructure handling routines are described more fully in [3]. Among the important topological routines are the traversal routines. There are multi-element traversals, such as finding all edges in a body, or all connected faces edges and vertices in a body part. These traverse the exact data structure and return a set of

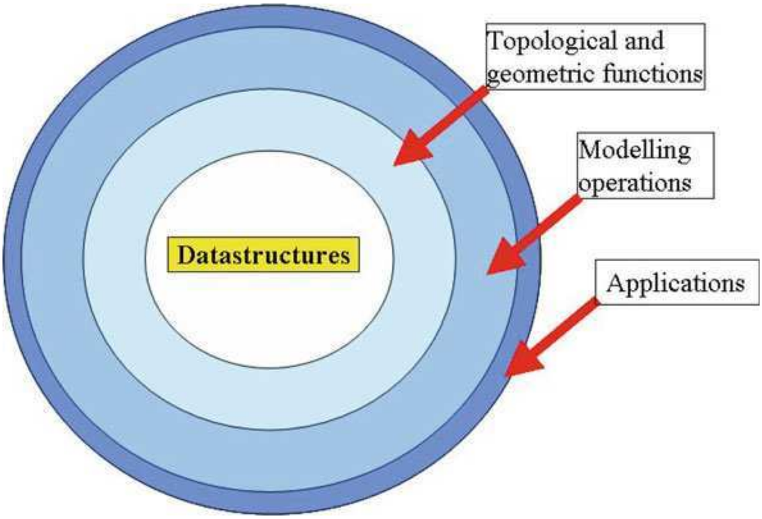


Fig. 2.22 Modelling "onion" or layered development

elements in the form of a simple list. The exact connections between the elements are handled by the routine, the user just sees the list, which is a well understood form. There are also single entity traversals to return neighbouring elements, such as the vertex at the opposite end of a given edge from a given vertex. The geometrical routines include such things as intersections, curve tangent at a point, surface normal at a point, creating surfaces by extruding, or sweeping, curves.

The single entity traversal routines, from [3] are:

Orientation using an edge as a "bridge"

vopev Find the vertex opposite a given vertex along a given edge.
lopel Find the loop opposite a given loop across a given edge.

Using an edge and a loop to find an edge or vertex

ecwel Find the edge clockwise around a given loop from a given edge.
eccel Find the edge counter-clockwise around a given loop from a given edge.
vcwel Find the vertex clockwise around a given loop from a given edge.
vccel Find the vertex counter-clockwise around a given loop from a given edge.

Using an edge and a vertex to find an edge or a loop

ecwev Find the edge clockwise around a given vertex from a given edge.
eccev Find the edge counter-clockwise around a given vertex from a given edge.
lcwev Find the loop such that the given edge is clockwise around that loop from the given vertex.
lccev Find the loop such that the given edge is counter-clockwise around that loop from the given vertex.

Using a loop and a vertex to find an edge

ecwlv Find the edge clockwise from a given vertex around a given loop.
ecclv Find the edge counter-clockwise from a given vertex around a given loop.

Miscellaneous

eclev Find the edge clockwise or counter-clockwise from a given edge around a loop in the direction of a given vertex.
vve Find the edge connecting two given vertices (if any).

The next level of operations above these are what are termed Euler operators. These are very useful for developing different algorithms so are dealt with in more detail.

2.7.2 Finding Sets of Connected Entities

As stated in Stroud [3], the structure traversal procedures provide a standard way of accessing the datastructure without having to know the exact relationship between the owner entity and the subsidiary entities. These take a structure known by the implementer and converts it to a simple list which can be handled by an application routine.

Using the classification from Stroud [3], the traversal procedures can be divided into four groups based on the usual type of datastructure.

Owner to single-level structure following:

- Vertices in body
- Edges in body
- Loops in face
- Faces in facegroup containing only faces
- Instances in a group of objects
- Notes in entity

Shared entity traversal:

- Surfaces in object
- Curves in object
- Single objects in a group of objects

Tree-structure traversal:

- Facegroups under object or facegroup
- Faces in body
- Faces in facegroup where the facegroup contains facegroups
- Instances under body

Topological set traversal:

- Edges in vertex
- Edges in loop
- Faces, edges and vertices in a shell

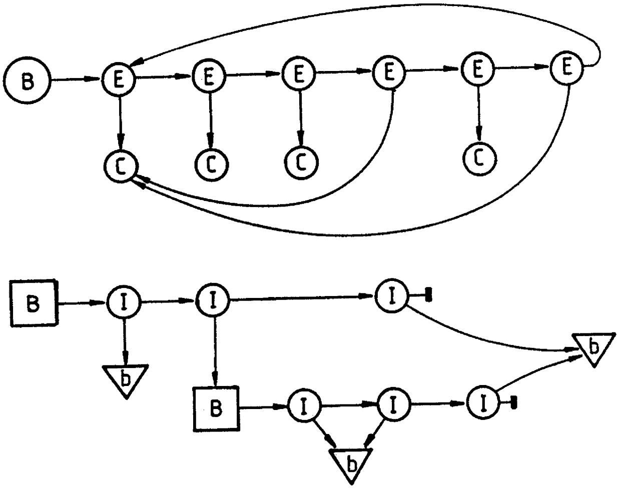


Fig. 2.23 Topological structures

Some entities can be shared in structures, so the shared entity traversals are needed to make sure that you do not process the same entity twice. For example, curves may be shared by several edges if, say, a single curve is broken into several pieces. A possible structure is illustrated in Fig. 2.23, top (from [3]). This can be a problem when transforming a body because it would be wrong to transform a curve once for each edge referring to it. Another possibility is that there are shared instances in assemblies, as illustrated in Fig. 2.23, bottom. The figure shows a structure with six instances (shown as circles with the letter I), two assemblies (squares with the letter B), and three basic objects (triangles with the letter b). The top-level assembly consists of three instances, the first referring to a basic, unshared object; the second to a sub-assembly; and the third to a basic object also referred to from the sub-assembly. The sub-assembly contains three instances, two referring to the same object, and the third to the same object referred to from the instance at the top level.

The tree-traversal algorithms are used to process all faces in an object, for example, or all single objects in an assembly.

The last group of traversal functions is important and uses topological relationships to traverse structures. The first two are more-or-less straightforward, but the third is more interesting, necessary to separate the shells in bodies.

2.7.3 Euler Operators

This section is based on a chapter from [3]. Some of this text is copied directly from there for completeness.

In mathematical terms the standard B-rep datastructure is a graph with certain properties. In the datastructure described so far, volume objects, sheet objects and combinations of these are called Eulerian objects. The name comes from graph theory because the elements in the datastructure form a graph in mathematical terms. Recent developments in non-manifold modelling mean that the nature of the graphs representing a model may be non-Eulerian. This chapter describes the basic Euler operators for the simpler datastructure, non-manifold datastructures will be discussed in Chap. 6. Extensions for creating basic manipulation operators for non-manifold datastructures are fairly straightforward. The importance of Euler operators lies in their use for low-level manipulation of the datastructure. They preserve the topological integrity of the object, making minimal changes only.

For Eulerian objects, the numbers of elements in the datastructure for a valid object or objects are related by a series of rules, described by Braid, Hillyard and Stroud [8] as:

1. $\underline{v}, \underline{e}, \underline{f}, \underline{h}, \underline{g}, \underline{b} > = 0$ [This is the condition that a valid object cannot have a negative number of elements.]
2. if $\underline{v} = \underline{e} = \underline{f} = \underline{h} = 0$, then $\underline{g} = \underline{b} = 0$ [This condition means that an object with genus 1, say, but with no other elements is disallowed.]
3. if $\underline{b} > 0$ then a) $\underline{v} > =\underline{b}$ and b) $\underline{f} > = \underline{b}$ [A valid object must have one or more vertices, and one or more faces.]
4. $\underline{v} -\underline{e} +\underline{f} -\underline{h} = 2(\underline{b} -\underline{g})$ [the Euler-Poincaré formula.]

where $\underline{v}$ is the number of vertices, $\underline{e}$ is the number of edges, $\underline{f}$ is the number of faces, $\underline{h}$ is the number of inner-, or hole-loops, $\underline{g}$ is the genus, $\underline{b}$ is the multiplicity or number of shells.

The Euler-Poincaré formula defines a five dimensional network in the six dimensional space defined by the six topological parameters. The nodes of this network, at positive integer values of the parameters, represent the valid Eulerian objects. The operators to transform the Euler object corresponding to one node into another object at any adjacent node are termed Euler operators. These were described by Baumgart [9], by Braid et al. [8], by Eastman and Weiler [10], and by Mäntylä [11, 12], and an extension to handle non-manifold models is described by Luo [13]. The description given by Braid et al. is most appropriate here, so what follows is based on that work.

There are 99 possible Euler operators which change the number of any element by at most one, i.e. perform transitions between adjacent nodes in Euler space. Of these, 60 are obvious combinations leaving 39 unique operators. Any change, any simple or complex operation can be described in terms of combinations of these Euler operators. Since the "null" point, where there are no topological elements, is part of the network, it also follows that any object can be built using a sequence of these. The full list of Euler operators as well as the shorter list are given in Appendix A.

The numbers of vertices, edges, faces and hole-loops can be easily determined from the datastructure. The multiplicity can be counted if object shells are recorded explicitly, but the genus is more difficult to determine. Robin Hillyard developed a package to calculate the Betti numbers (see e.g. Giblin [14] and Braid et al. [8]) from adjacencies in a model. This allows the genus to be determined explicitly, rather than implicitly to balance the Euler equation of an object. If the datastructure does not have a way of representing separate shells which form cavities within a body, then their existence has to be determined in some way. However, if the shells are recorded explicitly then low-level operations which potentially split off parts of an object, such as the operation to make a face and kill a hole-loop, have to make sure that a new shell has not been created.

From a practical point of view, Euler operators form a convenient building block from which to construct complex modelling operations. They are also interesting in that their use maintains consistency of the topology of a model. However, not all of the Euler operators need be implemented. Indeed, the effect of some of them is rather obscure.

2.7.3.1 Spanning Sets and Decompositions

As described by Braid et al. [8], it is possible to choose a set of five Euler operators which form a "spanning set", combinations of which can be used to create or modify the topology of Eulerian objects. To choose a spanning set it is necessary to find five independent vectors. One possible set is:

(1, 1, 0, 0, 0, 0)—MEV, Make an Edge and a Vertex
(0, 1, 1, 0, 0, 0)—MEF, Make a Face and an Edge
(1, 0, 1, 0, 0, 1)—MBFV, Make a Body (new shell), Face and Vertex
(0, 0, 0, 0, 1, 1)—MGB, Increase the Genus and Make a Body (shell)
(0, 1, 0,−1, 0, 0)—MEKH, Make and Edge and Kill a Hole

together with their inverses.

Writing the Euler operators in matrix form, together with the final row which corresponds to the coefficients of the Euler-Poincaré, formula gives the matrix, A:

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & -1 & 0 & 0 \\ 1 & -1 & 1 & -1 & 2 & -2 \end{bmatrix}$$

Euler objects, and transitions to change one Eulerian object into another can be described as combinations of these primitives, thus:

$$q = pA$$

where q is a vector representing the numbers of the elements in the Euler object or the change in the numbers of elements, and p is the vector of the numbers of times each primitive is to be applied. Multiplying by the inverse matrix, A^{-1}, gives:

$$qA^{-1} = pAA^{-1} = p$$

The inverse matrix to that representing the spanning set is:

$$1/12\begin{bmatrix} 7 & -5 & 4 & -2 & -1 & 1 \\ 5 & 5 & -4 & 2 & 1 & -1 \\ -5 & 7 & 4 & -2 & -1 & 1 \\ 5 & 5 & -4 & 2 & -11 & -1 \\ 2 & 2 & -4 & 8 & -2 & 2 \\ -2 & -2 & 4 & 4 & 2 & -2 \end{bmatrix}$$

For a cube, say, with topological element vector:

$$(8, 12, 6, 0, 0, 1)$$

the vector describing the number of times each primitive is to be applied is:

$$(7, 5, 1, 0, 0, 0)$$

A cube can thus be created with seven MEV (Make and Edge and Vertex) operations, five MFE (Make a Face and Edge) operations, and one MBFV (Make Body, Face and Vertex) operation. Since the cube has no hole-loops and a genus of zero the other operators are not needed. Also, if any operator is to be applied a negative number of times, −n say, this is equivalent to applying its inverse n times.

Note, though, that it is possible to create a series of cube creation sequences from different combinations of these thirteen elements. Some of these are shown in Figs. 2.24, 2.25, 2.26, 2.27 and 2.28.

The ratio of edges to vertices in the final object is 3:2 and the ratio of edges to faces is 2:1. This means that every vertex has three edges and every edge runs between two vertices. Similarly, every edge has two adjacent faces and every face has four edges. Figure 2.24 shows what happens when trying to maintain the ratio of edges to faces. Figure 2.25 shows what happens when trying to maintain the ratio of edges to vertices.

Figure 2.26 shows what happens when all the MEV operations are applied first followed by all the MFE operations.

Figure 2.27 shows the other extreme case, when all the MFE operations are applied first followed by all the MEV operations.

Finally, Fig. 2.26 shows a real sequence when creating a cube.

Figure 2.29 illustrates some of what happens. The Euler space is six-dimensional, as stated earlier, but taking just three of those, the edge, face and vertex number dimensions you can just about represent this on paper. The small dots represent all combinations of faces, edges and vertices in the cube range, that is: $0 \le v \le 8$, $0 \le e \le 12$, $0 \le f \le 6$. Not all these combinations are valid. The valid combinations are governed by the simplified equation $v - e + f = 2m, m$ is usually

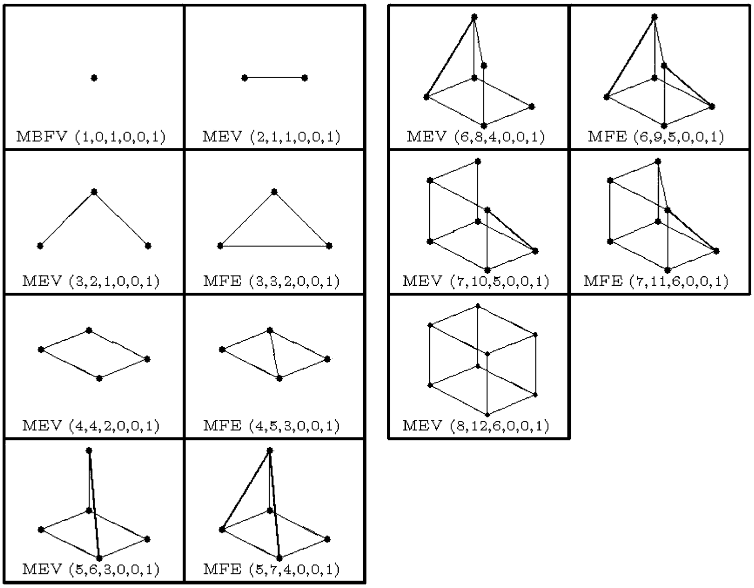


Fig. 2.24 Creating a cube with Euler operators (1)

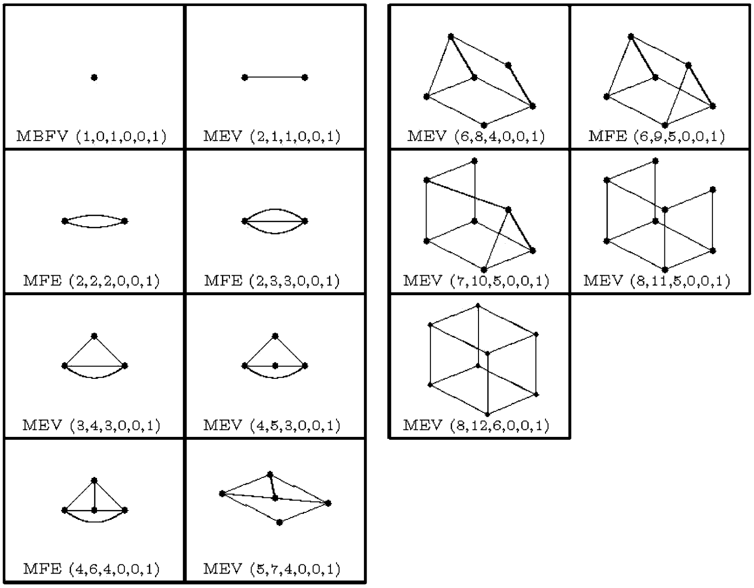


Fig. 2.25 Creating a cube with Euler operators (2)

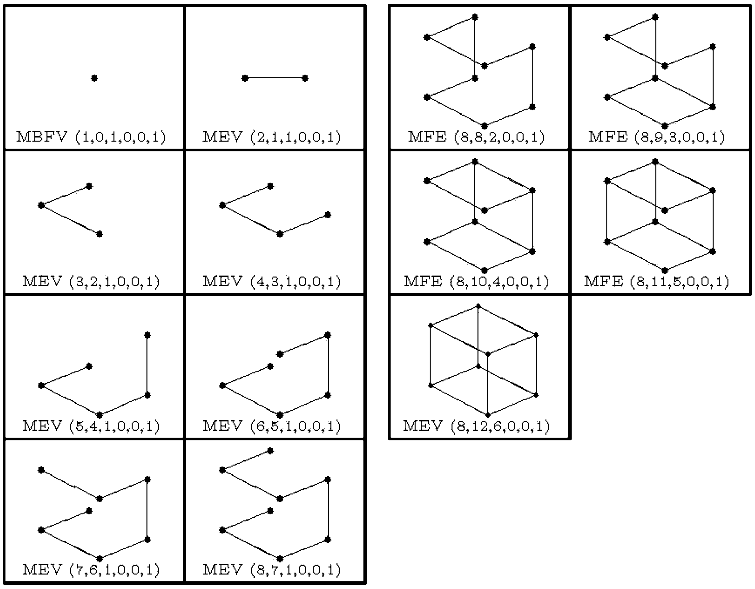


Fig. 2.26 Creating a cube with Euler operators (3)

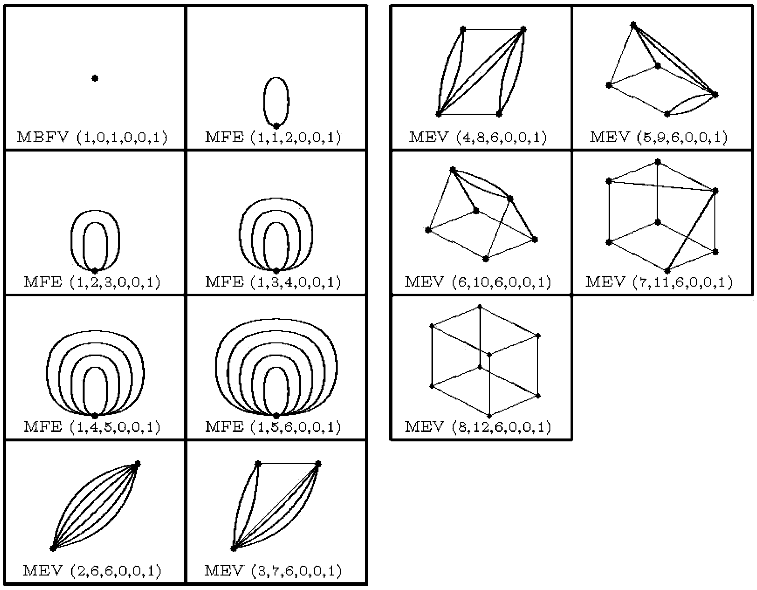


Fig. 2.27 Creating a cube with Euler operators (4)

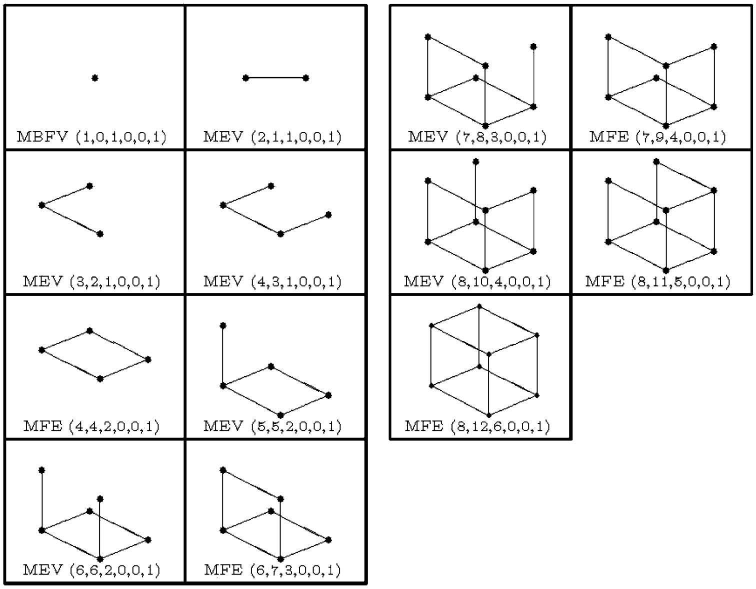


Fig. 2.28 Creating a cube with Euler operators (5)

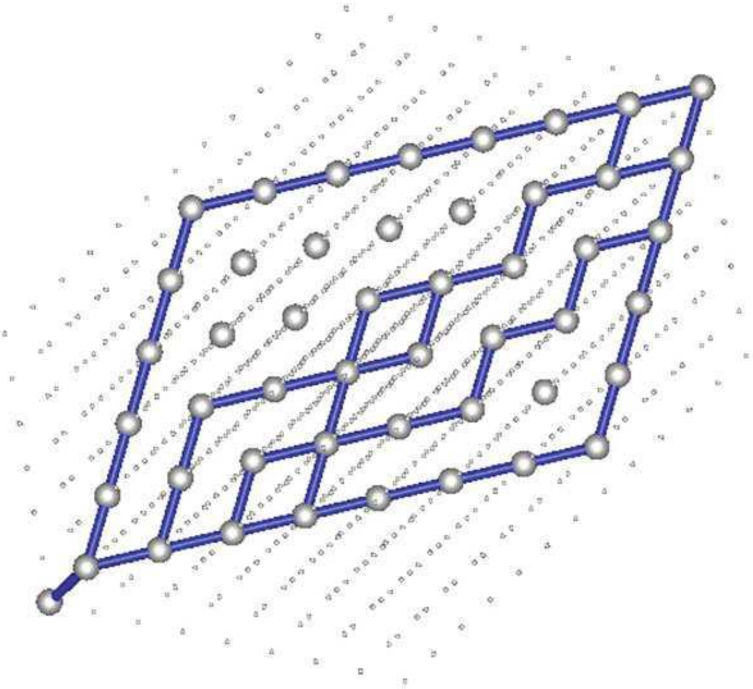


Fig. 2.29 Creating a cube with Euler operators (1)

1, but the origin is also a valid Euler point. The points which obey this equation are shown larger in the figure. The points with $v = e = 0, f = 2$ and $f = e = 0, v = 2$ are excluded by the rules defined previously. The process of building a cube means walking, more like staggering, through this point field in a particular sequence. The previous five figures show some of these sequences.

The next section describes how the Euler operators are used.

2.7.4 Stepwise Algorithms

Normally you don't see the Euler operators and the other lower level operators when you use a CAD system. The onion-layer that you see concerns high-level operations which make specific changes to a model. These, in their turn, are usually built on sequences of small changes, using Euler operators and other simple operations to make a set of small changes. Such a way of building an operation is termed a "stepwise algorithm".

More history. In the BUILD system, which established many basic principles of boundary representation modelling, there were two types of modelling operation: (1) Boolean operations; and (2) everything else.

Boolean operators are a powerful general tool performing a global comparison of two objects to produce a result. Many other operations were developed which performed specialised changes which were sometimes difficult to achieve except by using special code. This second category was important because, at that time, the Boolean operations were relatively slow because they performed a global check on objects. The other operations, sometimes called "local operations" performed localised changes, sometimes producing a global effect, but did not check the consequences. This meant that it was sometimes possible to create invalid objects termed "self-intersecting objects". Nowadays things have changed. Boolean operations have become faster, based principally on results by Smith in BUILD, and so several operations now create volumes and then apply a Boolean operation to add or subtract this from the part being modified. This is explained in Chap. 4, but since this is not the only way to do things, and since not every operation can be done that way, here is an explanation of the stepwise methodology.

2.7.4.1 Linear Extrusion

An example of this type of operation is the extrusion operation. The sequence for creating a cube is shown in Fig. 2.28. Another example, for a six-sided shape is shown in Fig. 2.30. Figure 2.30a shows the original figure. The first step in the extruding the base shape is to add an edge and a vertex in the extrusion direction using an MEV operation, Fig. 2.30b. A second edge is added in the same manner, Fig. 2.30c and then a cross-edge added, Fig. 2.30d. The sequence then continues,

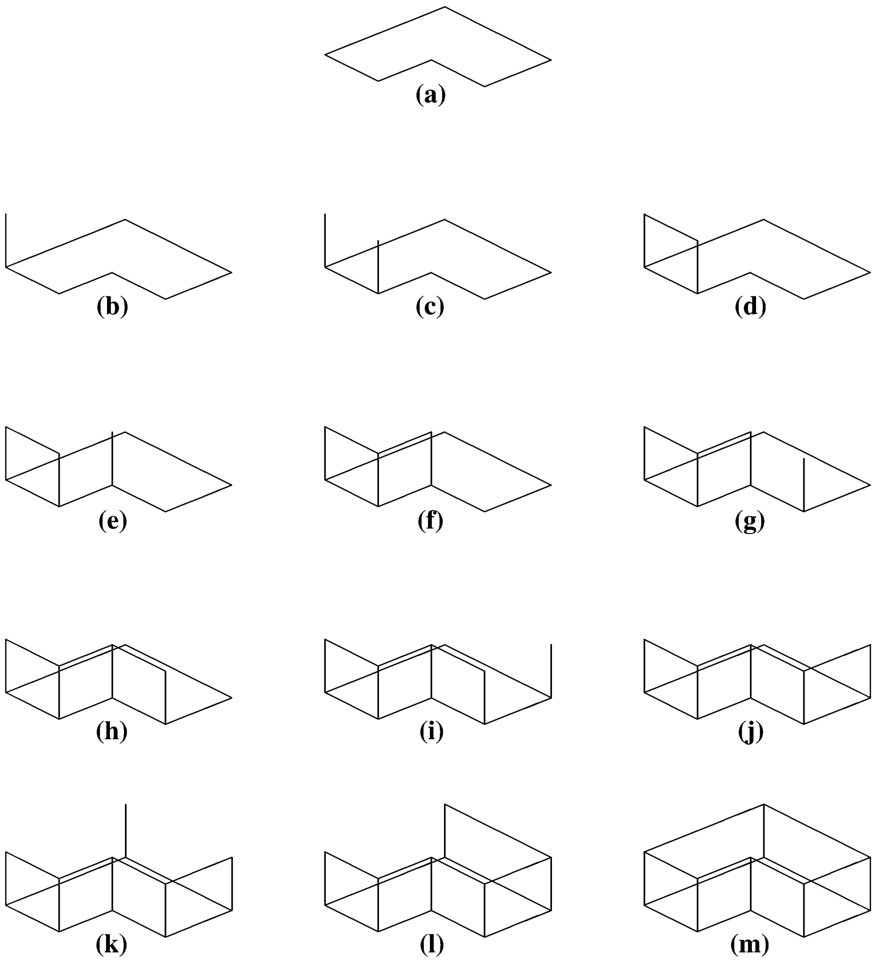


Fig. 2.30 Linear extrusion in a stepwise manner

add edge-in-extrusion-direction (Fig. 2.30e), add cross-edge (Fig. 2.30f), add edge-in-extrusion-direction (Fig. 2.30g), add cross-edge (Fig. 2.30h), add edge-in-extrusion-direction (Fig. 2.30i), add cross-edge (Fig. 2.30j), add edge-in-extrusion-direction (Fig. 2.30k), add cross-edge (Fig. 2.30l). Finally a single cross-edge is added to join the last two extensions, (Fig. 2.30m).

A linear extrusion of a face (a flat shape has two faces, one on top and one on the bottom) can be defined as a sequence of steps, one for each edge surrounding the face. The very first step is to create a single extrusion edge. Then, for each edge until the last, one extrusion edge and one cross edge are created. For the final edge a single cross edge is created between the last extrusion vertex and the first extrusion vertex created in the special first step. Each "step" is governed by an edge. The edges are found from the loops, or contours, around the face.

In pseudo-code form this might look like:

```
for all loops in face do
for all edges counter-clockwise round loop do
BEGIN
if (edge IS start OF loop)
v0 = oldv = MEV(vcwel(edge, loop), extrusion direction);
if (edge IS last OF loop) v = v0;
else v = MEV(vccel(edge, loop), extrusion direction);
e = MFE(oldv, v); oldv = v
END;
```

The function "vcwel" finds the vertex clockwise from a vertex from a given edge round a given loop. Similarly, the function "vccel" finds the vertex counter-clockwise from a vertex from a given edge round a given loop. These are two of a set of simple relational functions for traversing the data structure. The complete set is described in [3].

This is the very simplest form of extrusion. Nowadays this is the extrusion that is done. When you extrude a shape drawn on an existing face then the shape to be extruded is made into a 2D shape, extruded and then combined using a Boolean operation. More about this later.

2.7.4.2 Splitting an Object

Splitting an object with a plane can also be defined in a stepwise manner. In the extrusion example the edges were found and extruded in an ordered sequence. For the splitting algorithm here this is not done so there are some special cases which need to be recognised and handled.

The pseudo code is:

```
for all faces in object do split_face_with_plane(face, plane);
pull_results_apart;
```

This is illustrated in Fig. 2.31.

Most of the work is in the split_face_with_plane function. As can be seen from the outline of the split operation in Fig. 2.31 this inserts edges where the section plane cuts the face (e.g. Fig. 2.31b). These edges are then sliced (e.g. Fig. 2.31c). If one or both end vertices of the slice edges are also end vertices of other slice edges, the vertices are split (e.g. Fig. 2.31f). Eventually, splitting the vertices creates two disjoint loops which become the outer loops of two new faces.

A complete operation is a little more complicated. The basis is still a stepwise construction centred around the central function, split_face_with_plane, but this needs to identify different special cases. For example, what happens if they are several new section edges in the face? What happens if an edge of a face lies in the

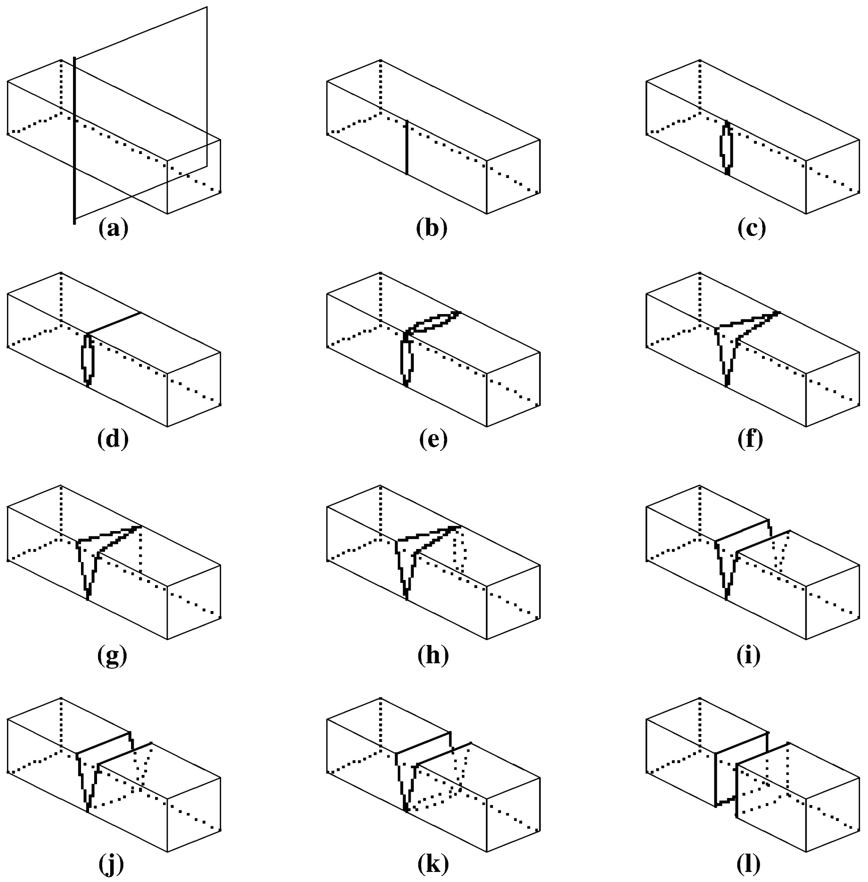


Fig. 2.31 Splitting an object with a plane

section plane? Extending the function to identify special cases keeps the stepwise character, though.

For the first question, if there are multiple section edges then these are created and sliced separately, as shown on the top line of Fig. 2.32.

Multiple section edges may also just touch a boundary, as in the case shown on the bottom line of Fig. 2.32. In this case, as well, the section edges are created and sliced, but there is an extra step to slice the common vertex, leaving the result in Fig. 2.32f.

For an edge lying in the section plane, such as that shown in Fig. 2.33, there are two cases. If the edge is convex then the object does not cut the section plane at that edge, so the edge is ignored. If the edge is concave, as in the figure, then the edge is sliced. Note, though, that the sliced edge has to be ignored when the other adjacent face is processed.

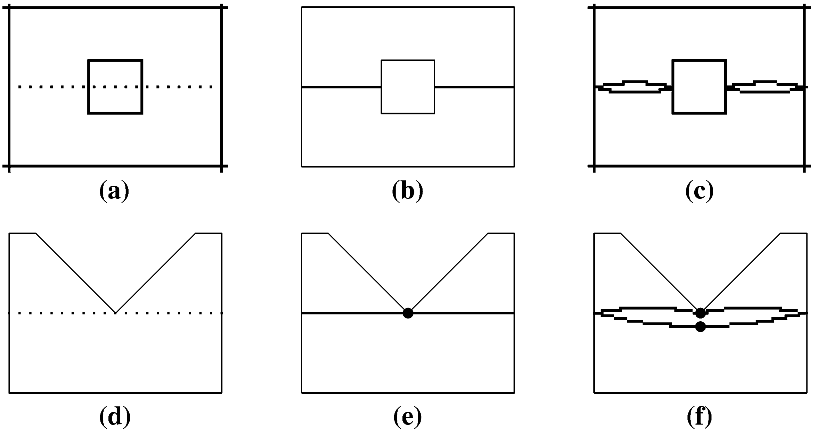


Fig. 2.32 Handling multiple section edges in a face

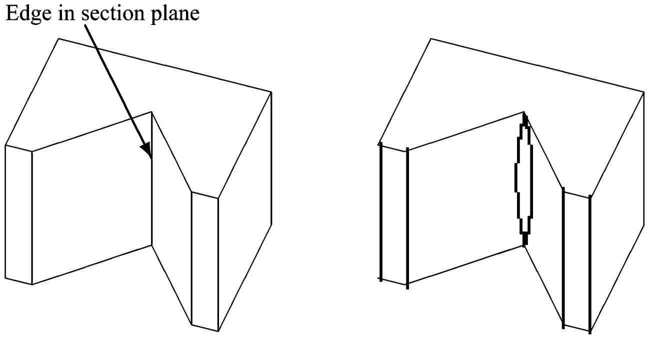


Fig. 2.33 Edge lying in section plane

The same set of simple steps are used to create the section seam. Once the section seam, or seams, are complete the connected sets of faces are moved into separate bodies.

These are just simple illustrations of how stepwise algorithms are built up from repetitions of basic steps.

2.7.4.3 Etcetera

The aim, her,e is not to run through a lot of modelling algorithms. These are described in Stroud [3] and short descriptions of several operations are given in Chap. 4. The aim is to illustrate the stepwise notion of building up operations as sequences of simple steps.

2.7.5 Complex Utilities

Finally in this section come the complex utilities. These are functions which perform well-defined common operations which are useful to higher level functions. Examples are copying objects, deleting objects, transforming objects, the dual function.

2.7.5.1 Copying

Copying is illustrated in Fig. 2.34. The entities in the object to be copied are renumbered temporarily so that they have consecutive numbers starting at zero. Lists of new entities matching the original entities are created. The original entities are then traversed and the corresponding new entities are linked into structures using the numbers as logical pointers to list elements.

Take a cube as an example. The cube might have one shell, six faces, six loops, twenty-four loop-edge links, twelve edges, eight vertices, six surfaces, twelve curves and eight points. The shell is numbered 0, the faces, loops and surfaces are numbered 0–5, the loop-edge links are numbered 0–23, the edges 0–11 and the

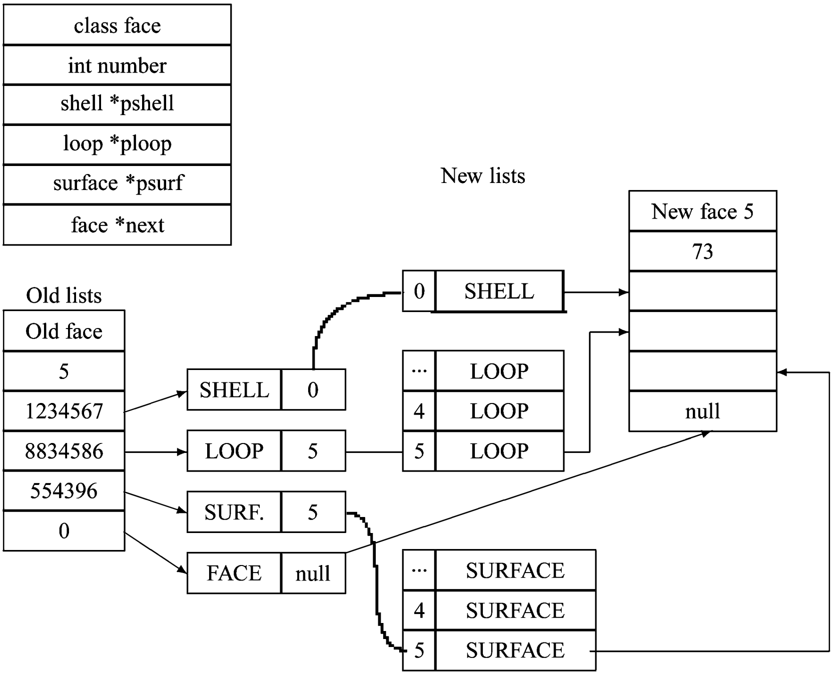


Fig. 2.34 Writing a face to disk

vertices and points numbered 0–7. A list of one new shell is created, a list of six faces is created and so on, for each type of entity.

Figure 2.34 shows what happens for face number 5 when linking the data. The datastructure definition is used to interpret the data. If there are pure numerical fields (except for the entity number), which there aren't in this case, they are simply copied. For the face, the first relevant field is the shell pointer field, *pshell*. In the original face this shell is numbered zero, so a pointer to the shell in position 0 of the list of new shells is copied into the *pshell* field of the new face. Note that the list indices start from zero, here. The next field is the loop pointer field, *ploop*. The loop referred to in this field from the original face is numbered 5, so a pointer to the loop in element 5 of the list of new loops is copied into the *ploop* field of the new face. Similarly for the surface, the surface referred to in the *psurf* field of the original face is numbered 5. A pointer to the surface in element 5 of the new surfaces is copied into the *psurf* field of the new face. The final field, the *next* field is a face pointer, but this is NULL, so the *next* field of the new face is set to NULL.

This process is repeated for all the entities of the original object.

2.7.5.2 Deleting

Deleting can be applied to single elements or to whole structures. It is useful to use Euler operations to delete connected edges, faces, vertices or hole-loops in an object. This means that the resulting structure left is correct. Once the element has been deleted from the topological structure it can be removed completely by disconnecting it from any lists in the object to which it belongs. This, again, is done with specialised low-level routines which handle the datastructure directly.

However, although it is possible to delete complete objects, or shells also, using Euler operators it may be more efficient to delete all entities more directly. At any rate, it is necessary to understand the difference between deleting using Euler operators, which disconnect elements, and pure deletion which probably doesn't tidy up completely surrounding elements.

2.7.5.3 Transforming

Transforming objects is done on three levels. What appears to be an object might be an instance in an assembly, which means that the transformation is multiplied into the instance transformation, and doesn't change the original geometry. Transformation matrices will be discussed in Sect. 5.2.2. If the object has a transformation attached then, again, the transformation may simply be accumulated with the object transformation instead of changing the real geometry. The third level really changes the geometry.

Accumulating transformations tends to be preferred to really changing the geometry for several reasons. First of all it is faster, second, some systems allow non-uniform scaling, which changes the nature of the geometry, which is usually transformed to general numeric forms. However, it is harder to go back from the numeric to the simpler forms, so you get what is called "geometric migration", to be discussed later, in Sect. 5.3.

Really transforming single objects changes the geometry of an object but not its topological structure. This is one example of where you need to identify elements which are used several times and transform them only once. To transform the geometry it is best to compile a list of geometric elements using the set traversal tool described earlier and then process each one, changing its nature if necessary, and applying the transformation.

2.7.5.4 Dualling

Dualling is a strange operation that normally you would not want to use, nor even know about, but it is useful as a background structure for some operations. An example is shown in Fig. 2.35.

In effect, the dual operation creates a new object with a vertex for each face and a face for each vertex, which means that the dual of a cube (with eight vertices, twelve edges and six faces) is an octohedron, with six vertices, twelve edges and eight faces. It is a graph theoretic operation which is used, maybe directly for some smoothing operations, or indirectly as a navigation aid for unfolding objects, for example.

The operation also uses a technique such as that described for copying. The first step, renumbering the entities, is the same as for copying. However, instead of creating lists of new entities of the same number of elements as in the original object, there is a switch. For a cube, a list of eight new faces and a list of six new vertices are created. Edges are reconnected appropriately. For example, take an

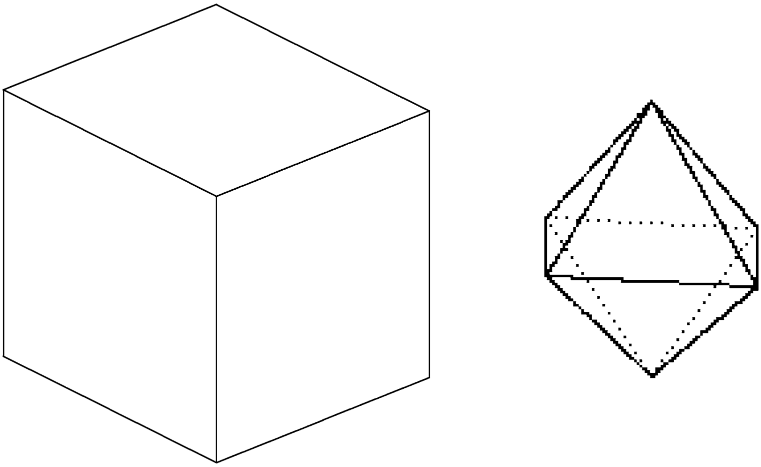


Fig. 2.35 Cube and dual (from Stroud [3])

edge lying between faces A and B, running between vertices C and D in the original object. In the new object, the corresponding edge will run between vertices A and B and lie between faces C and D.

2.8 Complex Geometry and Integration

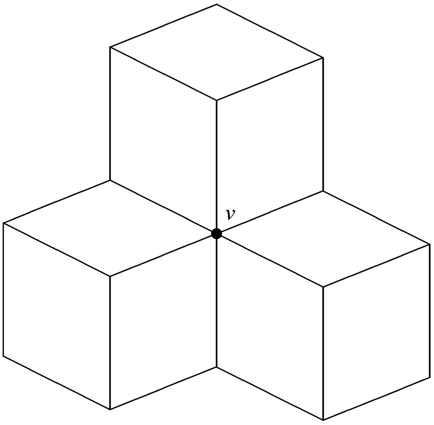
The next topic to mention briefly concerns the use of geometry in modelling. Geometry in modelling systems is usually of two types: analytic or numeric. Analytic geometry concerns specific forms, such as plane surfaces, cylinder surfaces, cone surfaces. Numeric geometry is a general form the shape of which is determined by a set of points called control points. Normally these are mixed in a manner which is transparent for the user. You should not need to know which is used because the system should use the appropriate one and modify it according to set rules.

As an example, try the example in Fig. 2.36.

I don't know who created this example, possibly Fjällström in his dissertation [15], but it is a useful example to show the use of complex surfaces. Create the object shown in the figure and then blend the six edges meeting at the central vertex, marked "*v*" in the figure. Even though the original geometry is simple, when the edges are blended it is usual that there is a complex blend surface, or surfaces, instead of the vertex. Similarly, intersections between curved surfaces may produce complex curves which are calculated automatically.

This is what happens in normal modelling and is taken care of automatically. Complex geometry is used to model all surfaces not modelled explicitly. There are other occasions, though, when numerical geometry is used explicitly. For example, for car bodies or other products with aesthetic shapes, or for functional forms like turbine blades. In this case the shape is usually created as a surface form, although

Fig. 2.36 Complex blend example



if these are used in isolation the problem is then to integrate them with solid models.

The integration of solid and surface modelling has long been the subject of research. One method was the sol-called "SETSURF" method, which sets a surface into a face (Braid [16]). Another interesting method was to smooth polyhedral models by inserting sort-of blend surfaces (Chiyokura and Kimura [17]). Hybrid modelling, with elements suitable for representing surfaces, was introduced by Kjellberg et al. [1]. This hybrid methodology is now common in CAD systems and is described further in Chap. 6.

2.9 The Cylinder Test

The "cylinder test" is a small diversion from the main theme of this chapter. It involves creating a cylinder and circular shape to identify how the system handles closed geometry. This should be transparent for users, but this is not always so. Identifying this gives clues to the way the system works.

In the original research system, BUILD, cylinders were represented with three side edges, because the "point-in-face" test counted angles. The first commercial kernel system, Romulus—a forerunner of today's Parasolid and ACIS systems, advanced this, but maintained a single edge down the cylinder side, presumably also for the "point-in-face" test. The ACIS kernel has no side edges but has two vertices on the top and bottom circular edges.

So, use your CAD system to create a cylinder and count the number of edges around the cylindrical surface.

- Zero – The modeller in the CAD system is modern and transparent for the user.
- One – The modeller is a little old-fashioned, you will get artefact edges of rotational objects.
- Two – CATIA has this, and this avoids another problem, that a single edge is a so-called "wire" edge with the same face on the left and right. These are also artefact edges and may not need to be at 180 degrees.
- Three or more – old-fashioned, maybe a BUILD derivative would have three, but there should be no need to have these in modern systems.

A similar problem can be found at the level of 2D. Both cylinders and circles are examples of "continuous" geometry, that is, geometry which curves back on itself. A true circle may have neither start nor end, but this is not true for CAD geometry. For practical purposes it is necessary to have a start and end, albeit at the same place. This is connected with parametrisation of geometry, which is a necessary mechanism in many algorithms. For a circle, the parametrisation may run from 0 to 2π.

A test you can do in CATIA v5 is to create a circular curve and then cut it with a vertical line, as shown on the left of Fig. 2.37. Instead of being divided into two pieces the circle is in three pieces, as can be seen by selecting the parts. The parts

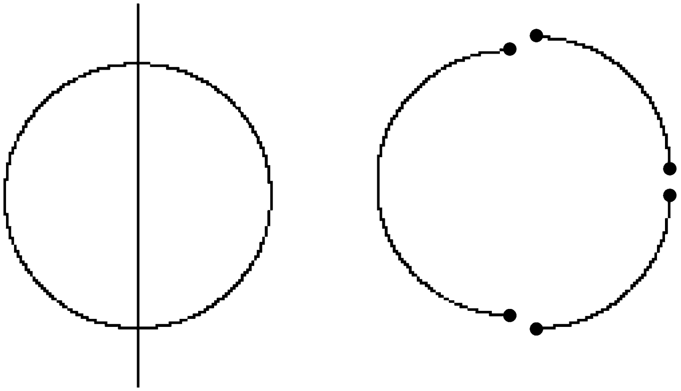


Fig. 2.37 Cutting a circle

are shown a little displaced on the right of Fig. 2.37. The reason is, apparently, that the point dividing the right-hand "half" of the circle is the start- and end-point of the circle. Instead of adjusting the circle and using one of the new intersection points when it is cut, CATIA maintains the original point. There may be a good reason for this, but I can't think of one. For the user it provides an oddity which does not seem logical at first glance. A personal opinion is that it would have been better to make this sort of artefact invisible to the user.

2.10 Assemblies

Assemblies are a frequent problem because the implementation method is not well understood. Chapter 13 explains more about assemblies. A modern assembly structure is shown in Fig. 2.38.

A major conceptual difficulty concerns how objects which occur more than once in an assembly are represented. There are two ways of doing this, by physically copying the object or by referring to it more than once. The latter

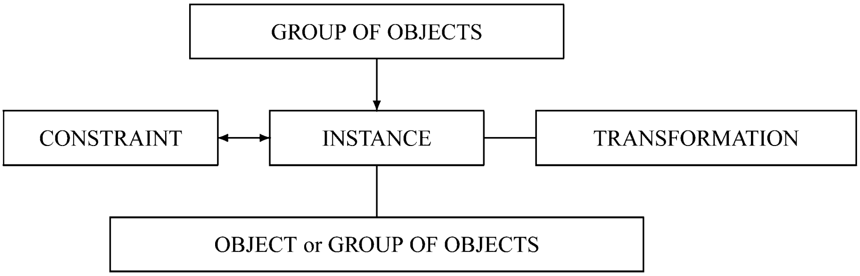


Fig. 2.38 General assembly datastructure

method is the "computer science" solution, which has several advantages, but many people seem to want to copy the object and then do not understand why the CAD system does not "know" that the copies are the same object. Figure 1.57. shows the expected structure of an assembly with multiple elements. A simplified assembly is shown in Fig. 2.39.

A simple rule is that, in an assembly, if multiple parts with the same shape are used, then there should be one object model of the part shape and multiple instances. Each instance has a transformation associated with it containing the information about repositioning of the part from its defined position to the position at which it is needed. Transformations are described in Sect. 5.2.2.

If one of the instances is to be modified then it is normal to copy the common object and to change the instance to point to the copy, which is then modified, as in Fig. 2.40. If the referenced object is modified without copying then all instances will show the change. It is, therefore, necessary to think about the effect you want before modification, if you want the change propagated to all instances or to change just one instance. Some systems let you perform the re-instancing operation explicitly, to "make-unique" one instance. This leaves you in control of the changes.

There is a special type of part used in assemblies which is "standard". These can be parts which are to be purchased from suppliers, such as nuts, bolts, motors, etc. Normally these should not be modified and so may be "blocked" from being changed. Such standard parts may come in 3D catalogues from suppliers. It is necessary, then, to consider the format in which they are communicated. This could be in a standard format, such as STEP (see Chap. 9) or in native format.

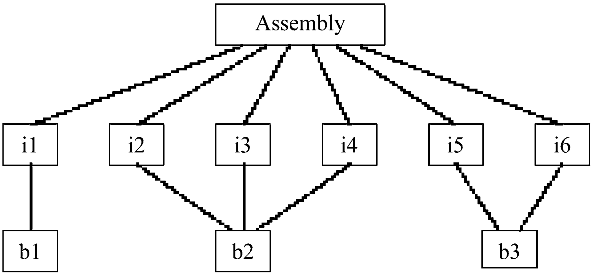


Fig. 2.39 Simple instance structure

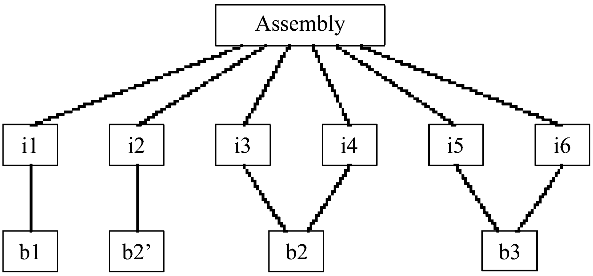


Fig. 2.40 Simple instance structure

If they are in a neutral format, like STEP, then you will probably only get the final shape anyway, which makes modification difficult.

If parts, standard or otherwise, are to be modified physically, say by drilling a hole in a baseplate, or cutting off a bolt, say, then the method mentioned above for copying instances loses the connectivity between the parts. This is a pity, since this is likely to increase the work in the manufacturing stage. A method for doing this would be to assign local modification "trees" to the instances themselves and to allow only material removal operations. I have not seen this in any CAD system yet, so part association would have to be done on paper or by some other means, such as an external database, or PLM (Product Lifecycle Management) system.

2.11 Chapter Summary

This chapter describes, briefly, solid modelling methods. The details of solid modelling are outside the scope of this book. However, the different modelling methods have different implications for the user, and hence a cursory knowledge of the methods is useful for understanding what is going on. The CSG method was described, which uses Boolean operators on primitive objects to build models. The method currently used for the majority of CAD systems, Boundary representation, was also described. Boundary representation was described in more detail because this is the current method used in most CAD systems. Specifically, some simplified datastructures, the layered manner of implementing systems as well as some of the more common operations were described.

2.12 Representation Exercises

These exercises are intended to test your understanding of representations.

2.12.1 CSG Decompositions

Figures 2.41, 2.42, 2.43 and 2.44 show simple objects to try decomposing into CSG trees of primitive objects and Boolean operations.

2.12.1.1 The Cylinder Test

Perform the "cylinder test", described in Sect. 2.9, using your CAD system.

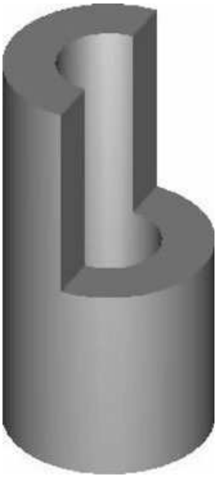


Fig. 2.41 Object 1 to decompose in CSG trees

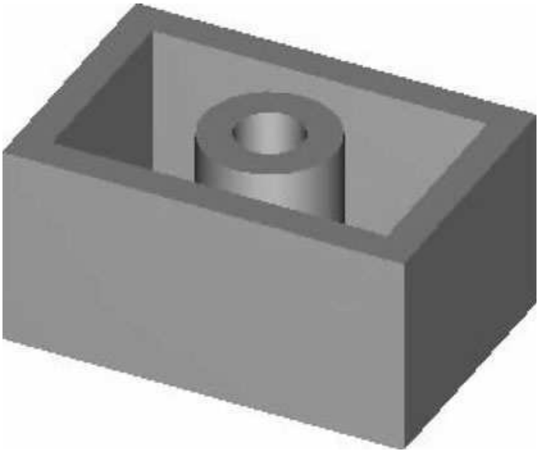


Fig. 2.42 Object 2 to decompose in CSG trees

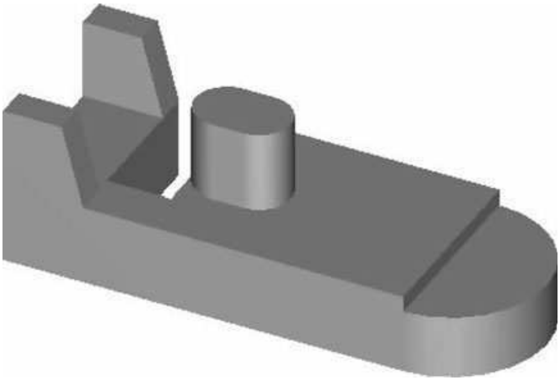


Fig. 2.43 Object 3 to decompose in CSG trees

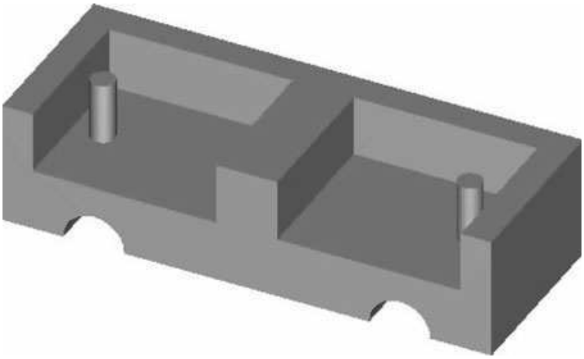


Fig. 2.44 Object 4 to decompose in CSG trees

References

1. Kjellberg, T.A., Lindholm, G., Sorgen, A., Haglund, G.: GPM Report 10: GPM—Specifikation Volymgeometri. Department of Manufacturing Systems. KTH, Stockholm, Sweden. Confidential document, ISBN 91-85212-54-7 (1980)
2. Jared, G.E.M., Dodsworth, J.: Solid modelling. In: Rooney, J., Steadman, P. (eds.) Principles of Computer-aided Design. Pitman Publishing in association with the Open University, ISBN 0 273 02672 0 (1987)
3. Stroud, I.A.: Boundary Representation Modelling Techniques. Springer, Heidelberg (2006)
4. Okino N., Kakazu Y., Kubo H.: TIPS-1: technical information processing system for computer-aided design, drawing and manufacturing. In: Proceedings of the Second PROLAMAT, vol. 73, pp. 141–150. (1973)
5. Requicha, A.A.G., Voelcker, H.B.: Constructive solid geometry. Technical Memo. No. 25, Production Automation Project, University of Rochester, NY (1977)
6. Katainen, A.: Monadic set operations. CAD J **14**(6) (1982)
7. Tilove, R.B., Requicha, A.A.G.: Closure of Boolean operations on geometric entities. Computer-Aided Des. **12**(5), 219–220 (1980)
8. Braid, I.C., Hillyard, R.C., Stroud, I.A.: Stepwise construction of polyhedra in geometric modelling, (1978). In: Brodlie, K.W. (ed.) Mathematical Methods in Computer Graphics and Design. Academic Press, London (1980)
9. Baumgart, B.G.: Geometric modelling for computer vision. AD/A-002 261, Stanford University (1974)
10. Eastman, C.M., Weiler, K.: Geometric modeling using the Euler operators. Research Report 78, Institute of Physical Planning, Carnegie-Mellon University (1979)
11. Mäntylä, M.: A note on the modeling space of euler operators. Comput. Vis. Graph. Imag. Process. **26**, 45–60 (1984)
12. Mäntylä, M.: An introduction to solid modeling. Computer Science Press, Maryland, ISBN 0-88175-108-1 (1988)
13. Luo, Y.: Solid modelling for regular objects renewed theory, data structure and Euler operators. Ph.D. Dissertation, Computer and Automation Institute (1991)
14. Giblin, P.J.: Graphs, Surfaces and Homology. Chapman and Hall, London, ISBN 0 412 21440 7 (1977)
15. Fjällström, P.-O.: Integration of Free-Form Surfaces and Solid Modelling. Ph.D. Dissertation, Department of Manufacturing Systems, PS-Lab, IVF/KTH, Stockholm, Sweden (1985)
16. Braid, I.C.: Notes on a geometric modeller. CAD Group Document 101, Cambridge University Computer Laboratory (1979)
17. Chiyokura, H., Kimura, F.: Design of solids with free-form surfaces. Comput. Graph. (SIGGRAPH 83 Proc.) **17**, 289–298 (1983)

Chapter 3
2D Shape Definition

In this chapter the intention is to start decomposing CAD systems into different functional units, to explain how they work, and to suggest experiments to try. The experiments are important so that you can see what happens in different cases and understand what is happening. The experiments are personal, they are things that I try. You should build up your personal set to try so that when you are faced with a new system, or a new version, you can start to analyse its functionalities.

Two dimensional shape construction is one of the basic building blocks of CAD systems. This construction is normally done on a plane. It could, in fact, be extended to drawing on curved surfaces by using the same tools to describe shapes in the parametric plane of a surface and then converting them to real space. However, while it might be interesting for some applications, especially for decorative work, the common use is to create planar shapes. These shapes are then extruded, in a line or circle, for example, to create base shapes for a design.

These techniques follow on from old drawing practice where the designer created construction lines in pencil and then inked them in for the final drawing. This 2D shape represents the projection of a shape, or the profile of a rotational shape. In solid modelling two dimensional shape definition has been used since the start. In Braid's original BUILD system [1] this was done using simple command interpreter commands. In further developments to the BUILD system the commands were extended and improved by Jared. As interactive graphics techniques have been developed the technique has changed somewhat, but the methods still have relevance for history files, for example.

3.1 BUILD's 2.5D

The BUILD system was the boundary representation modelling research system which was (is?) the grandfather of the systems used today. The main development period preceded the introduction of interactive graphics so BUILD worked with

I. Stroud and H. Nagy, *Solid Modelling and CAD Systems*,
DOI: 10.1007/978-0-85729-259-9_3, © Springer-Verlag London Limited 2011

textual command files. BUILD contained an elegant shape sketching package developed by Jared, which is described here.

The sketching package, called “2.5D”, worked on the Z = 0 plane in global space or on a specified face. When a face was specified as input then one of its vertices was used as the origin of a local coordinate system, otherwise the global origin was used. The system defined a geometrical framework of points, lines and circles which could be used to “ink-in” a shape to be extruded. You can see a real sequence from that period using the 2.5D command package in Sect. 12.1.1.

Points on a shape could be defined explicitly or implicitly as intersections of other elements. Lines and circles also had directions, so that extending round a circle implied going in the direction of the circle, counter-clockwise by default. If the opposite direction were needed then the circle would be referred to as “$-c1$”, for example.

A sequence to define a shape might be:

```
2.5d
circle c1 centre (70,70) radius 40
line l1 start (0,10) dir (1,0)
line l2 start (0,100) dir (1,0)
line l3 start (0,150) dir (1,0)
line l4 start (30,0) dir (0,1)
line l5 start (200,0) dir (0,1)
line l6 start (250,0) dir (0,1)
point p1 l × l l1 l6
point p2 l × c l2 c1 near (43.5,100)
start l × l l1 l4
extend to p1
extend to l × l l3 l6
extend to l × l l3 l5
extend to l × l l2 l5
extend to p2
extend round c1 to ltc l4 c1
joinup
```

The construction lines, circles and points are shown in Fig. 3.1. When “inked-in” with the “start...joinup” sequence you get the shape in Fig. 3.2.

This method has some aspects in common with the way in which 2D paper drawings were produce, with help geometries drawn in pencil and then the required shapes inked in. This was a forerunner of the methods in current systems although it was used in a static environment and not using interactive graphics.

A more complicated example, for the creation of the MBB Gehäuse Rohteil, is shown in Figs. 3.3 and 3.4. These are from the sequence shown in Sect. 12.1.1.

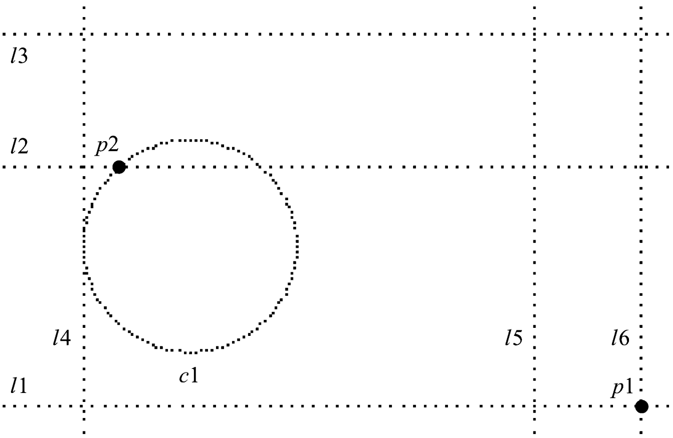


Fig. 3.1 2.5D construction elements

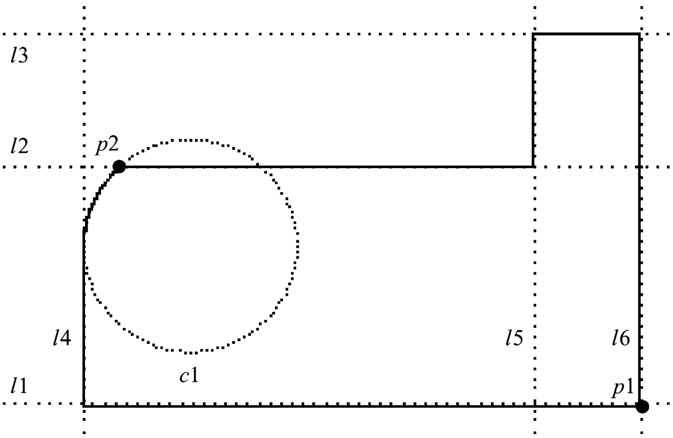


Fig. 3.2 Inked in shape

3.2 Two Dimensional Elements

This section introduces some of the basic elements used in two-dimensional shape definition. First the shapes and dimensions, later grids and rulers.

3.2.1 Lines, Curves, Dimensions and Constraints

Although there are several similarities with modern systems, interactive methods have introduced extra facilities. Modern methods set up a system of geometries linked by dimensions and constraints. This makes it possible to change shapes in

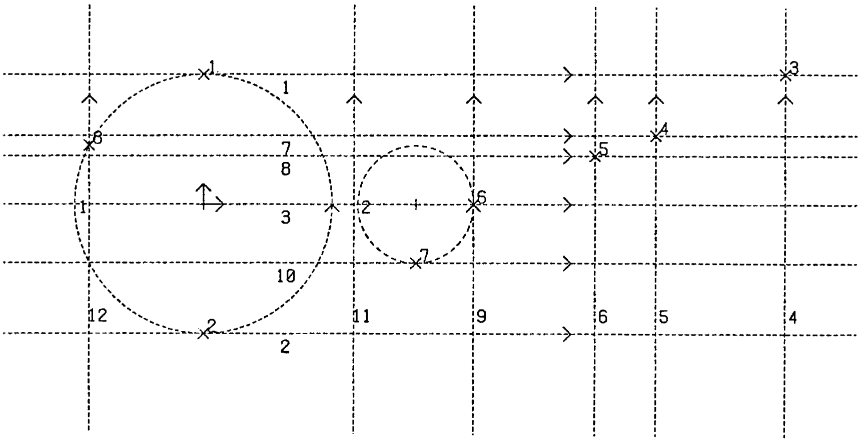


Fig. 3.3 Construction line framework for MBB Gehäuse Rohteil

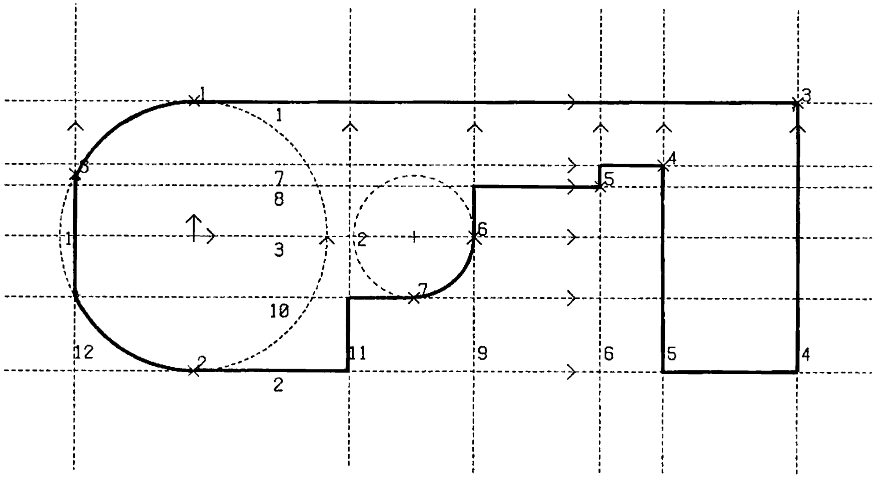


Fig. 3.4 Inked in shape for MBB Gehäuse Rohteil

real time rather than setting up systems of help geometries and rerunning command files to make changes.

First of all, identify which drawing elements you have available. Common drawing elements are lines, circles and splines. Polylines (connected sequences of lines) are also common. Other geometries are possible, it is just a question of how complicated the system developer wants to make the drawing facility. This in turn depends on the customer needs. The spline shape is a sort of catch-all, allowing any curve to be made if it doesn't exist explicitly. Note also how the system allows you to convert these shapes for extrusion. This subject is dealt with later, in Sect. 3.7 Some systems demand that the shape is closed. Usually the shape is not allowed to be self-intersecting, for practical reasons. The reason for this is explained in Sect. 3.7.

The dimensions give distances between elements, lines and points, circle radii etc. Sometimes these also imply constraints. A dimension between two lines, for example, is useless unless the lines are parallel.

A list of common (but not all possible) constraints that can be found in CAD systems is:

1. Vertical
2. Horizontal
3. Parallel
4. Perpendicular
5. Bisector
6. Midpoint
7. Fixed
8. Tangency
9. Coincidence
10. Distance constraints
11. Angular constraints

This are used for establishing relationships between elements in a two dimensional definition. Usually now geometry is visualised as being finite rather than infinite. The geometry is "held together" by a set of constraints. Figure 3.5 shows a square shape (Fig. 3.5a), what happens if the top edge is moved without the constraints between the end points (Fig. 3.5b), and the effect when the endpoints of the lines are constrained to be coincident (Fig. 3.5c).

Other common constraints are those of perpendicularity and parallelism, which can be considered as special cases of angular constraints. These are illustrated in Fig. 3.6. The original shape is shown in Fig. 3.6a. If there are no constraints and the node marked "v" is moved you might get something like Fig. 3.6b. Constraining "e1" and "e3" to be parallel and moving "v" might give you something like Fig. 3.6c. If "e2" and "e4" are also constrained to be parallel then you might get something like Fig. 3.6d. Adding an additional constraint, such as that "e1" and "e2" are perpendicular might give Fig. 3.6e. If, instead of moving "v", "e1" is inclined then the shape might be as in Fig. 3.6f.

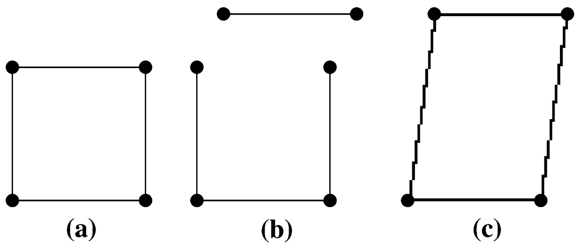


Fig. 3.5 Illustration of effects of point-point coincidence constraints

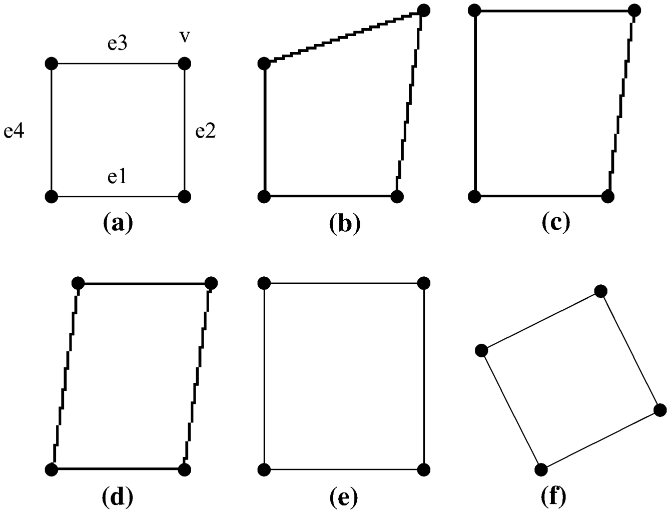


Fig. 3.6 Illustration of effects of line-line perpendicularity and parallelism constraints

The actual effects that you get vary from system to system because the “constraint solver” which is used to produce a consistent figure varies. Constraint solving will be described further in Sect. 3.5.

Another common constraint is the distance constraint, the dimension. Dimensioning and tolerancing have long been a part of geometric modelling. Robin Hillyard worked on this subject with the BUILD system [2, 3]. Gossard and Lin [4] worked with variational geometry. Bjorke [5, 6] is another well-known name in the area. A survey was produced by Bob Johnson for CAM-I [7] during the 1980s.

Dimensions are essentially distance constraints. The common ones are point–point, point–line, line–line or circle radii. Line lengths, for example, can be specified as the distance between the end-points. Spline curves can be constrained by setting up a set of points with distance constraints and then interpolating these. Ellipses can be constrained by setting them in a box with tangency constraints and dimensioning the box, and so on.

Note: point–point dimensions can be of three types. The default is to set a dimension on the shortest distance between the two points. However, you can also set vertical dimensions, which leave the points free to slide relative to each other parallel to the x-axis; or horizontal dimensions, which leave the points free to slide relative to each other parallel to the y-axis. All these three types may be useful. If there is a choice, it may be better to set a dimension between a point and a line rather than between points. This automatically defines the direction.

As an illustration of the use of dimensions, Fig. 3.7 shows examples of how a four-sided shape might be dimensioned. In the example on the left the distances of the corner points are fixed. In the example on the right, the distances between opposite sides are specified.

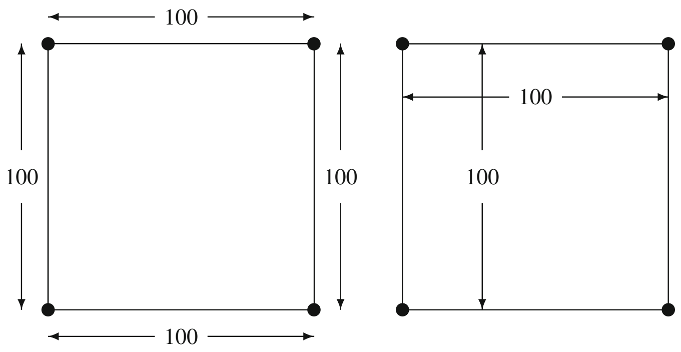
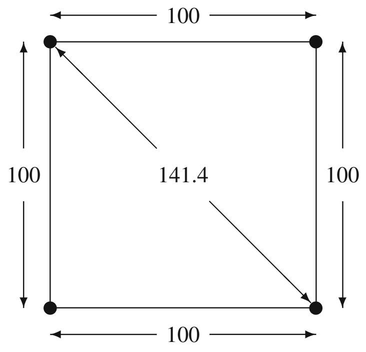


Fig. 3.7 Dimensioning schemes

Fig. 3.8 Dimensioning scheme with diagonal



The first thing to notice is that, in the example on the right of the figure, there is an implicit constraint of parallelism between the edges with the opposite edges. Were the edges not parallel then the distance constraint would have no meaning since the the distance between the edges varies according to the point at which it is measured. The figure on the left gives you more control over the shape, because it is possible to set up different distances. However, is this what you really would like? It is necessary to be careful when choosing the dimensioning scheme. Yet another thing to note is that the scheme on the left is not sufficient, it would be necessary to add at least an extra diagonal distance constraint, unless other constraints are used, Fig. 3.8. This is getting very clumsy, but the example is intended to show why it is desirable to use other constraints as well, and not just distances.

Angular constraints can be set up between two lines. Perpendicularity and parallelism are so often used that they are usually added automatically during dimensioning. This is the system trying to be helpful and guess what you want. This can often be helpful, but if you want lines that are nearly, but not quite, parallel, for example, then it may be necessary to delete them afterwards. Another help is to give you a grid of lines as a guide for shape definition. This is described in the next section.

3.2.2 Grids and Rulers

A grid is simply a pattern of vertical and horizontal lines which can be used as a framework for defining shapes. This means that the shape has known coordinates at the points. Take the example in Fig. 3.9.

The original grid is shown in Fig. 3.9a. When the use moves the cursor and selects a point, the closest point on the grid is selected, for example the point (0,40) in Fig. 3.9b. Selecting another position causes a new point, in Fig. 3.9c at (40,20), and a line to be drawn. Continuing, in drawing mode, the user selects (20, −20) in Fig. 3.9d, (−30,−20) in Fig. 3.9e and finally closes the shape in Fig. 3.9f. Using a grid means that the points are exact to the resolution of the grid. In Fig. 3.9 the resolution is 10 between grid lines, but this is usually variable and can be set according to the size of the object to be designed.

What the grid does is to give exact points for the first sketch. This can be helpful in defining the first gross shape but the point positions are not binding and will be changed according to dimension settings, if these are added. It can be helpful, as well, to ensure that a basic shape is correctly dimensioned even if the first sketch gives the right size. This is because it may be necessary to change the size or shape later. If the shape is even mildly complex then it should be dimensioned.

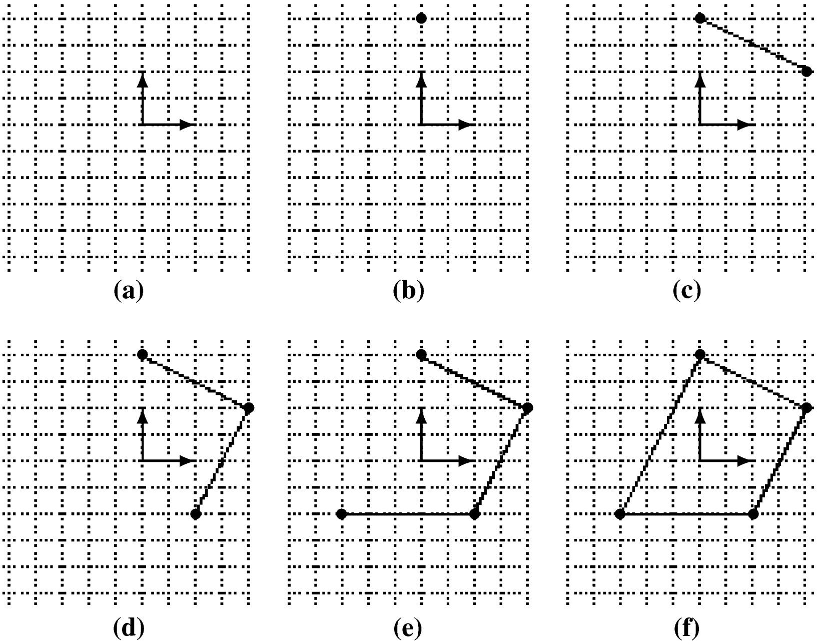


Fig. 3.9 Using a grid for shape definition

Another related topic is the use of virtual rulers as guides to defining shape. One version of this was implemented by Helldén and Andersson at KTH, Stockholm, Sweden, during the 1980s. The idea was that the measure along the virtual ruler from a fixed point is given, so that the user can draw lines of the required length. You sometimes see dynamic position information when you move the cursor about in the same way as the virtual ruler worked.

3.3 Shapes and Shape Modifications

There are some tools for modifying shape in two dimensions which can be useful to know about.

3.3.1 Corner Round-Off

This mechanism is convenient for some purposes. It takes a node between two shape elements, lines or circles, and replaces it by a circular element of specified radius, Fig. 3.10.

This operation puts a circle between the two elements, breaks the elements where they touch the circle and uses the portion of the circle between the two contact points, Fig. 3.11.

Care has to be taken that the radius given is not so large so that the intersection point on either line is not on the defined portion. Otherwise the CAD system should signal an error and refuse to do the operation.

Fig. 3.10 Rounded elements

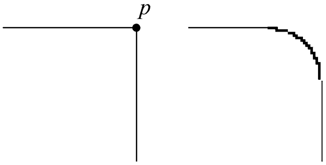


Fig. 3.11 Rounded elements

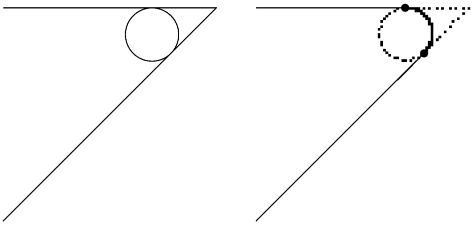
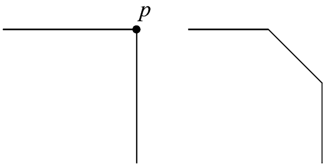


Fig. 3.12 Corner shaving



3.3.2 Corner Shaving (2D Chamfer)

This is a simple variant of corner round-off, as shown in Fig. 3.12. With both rounding-off and corner shaving it is important to distinguish between important shape elements of a two-dimensional shape and chamfers for other purposes. Even if you know that there will be a chamfer in that place, it is better to add it specifically as a chamfer. If it is created as a specific element in a two-dimensional shape and then extruded then the information that it is a chamfer is lost for later users.

3.3.3 Polygons

Polygons are simple regular shapes which may be useful. The principle regular polygons which are probably useful are the triangle, the square and the hexagon. However, any number of sides could be allowed. The parameters are: the number of sides, the radius of a circle, and a flag to say whether the circle surrounds the polygon or is surrounded by the polygon. The number of sides must be greater than 2. It is also necessary to know a start point for the calculation. Usually at least one side will be vertical or horizontal, but this is not really necessary.

Figure 3.13 shows some simple polygons. On the top row there are the polygons surrounding the sphere of given radius, on the bottom row the polygons enclosed in the circle of given radius.

Shape elements like these are easy to create as a CAD operation because the shapes are regular. The shapes could be created using basic tools, but direct operations provide a shortcut for the user.

3.3.4 Ellipses, Parabolae and Other Geometry

Ellipses are not difficult to produce, but need different control parameters. One possibility is to define a rectangular box into which the ellipse fits.

A basic quadratic Bézier curve is a parabola, and Bézier created them as a tool for designers who needed to specify only start and end points and tangents. This kind of curve is also easy to provide as a redefined shape, the problem is where to stop. Generally it should be expected that CAD systems provide a few basic shapes

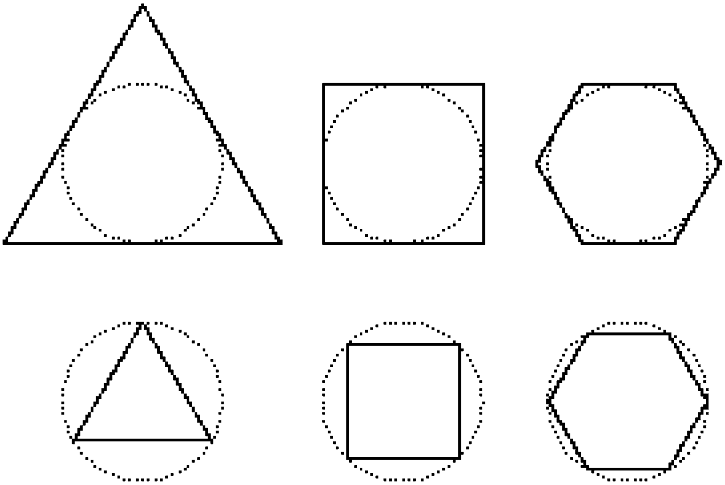


Fig. 3.13 Polygons

and then general tools for anything else. One exception may be the spiral curve. It is difficult to create correct spiral curves using general mechanisms because of the way that CAD systems perform interpolation through points. This is often done in a stepwise manner without global optimisation, leading to complex curves with uneven control points. Hence, because it is relatively common, the spiral curve may be provided as an explicit shape.

3.3.5 Patterns

Students sometimes ask how they can do patterns in two dimensions. The answer is that this mechanism doesn't exist. More correctly, I haven't seen it, but I haven't seen all CAD systems. The way that patterning works in CAD systems depends on the way that operations currently work, as will be explained in Chap. 4. Briefly, many operations create a separate solid object as an intermediate step and then unite this with the original body using a Boolean operation (ADD, SUBTRACT or INTERSECT). Defining a pattern means that the intermediate solid is copied and repeated at different positions before being combined with the object. The important thing about this is that the intermediate solid can be identified as a convenient subunit to copy and repeat. In two dimensions you need this notion, you would have to identify the subunit, probably by selecting the elements separately. These subunits would probably have to be named and put in a table so that they can be selected, copied and the copies repositioned. This is certainly technically possible and not very difficult, but at the moment I haven't seen it.

3.4 Constraint Systems

The whole set of constraints and dimensions defining a shape is termed a "constraint system". Setting up a proper system of constraints can be tedious. You will probably not believe this, but CAD systems try to be nice to you. They try and help by establishing probable constraints dynamically as you create your sketch. This is also helped if the system provides a grid structure of support geometry as a framework for drawing. The disadvantage of this is that they don't always set the constraints that you want. An example is shown in Fig. 3.14.

Here, if a shape such as that in Fig. 3.14a is drawn then the side edges risk being set to vertical. It is better to draw a caricature of a shape, such as that in Fig. 3.14b, and then setting the angle dimension separately. This is because systems use tolerances for deciding positions and orientations and, if you want a subtle shape, then slight angles or small details may be suppressed.

Another example is shown in Fig. 3.15.

This figure is based on an example used at the EPFL as a student exercise. The shape was relatively complicated, as illustrated on the left of Fig. 3.15 There were several cases where students could not define all the necessary constraints and dimensions because the system had set up an unwanted constraint between points and the diagonal line. On the right of Fig. 3.15 the dotted line shows the extension of the angled edge. One vertex is on the line, another is close to it and these were

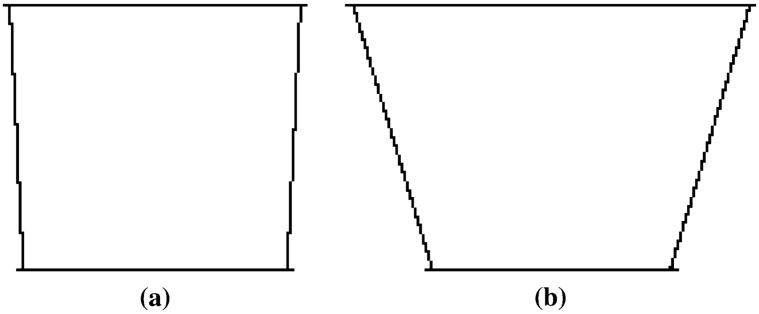
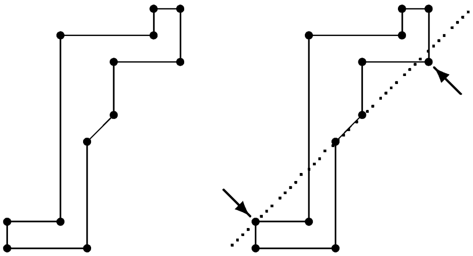


Fig. 3.14 Setting slight angles in a shape

Fig. 3.15 Unwanted constraints



defined by the system as being on the line, reducing the degrees of freedom and disrupting the desired constraint set.

There are two suggestions when setting up two dimensional shapes:

1. Use caricatures and don't try to get the exact geometry when sketching.
2. Keep shapes simple, combining several simple extrusions instead of packing everything into one.

For the first, this is to avoid problems such as those mentioned, where slight angles, or small distances are suppressed. The slight angles and small details can be controlled using dimensions and setting distances and angles after the shape has been defined. The second suggestion is to get round the other problem mentioned above. Defining a complex shape is possible if you know the result you would like, but often, a final shape is made up of many elements present for different reasons. It is often easier to define simple shapes and add in new elements later rather than cramming them into a complex two-dimensional form.

As an example, consider the linear extrusion shape from the first case study in Chap. 1. The basic shape, annotated, is shown in Fig. 3.16 Note that the labels are "l" for "line", and "p" for point to distinguish them from edges and vertices in a Boundary Representation model.

The lines in Fig. 3.16 are labelled l0–l11 in the order that they might be drawn. You could actually draw them in any order, the details of the description below will change, but not remarkably so.

When "l0" is drawn the system notes that it is approximately vertical, or really vertical if you are using a grid support, and constrains it to be vertical. Next, "l1" is drawn and there may be an automatic constraint of horizontality, or it may be set to perpendicular to "l0". After drawing the second line, examine the figure and try to work out whether the system has set constraints and, if so, how they are represented on the screen. The difference is in the effects. Setting a line to horizontal or vertical is a global constraint which means it will stay horizontal or global. A setting of perpendicularity is a relative contraint. Figure 3.17 illustrates this.

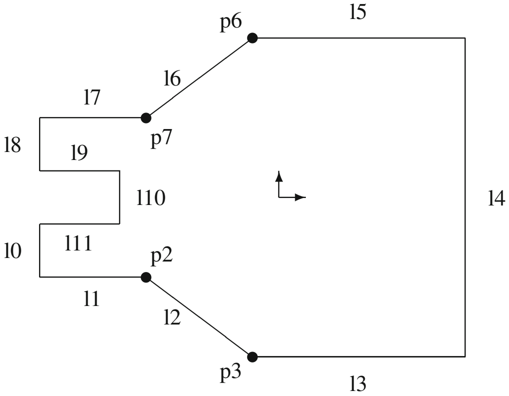


Fig. 3.16 Linear extrusion example, basic shape without dimensions and constraints

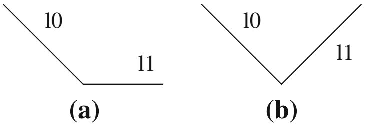


Fig. 3.17 Global versus relative constraints

In Fig. 3.17a “l1” is constrained as being horizontal. If the perpendicularity constraint on “l0” is removed and “l0” is rotated then “l1” remains horizontal. On the other hand, in Fig. 3.17b, if “l1” is constrained to be perpendicular to “l0” then rotating “l0” will change “l1” as well.

Returning to Fig. 3.16, “l2” is drawn diagonally down. There is no obvious relation to the existing elements so it is left unconstrained. “l3” may again be marked as horizontal or as parallel to “l1”. It is, perhaps, more useful to mark it as parallel to “l1”, because then you reduce the number of global constraints which is maybe more flexible. Again, check the figure as you draw it and determine which constraints have been set up by the system, global or relative. Determining this will help you understand what will happen when rotating elements in the 2D shape. Assuming that only one or two lines are globally constrained, “l4” will be set to be parallel to “l0”, “l5” will be parallel to “l1”. “l6” will not be constrained, “l7” will be parallel to “l1”. After drawing “l8” you have two alternative constraint possibilities. It may be just considered as parallel to “l0” or it may be considered to be “coincident”, or “collinear” with “l0”. Coincidence/collinearity is stronger than parallelism. If “l0” is moved then “l8” will move with it. If they are parallel then “l8” remains where it is. For this example it would be better to make “l8” collinear with “l0”. “l9” is parallel to “l1”, “l10” is parallel to “l0” and “l11” is parallel to “l1” again.

Exactly what you get will vary between systems, which is why the description above tries to stay on a general level.

Next come the dimensions. These are used to control the shape, including small details. This is why it is not important to have the exact shape that you want, even if you know it, and why you can use the caricatures mentioned earlier.

There are four small points to watch for. One is whether your system uses absolute geometry or relative geometry. The second thing is whether or not your system uses colour to indicate that elements are constrained or over-constrained. The third thing is that the dimensions controlling the shape are not necessarily the dimensions that are used to communicate the shape on a 2D drawing. The final thing is that systems do not like negative dimensions.

With absolute geometry the global position of your shape is important and it should be related to a global origin so that its position on the work-plane is well defined. With relative geometry, the absolute position of the figure in space is not important, only that the elements are positioned with respect to each other. Both methods are valid, you just have to be aware of which one your system supports.

Colouring the figure is a useful feedback mechanism which helps to check that everything has been constrained. If the figure is not fully constrained then, if a dimension is changed the two-dimensional figure may be deformed in unexpected

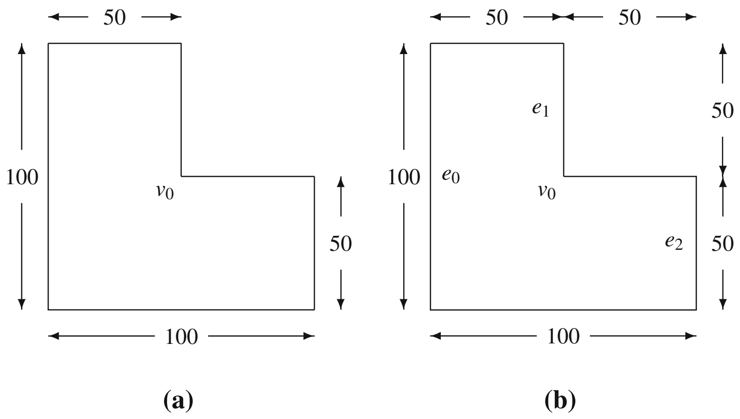


Fig. 3.18 Constrained and over-constrained figures

ways. An important aspect of this is when a figure is over-constrained. Figure 3.18 shows a correctly dimensioned figure and one which is over-constrained.

In Fig. 3.18a the figure is correctly dimensioned. If the top dimension of 50 is changed to 60, say, then there is no problem. Nor is there a problem if it is changed to 120, or the bottom dimension of 100 is changed to 200, the new geometry positions can be calculated. If, on the other hand, the dimensioning scheme in Fig. 3.18 is set up, with edges e_0, e_1 and e_2 parallel, then the dimensions are no longer independent. If one of the top two dimensions of 50 is changed to 60, say, then there is a conflict because one side of the figure is trying to be 110 units long while the other side of the figure is trying to be 100 units long. With the constraints of parallelism imposed on e_0, e_1 and e_2, this is impossible. Two visual indications of an over-constrained set, that I have seen, are to set the value of the over-constraining dimension in brackets (I-DEAS) or to draw the whole scheme in a violent colour, red or purple, say (CATIA). Both of these are acceptable and useful. With both of them you still have to work out what exactly the reason for the over-constraint. This is another reason for keeping shapes as simple as possible, as mentioned earlier, to make it easier to find the reasons for such problems.

The third thing mentioned above is that the dimensions controlling the shape are not necessarily the dimensions that are used to communicate the shape on a 2D drawing. It may help to have similar dimensions but it is not absolutely necessary. The dimensions used on engineering drawings are there for communicating something about how a shape is to be measured. They provide higher-level information as well as shape. This is also connected with the notion of tolerances which is mentioned later in this chapter. For 2D shapes, though, the dimensions are there simply for controlling shape and so can be chosen differently. This is also connected with the problem of negative dimensions, mentioned next.

CAD systems do not like negative dimensions. If you take the dimensioning scheme in Fig. 3.19a, by changing the top dimension 50–120 you can get the shape

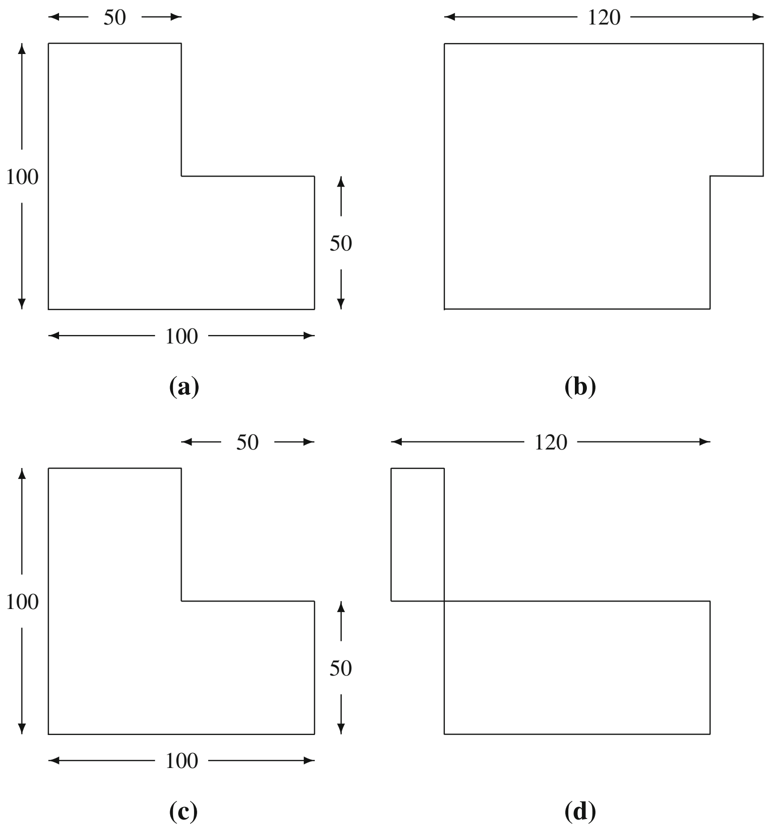


Fig. 3.19 Different dimensioning schemes

in Fig. 3.19b, but you cannot get the shape in Fig. 3.19d by attempting to change the 50 to −20. Similarly, from Fig. 3.19c you can get to Fig. 3.19d by changing the dimension 50–120, but you cannot get the shape in Fig. 3.19b. The logic is that a dimension represents a distance and a distance cannot be negative. This is a convention, though, because it is possible to interpret a negative value as a change in relation of the dimensioned elements and a positive distance. You should check whether your system allows this.

Consider the shape shown in Fig. 3.20. This is an example of a two-dimensional figure you might draw.

First of all, create a square and two circles. They do not have to be in exactly the right place or the right size (Fig. 3.21).

First of all, set up some constraints and dimensions for the circles. Circle c_0 should be set to be tangent to line l_0 and the centre of c_0 should be set to be 11 units from line l_3. The radius of c_0 should be set to be 15. Circle c_1 should be set to

Fig. 3.20 Two-dimensional shape to be made

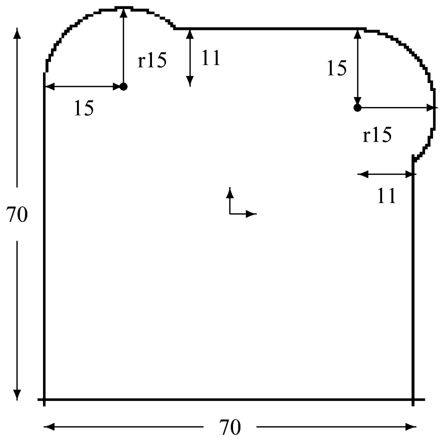
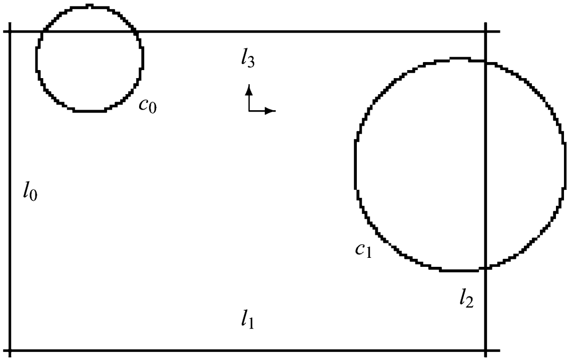


Fig. 3.21 First shape elements



be tangent to line l_3 and the centre 11 units from line l_2. The radius of c_1 should also be 15 (Fig. 3.22).

It is useful to fix the circle centres inside the rectangular shape before changing the shape of the rectangle so that the centres do not pass outside. The next stage is to set the dimensions of the rectangle. If the lines have not been constrained so that l_0 is vertical, l_2 is parallel to it, l_1 is perpendicular to l_0 and l_3 is parallel to l_1, then these constraints should be set first.

The distance between l_0 and l_2 should be set to be 70 and the distance between l_1 and l_3 also set to be 70. Personally, I prefer to set distances between lines rather than between end points of lines. For example, the length of line l_0 could be set to 70 to get the same effect, that is, by setting the distance of its endpoints to be 70. Although this would set the right size, one of the endpoints of l_0 will be deleted in a later step, and this would cause the dimension to disappear. Setting the distance between lines is more stable.

The rectangle should also be centred about the global origin. That is, line l_0 should be 35 from the origin as should l_1 (Fig. 3.23).

Fig. 3.22 Circular elements constrained

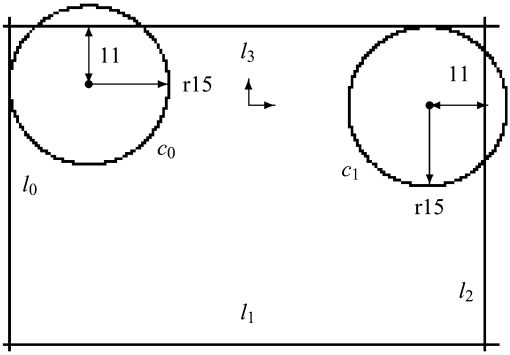
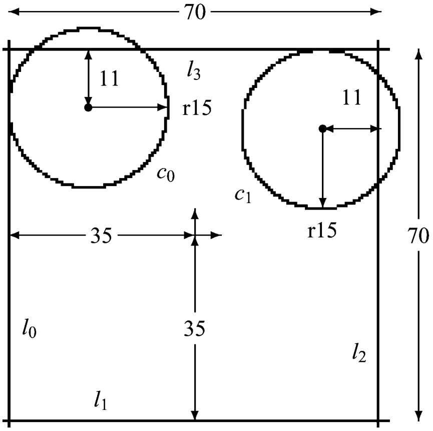


Fig. 3.23 Rectangular elements constrained



The geometric elements are now ready to "ink in" to create the shape. Some systems require you to trim the figure to remove extraneous elements. This will be described in Sect. 3.7. Other systems allow you to select elements dynamically where there is a choice of continuation. Before explaining about trimming, though, some words about how shapes and positions are changed using constraints and then something about dimensioning systems.

3.5 Constraint Solving

At the heart of the system of constraints and dimensions is the constraint solver. The constraint system solver is responsible for producing valid solutions from the set of constraints and dimensions which have been specified. If the constraint system is over-constrained it may be impossible to find a valid solution. If the constraint system is not complete then strange effects may occur.

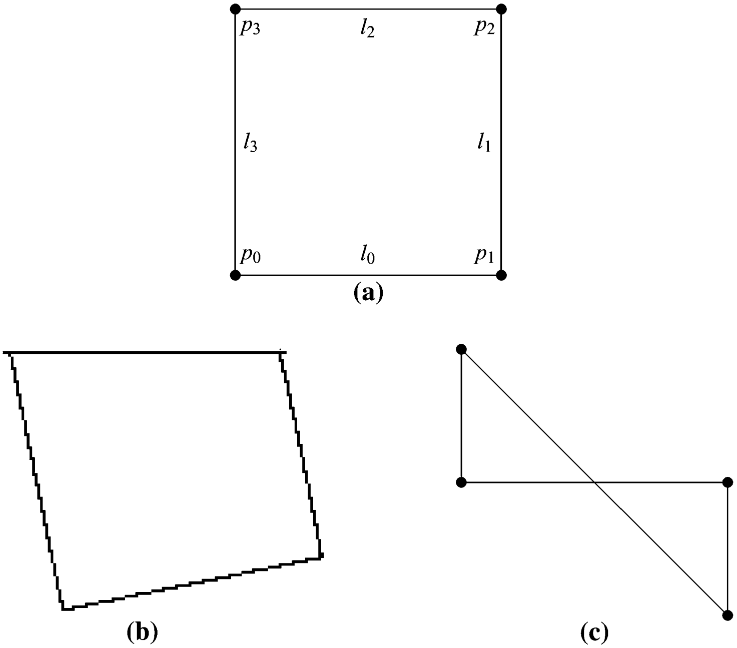


Fig. 3.24 Under-dimensioning

In Fig. 3.24 l_1 and l_3 are parallel and l_0 is perpendicular to l_1. If the line l_0 is rotated, for example, as in Fig. 3.24b, then l_2 is not constrained to be rotated so can stay where it is when the other constraints are satisfied.

The constraints stated above are also true in Fig. 3.24c. There is no way that the CAD system can tell what you want. The CAD system does not have a global view of the elements and cannot evaluate if one figure is better than another.

Another type of problem is shown in Fig. 3.25a. In this figure, l_0 is parallel to l_2 and l_1 is parallel to l_3. The distance between p_0 and p_3 is 100 and the distance between p_1 and p_2 is also 100. Although the figure is correct, it is over-dimensioned.

If the distance between p_1 and p_2 is changed to be 80 as in Fig. 3.25b then the system cannot maintain all the constraints, the distance is not respected. The desired change is ambiguous, because Fig. 3.25a and c shows another interpretation where the constraint of parallelism is not respected.

Looking at the points of the figure, each point has three degrees of freedom in this two-dimensional environment. A point can move in the X- and Y-directions and can also rotate about the Z-axis. Establishing a constraint of distance or other relation blocks different degrees of freedom. If the degree of freedom has already been blocked then you have a constraint loop and the figure is over-constrained.

The constraint system, or systems if you consider the different degrees of freedom separately, should form a tree structure. If there are elements which are not blocked, of if there are separate tree parts then the figure is under-constrained.

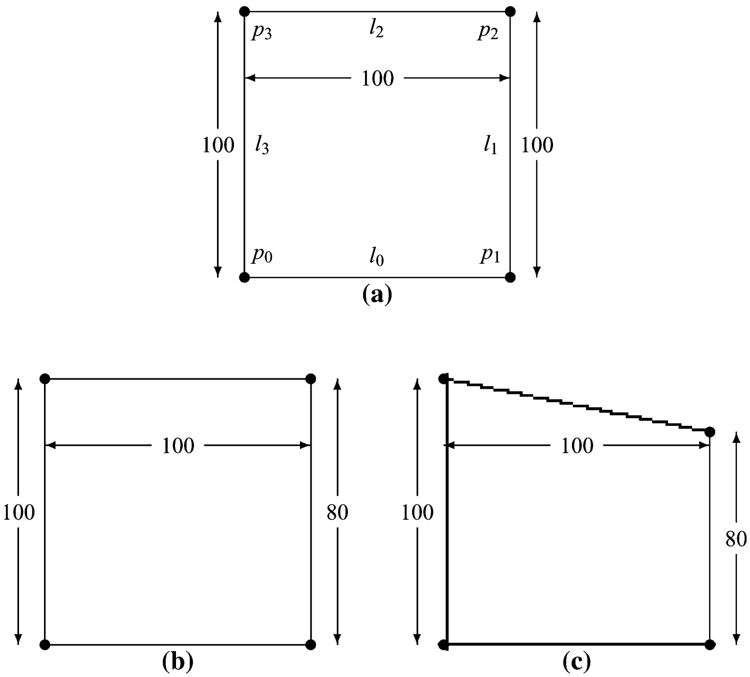


Fig. 3.25 Over-dimensioning

The root of the tree is an element which is “grounded”, that is, is fixed in global space. Without such an element the figure can move.

3.6 Tolerance Analysis

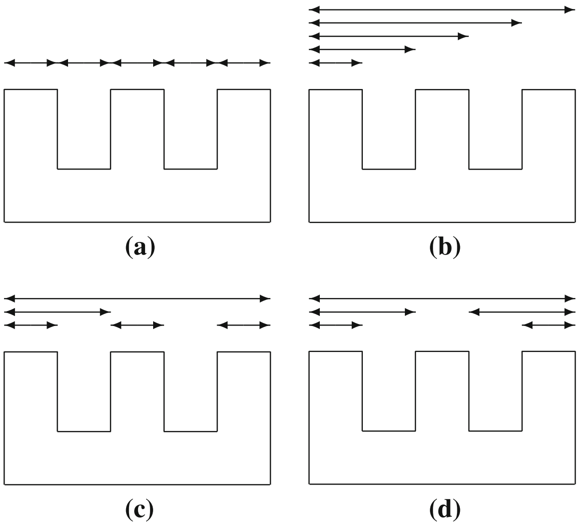
Tolerance analysis is a method to analyse the validity of a tolerance scheme. Figure 3.26 (not my figure) is something of a classical example used to explain the difference between tolerancing schemes.

If you add a control dimension and tolerance to these systems, to create an over-dimensioned scheme then it is possible to check the probability that the checked dimension will be within the tolerance limits. This is the way that I-DEAS analysed tolerances. Different tolerancing schemes with different dimension loops give higher or lower probabilities. The subject, though, is complex and not dealt with here. If your CAD system allows this, then try it. It may only work in two dimensions, though.

One question, though, is: “why add tolerance information?”.

Tolerance information is added to drawings as a way of communicating critical measurements between a designer and a manufacturer, say. Setting tolerances is a notorious source of errors, so that fine tolerances may be set on object parts where

Fig. 3.26 Dimensioning schemes



they are not needed, raising the cost of manufacture. In the past drawings were the means of communication between the designer and the manufacturer. However, nowadays it is possible to communicate not only the 3D part but also the assembly in which it is used. This gives more information for the manufacturer to choose manufacturing methods and set appropriate tolerances. At the moment, though, more work is needed to establish an improved flow of product information throughout the production chain.

The dimensioning scheme that you set up to control a shape is not necessarily the same as that you would communicate to someone else. However, look at the different options and consider which you think is important and closest to how you want to control the shape. This can be useful if you try and parametrise a part, as described in Chap. 12.

3.7 From 2D to Solid

The 2D environment uses, or may use, separate, unlinked or partially linked elements. Before being able to use such a model in three-dimensional work, for extruding for example, the shape has to be converted into a proper model, an "Eulerian" model (see Sect. 2.7.3). This is the "inking-in" procedure described in Sect. 3.1.

The normal requirement is that the elements to be made into a shape are linked with at most two lines at each node of the shape graph. If the system does not like mixing solids and partial models (see Sect. 6.3) then there may also be a requirement that there are exactly two lines at each point.

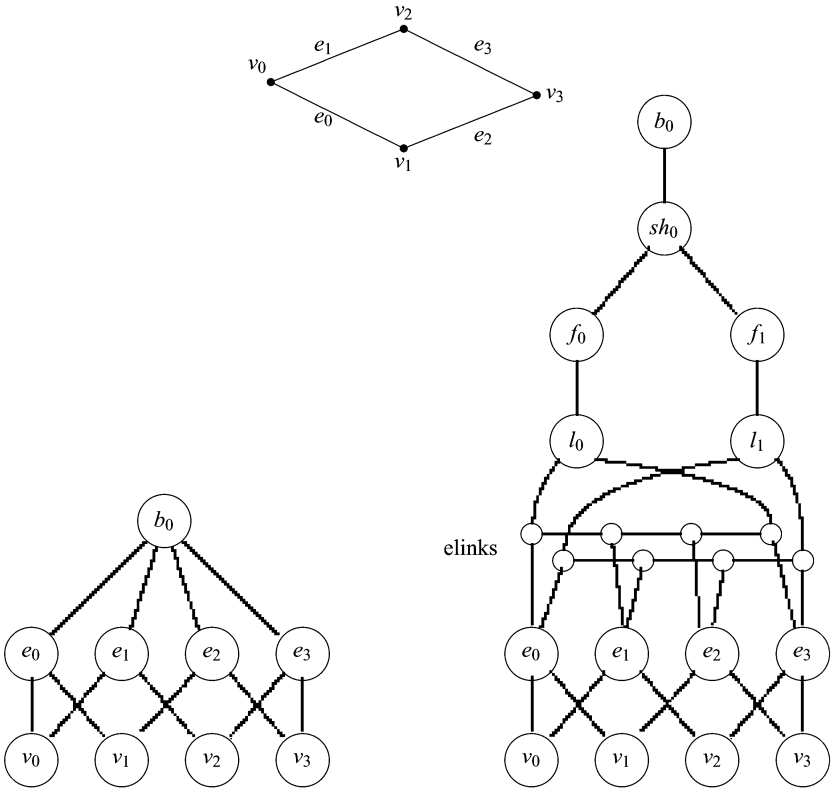


Fig. 3.27 Linking edges into loops

A simple example of what is going on is shown in Fig. 3.27. The lines and points are converted to edges and vertices, as at the top of Fig. 3.27. The structure is a simple wireframe structure, with just a body, edges and vertices (and their geometries) as shown at the bottom left of the figure. The structure needed after the conversion, with faces, loops and elinks added is shown on the bottom right of the figure.

The conversion phase involves adding elinks to all the edges, organising them into loops which are in turn associated with faces. The faces are part of the same new shell. Each edge has two elinks added, one is on the "left" side, the other on the "right" side. Starting with edge e_0, it runs from v_0 to v_1, say. The "counter-clockwise" edge on the left side is set to be one of the links of e_2. If e_2 runs from v_1 to v_3 then the link is the "left" link of e_2 otherwise it will be the right link. The "clockwise" edge from e_0 is set to be e_1 in the same way. For the "right" link, the counter-clockwise link is set to be one of the links of e_1, the left link if e_1 runs from v_0 to v_2, otherwise the right link.

Once all links are set up you have two loops which, read in counter-clockwise order, have the edges: e_0, e_2, e_3 and e_1; and e_0, e_1, e_3, e_2. This is easy if there are

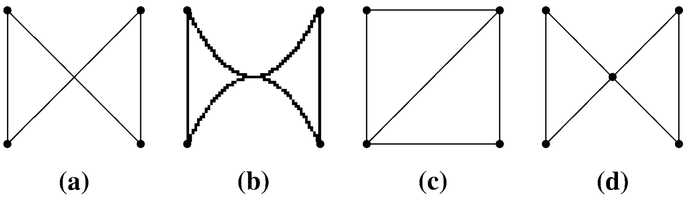
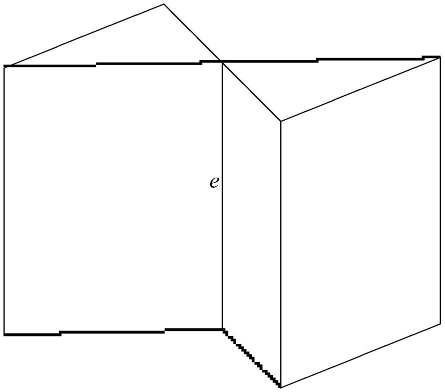


Fig. 3.28 Error cases and an ambiguous case for 2D sketches

Fig. 3.29 Extruded triangles



only two edges at each vertex, otherwise it is more complicated but still manageable. Most systems insist on having two edges at a vertex.

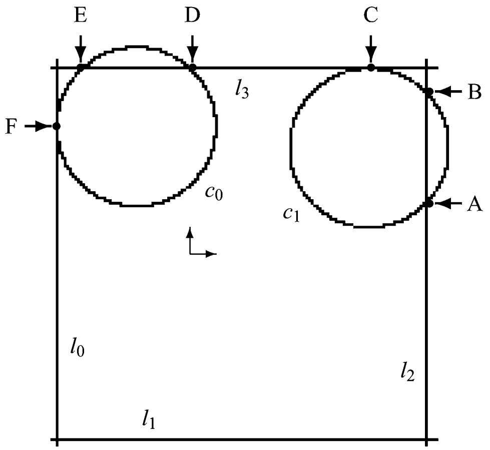
There is also a geometric check, edges are now allowed to touch each other except at vertices. Some error cases are shown in Fig. 3.28.

In Fig. 3.28a the two diagonal edges cut each other where there is no vertex. In Fig. 3.28b the curved edges do not cut each other, but they touch at a point where there is no vertex. In Fig. 3.28c two of the vertices have three edges, not two. The fourth example, shown in Fig. 3.28d is ambiguous. It might be two triangles with a common vertex, or it might be one body with what is termed a “non-manifold” vertex (see Chap. 6) Try this and determine which your CAD system produces. The result is shown in Fig. 3.29.

A simple test is to try and round (or fillet, or blend) the edge marked “*e*” in the Figure. If this is a non-manifold edge then CAD systems usually refuse. This will be explained further in Sect. 4.8.

Continuing the exercise above, in Sect. 3.4, with a rectangle and two circles, the final step of is to trim the shapes so that only the required geometric parts are present. This is not obligatory. I-DEAS has, or had, a user-friendly way of doing this at the stage of converting the 2D sketch into the 2D model for extrusion. Other systems, though, oblige you to trim off extraneous shape elements so that there is a clear form before doing the conversion.

Fig. 3.30 Extraneous element trimming



The extraneous elements can be identified as crossing points, where different shape elements intersect. In Fig. 3.30 the crossing points are marked A, B, C, D, E and F.

At point A, l_2 intersects c_1 and it is necessary to cut both. Note, though, that there are two intersections between the line and circle, so the system needs a little help to determine which one you mean. This is probably done by choosing the intersection point closest to the curve selection point. The other thing to note is which curve the CAD system chooses to cut, one or both. CATIA cuts the first curve selected. Test this on your system. If you cut the curve and then select it the curve is usually drawn in a different colour and lets you see what has been done.

So, cut c_1 by selecting the operation, then select c_1, click on the curve close to point A, and then click on line l_2, again close to point A just to be sure. Now repeat, clicking first on line l_2 close to A and then on c_1 close to A. Delete, or mark as a support geometry if you can, the upper portion of l_2.

Point B can be ignored, there is no need to cut anything there.

Now cut c_1 at point C by intersecting it with line l_3. Since l_3 is tangent to c_1 there is only one intersection point. Cut l_3, too, at point C. Now delete the right-hand part of l_3 and the left/bottom part of c_1 (Fig. 3.31).

A similar process is needed to cope with the other crossing points on the left hand side of the figure. Circle c_0 and line l_3 are broken at point D. Line l_0 is broken at point F and the appropriate parts deleted or marked as support geometry, giving Fig. 3.32.

As far as CAD systems go, this is about the end of the story. However, it should be mentioned that the restriction on having a maximum of two edges at every vertex is artificial. Work by Müller [8] shows that it is relatively easy to relax this rule and create valid shapes for extrusion from general graphs, assuming that they don't contain identical lines, circles and so on.

Take Fig. 3.33 as an example of a not tremendously complex figure.

Fig. 3.31 Extraneous element trimming

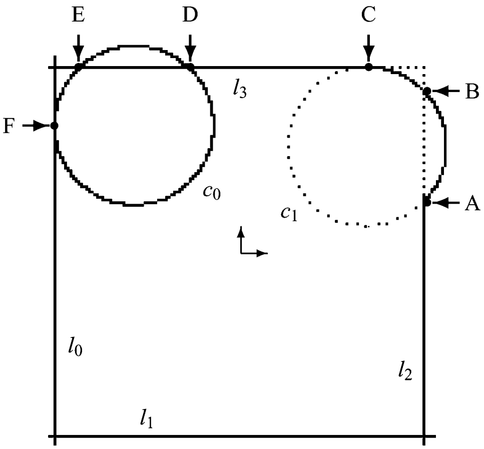
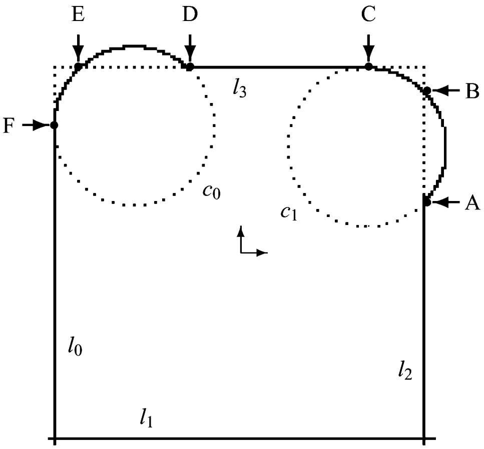


Fig. 3.32 More extraneous element trimming



Perhaps the main reason why these sort of figures aren't converted is because it is necessary to decide what to do at vertices v_0 and v_2 to have a correct figure. The trick is to notice that, when finding the next edge round the loop from a given edge, the next counter-clockwise around a loop is the edge clockwise around the appropriate end vertex from the edge with reference to the plane normal. This means that, looking at Fig. 3.33, from edge e_0, the next edge round the loop is e_1, the edge clockwise around v_1. After that, the next edge is e_4, the edge clockwise around v_2 from e_1. The next edge would be e_0 so the loop is closed. Similarly, starting from e_2 you would get the loop of edges: e_2, e_3, e_4. Computing the underside you use the normal in the opposite direction and get the loop: e_0, e_1, e_2, e_3.

This is not a complete solution, though. Look at the shape on the Fig. 3.34.

If you do the same thing then you would get a loop: e_0, e_5, e_3, e_4, e_2, e_5, e_1, e_4 because of the crossing edges. It is, therefore, necessary to perform a preprocessing step and cut all geometric elements at intersections and insert new vertices.

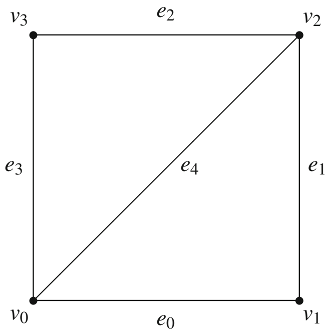


Fig. 3.33 Slightly complex 2D sketch

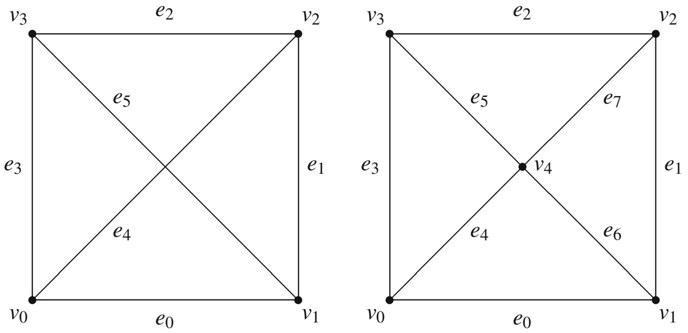


Fig. 3.34 Slightly more complex 2D sketch

This would result in the shape on the right of Fig. 3.34. Processing this figure would give you the four small triangular faces on the top and one four-sided face on the bottom.

Even with a correct figure, though, the simplified extrusion algorithms used in current CAD systems will not cope with sweeping multiple adjacent faces. In order to do this a commercial system would have to sweep each face separately to create a set of basic objects which would then be combined using a Boolean add operation.

3.8 Inscribing on Faces

It is usual in CAD that you draw profiles on faces and then extrude these profiles. The process of drawing the profile on a face is known as “inscribing”. There is a question, though, about what happens when the profile you draw on a face surrounds other profiles in the face.

Consider the object shown on the left of Fig. 3.35. This is a simple block with a toroidal shaped hole, shown in cross-section on the right of the figure.

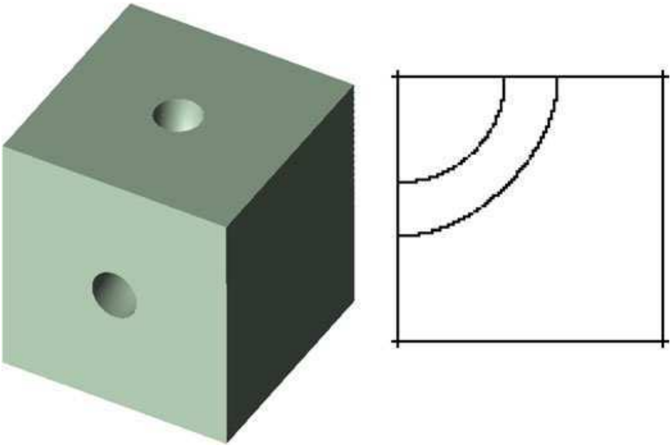
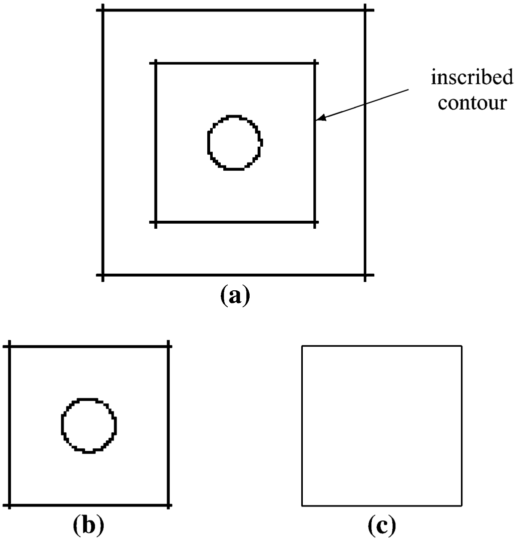


Fig. 3.35 Basic object for face inscribing

Fig. 3.36 Inscribed profile on face



(a)

(b)

(c)

What happens if you inscribe a profile on the top face of that object? Consider Fig. 3.36. In Fig. 3.36a you have the top face with the inscribed profile surrounding the hole. Should the resulting profile include the hole, as in Fig. 3.36b? Or should it just be the profile drawn, as in Fig. 3.36c?

If the system now extrudes the profile, if the hole is included then you get the result shown on the left of Fig. 3.37, if it doesn't then you get a result shown on the right of Fig. 3.37.

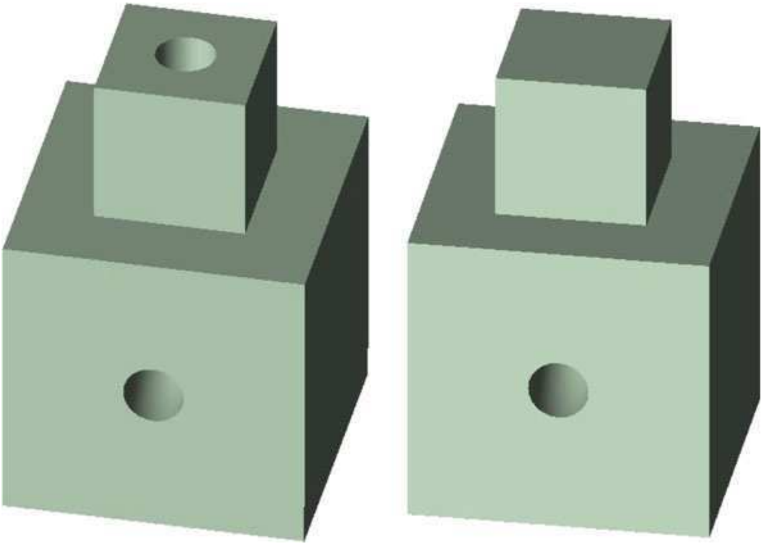


Fig. 3.37 Inscribing test object

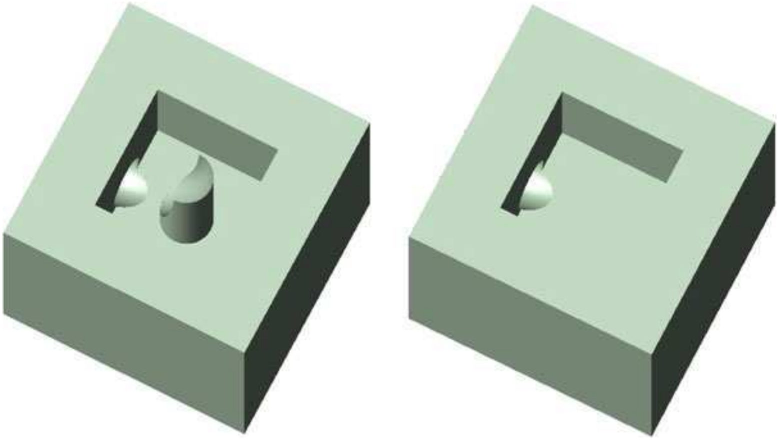


Fig. 3.38 Inscribing test object

When extruding downwards you get some other weird results, depending on the interpretation, as shown in Fig. 3.38.

You get another set of examples if the surrounded contour is the base of an extrusion outwards, as in Fig. 3.39.

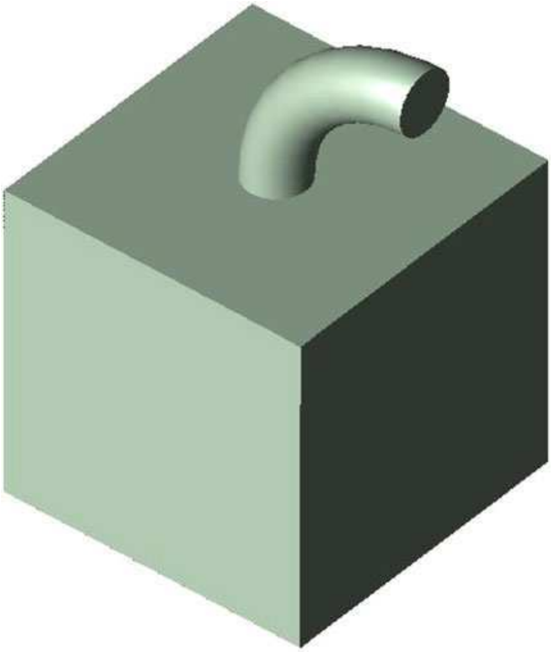


Fig. 3.39 Second inscribing test object

If the inscribing contour is extruded downwards you can get two results, on the left if the inscribed contour is not included and on the right, if it is.

If the inscribing contour is extruded upwards then there are, again, two alternatives, on the left is the case when the inscribed contour is not included and on the right when it is.

In all cases, the inclusion of the surrounded contours, or only the profiles explicitly drawn in the sketch, can be argued. CAD systems seem to use only the profiles drawn explicitly as being part of a shape definition. In the above examples, it seems logical to me to have the solution on the left of Fig. 3.37, the solution on the right of Fig. 3.38, the solution on the right of Fig. 3.40 and on the left of Fig. 3.41, but then I am an awkward person. Perhaps not. There is an algorithmic way of choosing these solutions. Put simply, when extruding a contour downwards, i.e. removing material, then surrounded contours with concave edges should be included while surrounded contours with convex edges should not. When extruding upwards, i.e. adding material, surrounded contours with convex edges should be included while surrounded contours with concave edges should not. Contours with a mixture of convex and concave edges can be divided into convex or convex sequences and treated accordingly.

It would, perhaps, be useful at the very least to ask the user what was meant, to include the internal shapes or to leave them out, rather than just imposing one or other solution.

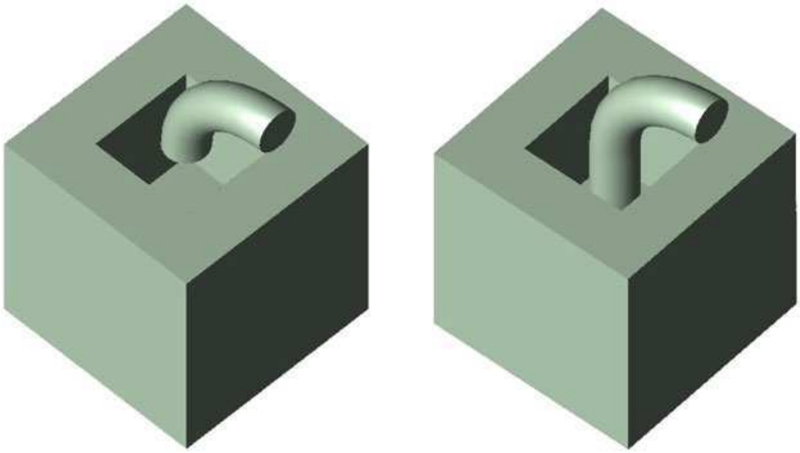


Fig. 3.40 Undercutting an extrusion

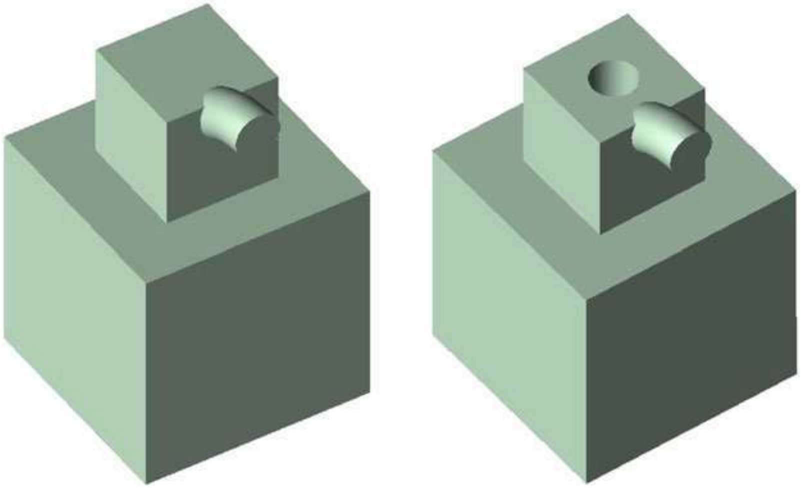


Fig. 3.41 Extruding outwards round an extrusion

3.9 Chapter Summary

This chapter describes how two-dimensional shapes are created and manipulated. The chapter explains the origins of this technique, how shapes were created with early non-interactive systems and how they are created in today's CAD systems. The chapter describes constraints and how they are used. The chapter also describes briefly constraint solving and tolerance analysis. The chapter describes how sketches are converted for use in solid modelling.

3.10 2D Exercises

3.10.1 Constraint Identification

Make a square and identify the constraints.

Make a rectangle centred about the origin. If there are horizontal and vertical constraints then delete them. Delete constraints of parallelism and perpendicularity. Now select the corner marked "v" in the shape on the left of Fig. 3.42 and drag it around. You should see something like the shape on the right of the figure. Do the same with other corners to show that they can be moved. Now re-establish some constraints. Establish a constraint of parallelism between one pair of opposing edges. Do the same with the other opposing pair. Finally establish a constraint of perpendicularity between two adjacent edges. The rectangle is re-established, but may not be oriented as before. It is possible to reorient it by setting one of the edges vertical or horizontal.

If your CAD system uses global positioning and has a set of axes, then establish a dimension between the x-axis and the bottom edge of the rectangle. Establish another dimension between the left hand side of the rectangle and the y-axis. Note whether the two edges change colour to show that they are fully constrained. Grab and move the top right-hand corner. Finally add dimensions to the bottom edge and the left hand edge and note whether or not the other edges change colour.

Note before starting the second part that there are different ways to dimension objects. A dimension between two vertices implies a simple distance while a dimension between two edges implies a distance and a constraint of parallelism. If the edges are not parallel then there is an angular dimension between them.

Create a new rectangle and delete the horizontal and vertical constraints. Establish dimensions between each pair of corner points (not the edges) and one pair of opposing points, giving you the diagram shown in Fig. 3.43.

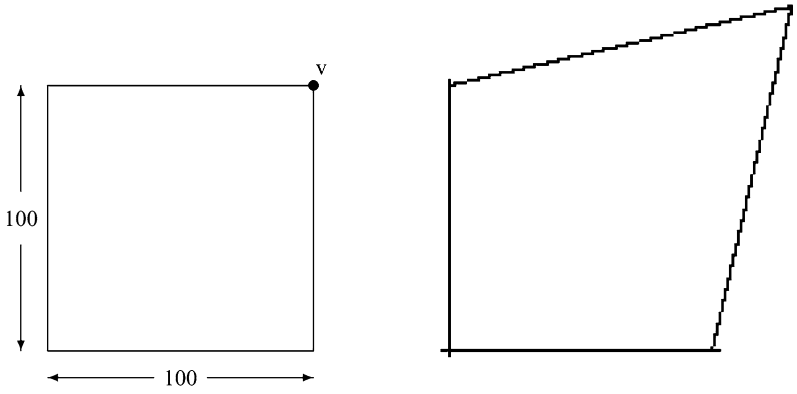


Fig. 3.42 Square and deformed figure

Fig. 3.43 Square and deformed figure

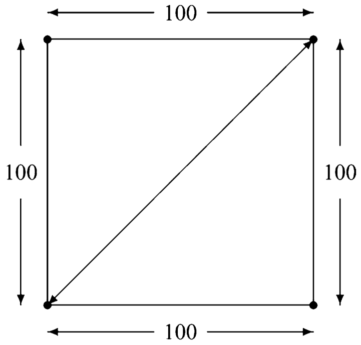
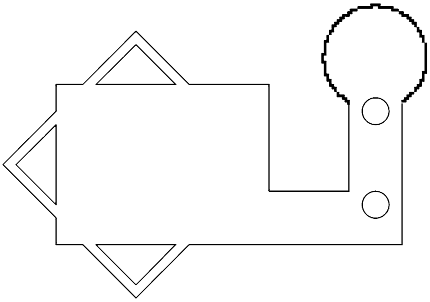


Fig. 3.44 Complex figure to decompose



The exact values are not important, but are shown as 100 here. Change the different values to see how the figure transforms.

3.10.2 *Shape Decomposition*

This exercise is to decompose the shape in Fig. 3.44 into simpler component shapes. Determine the order of defining them. This would correspond to making a simple shape, extruding it, drawing a second simple shape, extruding it, and so on to get a three dimensional figure with the shape defined in the figure. The shape would be created in steps instead of trying to define the complete contour in one step.

3.10.3 *Over-Constraint Analysis*

Make the shape shown in Fig. 3.45 and check what happens when the over-constraining dimension is added.

Fig. 3.45 Over-constrained figure

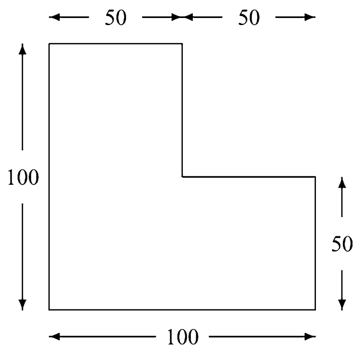
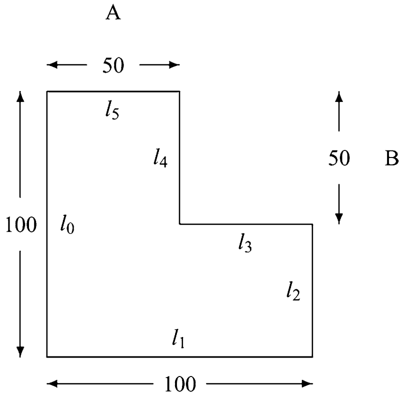


Fig. 3.46 Experimental figure



3.10.4 Dimension Games

If you did not try the experiments with dimensions earlier in the chapter, try them now. Make the shape shown in Fig. 3.46 with the dimensions as shown. Line l_0 should be set to be vertical. Lines l_2 and l_4 are set to be parallel to l_0. Line l_1 is set to be perpendicular to l_0. Lines l_3 and l_5 are set to be parallel to l_1. Assign the dimensions as shown in the figure.

Change the dimension 50 (marked "A" in Fig. 3.46 to 120. Change it back to 50. Try and change it to -20 and check if the system allows this and, if it does, what happens. Change the dimension back to get the original figure. Now do the same with the other dimension of 50, marked "B" in Fig. 3.46, Finally, change dimension A to be 100 and change it back. Is the figure the same? When the dimension is changed e_3 becomes zero length and e_2 and e_4 become collinear. Some systems may remove e_3 and merges edges e_2 and e_4. Really this should be done when converting the figure before extruding it. You should check whether your system performs the merge or not.

References

1. Braid, I.C.: Designing with volumes. Ph.D. Dissertation, Cantab Press, Cambridge Computer Laboratory, University of Cambridge, Cambridge (1974)
2. Hillyard, R.C.: Dimensioning and tolerancing in shape design. Ph.D. Dissertation, University of Cambridge Computer Laboratory, Technical Report No. 8, June (1978)
3. Braid, I.C., Hillyard, R.C.: Characterizing non-ideal shapes in terms of dimensions and tolerances. CAD J. **10**(3), 234–238 (1978)
4. Gossard, D.C., Lin, V.: Representation of part families through variational geometry, from advances in CAD/CAM. In: Proceedings of the 5th International IFIP/IFAC Conference on Programming Research and Operations Logistics in Advanced Manufacturing Technology PROLAMAT 82, pp. 47–53. North-Holland Publishing Company, Amsterdam (1982)
5. Björke, O.: Computer Aided Tolerancing. Tapir Publishers, Norway (1978)
6. Björke, O.: Computer Aided Tolerancing. ASME, New York (1989)
7. Johnson, R. H.: Dimensioning and tolerancing final report. Report R-84-GM-02.2, Computer Aided Manufacturing International (1985)
8. Müller, M., Stroud, I., Xirouchakis, P.: Preprocessing of degenerate slices in layered manufacturing. In: Topping, B.H.V. (ed.) Developments in Engineering Computational Technology, pp. 63–68. Civil-Comp Press, Scotland, ISBN 0-948749-70-9 (2000)

Chapter 4
Operations and Functionality

This chapter is about the main operations that are used in CAD systems to make objects. For each operation there is a brief description of the algorithm, a discussion of special cases and possible errors as well as experiments to do. A more detailed description of solid modelling algorithms are given in a separate book, [1] and are not repeated here. Hopefully the explanations here are enough to understand what is happening without repeating everything.

It has become usual in CAD systems to implement some of the operations so that they create a simple shape and then combine this with the model being created using Boolean operations. The reasons for this are partly historical. It wasn't always so, but this methodology means that models are more likely to be correct topologically and geometrically. Examples of these operations are the extrusion operations and the symmetry operation. An example of where this is difficult is, for example, the draft operation. An example of an operation which might or might not be implemented this way is the chamfer operation.

The early modelling operations were divided into two types:

1. the Boolean operations;
2. everything else, also called "Local operations".

The Boolean operations performed a complete check and worked very well, but were, in the original research system, a little slow and slowed down further as object complexity increased. This was later improved in two ways, firstly computers became larger and faster, but more importantly work was done on improving efficiency by using bounding geometry for fast checking. Now, Boolean operations are fast and can be used to add or subtract volumetric elements quickly. In contrast, the local operations were diverse, performing a number of specialised functions. The worked locally, hence the name, and so could create objects which cut into themselves, so called "self-intersecting objects". These are objects where one part of the boundary cuts or touches another part geometrically without being connected topologically.

I. Stroud and H. Nagy, *Solid Modelling and CAD Systems*,
DOI: 10.1007/978-0-85729-259-9_4, © Springer-Verlag London Limited 2011

Many of the Boundary Representation operations you find in CAD systems today have a long history, some more than 30 years old. To give an idea here are some operations and their origins.

- Boolean operations: Simplified Booleans in Braid's dissertation 1974 [2], general Booleans in BUILD 1976–1985.
- Linear extrusion: Braid's dissertation 1974 [2], generalised 1976–1980.
- Circular extrusion: Braid's dissertation [2], revised to allow partial circular extrusion in BUILD 1976–1980.
- Chamfering: BUILD 1976–1980.
- Implicit blending: BUILD 1976–1980.
- Draft angles: BUILD 1976–1980.
- Symmetry, or reflection: BUILD 1976–1980.
- Extruding along a path: BUILD 1976–1980.
- Feature operations (Boss, pocket, for example). BUILD 1980–1985.
- Shelling: GPM 1980–1985.
- Thickening a sheet object to a volume: GPM 1980–1985.
- Setting a wire object into a surface: GPM 1980–1985.

Further work has been done on these, notably in the commercial modelling kernels Parasolid and ACIS, at least. However, the ideas that generated these operations are relatively mature.

4.1 Boolean Operations

The Boolean operations are usually present somewhere explicitly, but, as stated above, they are used in the implementation of several other operations. For this reason, and for their historical importance, they are presented first. The Boolean operations, also known as "Set operations", or "Boolean set operations", combine two objects in one of three main ways. There are, in fact, other related techniques, but the three principle combinations are known as: ADD (or UNION), SUBTRACT (or DIFFERENCE) and INTERSECTION. These are illustrated in two dimensions in Fig. 4.1.

4.1.1 Parameters

Input:

Two objects

Output:

The result is combined with the original object, possibly as a collection of objects.

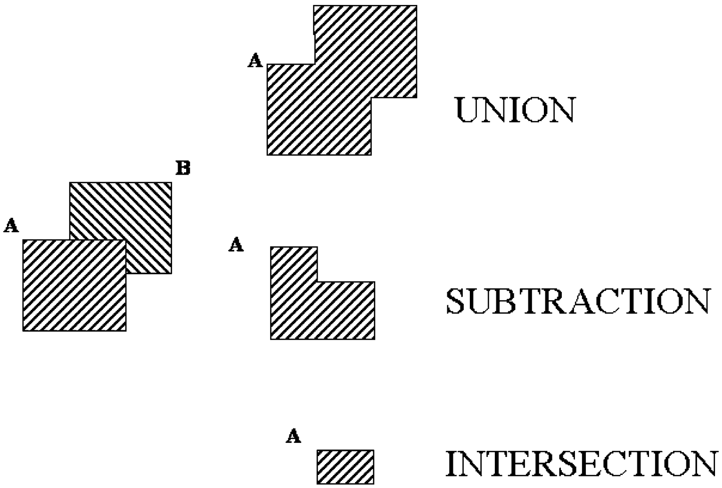


Fig. 4.1 Two dimensional Boolean operations

4.1.2 Potential Errors

The Boolean operations are fairly robust, general operations with simple parameters. As internal functions they should provide return codes so that, at the CAD level, the implementer can choose whether or not to print a message. Some optional warnings might be:

- One object inside another—if adding this may make no change or return the added object. For subtraction this might remove the whole object or create a cavity. For intersection this is OK.
- Objects do not touch—for addition this would create a multi-piece object. For subtraction this would mean no change. For intersection this is OK.
- Objects touch along an edge or vertex—only for addition, this means that a non-manifold element has been created.

4.1.3 The Boolean Algorithm

The Boolean algorithm is roughly:

1. Compute the interactions between two objects.
2. Separate the object boundaries at the interaction edges.
3. Recombine the objects by joining elements from the two boundaries.
4. Separate bodies into shells and clean-up.

To calculate the interactions, all faces in the first body are intersected with all faces in the second body. The method of calculating the intersection of a face pair is shown in Fig. 4.2, from [1].

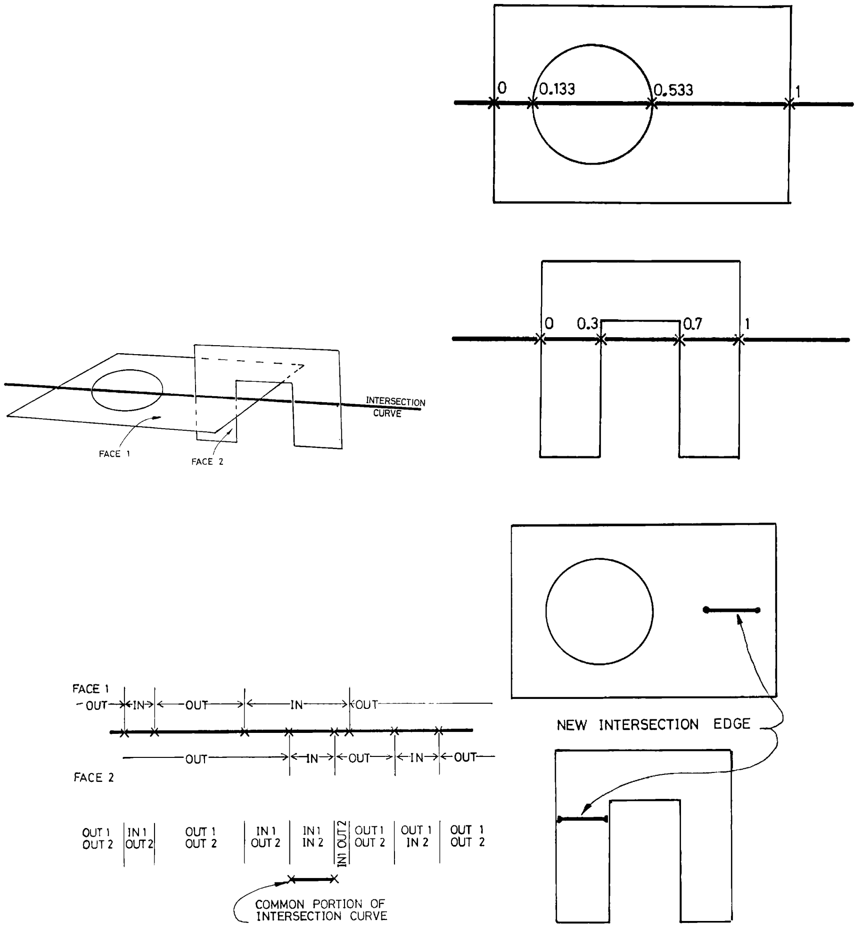


Fig. 4.2 Boolean operations—face–face comparison

This is shown in Fig. 4.3.

The subdivision method comes from Braid [2, 3]. The recombination described above is due to Mäntylä [4].

4.1.4 Special Cases

There are fewer special cases for Boolean operations than for other operations because they are general operations which work for a large range of inputs. However some can be identified so that it is possible to experiment with Boolean operations and learn about their limitations.

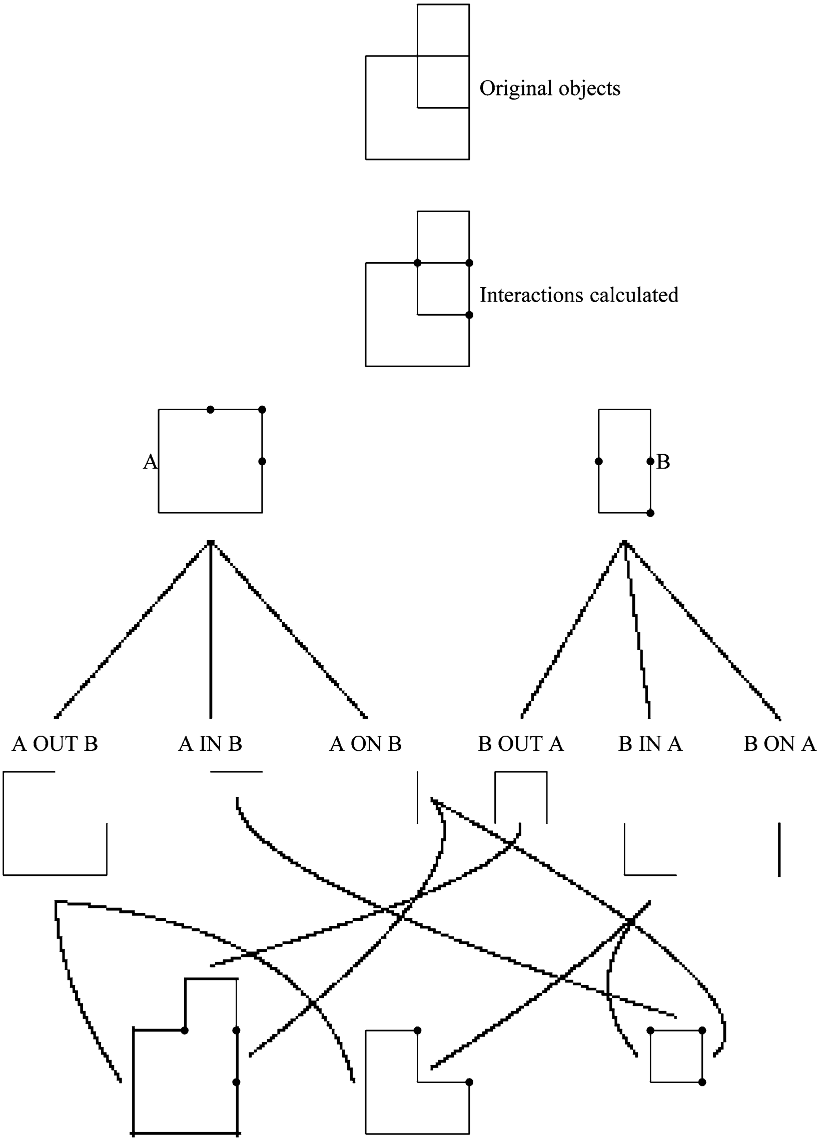


Fig. 4.3 Decomposition and recomposition of Boolean results (based on a figure from [1])

The special cases involve:

- One body contained in the other.
- One body separate from the other.
- Two bodies touching along a face set.
- Two bodies touching along an edge.
- Two bodies touching at a vertex.

4.1.5 Sectioning

This is a special type of Boolean operation which takes an object and a cutter surface or surfaces definition and returns two objects. Note, straightaway, that this goes a little against the single model per workspace philosophy, but this is a detail. This operation frequently exists as part of the engineering drawing production function but seems less common as a modelling tool.

It is possible to distinguish two types of sectioning:

1. Single surface sectioning.
2. Multi-surface sectioning.

The difference between the two comes in the calculation of the Boolean, or intersection boundary. For a single surface the surface is taken as infinite so the intersections inside the faces of the object being sectioned are taken as the boundary. The multi-sectioning surface is defined as a sort of sheet object (see Chap. 6) and so it is necessary to check the intersection curves against the topology of this object, as with Boolean operations. Note, though, that the sectioning objects are infinitely thin, so the operation is not the same as subtracting a very thin solid object, which will always leave a gap. Note that there is a problem if the surface used for sectioning does not traverse the object being sectioned, or if the multi-section surface ends inside the object.

Two last quick comments. The operation could optionally return both objects or only one. Perhaps the most useful variant is planar sectioning.

4.1.6 Experiments to Try

These experiments may be combined with extrusion, since extruding a shape drawn on a basic shape will have the same effect. See the section on extrusion.

4.1.6.1 One Body Contained in the Other

The arrangement is shown in Fig. 4.4.

You have three cases, according to the three Boolean operations: add, subtract or intersect. The object to be operated on is called “A” and the object to be added, subtracted or to intersect is “B”.

If A is larger than B the results are:

- For ADD the result is A.
- For SUBTRACT there will be a B-shaped cavity in A.
- For INTERSECT the result is B.

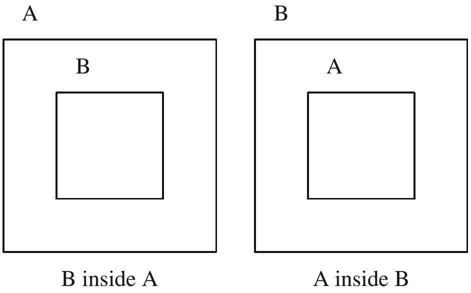


Fig. 4.4 Arrangement of overlapping objects

If B is larger than A the results are:

- For ADD the result is B.
- For SUBTRACT there will be nothing (A disappears).
- For INTERSECT the result is A.

The main difference in the result is what happens for the case of subtraction. If A is larger than B then there will be a cavity inside A. If A is smaller than B then the result is nothing. What is more interesting is what the system tells you about when it gives the result. For the add results, if B is contained in A the user should be warned that there is no change in A. If B is larger than A the system should warn the user that A has been swallowed up by B. For the subtract results, if A is larger than B then the result is correct but it would be useful to indicate to the user that there is now a cavity. A warning may be optional, but if the result is shown in shaded mode then no change will be visible and so a warning might help. If B is larger than A then the user should be warned that A has disappeared. Both the intersection results are the same, and correct. Intersection is used, among other things, for testing for interactions between objects in an assembly, so the result is useful. Testing these simple cases will tell you what to expect when bodies are combined with Boolean operations in other operations. If the system does not warn you about these special cases then you should check the results of operations and use wireframe images as well to check for containment.

A variant for subtract is if the bodies being subtracted are identical. The normal result should be nothing. A possible result is to create a sheet object with zero thickness. However, normally you should get nothing as a result.

4.1.6.2 One Body Separate from the Other

The arrangement is shown in Fig. 4.5.

For SUBTRACT the result should be A. For INTERSECT there should be no result, and perhaps a warning that there is no result. The main interest is what happens with ADD. What normally happens is that the separate objects are treated as one, at the datastructure level. This is not really correct. The justification is that the user may change the parameters and create a single object. Alternatively, the

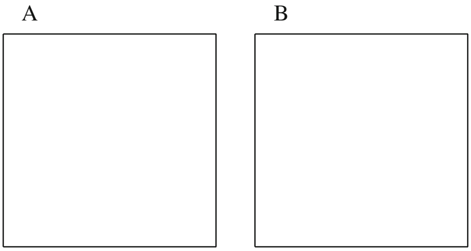


Fig. 4.5 Arrangement of objects with separation

user may join the separate parts in a subsequent operation. Why it is stated, above, that the result is not really correct is because the object is made of two distinct parts. At the very least, the result should be a compound object, not a simple object, but this would introduce a new element into the datastructure. At the moment there may not even be a warning for the user that the objects are separate which makes it difficult for the user to know.

On the datastructure level the current mainstream solution is to have a single object with two “shells” containing the separate object portions. It is possible to see the difference in the datastructure, but not necessarily at the user level, where small separations might not be noticed. This is done both in systems which insist on working on single objects as well as those systems which allow multiple objects in modelling space.

4.1.6.3 Two Bodies Touching Along a Face Set

This is more or less a normal case and should be handled easily. It is mentioned here only for completeness.

4.1.6.4 Two Bodies Touching Along an Edge

This is shown in Fig. 4.6.

SUBTRACT should make no change to object A and INTERSECT should give no result. The interesting question is what happens when ADDing the objects.

The “normal” result is to join the edges at the edge marked e_A, e_B and the common edge becomes a non-manifold edge. Non-manifold objects and edges will be described later in Chap. 6. The interesting thing is whether the system warns you, because these edges can cause problems for other operations. If it doesn’t then you risk having an edge which cannot be dealt with by other operations. The problem is that, if these are created, then they should be fully integrated into the CAD system and handled. Normally they are not. Another question is whether the CAD system should actually create the object. Such objects are not realisable, because you cannot make an object with zero thickness. They may be useful as intermediate objects, idealisations, as explained in Chap. 6. If this is the case, though, then there should be operations to expand them or adapt the edges,

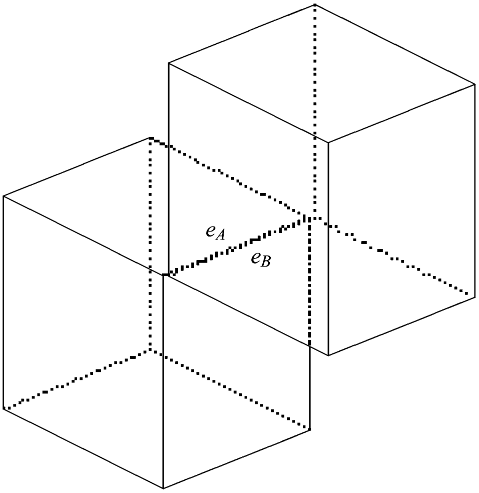


Fig. 4.6 Objects touching at an edge

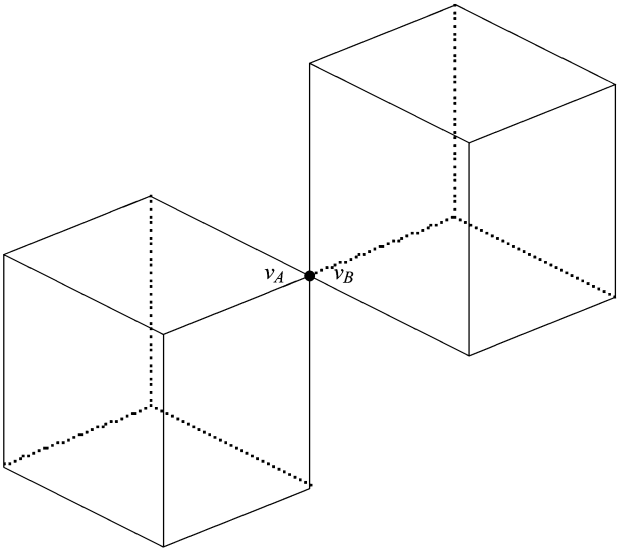


Fig. 4.7 Objects touching at a vertex

which is probably not the case. I have never seen such operations in a CAD system. If you create such an edge try chamfering it or rounding it, as explained in Sects. 4.7 and 4.8.

4.1.6.5 Two Bodies Touching at a Vertex

This is similar to the bodies touching at an edge, and is illustrated in Fig. 4.7.

Again, SUBTRACT should make no change to the object and INTERSECT should give no result. The interesting case is, again, what happens with ADD.

The "normal" result is a non-manifold vertex. The same comments as before are also relevant. It is not certain that the CAD system should do this without comment. There should also be operations to identify non-manifold elements and to expand them. The non-manifold case is more difficult to check that the edge case, but you can try blending all edges at the vertex too see what happens (normally the operation will be refused with some more-or-less obscure message).

4.2 Extrusion Operations

In modern systems there are two types of extrusion. The first type is where an extrusion is made from an initial 2D shape. The second type is where an extrusion is made as an increment to an existing 3D solid. The second type is more complicated because it involves a Boolean operation. The Boolean can be an add or subtract operation to create the result.

4.2.1 Parameters

Input:

Linear extrusion

Shape to be extruded
Vector extrusion

Circular extrusion

Shape to be extruded
Axis definition
Angle

Extrusion along a path

Shape to be extruded
Path definition

Output:

Result combined with the original object or perhaps a new multi-piece object.

4.2.2 Potential Errors

Some conditions that should be checked for are:

- Null vector—for linear extrusion.

- Null axis definition—for circular extrusion.
- Zero angle—for circular extrusion.
- Zero-length edge in path definition—for sweeping along a path.
- Closed path—for sweeping along a path. This could be handled, so is an optional error.
- Branching wire path—for sweeping along a path. Again, this could be handled so is an optional error.
- Open shape—only an error if the CAD system cannot handle mixed sheet- and solid-objects. If not an error should perhaps give a warning.
- More than two edges at a vertex of the shape being extruded—this is a common error.

4.2.3 The Simple Linear Extrusion Algorithm

The simple linear extrusion is a relatively simple operation. It has been around since the early 1970s and was part of Braid's original system reported in [2]. This was shown in Fig. 2.30. It simply involves stepping round the boundaries of the face extruding the edges.

The algorithm is, roughly:

```
LOOP *l = f->get_loop();
 while ( l != NULL )

    BEGIN
        EDGE *e, *xe, *qe;
        VERTEX *sv, *v, *xv, *yv, *sxv;
        e = l->get_edge();

        sv = vcwel(e,l); v = vopev(e, sv); xv = NULL;
        while ( e != NULL )

            BEGIN
                if ( xv = NULL )
                    BEGIN
                        sxv = xv = mev(sv, ecwel(e,l));
                       xv->set_pos(sv->get_pos()+extvec);
                        *qe = xv->get_edge();
                        qe->set_curve(vvstraight(sv, xv));
                    END
                if ( v IS sv )

                    BEGIN
                        xe = mfe(xv, sxv);
                        xe->set_curve(translate(e->get_curve(),extvec));
                        e = NULL;
                    END
                else
                     BEGIN
                          yv = mev(v, ecwel(e,l));
                          xv->set_pos(sv->get_pos()+extvec);
                          *qe = yv->get_edge();
                          qe->set_curve(vvstraight(v, yv));
                          xe = mfe(xv, yv);
                          xe->set_curve(translate(e->get_curve(),extvec));
                          e = eccel(e, l); v = vopev(e, v);
                     END
           END
      l = l->get_next();

END
```

Note the way that the Euler operators are used to create the new topology. This is a good example of a stepwise algorithm which builds up the model as a series of small steps.

This algorithm seems to be the only extrusion algorithm used in CAD systems. General extrusion is more complicated because there are more cases to take into account. If, having created the basic shape, you extrude a shape, upwards or downwards, from one of the faces of the base shape, then this is can be complex. The modern CAD solution seems to be to cut off the shape to be extruded, extrude this shape with the simple method and then add or subtract the new extruded shape to the base shape using a Boolean operation.

One last point, extrusion directions are often normal to the plane defining the shape to be extruded. This is not an absolute requirement, though, and some systems let you define the normal. The normal, though, should never be perpendicular to the normal of the plane of the shape being extruded.

4.2.4 Compound Linear Extrusion

Normally this is not dealt with, however it is mentioned here in case CAD systems change. Consider the shape shown in Fig. 3.34, reproduced in Fig. 4.8.

Using the simple extrusion algorithm, described above, gives something like the image in Fig. 4.9, though the geometry has been distorted to show the structure.

After one sweep the structure would be as shown in Fig. 4.10.

At this stage, the edges labelled e_0 and e_1 are “concave” edges. This means that, for example, when sweeping edge e_1 in face f_1 the edges e_2 and e_3 should be used as side edges rather than creating new side edges. This is possible, but the

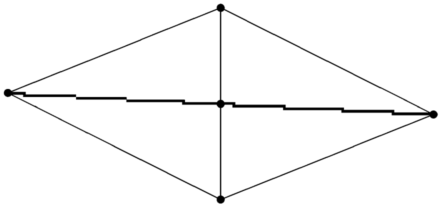


Fig. 4.8 2D sketch with multiple faces

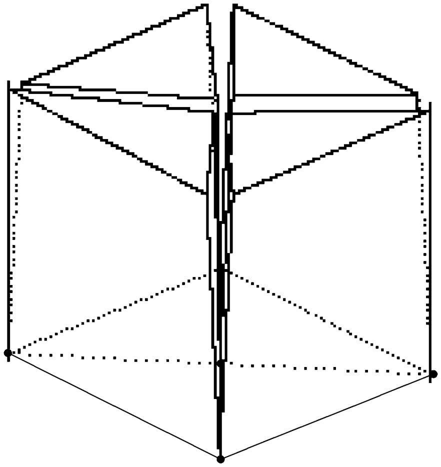


Fig. 4.9 Complex sketch with simple extrusion

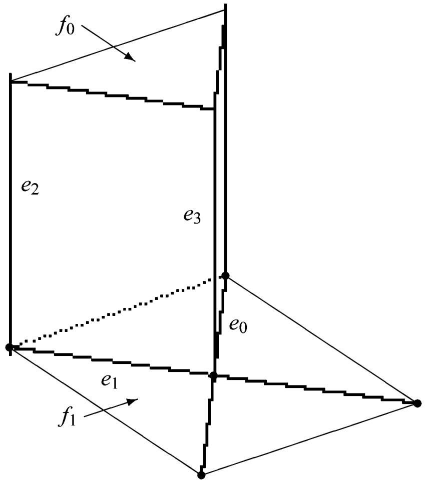


Fig. 4.10 Complex sketch with one swept face

algorithm is more complicated than the simple algorithm for linear extrusion described previously. A general method for extrusion was developed, though, towards the end of the 1970s in the BUILD system.

There are other methods to handle compound solids, though. The second method is shown in Fig. 4.11. From the original complex shape (Fig. 4.11a) the faces are separated (Fig. 4.11b). These are then extruded separately (Fig. 4.11c) and added (Fig. 4.11d).

This has a disadvantage, though, that the details of the complex shape are lost during the Boolean operation.

A third method is related to that for lofting, described later. Figure 4.12 shows the steps. First all the boundary edges are sliced (Fig. 4.12a). The matching corners are joined (Fig. 4.12b) creating the topology of the side faces. Finally the geometry is changed to create the final object (Fig. 4.12c).

The point of describing these complex shape extrusion methods is to demonstrate another "message". There are lots of extra things that can be done in CAD systems. There is not a theoretical complete set of operations and algorithms that you find. CAD systems do not allow complex 2D shape definition and complex shape extrusion now, probably because they don't want to. Chapter 3 explained

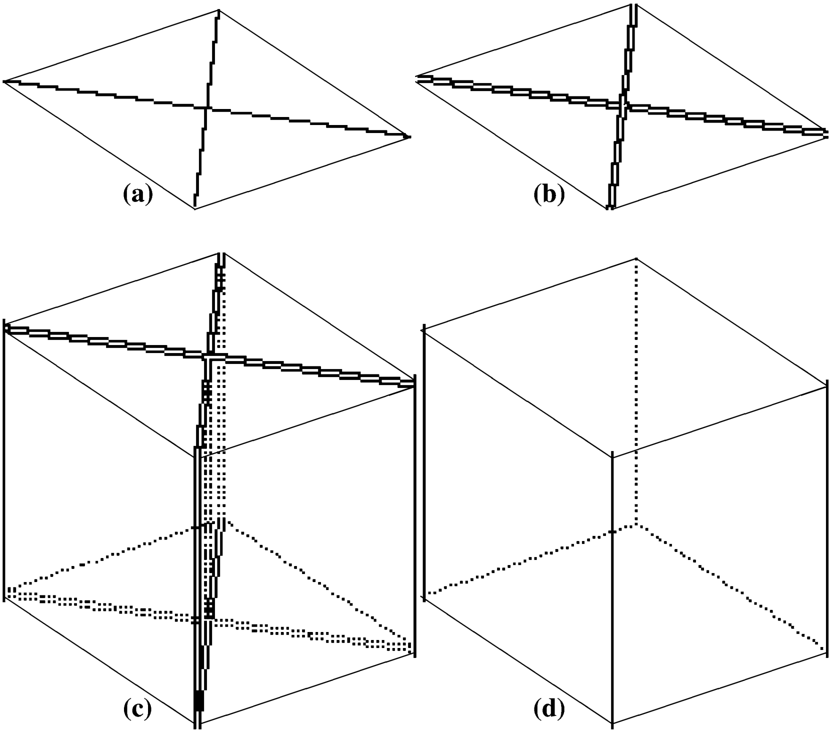


Fig. 4.11 Extruding 2D sketch with multiple faces, version 2

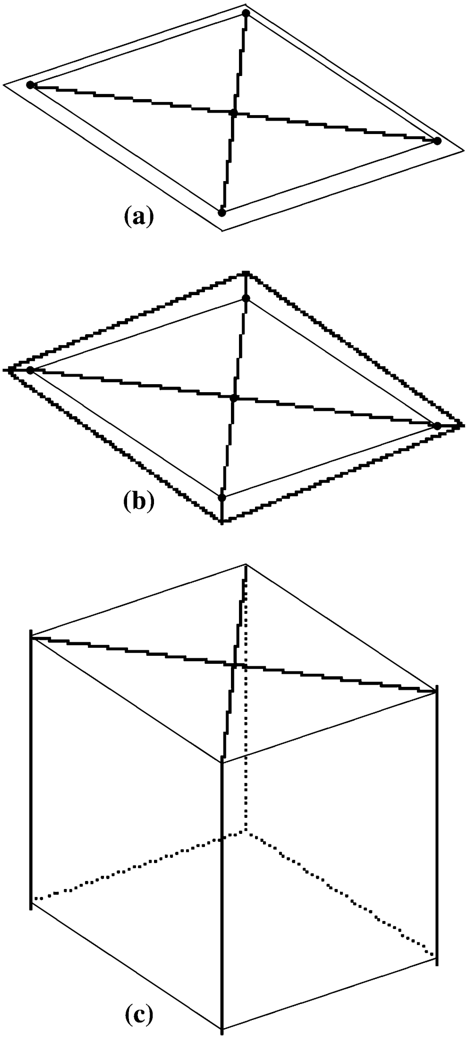


Fig. 4.12 Extruding 2D sketch with multiple faces, version 3

how it is possible to create complex shapes. This section explains how they can be extruded. The first method, of generalising extrusion to handle the special cases was implemented. The other two methods are other possibilities which are equally possible to implement. If you want to work in a different way to the way permitted then you should communicate this to the software vendor, if possible, to encourage continuing development.

4.2.5 Incremental Extrusion Limits

Once the basic shape has been created it is common that new extrusions are made to add or subtract elements to a model. There is a distance parameter but it is also possible to use geometric reference elements to determine the extrusion extent. Common ones are:

- Until last
- Until next
- Until surface
- Until face

It may not always be clear what the difference between these is and when one should be used instead of another. Figure 4.13 is an attempt to illustrate these. With a distance the length is what the user specifies. "Until last" calculates an extreme point which is the extent of the object in the extrusion direction. "Until next" calculates the extent until the next face in the extrude path. "Until face" calculates the extent to the specified face. "Until surface" calculates the extent to a specified surface, and the entire shape should cut this surface.

There are two possibilities for calculating "last", "next", etc. One method is that the extruded element is limited completely by a face, the other that the extrusion limit is the surface of a face. To understand these, consider Fig. 4.14.

Make the sort of E-shape shown in the figure. Make the middle prong half the height of the other two. Draw a square on the inside of one of the end faces and extrude it "until next". Check whether it goes half way inside or the whole way inside. There are two ways of interpreting the "next". It could be the next surface that any of the contour being extruded touches or it could be the next face that surrounds the extruded contour. It is necessary to do this kind of experiment to find out how your CAD system interprets "next", "last" and so on.

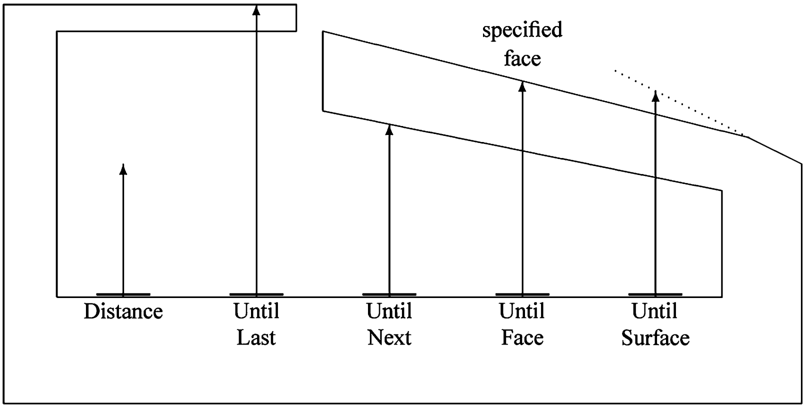


Fig. 4.13 Extrusion limits

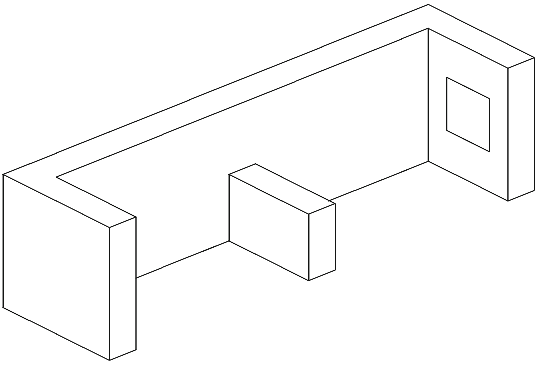


Fig. 4.14 Extrusion limits

Experiments with these options are suggested in Sect. 4.2.10.3. Essentially what they are doing is providing a functional definition of the extrusion distance, but the "until" types are slightly more complicated because they perform a "setsurf" operation as well, to set the final geometry into a new surface. This is needed to set the extrusion limit correctly. If they are just simple extrusions then there is a risk that some of the extrusion will extend outside the existing surfaces.

4.2.6 The Simple Circular Extrusion Algorithm

Originally circular extrusion was a separate operation, but circular and linear extrusion are closely related so they are now done in the same way. There is a difference in the geometry of the side elements, which can be seen in Fig. 4.15.

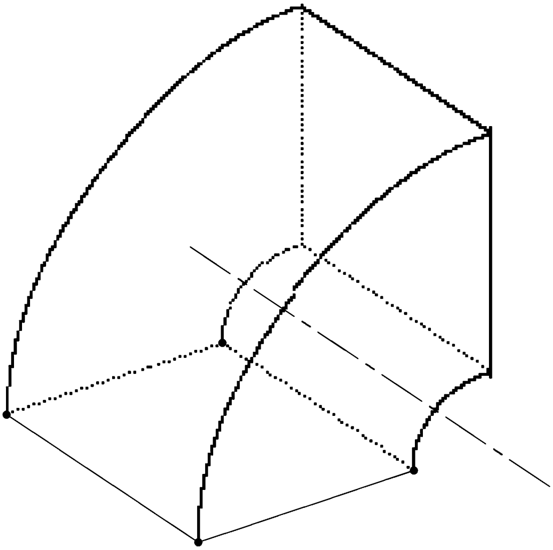


Fig. 4.15 Simple circular extrusion

The relation may be less obvious if an edge, or edges lie on the rotation axis, as in Fig. 4.16 but this is also related.

For the edge on the face, were the same algorithm used as used for linear extrusion, then there would be a degenerate face. To avoid this it is necessary to check for some special cases.

The algorithm is, roughly:

```
l LOOP *l = f->get_loop();
 while ( l != NULL )
    BEGIN
        EDGE *e, *xe, *qe;
        VERTEX *sv, *v, *xv, *yv, *sxv;
         e = l->get_edge();
         sv = vcwel(e,l); v = vopev(e, sv); xv = NULL;
          while ( e != NULL )
             BEGIN
                 if ( xv = NULL )
                     BEGIN
                         if ( NOT edge_on_axis(e, pt, axdir) ) then
                             BEGIN
                                 if ( vertex_on_axis(sv, pt, axdir) then
                                     sxv = xv = sv;
                                 else
                                     BEGIN
                                         sxv = xv = mev(sv, ecwel(e,l));
                                         xv->set_pos(sv->get_pos()+
                                                 extvec);
                                         *qe = xv->get_edge();
                                         qe->set_curve(vvcircle(sv,
                                                 xv, pt, axdir));
                                     END
                             END
                     END
                  if ( v IS sv )
                     BEGIN
                         if ( NOT edge_on_axis(e, pt, axdir) ) then
                             BEGIN
                                 xe = mfe(xv, sxv);
                                 xe->set_curve(
                                     translate(e->get_curve(),extvec));
                             END
                         e = NULL;
                     END
                 else BEGIN
                         if ( NOT edge_on_axis(e, pt, axdir) ) then
                             BEGIN
```

```
                        yv = mev(v, ecwel(e,l));
                        xv->set_pos(sv->get_pos()+extvec);
                        *qe = xv->get_edge();
                        qe->set_curve(vvstraight(v, xv));
                        xe = mfe(xv, yv);
                        xe->set_curve(
                            translate(e->get_curve(),extvec));
                        e = eccel(e, l); v = vopev(e, v);
                    END
                else BEGIN
                        e = eccel(e, l); xv = v; v = vopev(e, v);
                    END
                    END
        END
  l = l->get_next();
END
```

This method also means that it is possible to extrude round an axis to a degree less than 360°. If the modeller does not like extruding round 360° in one step then you might have two or three applications of the procedure to create the whole object.

If a shape is extruded through a complete circle, there is one final step at the end: to join the swept face and the back face of the original 2D shape to create a closed solid.

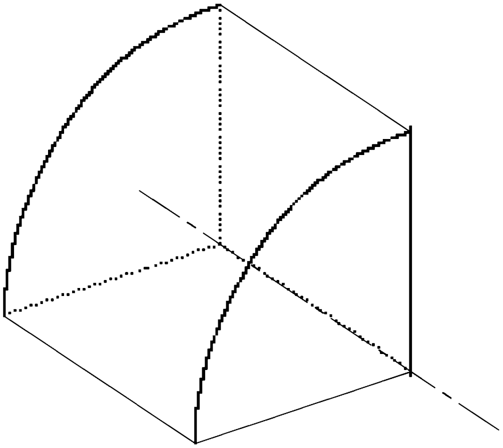


Fig. 4.16 Simple circular extrusion

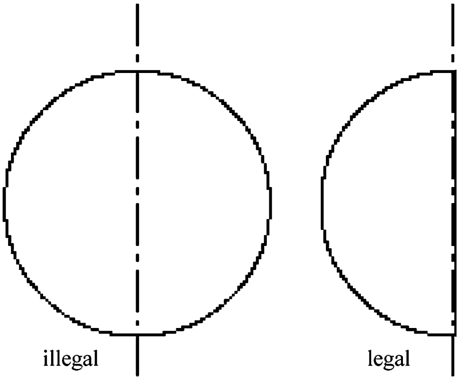


Fig. 4.17 Example of illegal and legal extrusion shapes to create a sphere

The rotation axis should not cut through the shape being swept, otherwise the operation would create a self-intersecting object.

An example of this concerns a common error for producing a sphere, shown in Fig. 4.17.

Why this is worth mentioning is that it is an example of where the user and implementer philosophies differ. It is not that the user is wrong, it is rather a case where CAD systems should be improved to cope with such cases.

The problem, or potential problem is that some users find it natural to draw both sides of an object, it is a matter of perception. A workable solution to this is to cut the 2D shape into two pieces, using the rotation axis as a separator, extrude each shape through half the angle and then add the results. This is illustrated in Fig. 4.18. Note, though, that normally the two halves will be the same. They are shown here as different just to emphasise the solution.

On the left of Fig. 4.18 is the original shape, on the right is how the shape can be adapted to produce a solid made of two parts. The result object might be as shown in Fig. 4.19.

The point of explaining this is not to criticise the way things are done in CAD systems. The point is that you can make different basic assumptions and get different results for what appears to be the same operation. The way things are done in CAD systems, that is, by defining half the shape and rotating it, has some significant advantages. One of these is that the shape is rotationally symmetric and, if swept through 360°, you get a shape which is appropriate for turning. The shape shown in Fig. 4.18 is not symmetric and, therefore, there is a discontinuity. Leaving that apart, what I am trying to explain here is the way in which the implementer works. That is, the implementer makes some decisions about the input parameters and the functionality of the operation and that is what you see. Sometimes this is arbitrary, so different implementations may

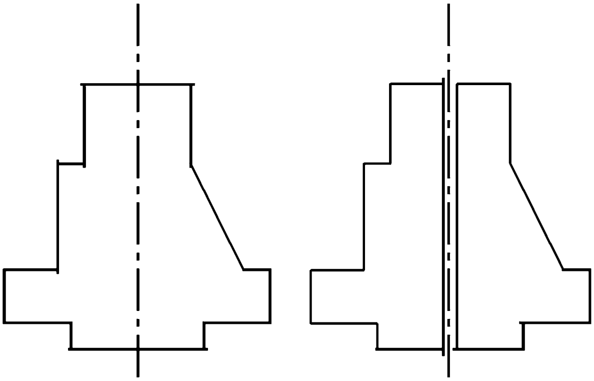


Fig. 4.18 Handling 2D shapes crossing the rotation axis

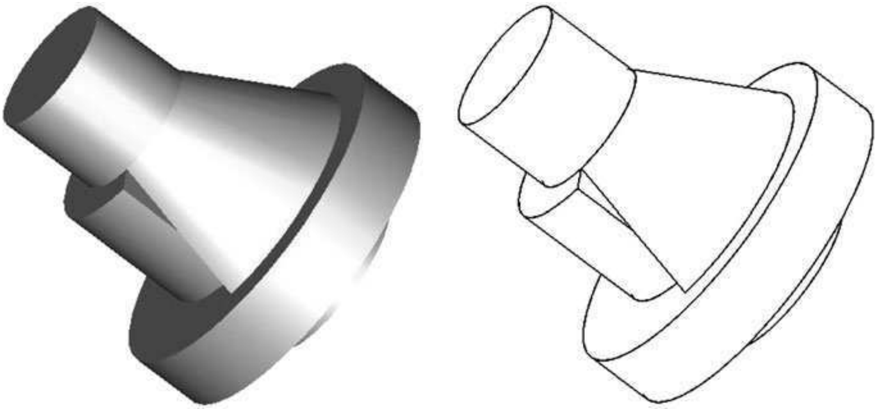


Fig. 4.19 Circular extruded two-halved shape

work slightly differently. This effect is lessened because several systems use the same geometric engines, or modelling kernels, and hence have the same basic functionality. However, it is useful, when starting with a new system, to perform some simple experiments to determine the limits of the functionality in simple cases and hence avoid problems. This is one of the purposes of this book. You should also be very clear about the fact that it is usually the case that the software developers are computer scientists. Their perceptions may well be different from those of the users. Another aim of this book is to try and explain these differences.

4.2.7 Extruding Along a Curve

Yet another variant of extrusion is extrusion along a curve, Fig. 4.20. This is something like linear extrusion, but the sidewall geometry is more complex. There are also more special cases. If the curve is too curved then there is a risk of creating a self-intersecting object during the operation. This may or may not be detected automatically by the system.

Looking sideways at the object, shown in Fig. 4.21 gives a clearer idea of the potential problem.

The thick line at the bottom on the left represents the shape to be extruded along the curve, on the right of the figure you see what would happen, with a small self-intersecting part. This is a simple and rather artificial case, but the problem is to find where the centre of curvature of the extrusion path curve lies inside the shape being extruded. When this happens there is a self-intersection of the side surfaces as well as the object. This also depends on how the path curve is positioned relative to the object. This is a parameter of the operation.

Fig. 4.20 Extruding a 2D shape along a curve

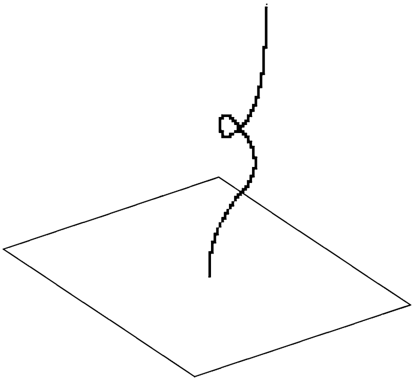
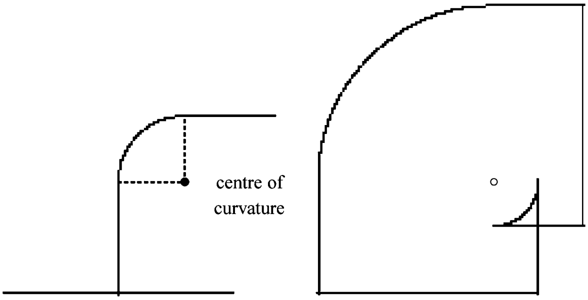


Fig. 4.21 Extruding a 2D shape along a curve, from the side



4.2.8 Extruding Along a Path

Some CAD systems allow extrusions along a path composed of a set of edges. If the edges are a sequence of straight and circular arcs then the operation can be interpreted as a sequence of linear and circular extrusions. With other edge geometries you extrude along a curve.

The operation can be useful for creating pipework, for example. The pipework would be represented by a set of edges and the pipe cross-section extruded along these to create a volumetric pipe shape which can then be hollowed out using a shelling operation.

As with extruding along a curve, this operation can also be used sometimes to create self-intersecting objects. It is normal to do some checking for each edge, but this is not sufficient. The case shown above, Fig. 4.21, in Sect. 4.2.7 is also relevant for sweeping along a path.

In Fig. 4.22 the "shape" to be extruded is represented by a vertical bar, with the path the path on the left-hand side of the figure is not allowed because the second element of the path, the vertical line in the figure, is perpendicular to the shape being extruded. The resulting object would have a degenerate middle portion, as shown on the right.

To manage the corners it is often necessary to round off the corners so that the shape turns to be perpendicular to the straight sections, as shown in Fig. 4.23. It is, of course, possible to have a path with only straight sections, but the must not be perpendicular to the shape being extruded.

What happens is that the straight edges are interpreted as vectors for linear extrusion, the circular arcs as defining an axis for circular extrusion. The axis passes through the centre of the circle, with direction normal to the plane of the circle. The individual extrusions may or may not be checked. It is more efficient not to do too much checking, but this leads to other problems, as for the path in the experiments, later.

It is also interesting to note what happens when the path starts in the plane of the shape being extruded, and what happens when it is offset slightly. The vectors defined by the straight edges can easily be taken as being relative to the face, but the circle centre points defining the axes may be taken as either relative or absolute positions.

4.2.9 Possible Variants

The older versions of extrusion performed extrusions "in situ" instead of creating separate volumes and combining these with the base object using Boolean operations. This is still an option but would need a self-intersecting object evaluation as a final step to check for interactions.

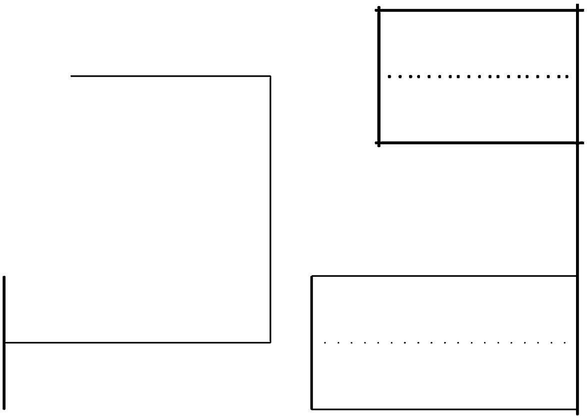


Fig. 4.22 Illegal path and resulting object

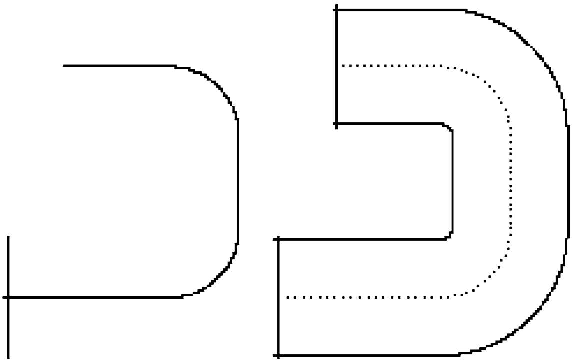


Fig. 4.23 Legal path and resulting object

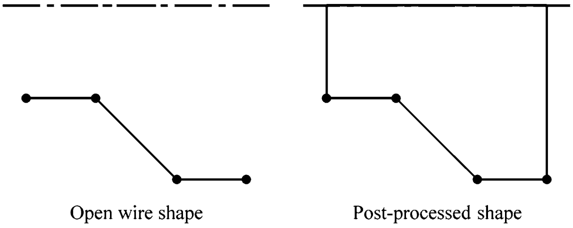


Fig. 4.24 Circular extrusion defined by a wire

For circular extrusions an early version did not need to close the shape. As shown in Fig. 4.24, the initial wire, on the left defined the outside shape and was processed to produce a closed shape before extrusion.

Some systems allow you to define complex relationships for defining extrusions, which can be termed: “functional extrusion”. Examples might be:

- Allowing a scaling for the extrusion so the final shape is larger or smaller than the starting shape.
- Allowing a displacement so that the final shape is displaced in a certain direction, by a certain amount.
- Allowing a rotation about an axis normal to the plane of the extrusion shape.

These are just a few possible functional extrusion types. They allow you to define some regular shapes relatively simply. If such functions do not exist it may be possible to produce the shapes using sweeping along a curve, or lofting (see Sect. 4.14) instead.

4.2.10 Experiments to Try

The first two experiments repeat some of the Boolean operation experiments to demonstrate the use of the Boolean operations for extrusion.

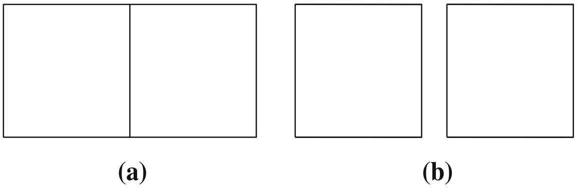
4.2.10.1 Touching Objects

One experiment is where two objects just touch. Create a square shape, 100 × 100 and extrude it by 100. On the top of this, create another square, 100 × 100 which touches one of the edges of the base shape, see Fig. 4.25a. Extrude this new shape by 100. Verify whether the touching edge becomes a non-manifold edge or not, as with the Boolean experiment.

4.2.10.2 Missing Objects

Another experiment is where the objects do not touch at all. Create a square shape, 100 × 100 and extrude it by 100. On the top of this, create another square, 100 × 100 which is outside the top at a distance of ten, say, from one of the edges of the base shape, see Fig. 4.25b. Extrude this new shape by 100. Check whether or not the CAD system signals that the result is a multi-body result, as with the Boolean experiment.

Fig. 4.25 Touching and non-touching extrusions



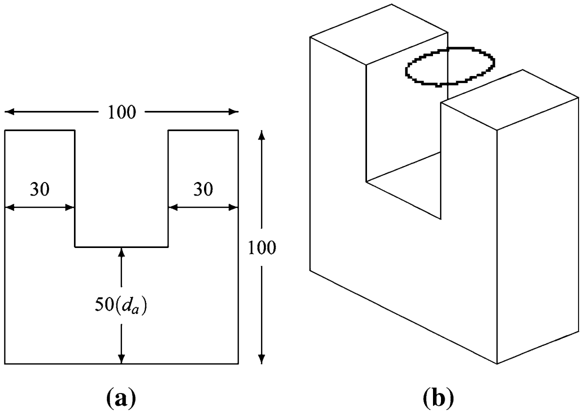


Fig. 4.26 Experiment with extrusion length definitions

4.2.10.3 Experiments with Limits

It is worthwhile understanding how to use the limits defined in Sect. 4.2.5. There is a longer exercise on this topic at the end of the chapter. Make the shape shown in Fig. 4.26a, extrude it, say 40 units. Now select one of the two prongs for sketching and draw a circle, lying between the two prongs but not touching them, as shown in Fig. 4.26b.

Extrude the circle downwards 50 units. Now use the part construction history to change the dimension marked d_a in the figure to 49.99. Does the extruded part still make contact with the base object? Can you tell? Does the system report that you have a multi-body result? Change dimension d_a to 25 and look again. What is to be expected is that you do not always see whether or not the parts are connected and that the CAD system does not tell you. However, if you can, use the part construction history to change the parameter of the extrusion from a fixed length, 50, to a calculated length, "until next" for example. Redo the experiments and check that the extrusion length changes.

4.2.10.4 Multiple Components

The next experiment is with a contour with internal parts. Make a shape such as that shown in Fig. 4.27.

Extrude the shape and look at the result. You might get something like that shown in Fig. 4.28.

There are several possible variants. In Braid's original BUILD system the contours destined to be holes had to be explicitly "punched" through to make them holes (using Euler operator Kill Face Make Hole Genus—KFMHG). The current strategy is that the internal contours become holes through the shape. The left hand circular

Fig. 4.27 Experiment with multiple pieces

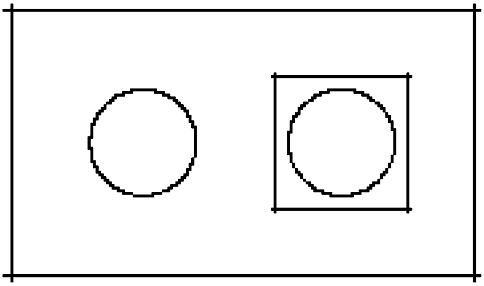
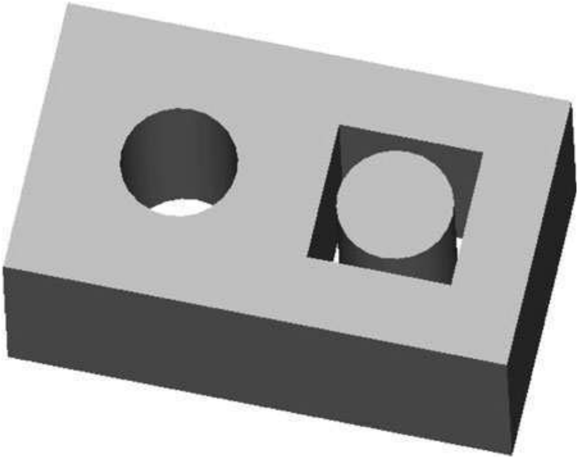


Fig. 4.28 Multi-piece result



shape becomes a hole as does the square shape. The second circular shape has to be solid because it is then outside the shape and so becomes a positive shape.

4.2.10.5 Apex-to-Apex Triangles

Another experiment is with a contour consisting of two triangles touching at one corner. This was mentioned in Sect. 3.7, the result is shown in Fig. 3.29. Make a shape like that shown in Fig. 4.29. Make sure that there are six edges, the two crossing edges should be broken at their intersection point.

Extrude this figure, it is not important exactly how much, the interesting thing is what happens to the edge which derives from the common vertex. Try blending or chamfering this edge to see if it is non-manifold or a duplicated edge. See also Sect. 4.8.

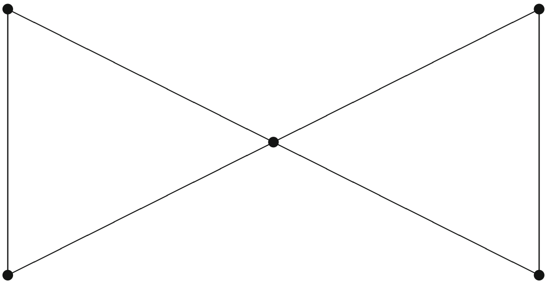


Fig. 4.29 Touching vertices

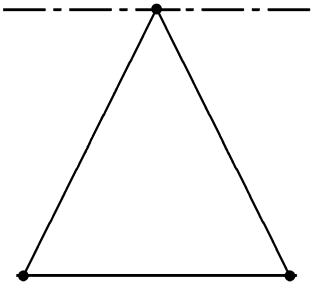


Fig. 4.30 Vertex on axis

4.2.10.6 Vertices on Axes for Circular Extrusion

Try creating a figure for circular extrusion with one vertex on the axis, as shown in Fig. 4.30.

The question is what happens at the vertex on the axis when the shape is extruded by 360° around the axis. The vertex could be a non-manifold vertex or two vertices. Blend all the edges at the vertex and look at the result.

4.2.10.7 Along-Path Extrusion

The experiment with extrusion along a path is to try and create a self-intersecting object. The path is shown in Fig. 4.31, the profile to be extruded is a 10 × 10 square. First try with a corner radius of 5 to see if the system performs checks to see if the axis intersects with the profile. (In the two systems I have tried this with, one did and one didn't).

Obviously, if you examine the path, there will be an overlap of the objects at the end. The path itself does not intersect but extruded object does, as shown in Fig. 4.32. On the left is the whole object and on the right is a detail of the overlap. This is a self-intersecting object, and this is one of the few ways to

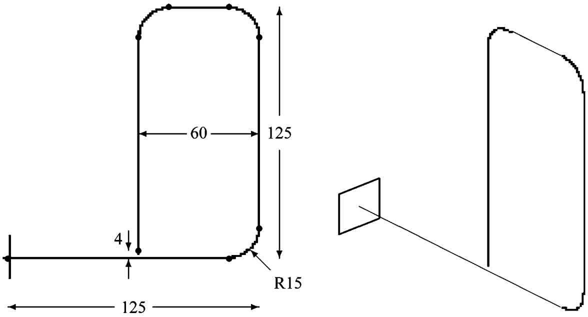


Fig. 4.31 Path creating a self-intersecting object

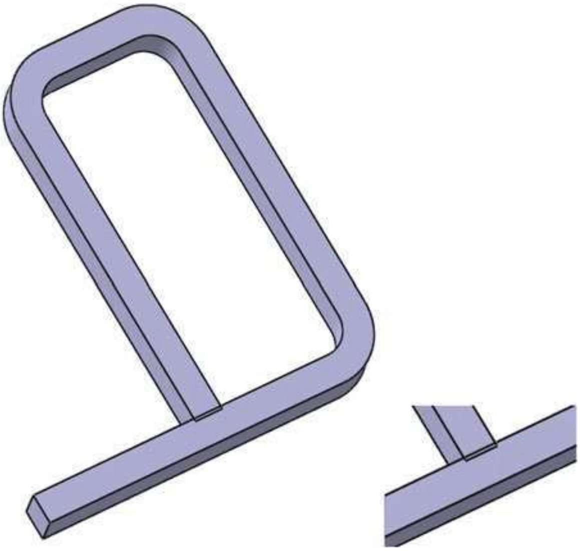


Fig. 4.32 Self-intersecting path extrusion object

show it in commercial software. This problem has almost completely disappeared, though it is useful to be able to show it for historical purposes. If you try subtracting a cylinder, or rectangular block from the overlapping region you may well see more problems as the Boolean operations get lost in the object reconstruction.

An interesting variant is if the start of the path is offset from the plane of the profile shape. Figure 4.33 shows what happens if the offset is 10 mm.

In the implementation illustrated, the straight edge vectors are used to extrude the profile shape from its current position. However, the circular sections are calculated using the absolute centre of the circular arc as an axis point, hence the geometry is a mixture of relative and absolute elements and not what would be expected.

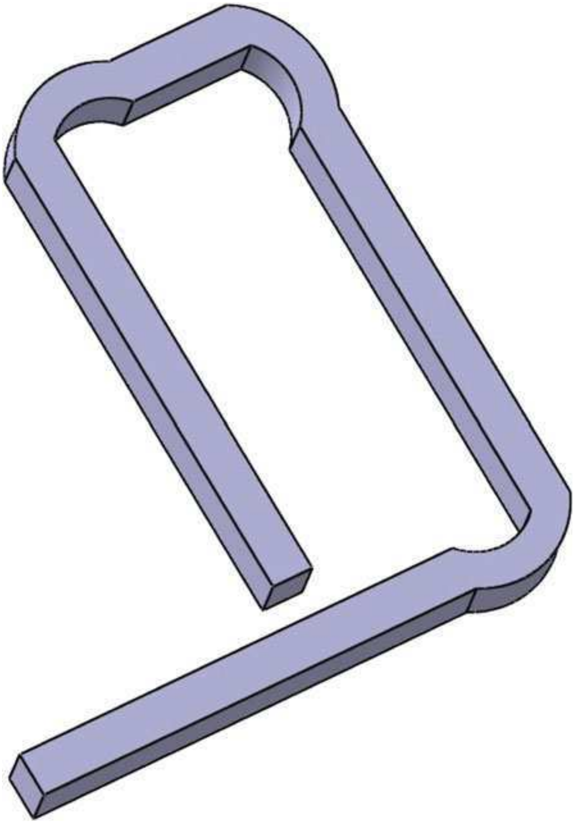


Fig. 4.33 Path not starting on profile plane

4.2.10.8 Along Curve Extrusion

This experiment is to create a spiral, or a screw thread by extruding along a spiral curve. The purpose is just to show that this method can be used for comparison later with lofting.

The spiral path is simply an interpolation of nine points, in this case:

(50, 0, 0)
(0, 50, 20)
(−50, 0, 40)
(0, −50, 60)
(50, 0, 80)
(0, 50, 100)
(−50, 0, 120)
(0, −50, 140)
(50, 0, 160)

Note that this does not give a very good spiral, the interpolation does not always give good results for regular shapes. Interpolation will be explained in Chap. 5. For a more serious example it would be necessary to use more points and, perhaps, tangents at the start and end of the curve.

The profile is defined in a plane perpendicular to the path curve and simply extruded up to get the object. The original path is shown on the left of Fig. 4.34 and the final object on the right of the figure.

4.2.10.9 Sweeping an Open Figure

Some systems allow you to extrude open figures and mix surface and solid models. This is more flexible, but the risk is that you create a sheet object instead of a solid.

Make an open shape like that in Fig. 4.35a and extrude it using a straight extrusion. If that works then you can try two circular extrusions. Make a shape like that shown in Fig. 4.35b and extrude it round an axis. If the system allows this, then try making a hole through the thick part. If you get a result then it should be two circles, but there is no linking surface. Try a third variant, based on the shape in Fig. 4.35c. This time the open points lie on the rotation axis, so when you

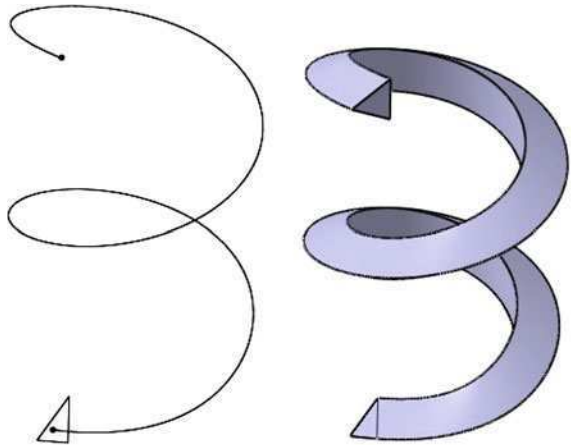


Fig. 4.34 Spiral path and profile, final object

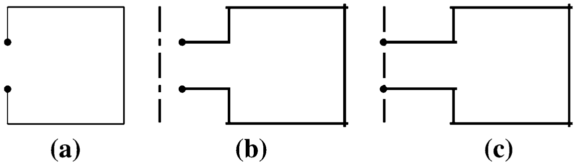


Fig. 4.35 Open shape for extrusion

extrude it it may look solid, but it is not certain whether or not it is. To check, punch a hole through it and see whether the hole is really a hole or two isolated circles.

Note that some systems, like CATIA v5, use a different colour to indicate a surface or sheet model, other systems may not show any difference so make sure that sketches are closed. You can convert open sheet objects to solids by giving them thickness (Sect. 4.11). Closed sheet models can be made solid like that, or you may be able to fill them (Sect. 4.12).

4.3 Reflect or Symmetry

Reflection, sometimes known as symmetry, makes it possible to create part of the object and complete it in a single operation by creating and joining the symmetrical part. This is useful if you know beforehand, but this is not always the case.

4.3.1 Parameters

Input:

Body to be reflected
Reflection plane

Output:

Result is combined with the original body, possibly as a multi-piece solid.
Optional new body if allowed as an option.

4.3.2 Potential Errors

Some conditions that should be checked for are:

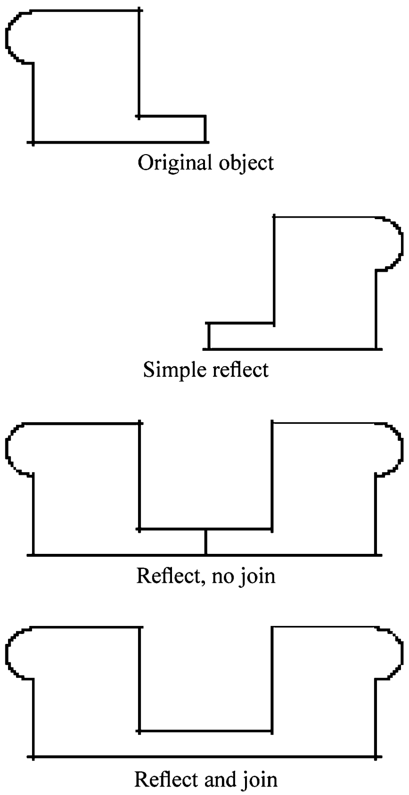
- Non-planar reflection surface.
- Possibly object crossing reflection plane.

4.3.3 Reflection Algorithm

The original reflect/symmetry algorithm was:

1. Create reflect transformation matrix.
2. Copy object.

Fig. 4.36 Reflection operation and options



3. Transform copy with reflect transformation matrix.
4. Join object and copy at coincident faces (3D object) or edges (2D object).

Modern tools have more options, as summarised in Fig. 4.36.

- The first option is simply to transform the original body.
- The second option is to create the copy and transform it without joining.
- The third option is to create the copy, transform it and join the copy and original object.

4.3.4 Creating the Transformation Matrix

Transformations are described in Sect. 5.2.2. The transformation matrix is created from a plane surface. Originally, in the BUILD system, this was a face of the object being reflected, thus guaranteeing a join. In the Swedish system, GPM, free-standing geometry was allowed and hence it was no longer a requirement to pick a planar face. Now, any available plane can be used.

The reflection surface is currently only allowed to be planar. This is a realistic limitation because otherwise the object shape would change. In the original implementation the planar surface was not allowed to cut the object, as this would have resulted in a self-intersection object. Now, with the use of Boolean operations for joining this restriction has been lifted.

Reflection transformations along the three principal axes are simple:

Reflection along X:

$$\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Reflection along Y:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Reflection along Z:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

For reflection about a general plane it is possible to decompose the matrix into a series of rotations, a reflect transformation and another series of rotations. Assume that the vector normal to the plane is:

$$(x, y, z) \quad \text{where } x^2 + y^2 + z^2 = 1$$

and a point through which the plane passes is:

$$(P_x, P_y, P_z)$$

If x and y are both zero then the reflection is along the Z-axis. Otherwise it is possible to rotate the object around the Z-axis to align it with the X-axis and then reflect it along the X-axis.

First, for the case when x and y are zero, the sequence can be decomposed into a translation, a reflection and another translation. The three matrices are:

$$\begin{bmatrix} 1 & 0 & 0 & P_x \\ 0 & 1 & 0 & P_y \\ 0 & 0 & 1 & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -P_x \\ 0 & 1 & 0 & -P_y \\ 0 & 0 & 1 & -P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

which, when multiplied together, give the single matrix:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 2P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This also shows that the movement in the *X*- and *Y*-directions is not necessary, as would be expected.

Taking the general case you get seven matrices. These correspond to aligning a vector with the *X*-axis, reflecting, and then the inverse to the alignment.

Translation to the origin:

$$\begin{bmatrix} 1 & 0 & 0 & -P_x \\ 0 & 1 & 0 & -P_y \\ 0 & 0 & 1 & -P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation to the *X*–*Z* plane (using $q = \sqrt{x^2 + y^2}$):

$$\begin{bmatrix} x/q & y/q & 0 & 0 \\ -y/q & x/q & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation to the *X*-axis is simplified because the plane normal vector is normalised:

$$\begin{bmatrix} \sqrt{1-z^2} & 0 & z & 0 \\ 0 & 1 & 0 & 0 \\ -z & 0 & \sqrt{1-z^2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Reflection along *X*:

$$\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation back from the *X*-axis:

$$\begin{bmatrix} \sqrt{1-z^2} & 0 & -z & 0 \\ 0 & 1 & 0 & 0 \\ z & 0 & \sqrt{1-z^2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation back from the X–Z plane:

$$\begin{bmatrix} x/q & -y/q & 0 & 0 \\ y/q & x/q & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Finally, translation back:

$$\begin{bmatrix} 1 & 0 & 0 & P_x \\ 0 & 1 & 0 & P_y \\ 0 & 0 & 1 & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Multiplying these together gives the final transformation matrix.

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where:

$a = -x^2 + \frac{(x^2z^2+y^2)}{(x^2+y^2)}$
$b = -xy + \frac{(xyz^2-xy)}{(x^2+y^2)}$
$c = -2xz$
$d = P_x + x^2P_x + xyP_y + 2xzP_z + \frac{(-xyz^2P_y-x^2z^2P_x-y^2P_x+xyP_y)}{(x^2+y^2)}$
$e = -xy + \frac{(xyz^2-xy)}{(x^2+y^2)}$
$f = -y^2 + \frac{(y^2z^2+x^2)}{(x^2+y^2)}$
$g = -2yz$
$h = P_y + xyP_x + y^2P_y + 2yzP_z + \frac{(-xyz^2P_x-y^2z^2P_y-x^2P_y+xyP_x)}{(x^2+y^2)}$
$i = -2xz$
$j = -2yz$
$k = x^2 + y^2 - z^2$
$l = P_z + 2(xP_x + yP_y)z + P_z(z^2 - x^2 - y^2)$

For example, if the reflection plane:

$$\text{point: } (6,7,8) \text{ normal: } (0.3\sqrt{2}, 0.4\sqrt{2}, 0.5\sqrt{2})$$

you get:
Move to plane point and rotation about Z-axis:

$$\begin{bmatrix} 0.6 & 0.8 & 0 & 0 \\ -0.8 & 0.6 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -6 \\ 0 & 1 & 0 & -7 \\ 0 & 0 & 1 & -8 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.6 & 0.8 & 0 & -9.2 \\ -0.8 & 0.6 & 0 & 0.6 \\ 0 & 0 & 1 & -8 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Multiply by the rotation to align the x-axis:

$$\begin{bmatrix} 0.5\sqrt{2} & 0 & 0.5\sqrt{2} & 0 \\ 0 & 1 & 0 & 0 \\ -0.5\sqrt{2} & 0 & 0.5\sqrt{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.6 & 0.8 & 0 & -9.2 \\ -0.8 & 0.6 & 0 & 0.6 \\ 0 & 0 & 1 & -8 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.3\sqrt{2} & 0.4\sqrt{2} & 0.5\sqrt{2} & -8.6\sqrt{2} \\ -0.8 & 0.6 & 0 & 0.6 \\ -0.3\sqrt{2} & -0.4\sqrt{2} & 0.5\sqrt{2} & 0.6\sqrt{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Reflection along X:

$$\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.3\sqrt{2} & 0.4\sqrt{2} & 0.5\sqrt{2} & -8.6\sqrt{2} \\ -0.8 & 0.6 & 0 & 0.6 \\ -0.3\sqrt{2} & -0.4\sqrt{2} & 0.5\sqrt{2} & 0.6\sqrt{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} -0.3\sqrt{2} & -0.4\sqrt{2} & -0.5\sqrt{2} & 8.6\sqrt{2} \\ -0.8 & 0.6 & 0 & 0.6 \\ -0.3\sqrt{2} & -0.4\sqrt{2} & 0.5\sqrt{2} & 0.6\sqrt{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation back from the X-axis:

$$\begin{bmatrix} 0.5\sqrt{2} & 0 & -0.5\sqrt{2} & 0 \\ 0 & 1 & 0 & 0 \\ 0.5\sqrt{2} & 0 & 0.5\sqrt{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -0.3\sqrt{2} & -0.4\sqrt{2} & -0.5\sqrt{2} & 8.6\sqrt{2} \\ -0.8 & 0.6 & 0 & 0.6 \\ -0.3\sqrt{2} & -0.4\sqrt{2} & 0.5\sqrt{2} & 0.6\sqrt{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & -1 & 8 \\ -0.8 & 0.6 & 0 & 0.6 \\ -0.6 & -0.8 & 0 & 9.2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation back from the X–Z plane:

$$\begin{bmatrix} 0.6 & -0.8 & 0 & 0 \\ 0.8 & 0.6 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & -1 & 8 \\ -0.8 & 0.6 & 0 & 0.6 \\ -0.6 & -0.8 & 0 & 9.2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.64 & -0.48 & -0.6 & 4.32 \\ -0.48 & 0.36 & -0.8 & 6.76 \\ -0.6 & -0.8 & 0 & 9.2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Finally, translation back:

$$\begin{bmatrix} 1 & 0 & 0 & 6 \\ 0 & 1 & 0 & 7 \\ 0 & 0 & 1 & 8 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.64 & -0.48 & -0.6 & 4.32 \\ -0.48 & 0.36 & -0.8 & 6.76 \\ -0.6 & -0.8 & 0 & 9.2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Giving the final matrix:

$$\begin{bmatrix} 0.64 & -0.48 & -0.6 & 10.32 \\ -0.48 & 0.36 & -0.8 & 13.76 \\ -0.6 & -0.8 & 0 & 17.2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4.3.5 Copying and Transforming the Object

These are standard operations which are part and parcel of solid modelling systems and are described in Sect. 2.7.5. For single reflections, though, the determinant of the matrix is -1, which means that the transformed object is negated.

4.3.6 Joining the Objects

The original operation used a local face/face or edge/edge join, but this has now also been replaced by a Boolean add operation (for better or worse …). One advantage of the Boolean operation is that any plane can be used as a basis of the transformation. Nor is it necessary to perform the check that the object does not cross the plane, as with the local joining operation. A disadvantage, though, is that Boolean operations lose information about why they have been applied. Another disadvantage, which was already described for Boolean addition and extrusions, is where the objects being added do not touch. This can happen where the reflection plane is not a plane of the object, but a free-standing plane outside the object.

4.3.7 Sub-Operation Symmetry

Because of the tendency to perform modelling operations as a volume creation task together with a Boolean operation it means that the volumes created can also be subjected to symmetry operations. The volume created by the operation is simply copied, transformed with a reflection transformation and the same type of Boolean operation (addition or subtraction) as for the operation.

The same comments apply as for Boolean operations, if the reflected sub-operation body lies outside the original body, and is to be added, you may get multiple objects.

4.3.8 Experiments

4.3.8.1 Reflection in a Curved Face

First of all, make an object with a curved face. Check whether or not you can reflect the object in this face. You should not be able to, but it is worth checking once in case. The reason to check this is, if it is possible, then the CAD system will go ahead and reflect an object in any face, which may cause problems if you inadvertently select a curved face when using it. It is not reasonable to expect the CAD system to be able to do this, and it is not certain that it is very useful, but check it.

4.3.8.2 Reflection Creating a Cavity

This is a sneaky way of creating an internal cavity in an object. Make a rectangular block, 100 × 100 × 50 mm, say, but this is not too important. Sketch a 50 × 50 mm square on the top face and extrude it downwards 25 mm to make a "pocket". See Fig. 4.37.

Now reflect the object using the top face as the symmetry plane. You have created an object with a cavity. The same comments apply as for subtracting one object from another where the object being subtracted is entirely enclosed by the other object. It would be useful to have a warning, or some indication that there is now a cavity in the object. Otherwise, you might have trouble manufacturing it.

4.3.8.3 Reflection Creating Disjointed Objects

Create a sketch as shown in Fig. 4.38.

Extrude this shape 20 mm to create the three dimensional basic object (Fig. 4.39).

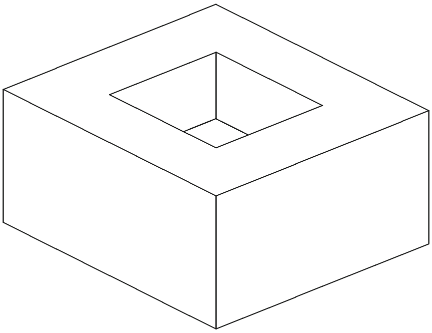


Fig. 4.37 Basic object for making a cavity

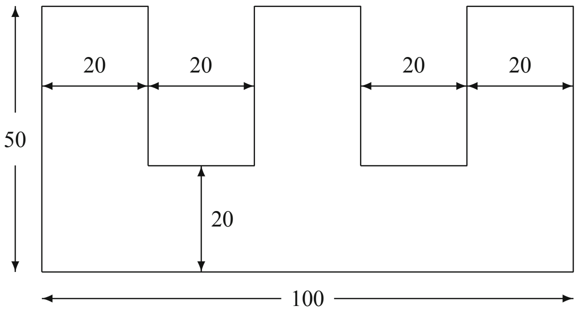


Fig. 4.38 Basic reflection shape

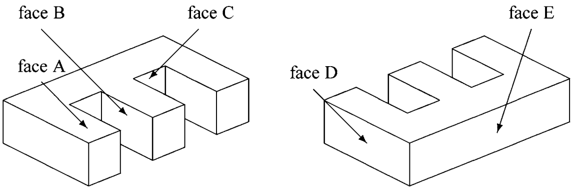


Fig. 4.39 Extruded reflection shape

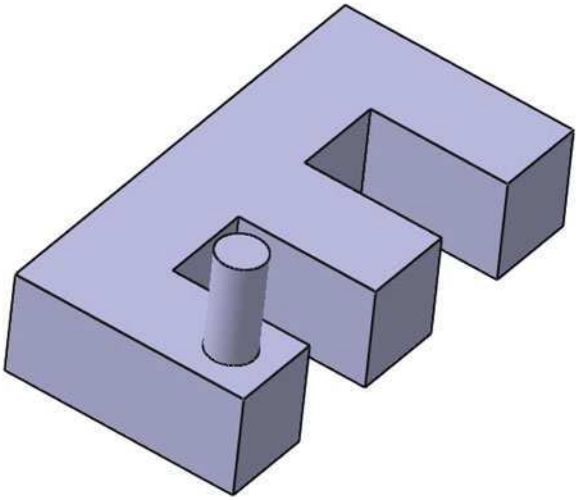
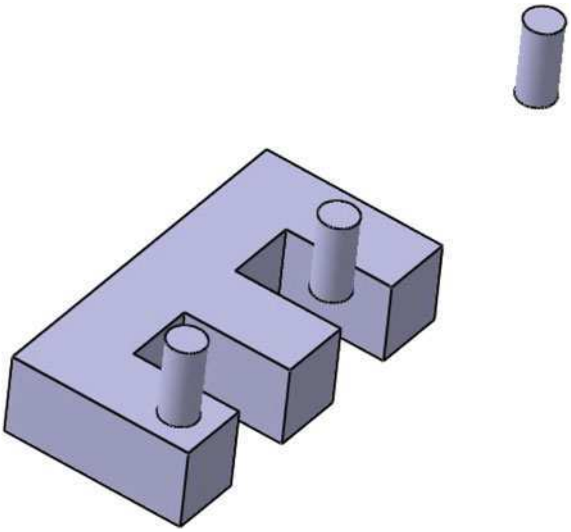


Fig. 4.40 Basic object with extrusion for reflection

Make a circle, radius 5, on face A. The centre should be ten units from each of the closest edges, though so long as it is on the "prong" it is OK. Extrude this shape 20 units, to give the shape in Fig. 4.40.

Now reflect the extruded cylinder in face B and face D to get the shape shown in Fig. 4.41. Does the system warn you that the reflected elements are separate from the body?

Fig. 4.41 Basic object with extrusion reflected in different faces



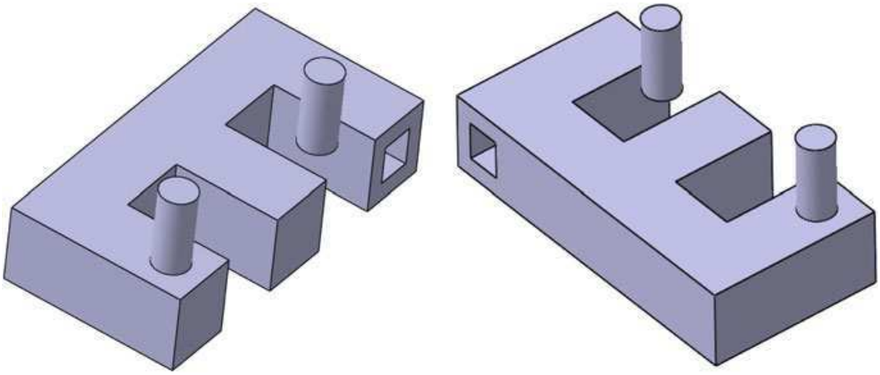


Fig. 4.42 Object with intrusive shape (*left*) and reflected intrusive shape (*right*)

Now make a small extrusion inwards on another of the prongs, as shown on the left of Fig. 4.42. Extrude this inwards 20 mm.

Reflect this negative extrusion in face D and in face C. Does the system warn you? It may be different for a negative extrusion, because this should remove material and when you try reflecting in face D there is no material to remove. The second reflect in face C makes an intrusion, but the depth is not the same, only 10 mm. Although it is an intrusion it is not really a symmetrical intrusion to that from which it was created. This makes the information in the construction history of dubious value for downstream applications. If machining the pocket, say, the pocket does not have depth 20 unless you start from a plane outside the object and machine air.

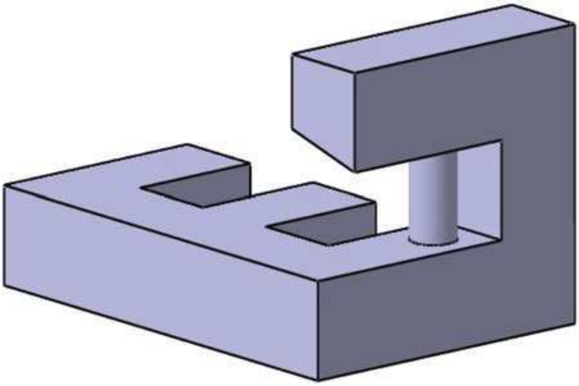


Fig. 4.43 Basic object with extrusion to a limit face

These small exercises are meant to show the effects of using Boolean operations to unite objects or subtract objects. There may be no real check that the operation has a symmetric result, and the same problems with multiple-piece results emerges again.

4.3.8.4 Reflection with Limited Sub-Objects

Try a variant of the above. Make an object like that shown in Fig. 4.43. This is similar to the object in the previous experiment, but there is an arm upwards with a shaped extrusion. Make the same circular shape, as before, but extruded it upwards to the under-surface of the shaped extrusion, as in the figure.

Now reflect the extruded shape in the faces equivalent to face E and face B in Fig. 4.39. What happens? In CATIA version 5, which I used for this exercise, you get the result shown in Fig. 4.44.

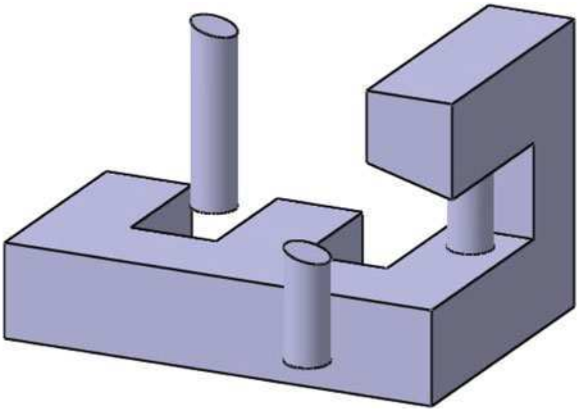


Fig. 4.44 Object with reflected sub-shapes

Reflecting in face E is correct, but reflecting in face B gives a mixed result. The position of the base of the circular extrusion is symmetrical about face B, but the upper limit is strange. Instead of reflecting the limited extrusion with its geometry, the top geometry is recalculated to be "up to the under-surface of the shaped extrusion". This is an interesting result. It is not exactly a symmetrical copy of the original sub-object. It might or might not be what you want and CATIA at least gives a warning.

4.4 Giving a Face a New Surface (SETSURF)

SETSURF was one of the early methods for setting complex geometry into a model and for manipulating geometry in a controlled way. This was developed by Graham Jared in the BUILD system.

Similar operations appear in extrusion, when using the "extrude to …" or "extrude until …" options.

4.4.1 Parameters

Input:

Face in which the new surface is to be set
The new surface

Output:

The changed elements are modified in place.

4.4.2 Potential Errors

Some conditions that should be checked for are:

- Surface does not intersect surrounding geometry.
- Side elements disappear, possibly a warning.

4.4.3 SETSURF Method

What SETSURF does is illustrated in Fig. 4.45. You take an object, shown on the left of the figure, a surface, shown in the middle and replace the surface of a selected face with this, giving the object on the right of the figure. At the same

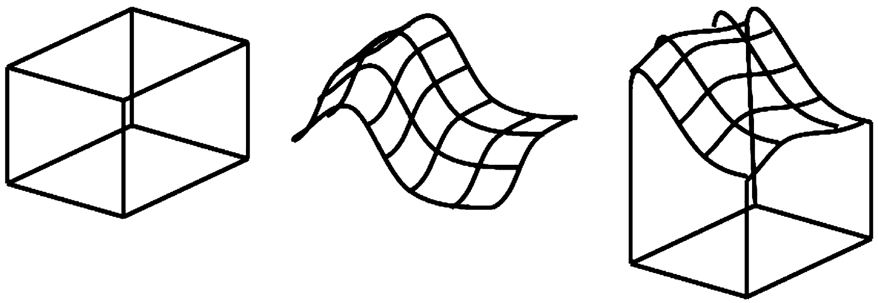


Fig. 4.45 Setting a surface into an object

time, all the geometry surrounding the face being resurfaced has to be adjusted to ensure consistency.

The positions of the vertices are calculated by intersecting the side edges, that is, the edges not in the face, with the new surface. This is shown in Fig. 4.46.

Then, the curves of all the edges around the face are calculated by intersecting the side surfaces with the new surface. Finally the new surface is attached to the face giving the final result (Fig. 4.47).

This is a very simplified explanation. There are lots of special cases which have to be handled. A requirement is that the new surface is at least as big as the surface in which it is being set. Another requirement is that the geometry of the elements surrounding the face in which the surface is being set also intersect the new surface.

All the elements to be changed are found from the face given as input parameter.

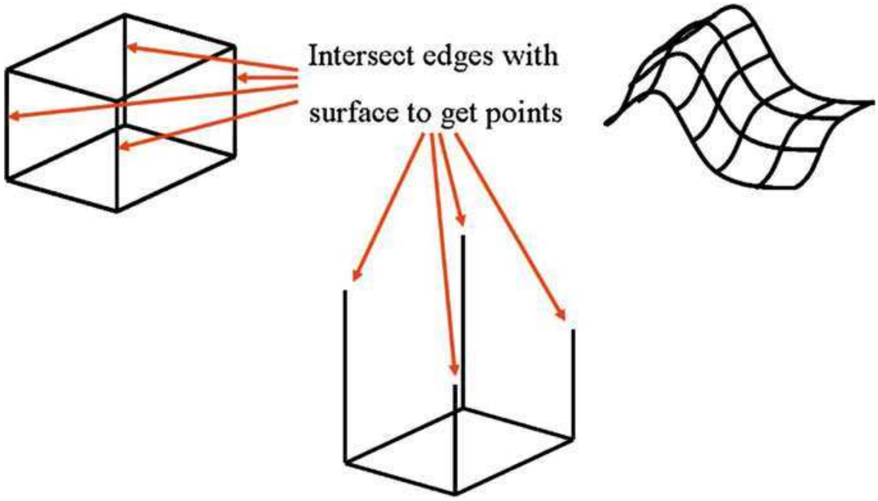


Fig. 4.46 Adjusting vertex positions

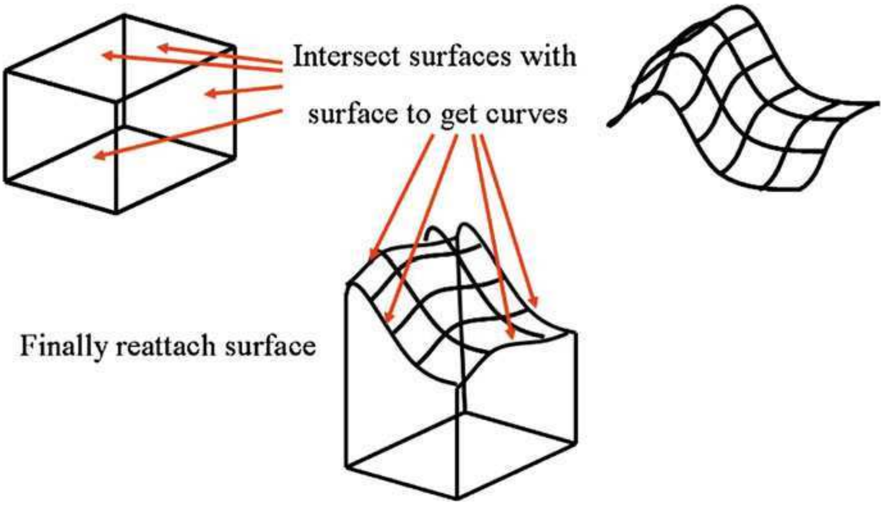


Fig. 4.47 Adjusting side curves

4.4.4 Experiments

4.4.4.1 Disappearing Topology

A simple experiment is to change the surface of a face to that of a neighbouring face to see if the object elements between them disappear. Make an object like that shown on the left of Fig. 4.48. Set the surface of face B into face A and see what happens.

There are four likely results. The first is that the system tells you that it won't perform the operation because a face will disappear. The second is that the system changes the geometry and leaves the topology in a degenerate form. The third is that the system changes the geometry, removes the degenerate face and leaves an edge in the middle. The fourth is that the system makes the change, removes the degenerate face and also removes the edge between the now coplanar faces.

4.4.4.2 Inserting a Doubly Intersected Surface

Make a tall block and insert a cylindrical surface into the face. This is illustrated in Fig. 4.49. The new surface could be interpreted either as adding a cylindrical element to the top of the block (Fig. 4.49 middle), or making a cylindrical cutout (Fig. 4.49 right). But, which of the results is given? What distinguishes the cases shown is the direction of the surface normal. If the cylindrical surface is normal, when the surface normals point away from the cylinder axis, then the result in the middle should be given. If the surface

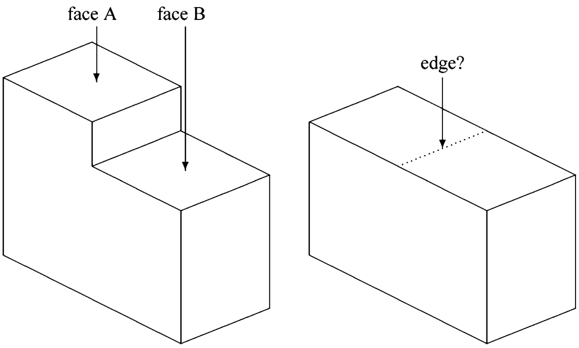


Fig. 4.48 Changing a surface and removing topology

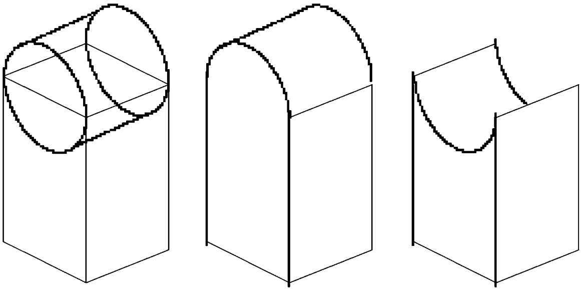


Fig. 4.49 Changing a surface which covers twice

normals point inwards, i.e. towards the axis, then the result on the right should be given. One question is how to specify a negative surface for the operation. If the system allows you to create a surface by extruding a circle then you may have two sides to the cylinder. If you pick the cylindrical surface where you see the inside then the surface may be negative. Picking the outside gives the normal cylinder.

Note that this can be done with a spherical surface as well.

4.4.4.3 Inserting a Triply Intersected Surface

This is harder to arrange than with the doubly intersected surface, but is possible with free-form geometry. The question is whether the CAD system allows this or refuses on the grounds that the change is ambiguous. If the system does allow the operation, then what is the result? Fig. 4.50 shows a block and a kind of S-shaped surface to give a triple intersection result.

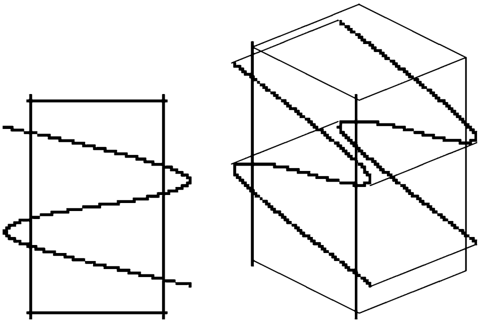


Fig. 4.50 Changing a surface which covers three times

4.4.4.4 Inserting a Non-Covering Surface

This is to check how the CAD system reacts to a critical case. Make a surface patch which does not cover the face for changing the surface, as shown in Fig. 4.51. The system should complain, but make sure that you use a finite surface for the experiment and not an infinite surface, like a plane.

4.5 Tweaking

Tweak is an operation which was present in the BUILD research system and still exists but seems less common in modern CAD systems. Its use is illustrated in the command files in Sect. 12.1.1 where it was used to adjust the geometry of certain faces, like the example shown in Fig. 4.52. Tweaking is an application of the SETSURF operation described in the previous section and was an example of the local operations, where small incremental changes were made based on local

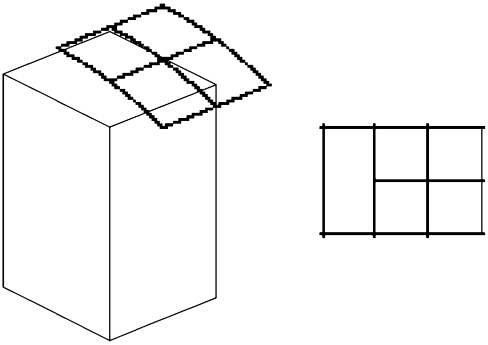


Fig. 4.51 Changing a surface which does not cover the face

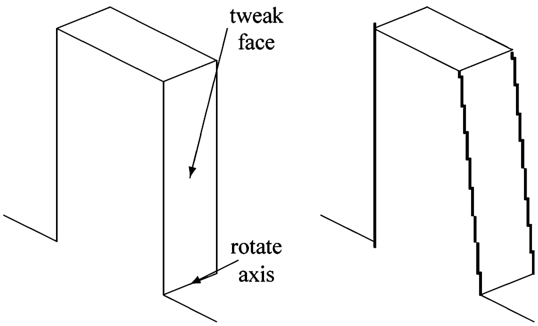


Fig. 4.52 Tweaking a face to produce a sloping element

model neighbourhoods. It is easiest to apply to a face directly, if applied to an edge or vertex then it is necessary to adjust the surrounding faces as well by recalculating their surfaces. In Fig. 4.52 the face marked as the tweak face in the figure on the left is rotated slightly about the rotation axis.

4.5.1 Parameters

Input:

Element to be tweaked
Tweak transformation

Output:

The changed elements are modified in place.

4.5.2 Potential Errors

Some conditions that should be checked for are:

- Invalid tweak type.
- Trying to tweak over-constrained element.

4.5.3 Tweaking Method

The tweak method is just an application of the SETSURF operation where the new surface (or surfaces) is calculated from existing elements. The most straightforward version is face tweaking, where the face surface equation is modified and reset into the face. Edge and vertex tweaking may be difficult because edges and vertices are constrained by more than one surface.

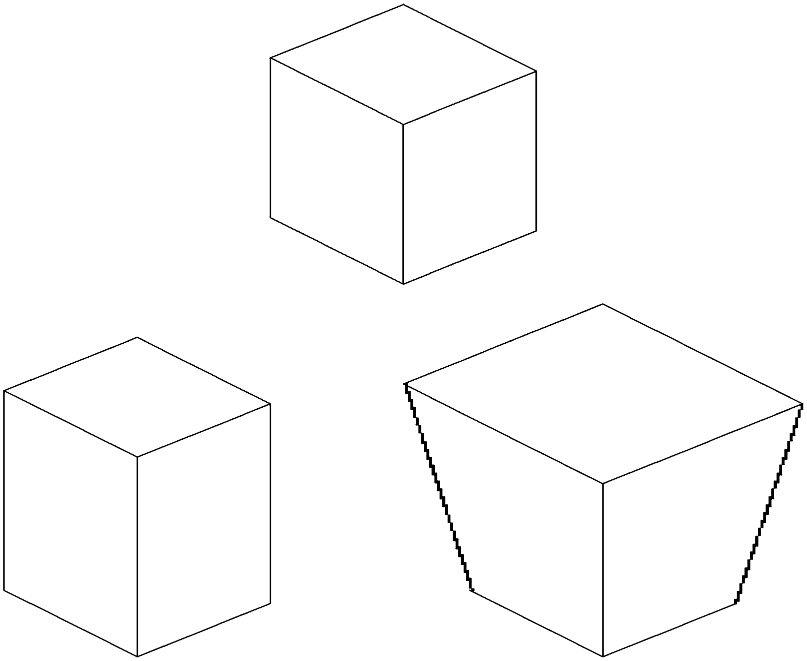


Fig. 4.53 Scaling a face in a cube

The change is specified by a transformation, i.e. a rotation, scaling or translation. At the user interface this may be given explicitly, for the operation the information may be passed as a transformation matrix or as separate transformation elements. The difficulty comes in interpreting the transformation as an action. If, for example, you scale a face, should that mean that the surface equation of the face is scaled or should the elements surrounding the face be scaled? This is illustrated in Fig. 4.53. If the top face of the cube shown at the top of the figure is scaled by 1.5, should that mean that the surface is moved slightly in the Z direction, as shown on the bottom left of the figure, or should it mean that the edges and vertices surrounding the face be scaled, as shown on the bottom right?

It is necessary to experiment with a tool, if one exists in your CAD system, to see what the result is in order to understand how it is implemented. A general rule, though, is to make simple changes rather than complex ones which may lead to unexpected results.

4.5.4 Experiments

As stated at the beginning, you may not find this operation in your CAD system so you may not be able to perform these experiments.

4.5.4.1 Tweaking a Face

Build an object line that shown in Fig. 4.52 and tweak the top face by 15°. For example, as illustrated in Fig. 4.54, make a square 100 × 100 and extrude this 30 units. Draw a rectangle on the top face, 50 × 35 and extrude this upwards 60 units. The rectangle need not be aligned with an edge, it was done this way because of the geometry of the MBB object.

Now tweak the front face of the prong by 15° about its bottom edge (from the view direction shown in the figure). The result should be as in Fig. 4.55. This is intended to be a tweak that works just to illustrate how it is used.

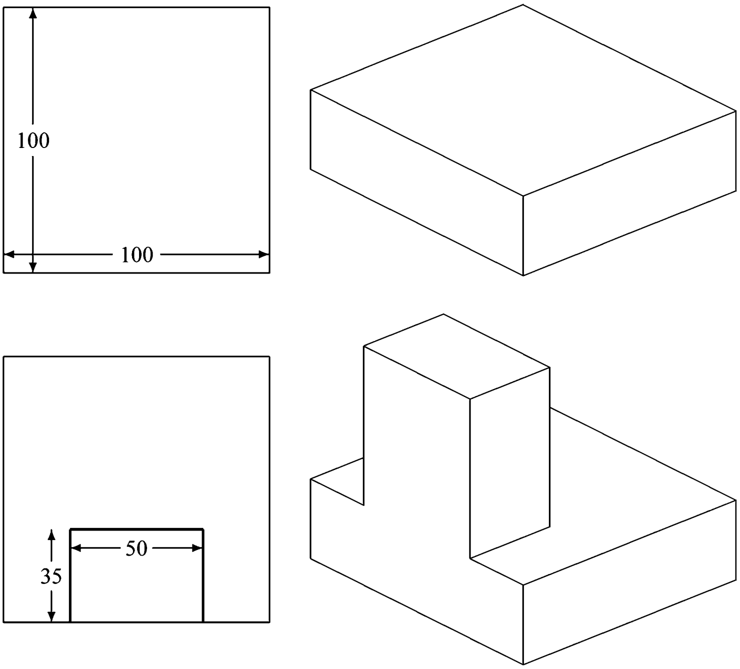
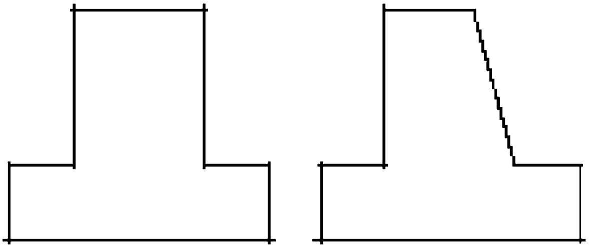


Fig. 4.54 Making an object with an extruded prong for tweaking

Fig. 4.55 Tweaking the prong face



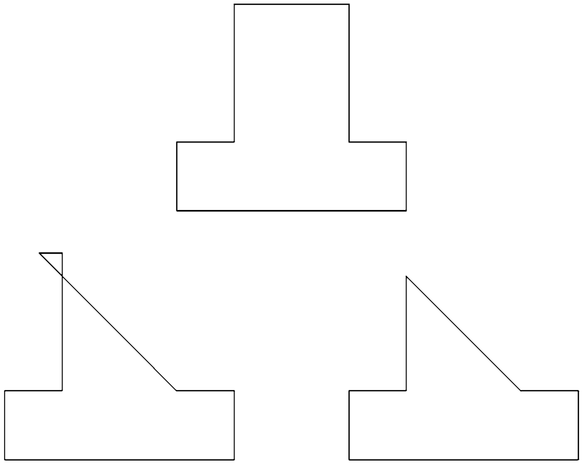


Fig. 4.56 Tweaking the prong face by an excessive amount

4.5.4.2 Tweaking a Face Too Far

Repeat the previous experiment, but this time tweak the face by 45°, as shown in Fig. 4.56.

One possible result is that the system simply refuses to perform the operation, which may be the best solution. Another possibility is that the system simply performs the operation without checking that the result is self-intersecting, as shown on the left of Fig. 4.56. A third possibility is that the system trims the object, as on the right of Fig. 4.56. However, this means that the topology of the object changes.

4.5.4.3 Tweaking a Curved Face

What happens when tweaking a curved face? Make the object shown in Fig. 4.57. This consists of a 100 × 100 square extruded 30 units with a circle, radius 30, drawn on the top face and extruded 60 units. Make sure that the circle is broken into two pieces so that the extruded shape has two side edges.

Now tweak the front curved face 15° about the front bottom edge. What happens? The system might complain that the front bottom edge is curved or it might take the end points as defining an axis. The system might complain that it cannot rotate the surface or it might add extra topology to make it possible to rotate the face.

4.5.4.4 Tweaking a Face with Internal Loops

This experiment is to see what happens if you tweak a face and change the topology of surrounding elements. Make an object like that shown in Fig. 4.58,

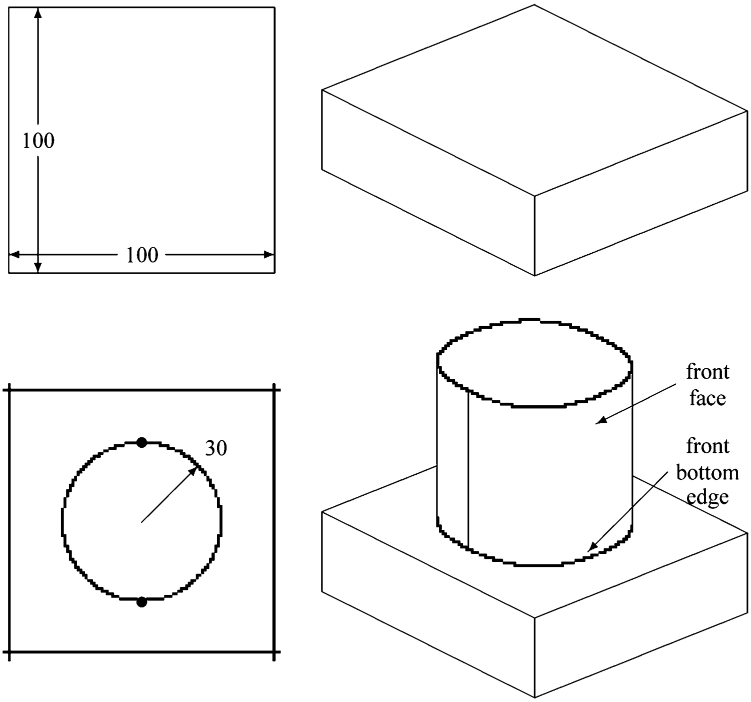


Fig. 4.57 Making an object with a cylindrical prong for tweaking

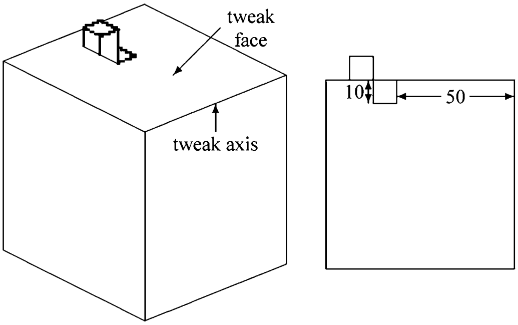


Fig. 4.58 Object with internal loops for tweaking

which is like Jared's well-known feature object. This is basically a 80 × 80 × 80 cube with a 10 × 10 × 10 pocket, 50 units from the front face and a 10 × 10 × 10 boss.

Tweak the face marked "tweak face" by +15° and by −15° about the axis shown in the figure and check what happens. For both these values one or other of the features is swallowed up by the change. The system may refuse to do the operation or may make an arbitrary decision about what to do.

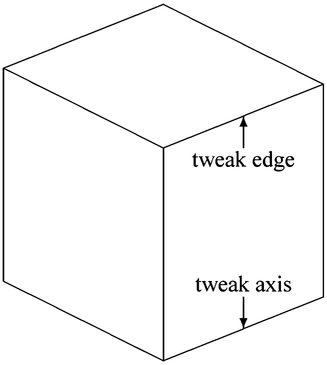


Fig. 4.59 Object with internal loops for tweaking

4.5.4.5 Tweaking an Edge

Face tweaking is probably the most useful operation, but edge and vertex tweaking may also be possible. Try different tweaks on the edge shown in Fig. 4.59. Tweak the edge by moving it by the vector(1, 0, 1), by rotating it ±10° about the axis shown and by scaling it by 1.5 and by 0.75.

You could try similar tests for a curved edge and for a vertex to check what happens with these cases.

4.6 Adding a Draft Angle

Adding a draft angle is an operation closely associated with mould making. From this point of view it is a little strange to include it as part of Computer Aided Design. The reason that it is present may be historical. One of the early local operations in the BUILD system developed at the end of the 1970s was an operation to add a draft angle, as shown in Fig. 4.60, from a paper from 1979. The draft operation was originally based on tweaking and hence is related to this operation.

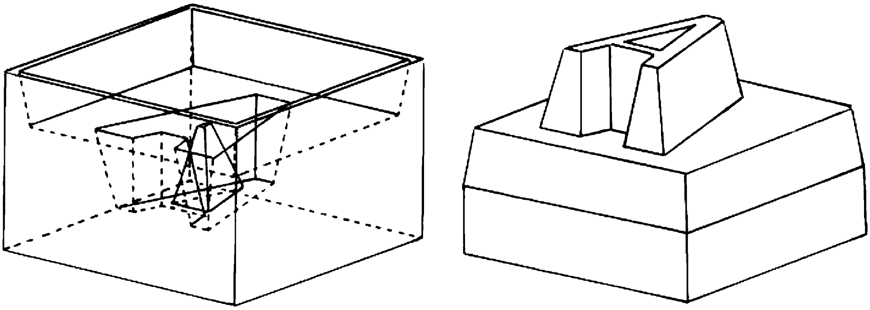


Fig. 4.60 Object with draft angle

4.6.1 Parameters

Input:

Neutral elements (elements which stay unchanged)
Possibly faces to draft (if these are not derived from the neutral element)
Possibly the removal direction
Draft angle

Output:

The changed elements are modified in place.

4.6.2 Potential Errors

Some conditions that should be checked for are:

- Zero draft angle—not an error but silly.
- Possibly modified topology.

4.6.3 Drafting Method

At that time the operation was used as an illustration of the advantages of Boundary Representation compared to CSG, at that time the other main solid modelling method. The operation took a face, now usually called the "static face", stepped round the edges bounding it and tilted the neighbouring faces by given angle, as shown in Fig. 4.61.

This is reasonable enough if all the neighbouring faces are planes, but what happens if the neighbouring face is curved?

The usual solution, at least if the neighbouring faces are cylindrical, such as those created when blending side edges, is to convert these to cones, as illustrated in Fig. 4.62b. Is this correct? If you perform the draft first and then blend the object you get inclined cylindrical faces.

The purpose of pointing this out is to illustrate a difference between computer logic and manufacturing logic.

4.6.4 Choice of Parameters

Setting a draft angle is an example of an operation with a more complex interplay between the parameters. You can set positive or negative angles, choose different elements as static elements, and so on.

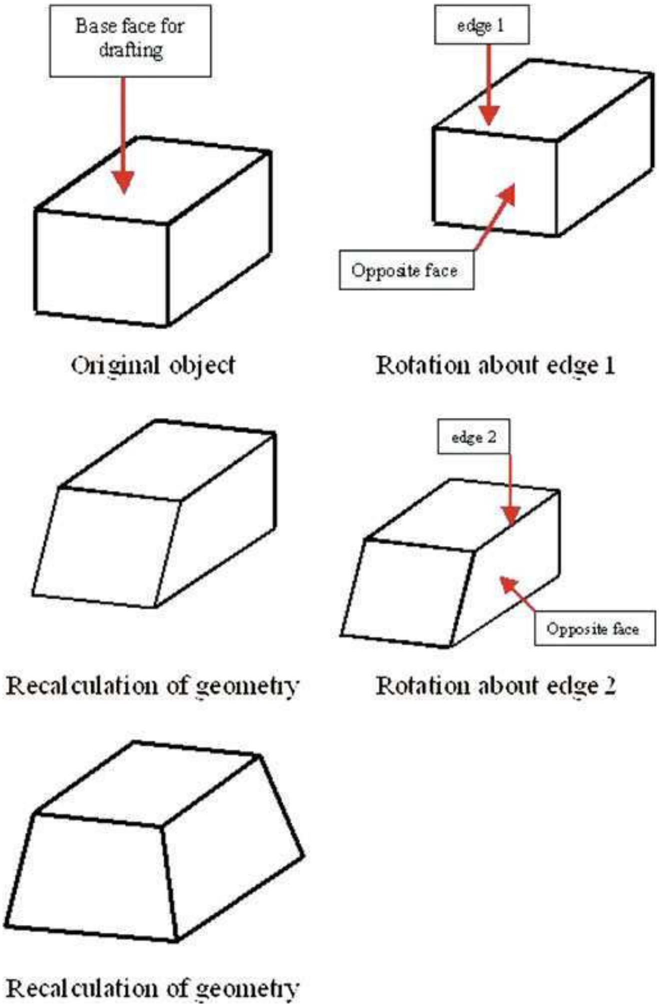


Fig. 4.61 Adding a draft angle to a block

Figure 4.63 illustrates an example of this.

If the top face, from the point of view of the extraction direction, is chosen as the static face then the object will get slightly smaller. However, if the bottom face is chosen the object gets slightly larger. This may be important if the side faces have a tight tolerance. However, if there is no restriction then it is not so important. A slightly large object can be machined down to size, if necessary. However, if this is not necessary then this would be a waste of material. This is not easy to generalise and it is necessary to take this into consideration. At any rate, be aware of this effect.

Fig. 4.62 Adding a draft angle and blends

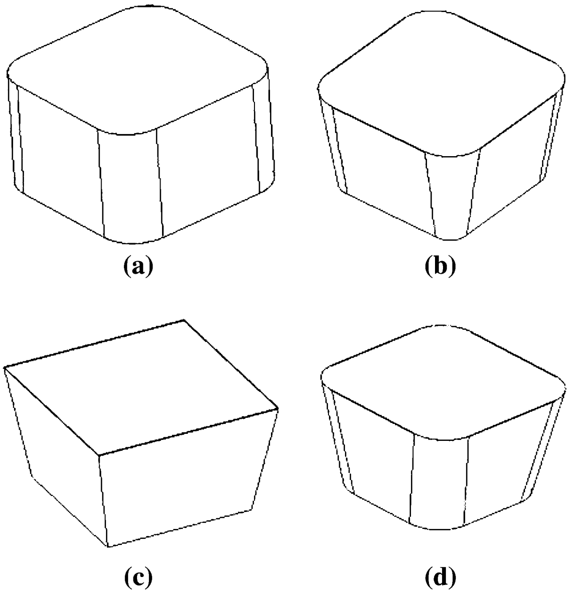
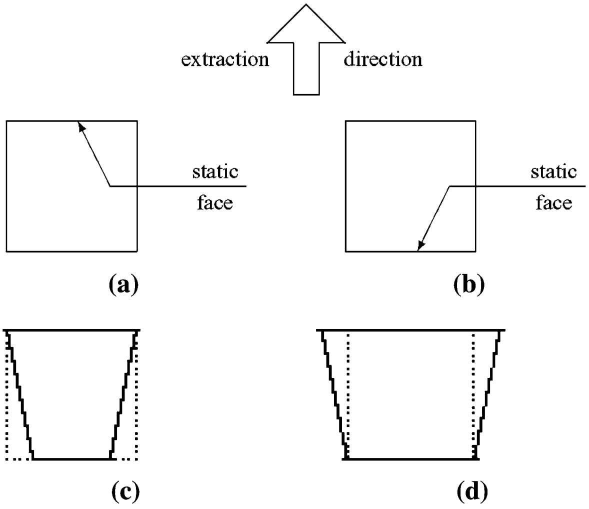


Fig. 4.63 Influence of static face choice



Before applying the final draft angle it may be useful to check with a larger tolerance which makes the change visible to make sure that you get the effect required. Angles can be positive or negative and this may interact with the extrusion direction to give you the opposite of what you want.

4.6.5 Drafting Using Warping

This technique sort-of exists. It exists in the research environment and at least one of the kernel modelling systems (ACIS) has a mechanism of applying “laws”, or rules to model elements. What this means is that you set up a rule to say how the geometry should be modified rather than using model elements directly. This is very useful for modifying facetted objects, where there are no convenient model elements which can be used to guide the change. Figure 4.64 illustrates this.

With facetted objects there are too many elements to allow picking. There is also a great risk of error with picking. Instead it is possible to use a separate plane as a neutral element. Vertex points are then moved inwards, compared with the average normal vector of the facets surrounding the vertex, to a distance which depends on the distance of the vertex from the plane. The object on the left of Fig. 4.64 shows this.

This can also be done with a plane which cuts through the object, as shown on the right of Fig. 4.64.

4.6.6 Adding Draft Angles and Mould-Making

Note, though, that adding a draft angle is only one element of mould-making. Separation lines are not always planar contours. There are also extra elements to

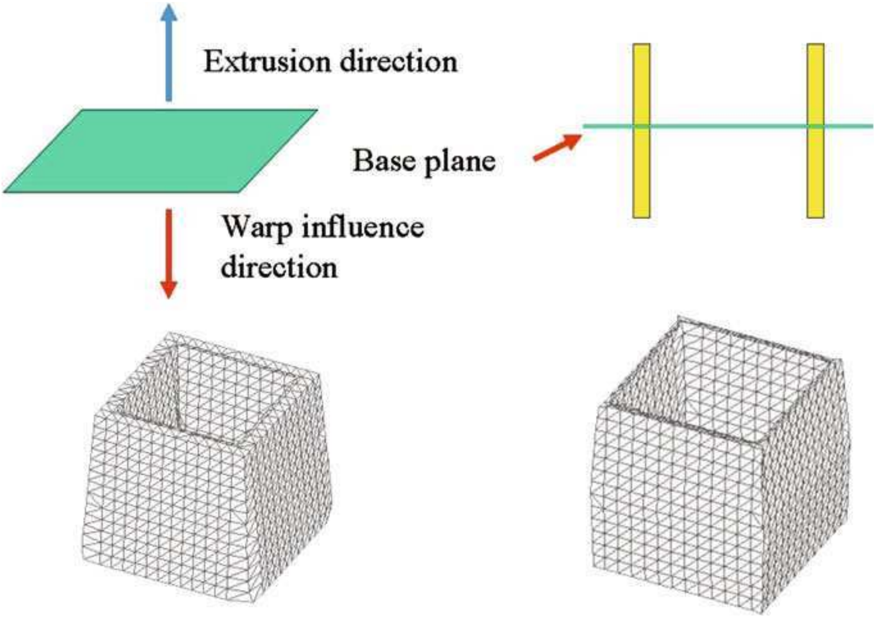


Fig. 4.64 Adding a draft angle using a rule

add, ejector pins and cooling channels for example. Adding a draft angle is an interesting operation, from one point of view, but it is not complete. It remains, in my opinion, something of an oddity as part of design systems.

4.6.7 Experiments

Note that these experiments use a much larger draft angle than would be used in practice. The reason is to illustrate what happens in extreme cases so that you can better appreciate what the operation is doing.

4.6.7.1 Drafting with Volume Calculation

Make a cube 100 × 100 × 100. Set a draft angle of 5° on the side faces using the top face as a static element. Measure the volume with the utility for this, which should be present in the CAD system. Note the volume calculated. Redo the draft using the bottom face as a static element and recalculate the volume. Note the change. Now redo the experiment for a draft angle value of 1°.

4.6.7.2 Drafting with Blends

The next experiment is to try the experiment with blends, illustrated in Fig. 4.62. Make a cube 100 × 100 × 100. Blend the side edges with a radius of 10, say. Now add a draft angle of 5° and not the shape. Make a second cube 100 × 100 × 100, add the draft angle and then blend the side edges and note the difference with the first object.

4.6.7.3 Drafting with Undercut

Make a cube 100 × 100 × 100 mm. On the bottom face, make a square 80 × 80 mm, that is, with a 10 mm border all round. Extrude this 100 mm. The resulting object is shown in Fig. 4.65.

Using the top face as a neutral face, add a draft angle to the side faces of the large cube, marked “face 0”, “face 1”, and the two hidden faces at the back. Add a draft angle so that the size of the block decreases downwards. Try it with angles of 5, 7.5 and 15°.

With 5°, the offset at the bottom of the large block is about 8.716 mm, so the 10 mm border is not violated. With 7.5° the offset is about 13.053 mm, so the offset faces are inside the lower extrusion. Finally, with 15°, the offset is about 25.882 mm, clearly inside the lower extrusion.

There are, at least, two interpretations of what to do, shown in Figs. 4.66 and 4.67.

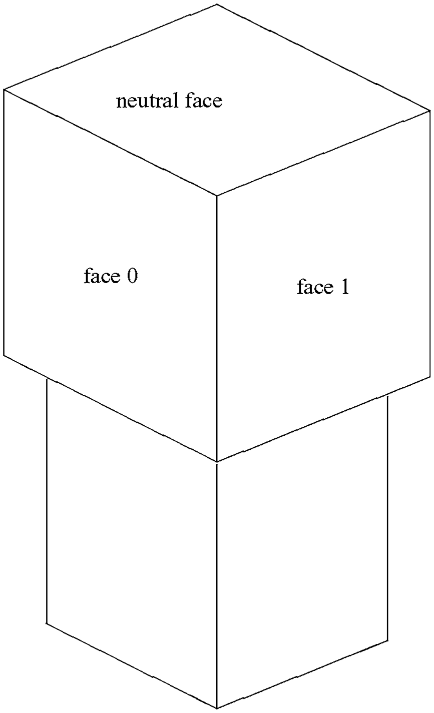


Fig. 4.65 Draft undercut experimental object

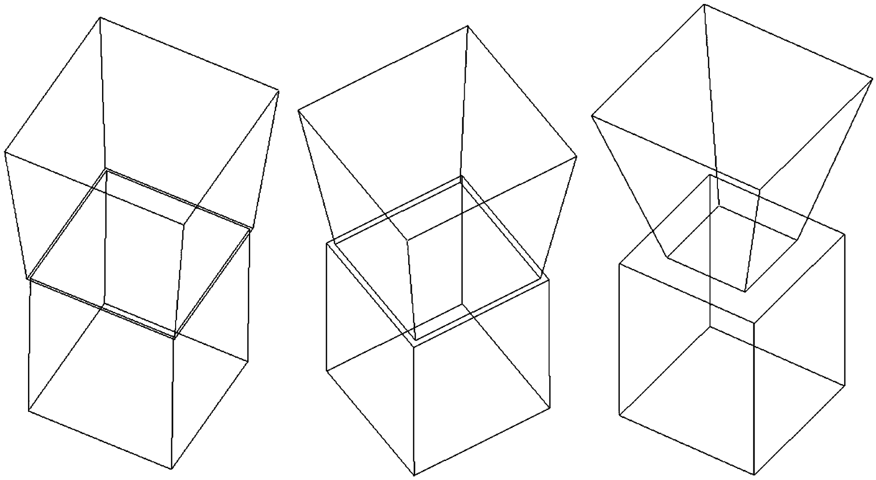


Fig. 4.66 Adding a draft angle across a step. Series 1

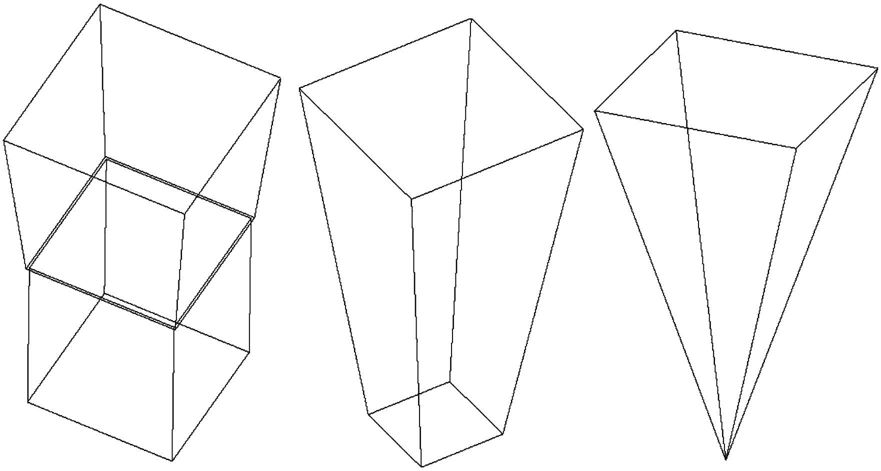


Fig. 4.67 Adding a draft angle across a step. Series 2

In Fig. 4.66 only the top faces are drafted, necessitating a changed orientation of the lower face.

In Fig. 4.67 only the top faces are drafted with an angle of 5°, while the lower faces are also drafted when the offset is so large that the borders are violated.

Which do you think is correct?

4.6.7.4 Draft Undercutting Again

Now repeat the experiment, but with the second extrusion not centred on the original object. Make a rectangle 120 × 110 mm. Extrude this 100 mm and, on the bottom face, create a 80 × 80 mm square, offset 10 mm in both *X* and *Y* from one corner. See Fig. 4.68.

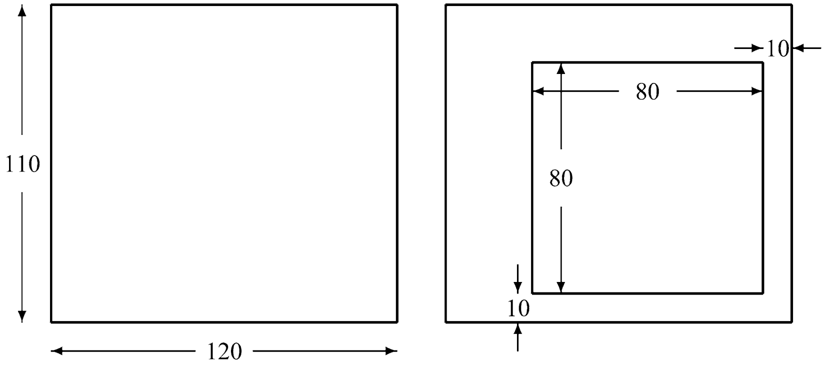


Fig. 4.68 Second draft undercut experimental object

Extrude this square shape 100 mm. Now repeat the previous experiment with draft angles of 5, 7.5 and 15°. If you got the second series, in Fig. 4.67, look how the draft is propagated to the different parts of the object.

Repeat this experiment, but instead of extruding the square shape to create a positive addition, extrude it back through the object to make an object with a hole.

4.7 Chamfering

Chamfering is an operation which replaces sharp edges by small planar faces at an angle to the faces adjacent to the original edge. It may be better to explain that with some examples as the previous sentence, although true, is a little opaque.

This might be done to remove a sharp edge, to add material at an extrusion or for aesthetic reasons, for example.

4.7.1 Parameters

The parameters are illustrated in Fig. 4.69.

Input:

A face, edge or vertex
A depth value
Optionally an angle

Output:

Object shape modifications done in place.

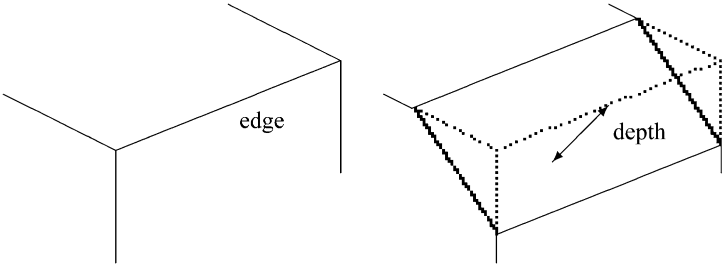


Fig. 4.69 Chamfering parameters

Normally the chamfer angle is symmetric to the two faces meeting at the edge. The optional angle can be used to vary this.

4.7.2 Potential Errors

Some conditions that should be checked for are:

- NULL face, edge or vertex.
- Zero or negative depth.
- Disappearing elements surrounding the edge or vertex being chamfered.

4.7.3 Chamfer Algorithm

The chamfer operation is applied to edges or vertices. If a face is given then this is interpreted as meaning that all edges surrounding the face should be chamfered. If a vertex is given, then this may mean that only the vertex is chamfered. Alternatively it might mean that all edges at the vertex and the vertex itself are chamfered.

The chamfering direction depends on whether the edge (or vertex) being chamfered is "convex" or "concave". The edge marked "e" in Fig. 4.70a is convex, and the chamfer operation removes material, as shown in Fig. 4.70c.

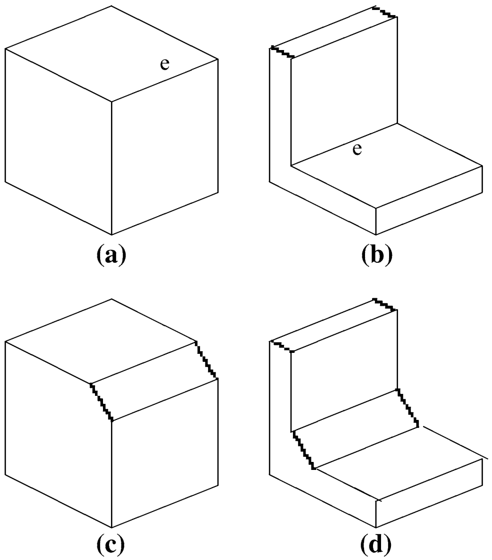


Fig. 4.70 Chamfering convex and concave edges

In Fig. 4.70b the edge marked “e” is concave and the chamfer operation adds material, as shown in Fig. 4.70d.

There are two ways that an implementer might create this operation: (1) with local changes; or (2) as a volume creation step and a Boolean operation.

For the local change version the edges around the start and end vertices of the edge are cut at the appropriate positions and these are joined together creating a face at the appropriate depth, as shown in Fig. 4.71.

The surface of this new face is calculated from the shape of the original edge. If the edge is straight then this surface is planar. If the edge is circular then the surface will be a cone, for example. If the edges at the vertex do not cut this chamfer surface then “eaves” are created in the algorithm devised by Jared and reported in [3].

For the Boolean version, the easiest way is to do something similar but, instead of modifying the topology directly, build a small shaped volume to subtract from the original part. Figure 4.72. In the middle is the original object model. The left hand image shows the crossing points where the chamfer plane cuts the adjacent

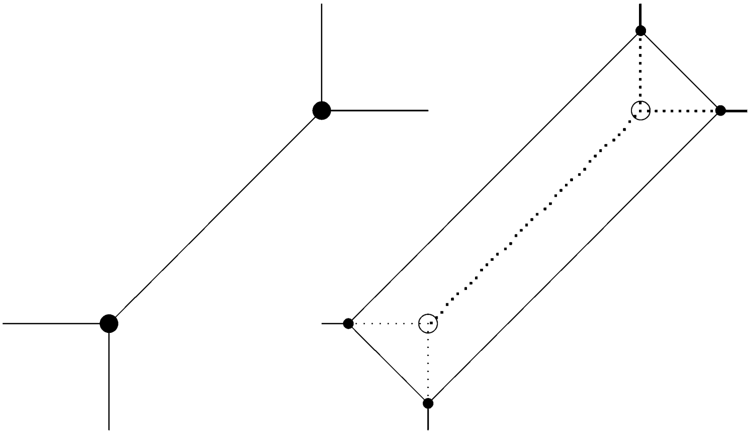


Fig. 4.71 Local modification chamfering

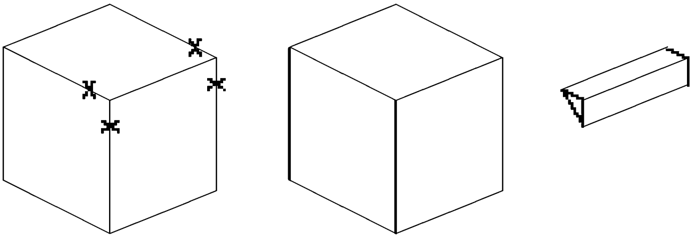


Fig. 4.72 Creating a tool body for Boolean operation chamfering

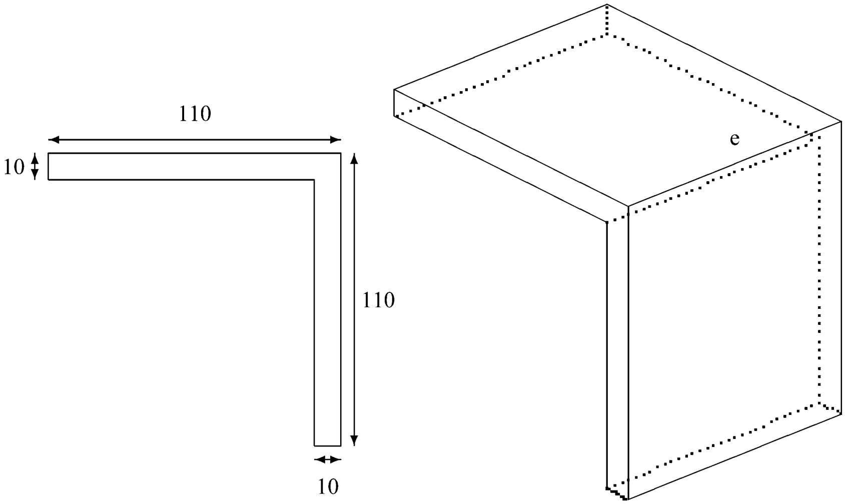


Fig. 4.73 L-shape

edges and the resulting body, the tool body, which is subtract from the original body is shown on the right.

The two methods could be merged, to make a local change and then to modify the object with a special kind of Boolean operation. This is an implementation detail, though.

To see which method is used in your CAD system, try an experiment.

Create an object like that shown on the left of Fig. 4.73.

Extrude the object a distance of 100 to create an L-shaped block. Now chamfer the edge marked "e" in the object on the right of Fig. 4.73.

Two of the results that might be expected are shown in Fig. 4.74. If the operation has been implemented as a local operation you get the result on the left. If the operation has been implemented using a Boolean operation you get the result on the right.

Obviously the result on the right is correct and the one on the left is wrong.

Or is it?

Actually, no. The one on the right may be topologically and geometrically correct but I somehow doubt that a manufacturer will thank you if the object being made suddenly separates into pieces. The implementation on the left does, at least mean that an object check may reveal the problem. The result on the right would not reveal an error. Of course, both these operations can be modified in modern systems by changing the parameters and reapplying them, if the fault is noticed. In the experiment above the CAD system should, at the very least, warn the user that there is a problem. If it didn't, then it is up to you to check the results. Note, as well, that in CATIA v5 you get different results if the L-shaped part is extruded in a straight line or in a circle. Try it on your system.

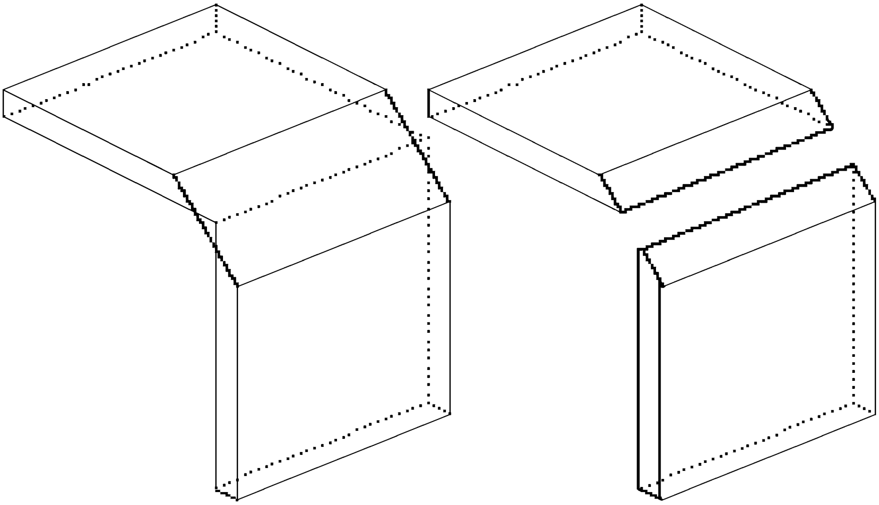


Fig. 4.74 Chamfered L-shaped block

4.7.4 Chamfered Edges at a Common Vertex

For both chamfering and blending there are potential problems with multiple modified edges meeting at a vertex. A traditional way of handling this was a two stage approach. In the first step, a chamfer or blend depth value was attached to each edge to be chamfered. In the second step the operation made the topological changes necessary for the given depth values.

From an implementation point of view this is a good approach because the changes can be made consistently. This was the method used in research system and in I-DEAS, for example. However, from the user point of view it was less successful because, at least at EPFL, students did not understand that, when they assigned the depth value, there was no visible change.

From a user point of view it is useful to have a visual effect at once, but this causes other complications. If you chamfer individual edges at a vertex one after the other then the effect is not the same as if you chamfer all edges at the same time.

Figure 4.75 shows the difference between chamfering just edges and chamfering the vertex as well.

On the left of the figure is the original arrangement, in the middle the result of just chamfering the edges and on the right the result if the vertex is also chamfered.

Another problem of chamfering edges one-by-one, especially with different parameters, instead of all at once is that, after the first chamfer, there may be short edges which cause subsequent operations to fail because the next chamfer misses the short edge.

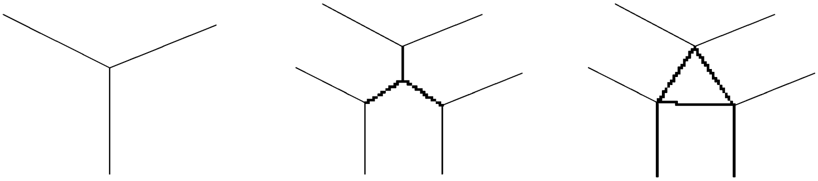


Fig. 4.75 Chamfered edges and vertices

4.7.5 Chamfering Non-Manifold Edges

Verboten!

Non-manifold edges are normally not chamfered, they are rejected when selected. If a face to be chamfered contains non-manifold edges then these are not included.

This is a pity.

It is also unnecessary to refuse these.

The problem is that, at non-manifold edges, more than two faces come together and, with the currently popular representation method, the meaning is ambiguous. This is illustrated in Fig. 4.76.

Chamfering the non-manifold edge in the object on the left of the figure could lead to either of the results on the right. The object on the top right interprets the non-manifold edge as where two objects touch but just miss. The object on the bottom right interprets it as where two objects touch and just join. There is no obvious answer because there are many arrangements which might arise.

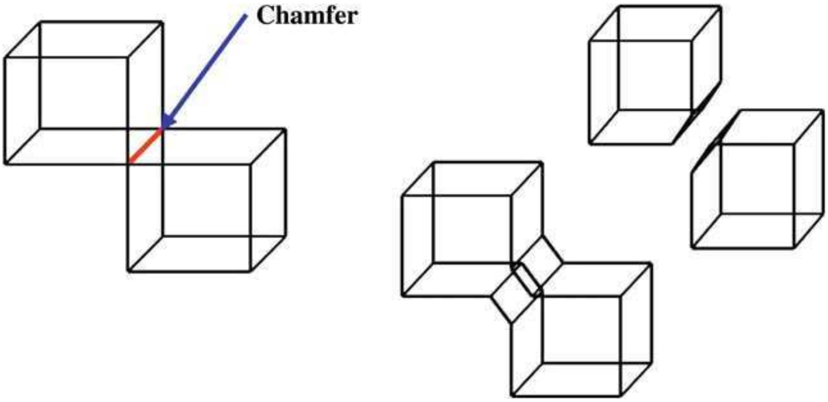


Fig. 4.76 Chamfering a non-manifold edge

As will be described later, in Chap. 6, there are links between the edge and loop of each face meeting at the edge. For a valid object there has to be an even number of links arranged in sequence around the edge. In order to handle the results it is necessary to pair up the links and assign them to new edges. Then the original edge and each new edge can be chamfered. Depending on how the links are paired you get the two cases of bodies just missing each other or just touching. These could be resolved by asking the user to define which is the desired result. It needs user intervention but can be resolved.

An unlikely option is that the user may want to mix the missing/touching cases if there are more than four links round the vertex, but this is not so likely. It should, then, be possible to handle chamfering of non-manifold edges with a single supplementary piece of information from the user. At the moment CAD systems do not do this, but it may happen in the future.

4.7.6 Experiments

The following experiments are similar for both chamfering and blending, which have several modelling similarities but use different geometry.

4.7.6.1 Simple Check

Check which elements your system allows you to chamfer. Just edges? Or are vertices and faces allowed?

4.7.6.2 Meeting Chamfers

Make the objects shown in Fig. 4.77. The vertices marked with the dot in the objects have zero (Fig. 4.77a), one (Fig. 4.77b), two (Fig. 4.77c) and three (Fig. 4.77d) concave edges, respectively.

The exact geometry is not important, it is the arrangement of the edges which should be considered.

However, make the object in Fig. 4.77a 100 × 100 × 100 cube.

The object in Fig. 4.77b can be made as a 100 × 100 × 100 cube with a 80 × 100 × 80 cutout.

The third object, in Fig. 4.77c could be a 100 × 100 × 15 base with an 80 × 80 × 85 extrusion.

Finally, the object in Fig. 4.77d can be made as a 100 × 100 × 100 cube with a 85 × 85 × 85 cutout.

Chamfer the edges at the marked vertices with chamfer depths of five and look at the results. You should examine how the system arranges the topology at the vertex.

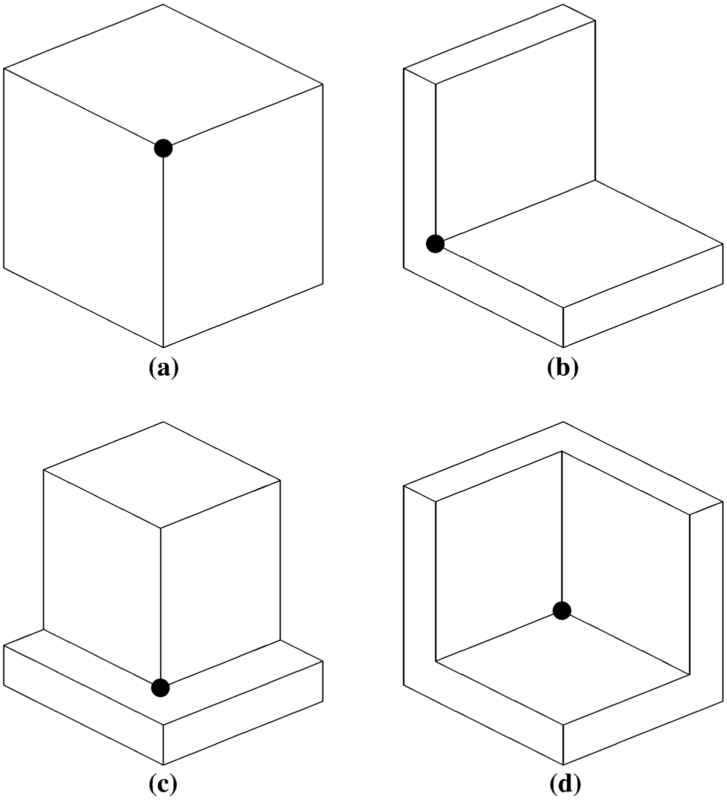


Fig. 4.77 Objects for meeting chamfers experiment

Chamfer the edges with depth 5. If your CAD system allows, do this twice, once all the edges at the same time and the second time one edge at a time and check whether they are different. Has the vertex also been chamfered?

Depending on the algorithm used you might get different results, and it is important to know this because when you chamfer a real object. Real chamfers with small depth are less visible and it is important to know what the CAD system produces so that you can tell whether to apply the operation all at once or separately. In general you should try and chamfer as many edges as possible at the same time.

Now try the experiment with different depths for the edges, with depths of 5, 10 and 15, say. Again, do this for each edge separately and then for all edges at once. Try it once for individual edges with depth order 5, 10, 15, and once giving the depths in the reverse order, 15, 10, 5. You should be able to do this using history tree modification so that you don't have to make new objects always.

4.7.6.3 Thin Part Cutting

If you didn't try the experiment proposed earlier, in Figs. 4.73 and 4.74, then do it. You should check what your system does for these cases. Try it for a thin shape with linear extrusion and circular extrusion to see if they do the same things. At least one system, CATIA v5, does different things for these two cases depending on whether the circular extrusion is through 180 or 360°.

4.8 Blending

Blending is sometimes called filleting, or rounding. The term blending is used here.

Blending is similar to chamfering, but the geometry is different. Blending and chamfering could almost be dealt with in the same section, but are not here because it would get complicated to distinguish the cases. However, if you have already read the section on chamfering you may have a sense of déjà vu when reading about blending.

4.8.1 Parameters

Input:

Face, edge or vertex to be blended
Blend radius, or radii if non-uniform blending is required
Optional set-back offset

Output:

Object modifications are done in place

4.8.2 Potential Errors

Some conditions that should be checked for are:

- Zero or negative radii. (One zero radius may be allowable for variable radius blending.)
- Too complex blend geometry.
- Partially disappearing blends.
- Blend radius too large so that start or end topology disappears.

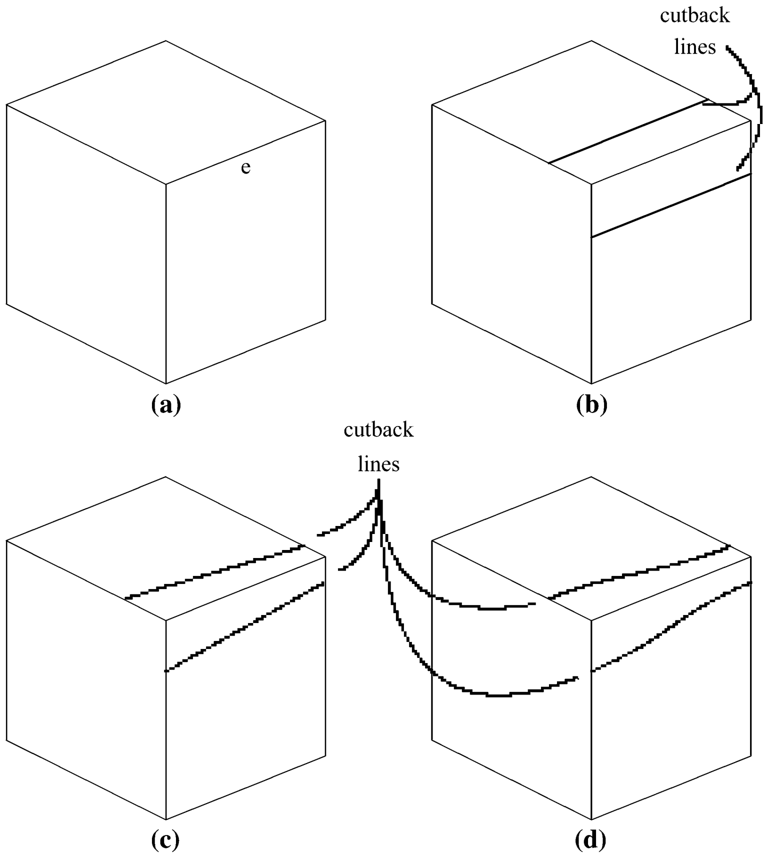


Fig. 4.78 Cut-back edges

4.8.3 Blending Algorithm

Note that there are different types of blending. The common one is "constant radius blending", which can be thought of as a portion of the surface of a ball, or sphere, rolling along between two given surfaces. Another important type of blending is "variable radius blending", where the blend radius varies from one given value to another given value.

The general method is to create two cutback curves on the faces on either side of the edge being blended, delete the original edge and then create a new surface between the cutback curves. Figure 4.78 shows some examples of cutback curves.

Figure 4.78a shows the original object, with the edge to be blended marked "e". In Fig. 4.78b the cutback lines are for constant radius blending. Figure 4.78c shows the cutback for variable radius with constant change. Figure 4.78d shows the cutback lines for variable radius blending with variable change.

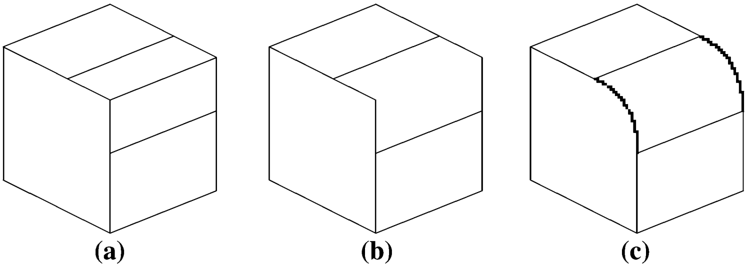


Fig. 4.79 Blend sequence

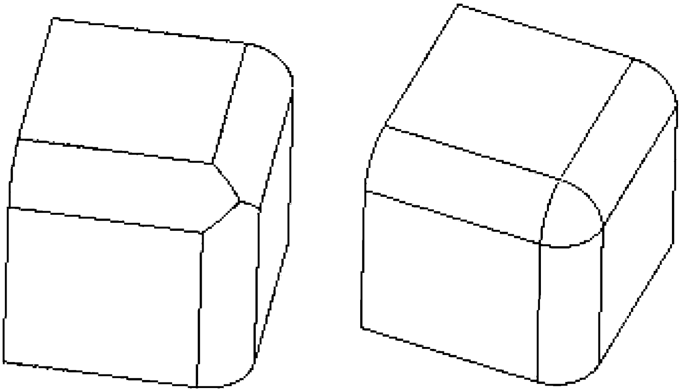


Fig. 4.80 Blending edges meeting at a vertex

Determining the cutback curves for general surfaces can be complex. For numerical, or sculptured surfaces, this may be done by calculating a series of points on the curve and fitting a curve to these points. The general condition is that the points on the curve are where a sphere, of given radius, touches the two surfaces and is tangent continuous to them.

The sequence for blending a single edge is shown in Fig. 4.79.

Once the cutback edges have been created (Fig. 4.79a) the edge is deleted (Fig. 4.79b) and the new blend surface and end curves added (Fig. 4.79c).

4.8.4 Blending Edges at a Common Vertex

As with chamfering, edges meeting at a vertex should be blended at the same time. Figure 4.80 shows what happens in the two cases. If you do not blend the

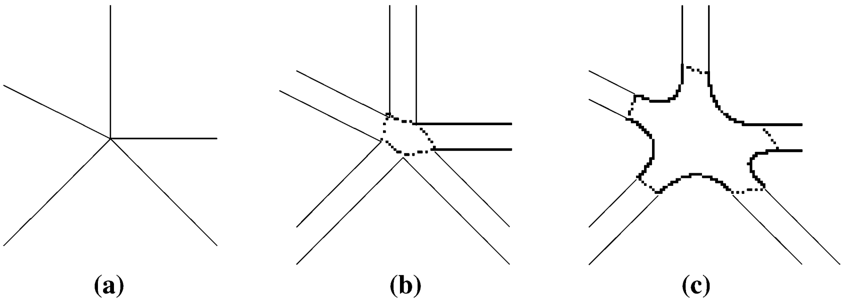


Fig. 4.81 Vertex blend set-back

edges at the same time you get the figure on the left, if these are blended at the same time you get the figure on the right.

There is an extra parameter that some systems provide for blending vertices and that is the "set-back" parameter. With different set-back values the vertex blend starts further back and you get different rounding quality of the vertex blend. Figure 4.81 shows roughly what happens with set-back curves.

The original vertex might look as in Fig. 4.81a. The result of simple vertex blending is shown in Fig. 4.81b, with the dotted edges marking the vertex blend boundary. A vertex blend with set-backs is shown in Fig. 4.81c.

What the set-back does is to move the boundary of the edge blend back and replace it with the surfaces corresponding to the vertex blend. This sets up extra constraints on the geometry of the vertex blend and hence changes the form of the vertex blend. This is a mechanism for allowing users more control over the shape of the vertex blend without necessitating them to manipulate control points directly.

There is some variation over the parameters of the set-back. Obviously the set-back cannot diminish the size of the blend radius, it must move the parameters back. The setback distance could be consider as simply a positive value, determining how far back from the blend boundary the setback starts. However, some systems interpret the set-back distance as the distance from the vertex, and hence it should be not less that the blend radius. Check this.

4.8.5 Blending Non-Manifold Edges

The same comments apply to blending non-manifold edges as were made for chamfering non-manifold edges. To handle these it is necessary to convert the non-manifold edge as a set of simple edges which are then blended individually. This is explained in the section on chamfering, so will not be repeated here.

4.8.6 *Experiments*

4.8.6.1 Meeting Blends Experiment

This is similar to the experiment for chamfers which meet, except that the geometry is different. Make the figures shown in Fig. 4.82 and blend them with blend radius 5.

Perform the same tests, that is, blend the edges all together and then separately, using the construction history to repeat the task. Also try blending the edges with different radii, say 5, 10 and 15. As before, do this for all edges together and then singly, noting where errors occur.

Experiment, also, with the set-back mechanism as described earlier.

4.8.6.2 Complex Geometry Experiment

A classic figure (which I did not invent) for showing blending was shown in Fig. 2.36 and is shown in Fig. 4.83.

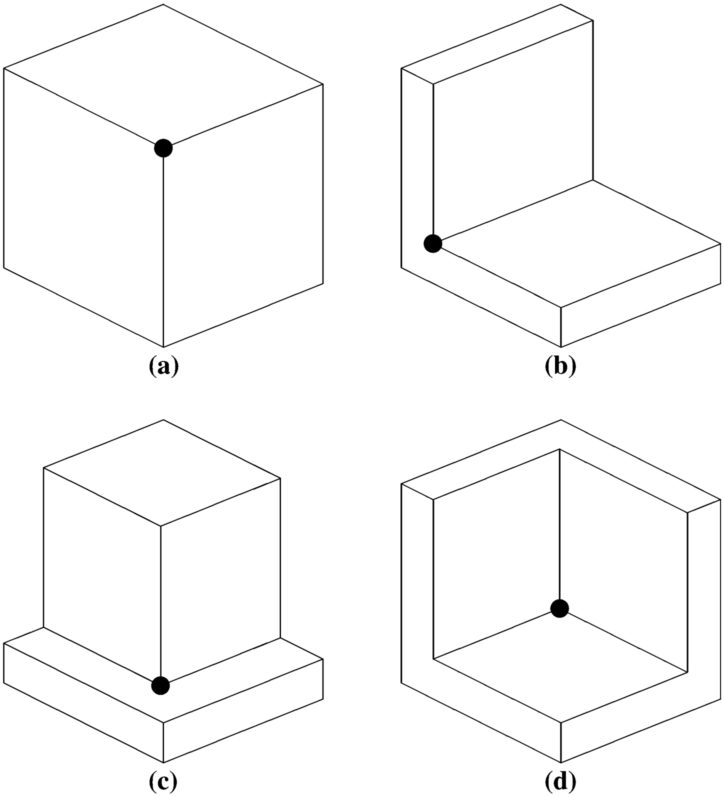


Fig. 4.82 Objects for meeting blends experiment

Make an L-shaped figure, as shown in Fig. 4.84a. Make the long sides 100 and the short sides 50, for example. Extrude the figure upwards 50 mm. On the top face of the extrusion make a 50 × 50 mm square, with two sides common to the extruded figure and a corner touching the concave corner of the L-shape, as in Fig. 4.84.

Extrude the square shape upwards 50 mm to complete the work object. Note, the exact dimensions are not too important, it is the arrangement of edges around the vertex marked with a dot in Fig. 4.83. There are six edges, alternatively concave, convex, concave, convex, concave, convex, marked e0–e5 in the figure.

If you blend all these edges, with radius 5, for example, you get an interesting surface which replaces the vertex. Even though the geometry of the original figure is planar, and the blend surfaces are all cylindrical, the surface where they meet is a complex surface. It looks like a six-sided patch but is represented as

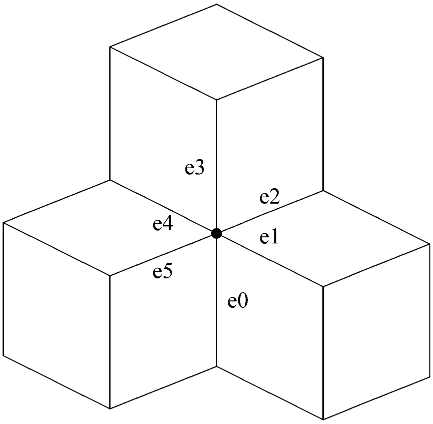


Fig. 4.83 Blending which creates complex geometry

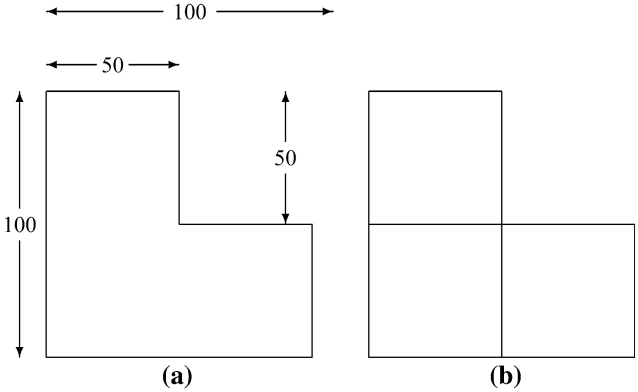


Fig. 4.84 Blending which creates complex geometry

six four-sided patches using a subdivision scheme which will be described in the next chapter.

Note that with CATIA v5 there is a strange variant. You get different results depending on the order in which you select the edges, even though they are all selected at the same time. If you select in the order e0, e1, e2, e3, e4, e5 you get one result, if you select in the order e0, e2, e4, e1, e3, e5 you get a different result. CATIA does not blend the common vertex by default and, apparently, the edges are blended one after the other. Vertex blending is an option and, if blended, you get the result with the six-sided complex surface area mentioned above.

4.9 Shelling an Object

The operation to shell a part is to create thin-walled objects from solids. As described here, the operation creates a sheet object and then adds thickness, so is related to the operation described in Sect. 4.11. The operation came about, possibly originally, as a result of how to design parts in a refrigerator. There was a desire to design a box containing electrical components as a solid for the overall design and then convert it to a thin-walled object as part of the detailing process.

4.9.1 Parameters

Input:

Body to be shelled
Outer thickness
Inner thickness

Output:

Object modifications are done in place

4.9.2 Potential Errors

Some conditions that should be checked for are:

- Sheet object given. Sheet object thickening is taken here as a separate operation.
- Wire-object given.
- Inner or outer thickness is negative.
- Inner and outer thicknesses are both zero.

4.9.3 The Shelling Algorithm

The original shelling algorithm was quite simple. It can be summarised thus:

1. Copy the object.
2. Negate the copy.
3. Merge the two objects with the negated copy as a separate shell in the original object.
4. Offset the geometry.
5. Check for collisions.

The first step is familiar from Sect. 4.3 on reflection or symmetry.

The negation stage is just to turn the copy "inside-out", to make it an object-shaped cavity in a universe full of material. This is done by reversing all the surface normals and exchanging the left- and right-loop pointers of each edge.

The third step, to merge the original and negated copy, is a simple data-structure manipulation. The negated object becomes a new shell in the original body, since there are no faces common between the two objects. There may be an option to remove some faces in the shell object before thickening.

4.9.3.1 Offset Geometry

The first complicated step is creating the offset geometry for the object. Several common surfaces have exact offsets but some need to be converted before offsetting. Planes, cylinders, spheres and torii can be readily offset, which are the common analytic surfaces, any others can be converted to numerical surfaces and offset.

Once the offset surfaces have been created the offset curves for the edges can be recalculated as intersections of the offset surfaces. Similarly, the positions of the vertices can be calculated from the offset edge curves, or directly from the offset surfaces.

In the normal CAD system, the offset geometry does not give an exact offset of the object. This is illustrated in Fig. 4.85.

The original shape is shown in Fig. 4.85a. The normal offset shape is shown in Fig. 4.85b. The real offset shape is shown in Fig. 4.85c. However, the result you get from shelling, that is the shape shown in Fig. 4.85b, is probably the correct result, even though it is not technically the offset. It is more realistic to produce this result and then let the user decide which edges should be blended. The choice of which to produce could, though, be an option.

4.9.3.2 Checking for Collisions

In general it is best to use this operation with small offsets. With large offsets there is a risk of collision between the offset parts of the object. However, it is not

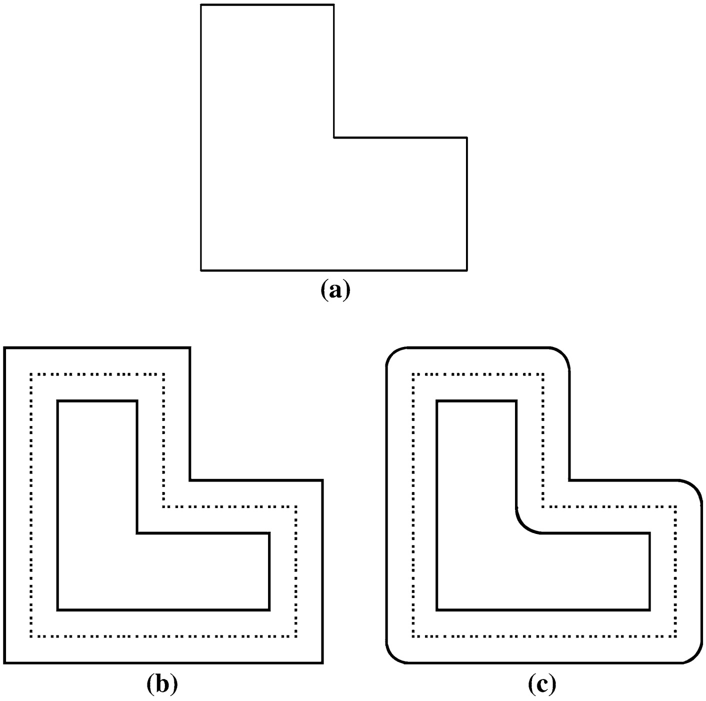


Fig. 4.85 Offsetting the geometry

possible to exclude the use of this operation with large offsets, so it is necessary to have a collision checking step to avoid self-intersecting operations.

Collision checking is an operation similar to the Boolean operations. There are two variants:

1. Checking all internal faces against all internal faces and all external faces against all external faces.
2. Checking all faces against all faces.

Figure 4.86 illustrates the difference between the two. The shape in Fig. 4.86a is a 100 × 50 mm rectangle with a 20 × 45 cutout.

The shelling operation is applied with an internal offset of 10 mm and an external offset of 0 mm.

In the first type of check, the internal faces are checked only against internal faces and you get an object such as that shown in Fig. 4.86b. If all faces are checked against all faces then you would expect a result such as that shown in Fig. 4.86c.

Which you consider correct is a matter of opinion, there are advantages with both of these. The example is just an illustration of two variants you might expect from a CAD system. It is, however, preferable to avoid collisions altogether and use small offsets in practice.

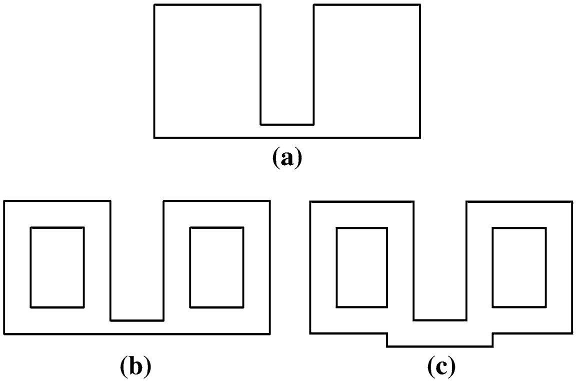


Fig. 4.86 Offsetting the geometry

Note, also, that some systems allow different offsets for specified faces. This adds more flexibility to the operation but goes against the original meaning of the operation, to convert solids into thin-walled parts which can be made by folding up material.

4.9.4 Deleting Faces

It is common to allow some faces of the object to be deleted during the operation. Figure 4.87 illustrates a simple case where the top face of a cube is deleted when applying the shell operation.

Face deletion can be done at the stage where the original object and negated copy are merged. The face to be deleted may be denoted as f_p and the corresponding face in the negated copy by f_n. The outer loop of f_n is moved to f_p as a hole-loop. The offsetting takes care of the rest.

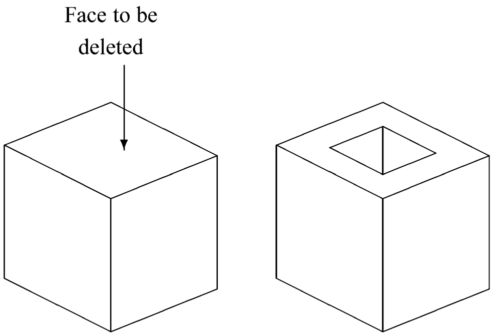


Fig. 4.87 Deleting a face when shelling

A special case is if the face being deleted has hole-loops. In this case it is necessary to block off either the hole-loop of f_p and make the corresponding hole-loop of f_n a hole-loop in the new face, or vice versa. If the hole-loop of f_p is surrounded by concave edges, then that hole-loop is blocked off with a new face, otherwise the hole-loop of f_n is blocked off.

Note, though, that if the face being deleted does have hole-loops then this will usually result in the creation of a new separate body.

4.9.5 Designing the Interior

This option does not exist, as far as I know, but could. There was a student project at the Ecole Polytechnique Fédérale de Lausanne (EPFL) to create a model of an aircraft, the Smartfish designed by Schafroth, to be built using Stereolithography. The students had great difficulty producing the shell model because the shape to be shelled had thin parts which disappeared during the offset process. It would be possible, after the negation step above, to allow the user to modify the negative shape before it is merged with the original object and offset. This is mentioned here because it might be an option in a more specialised CAD system which you can check for.

In effect this option would allow you to add material to the negated shape. Since this is a negative shape, an object sized void in a universe of material, adding material will affect only the cavity and not extrude through the outer skin. Such an option would allow a user to trim off difficult thin parts or to build internal support structures at the shelling level.

4.9.6 Experiments

4.9.6.1 Offsetting with Collisions

Make the shape shown in Fig. 4.88 and extrude it upwards 50 mm.

The idea of this shape is that you can examine the internal and external collisions in one object. Shell the object with internal and external thicknesses of 2.5 mm. Change to wireframe drawing mode and change the internal and external offsets to 5 mm, using the construction history. There should now be coincident parts to the object, though you should not be able see if the object is correct unless you section through those parts, possibly in an engineering drawing. Finally change the offsets to 7.5 and there should be overlap, which will disappear if the collisions have been detected and handled. If they have not, then be careful to keep offsets sufficiently small when using this operation in practice, otherwise you will get self-intersecting objects.

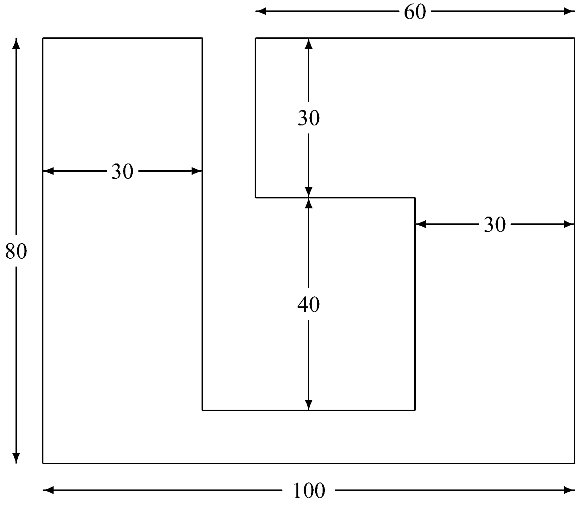


Fig. 4.88 Shelling shape for collisions

Try offsetting the outside by 12.5 and the inside by 0. This should show you whether the collision check is performed for all faces against all faces. If it is then the outside offset should intrude into the inner space. If it doesn't, then the outside will just merge into the outside. Similarly, if you offset the inside by 12.5 and the outside by 0 you can check whether the inside offset protrudes outside the object.

4.9.6.2 Offsetting Objects with Cavities

Cavities are interesting because they should give rise to separate objects when shelling. Create a 100 × 100 × 100 cube and subtract a 50 × 50 × 50 cube from it. Apply the shelling operation and check whether the CAD system indicates that the result is a multi-piece object.

Another exercise with voids is to create a 100 × 100 × 100 cube and subtract a 50 × 50 × 50 cube from it, as before, and then to put 20 × 20 mm intrusions on the top and bottom faces, depth 20 mm. You get an object with a cross-section as shown in Fig. 4.89.

Shell the object with an internal offset of 5 and check whether or not the void merges with the object.

4.9.6.3 Difficult Geometry

Some geometrical arrangements can cause problems when offsetting. Make an object like that shown in Fig. 4.90.

Fig. 4.89 Object with void for shelling

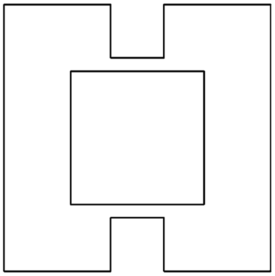
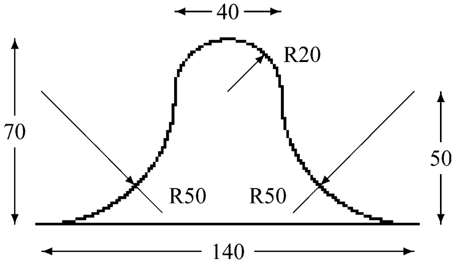


Fig. 4.90 Shelling objects with thin parts



The two large, radius 50, circles should be tangent to the baseline, but try to do this without an explicit constraint. You should make the centre points of the circles 50 mm from the base line and 140 mm apart. Extrude the shape 40 mm.

Offset the exterior of this figure by 5 mm. At the two extreme points there is a potential geometric problem. There, the surface normals at these points are in opposite directions. When offsetting, the offset surfaces do no intersect, and so a bridging face has to be constructed. The question is whether the CAD system can handle this problem and, if so, what shape the bridging surface is, a plane or a cylinder.

4.9.6.4 Deleting Faces with Hole-Loops

If a face with a hole-loop is selected for deletion, how should the hole-loops be handled? Make the objects shown in Fig. 4.91. Shell the parts, specifying the faces indicated as to be deleted and with a small offset, 5, say.

Does the CAD system indicate that a multi-piece solid has been created?

4.10 Unfolding Objects

This is a conversion operation in the opposite direction. The first time I heard of unfolding was when it was used for a Christmas card in the early 1980s from Shape Data, who produced Romulus and Parasolid. It takes a volume object and produces the flattened shape which can be bent upwards to create the volume. Figure 4.92 shows a simple example of unfolding a cube.

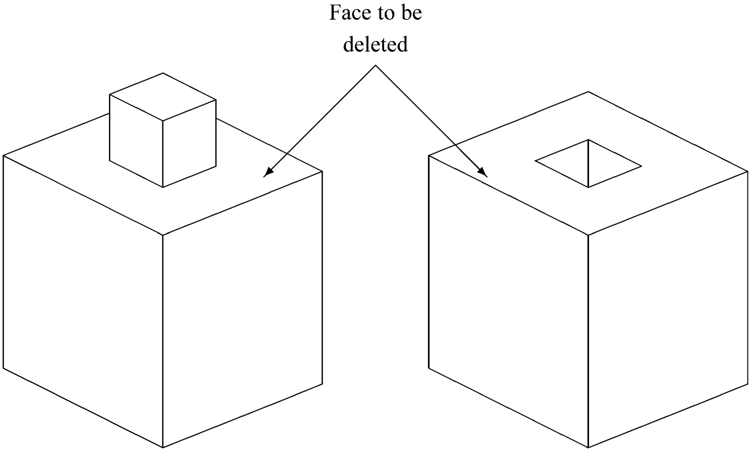


Fig. 4.91 Deleting a face with a hole-loop when shelling

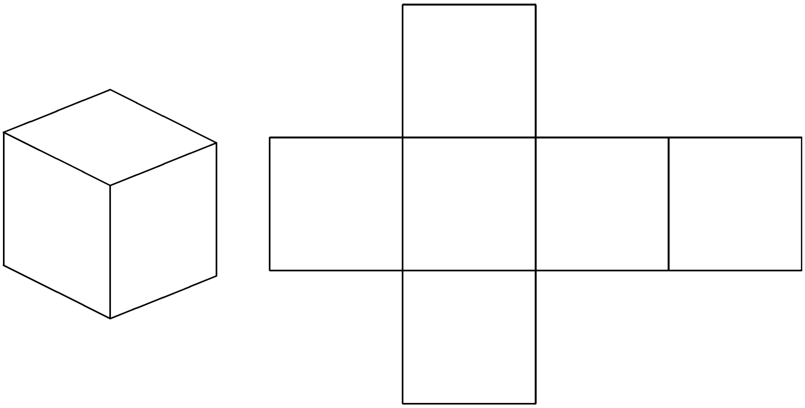


Fig. 4.92 Unfolding a simple cube

4.10.1 Parameters

Input:

The object to be unfolded
Interactively, the edges to be cut

Output:

Probably a new model of the flattened shape

4.10.2 Potential Errors

One error is if the object given is a wire object, but the algorithm should be able to handle sheet- or volumetric-objects.

4.10.3 The Unfolding Algorithm

The algorithm described here is not necessarily that which is used in commercial systems, it was developed for testing.

This algorithm uses a dual representation as a "map" to determine edges to be cut. However, in any realistic object this has to be done under user control. The dual is described in Sect. 2.7.5.4. For a cube, the object and its dual are shown in Fig. 4.93.

The manner in which the dual is used is shown in Fig. 4.94.

The first steps, as with the shelling operation, are to copy the object to be unfolded, negate it and merge it back to create the interior of the object.

The dual is then used to control the cutting. The original status is shown in Fig. 4.94a. The first edge in the dual (d0 in Fig. 4.93) is not a wire edge, so the

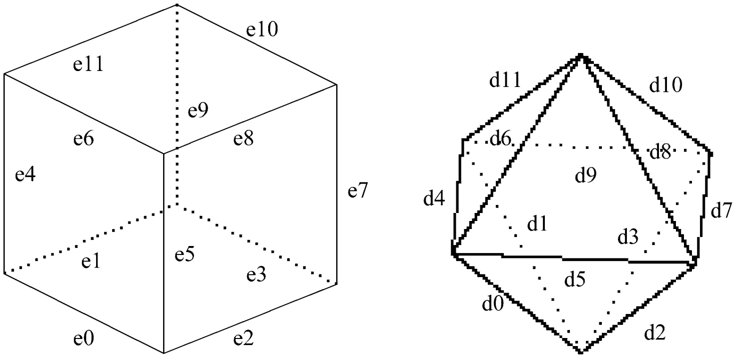


Fig. 4.93 Cube and dual

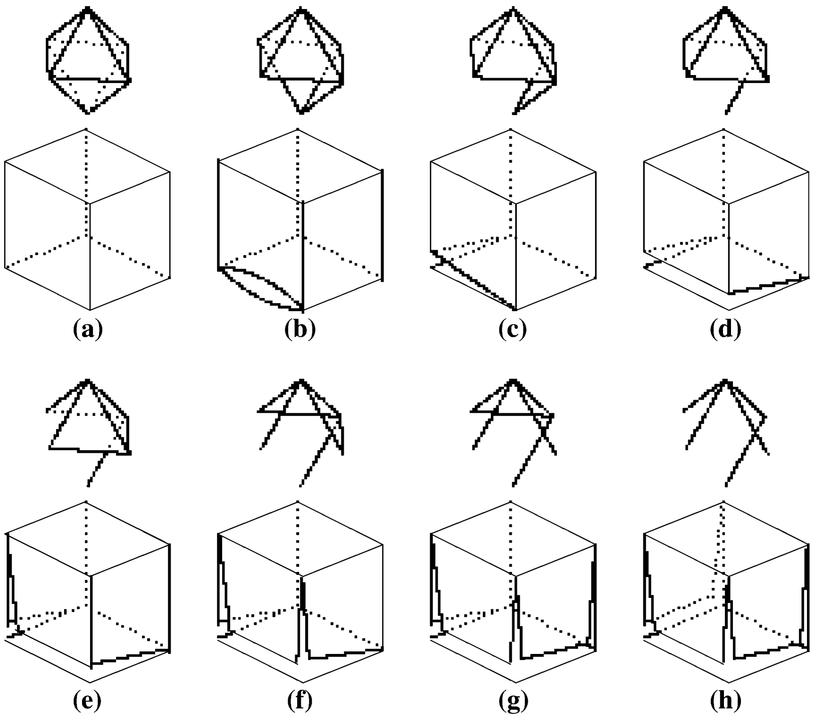


Fig. 4.94 Splitting using the dual as a map

dual edge is deleted and the cube is sliced along the corresponding edge, edge e0 in Fig. 4.93, as shown in Fig. 4.94b.

Here, slicing means combining edge e0 with its matching edge in the object interior, just created. A "wire" edge is one which has the same loop to the left and to the right.

The next edge in the dual, d1, is checked. This is not a dual, so d1 is deleted and edge e1 is sliced as in Fig. 4.94c. Similarly, dual edge d2 is deleted and edge e2 is sliced, Fig. 4.94d.

The next edge in the dual, d3, is now a wire edge, so it is left and the corresponding edge, e3, left untouched. The object will be flattened out around this edge.

Dual edge d4 is not a wire edge, so it is deleted and e4 sliced (Fig. 4.94e). Similarly dual edge d5 is not a wire, so is deleted and edge e5 sliced, Fig. 4.94f. Dual edge d6 is a wire so is left untouched. Dual edge d7 is then deleted and e7 sliced, Fig. 4.94g, and finally, dual edge d9 is deleted and e9 is sliced, Fig. 4.94h.

All dual edges are now wire edges and the object has been sliced to produce a single unfoldable part. The remaining dual graph is termed a "spanning graph" of the original dual if you want to be formal about it.

One face is then taken as a base face and the faces (and their attached object parts) unfolded to this base plane. The process is repeated for each face, flattening adjacent faces which have not already been dealt with until all faces have been handled.

Obviously this handles only simple planar polygonal cases and it is necessary to generalise the algorithm to handle curved geometry as well. Closed geometry, such as cylinders, need to be handled specially. Curved geometry needs to be flattened into planar elements.

In general, though, not every object can be flattened as a single connected piece. There are simple examples, though, objects which cannot be flattened in such a way. Objects with pockets or through holes are simple examples. Any object which is not convex, though, has potential problems with collisions between flattened parts. The use of an automatic algorithm is not advisable because edge slicing is better done interactively.

Unfolding is not one of the major shape creation tools and may only be in your CAD system as part of a sheet-metal application. The purpose of this section is partly to illustrate the easy flow between solids and sheet objects and partly to show the use of the dual graph as a support tool for other operations.

4.10.4 Experiments

4.10.4.1 L-Shaped Objects

Make an L-shaped object such as that shown in Fig. 4.95. Extrude it 40 mm and apply the unfolding operation on it.

Accept any proposals for cut edges that the system proposes. Check visually for collisions. How is it possible to unfold this object without collisions? Can you use the tool interactively to produce your solution? How does the system ask you which edges to slice.

4.10.4.2 Objects with Pockets and Through Holes

Make a 100 × 100 × 100 mm object and put a 50 × 50 × 20 pocket in the top face. Use the unfolding tool to unfold it, accepting the slice proposals proposed by the

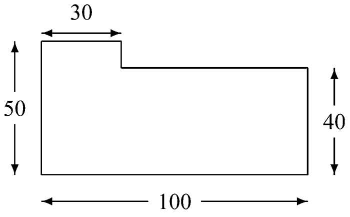


Fig. 4.95 L-shaped object for unfolding

CAD system. Examine the result to check visually for collisions. Did the system warn you that the unfolded parts interact?

Do the same for a 100 × 100 × 100 mm block with a 20 × 20 hole through it.

4.11 Giving Sheet Objects Thickness

This is a conversion operation the other way from sheet objects to volumetric objects. The effect is shown in Fig. 4.96.

4.11.1 Parameters

Input:

Object to be converted
Outer thickness
Inner thickness
Optionally different face thicknesses

Output:

Object modifications done in place or new object

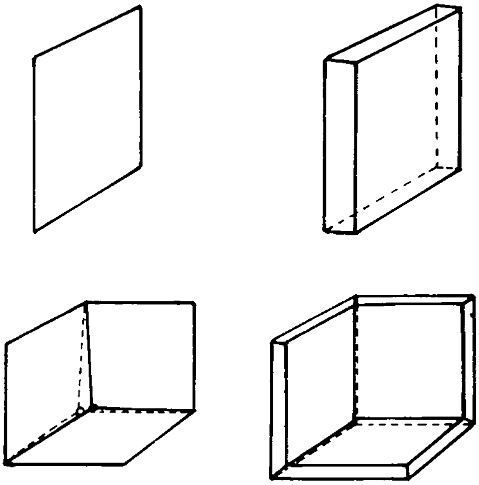


Fig. 4.96 Converting sheet models to volumetric models (from Stroud [1])

4.11.2 Potential Errors

Some conditions that should be checked for are:

- Either thickness negative
- Both thicknesses zero
- Wire body

4.11.3 The Shell-Thickening Algorithm

In a sheet object, certain edges can be characterised as "sharp", that is, the normal vectors of the adjacent faces point in opposite directions. Such edges correspond to small side faces which have been collapsed to edges for the sheet object. Once these have been converted into faces again, the geometry can be offset to create the volume object. In summary, the algorithm is:

1. Handle multi-link non-manifold edges.
2. Slice all "sharp" edges to create side faces. If a vertex of the edge has another slice sharp edge then slice the vertex as well.
3. Offset the geometry.
4. Check for collisions.

4.11.3.1 Handling Multi-Link Non-Manifold Edges

The later parts of the algorithm assume that all edges are adjacent to two faces so the first step needs to be to convert any non-manifold edges with more than two faces adjacent, and possibly their vertices. The conversion process simply involves duplicating the edge, once for each extra pair of links, and then transferring these link pairs to the duplicated edges. It is also necessary to duplicate vertices as well which have multiple edge sets so that each vertex has a single connected edge set.

4.11.3.2 Sharp Edge Slicing

Sharp edge "slicing" simply means putting another edge alongside the sharp edge to create a small face with two edges. However, it is also necessary to separate vertices, as shown in Fig. 4.97. If, after the geometry of these side faces has been created, the edges connecting the split vertices lie between faces in the same surface then they can be removed.

The left of the figure shows a vertex with two sliced edges. The vertex has to be split into two, moving some edges to the new vertex, as shown on the right of the figure. This has to be done for every vertex with two or more sliced edges.

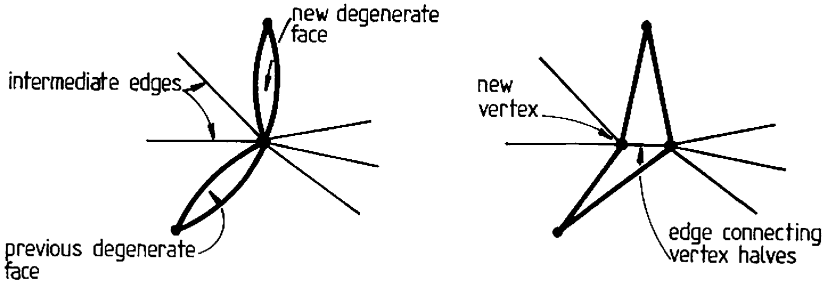


Fig. 4.97 Splitting a vertex (from Stroud [1])

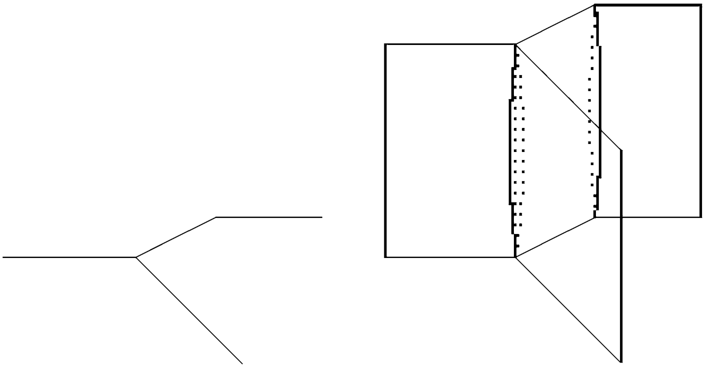


Fig. 4.98 Branching wire for creating branched sheet object

4.11.3.3 Offsetting the Geometry

Offsetting the geometry is the same as for shelling. However, it may not be clear which side has which thickness, so CAD systems should indicate this, probably graphically. You can perform a test by creating a simple sheet object, applying one-sided thickness and looking at the offset direction.

The geometry of the side surfaces should be created as well, but not offset. The surface is a ruled surface which can be generated as a line segment being swept along the curve of the split edge. For straight lines and circular arcs this can be generated directly, for other curves the resulting surface will probably be a numerical form.

4.11.3.4 Collision Checking and Finishing

Again, this has already been described in the object shelling operation. Once the geometry has been offset it is necessary to test all modified faces against each other to discover if there are collisions. If there are, then the topology of the object should be modified.

Any extraneous edges, edges lying between the same surface can also be removed at the end of the operation. Not all CAD systems do this.

4.11.4 Experiments

4.11.4.1 Branching Sheet Objects

It may not be easy to generate branching sheet objects in your CAD system. The simplest way of doing this is to extrude a branching wire, as in Fig. 4.98.

The problem for CAD system developers is to know what to do about the places where more than two edges meet. For this reason there may be a limitation on what you can extrude.

Another possibility for making a branched sheet object is to sweep two non-branched wires separately and then join them, as in Fig. 4.99.

There is a similar problem to the extrusion problem, that is, to find with which edge to merge the sheets. This is possible by using face normals, so it is not inconceivable that it exists, it depends on the level of implementation.

If successful in creating the branched sheet object then add a thickness and check whether the CAD system can handle it.

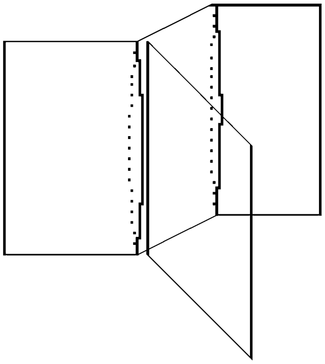


Fig. 4.99 Branching wire for creating branched sheet object

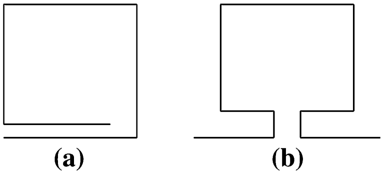


Fig. 4.100 Collision objects

4.11.4.2 Colliding Object Parts

Make two shapes like those shown in Fig. 4.100.

The two figures have close elements. In Fig. 4.100a the collision should be between different sides, whereas in Fig. 4.100b the collision is between the same side of the object.

4.12 Filling Closed Shell Objects

This is another sheet conversion operation which is more or less trivial. It takes a sheet object which has no sharp edges and pulls out the interior. Exactly the opposite of the first stages of the shelling operation.

4.12.1 Parameters

Input:

Object to be filled

Output:

Possibly new object or modification in place

4.12.2 Potential Errors

The most common problem is that a non-closed sheet object has been given. The boundaries should be indicated graphically. Another potential problem is if a flat, but closed, sheet object is given, think of a deflated balloon. Checking for an open sheet is done by checking for the "sharp" edges described in sheet thickening.

4.12.3 The Filling Algorithm

The requirement that the sheet object to be converted should be closed. If this is so, then it is necessary to handle all non-manifold edges and vertices, which will leave the object with distinct shells, and then pull out the interior shell. This is then deleted and the result object is a solid.

Handling the non-manifold is as for thickening sheet objects: all non-manifold edges are traversed and the links divided into pairs in the same way. Then, all non-manifold vertices are traversed and multiple edge sets assigned to different vertices.

Pulling out the interior is not as simple as it sounds, because you have no means of telling which shell is the interior and which the exterior just by their arrangement in the original body. It is necessary to perform a geometric test using a standard function: “point-in-body”. The simplest way is to take a point well outside a box surrounding the shell being checked and test if it lies with the body associated with the shell. If it does, then this shell is the negative one and should be deleted, because the shell represents a cavity in a universe of material.

4.12.4 Experiments

There are no special experiments described here. It is possible to check this using a simple extrusion. Make a shape such as that in Fig. 4.101.

Extrude it round the axis 270° and try to fill the resulting sheet. The system should complain and tell you that it is not closed, check whether the system indicates visually the open boundaries.

Next redo the extrusion through 360° and ask the system to fill the resulting solid.

4.13 Giving Wire Objects Thickness

Another conversion operation, but this one may not exist. It may exist in your CAD system as part of an application package, pipe routing is one possibility. The following description is based on an experimental version done to demonstrate the technique. Some results are shown in Fig. 4.102.

The wireframe model is created as a set of edges with assigned profiles. The idea is to allow users to create simple skeletal shapes and then perform an automatic conversion to volumetric models.

4.13.1 Parameters

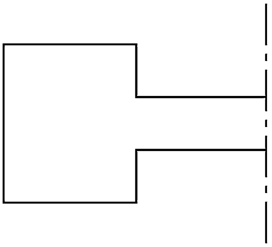
Input:

Wireframe model

Output:

Volume model

Fig. 4.101 Open wire for extrusion



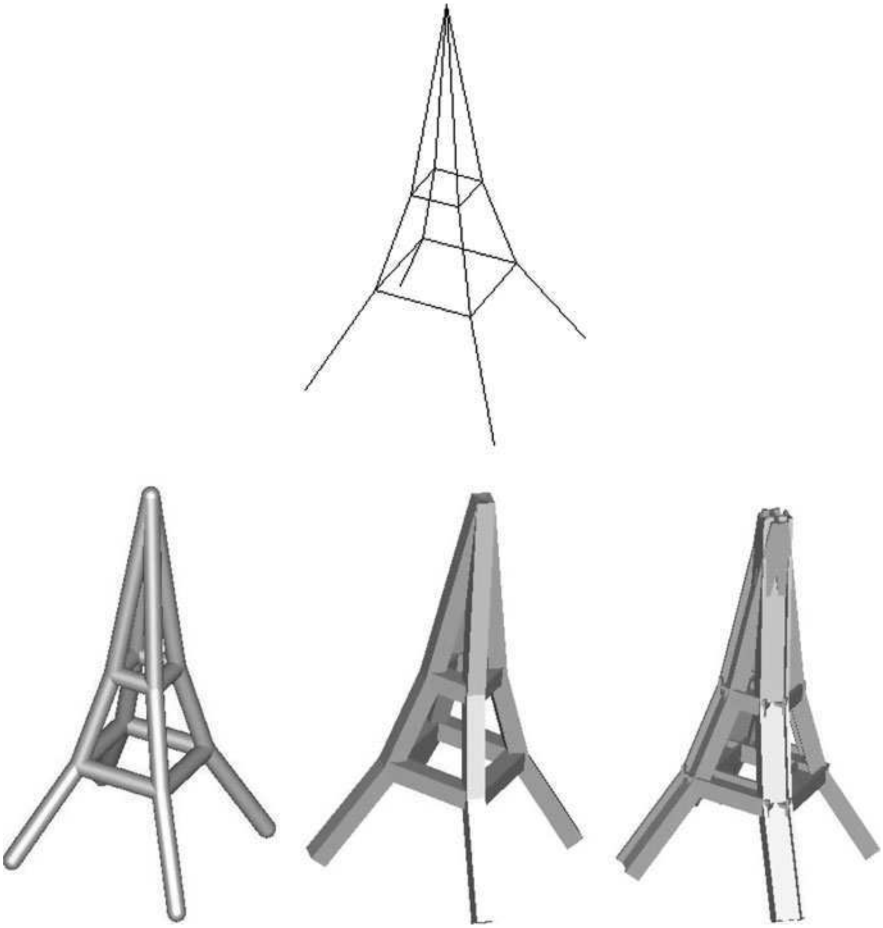


Fig. 4.102 Converting wireframe models to volumetric models

4.13.2 Potential Errors

One condition to check for is an edge or edges with non-assigned profile, or degenerate profiles.

4.13.3 The Wire-Conversion Algorithm

The simplest way to perform this is as a set of edge conversions and Boolean operations to join the sub-volumes. This was done for the model in Fig. 4.103.

The model has mixed cylindrical and square profiles.

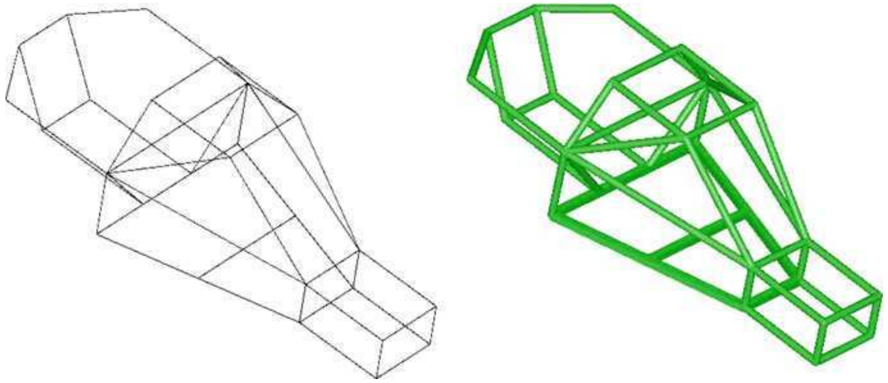


Fig. 4.103 Converting a wireframe models to a volumetric model

The awkward thing about this method is handle the junctions where the shaped wires meet. For circular profiles this was done by adding spherical endcaps. The junctions for square and girder-like profiles is less clear. Ideally this would be done by expanding the vertices rather than using endcaps.

4.13.4 Experiments

You can simulate the process using extrusion along a path curve instead of wire conversion, if your system does not have this.

This experiment is based on a CATIA exercise I use for students, so the information given for creating the geometry is what CATIA asks for. Create six points with coordinates:

(−100, −100, 0)	(100, −100, 0)	(100, 100, 0)
Point 1	Point 2	Point 3
(−100, 100, 0)	(0, 0, 250)	(0, 0, 600)
Point 4	Point 5	Point 6

Create lines as follows:

Line 1	Point 1	Point 5
Line 2	Point 2	Point 5
Line 3	Point 3	Point 5
Line 4	Point 4	Point 5
Line 5	Point 5	Point 6

These lines form a geometric framework around which a structure will be created.

Next come some planes. This exercise was developed for a course based on CATIA and the planes were needed as supports for the conics. This may be because the conic creation parameters are projected to the plane to ensure that a conic is created and not a space curve. Create planes as follows:

Plane 1	Point 1	Point 2	Point 5
Plane 2	Point 2	Point 3	Point 5
Plane 3	Point 3	Point 4	Point 5
Plane 4	Point 4	Point 1	Point 5
Plane 5	Point 6	Point 5	Point 1
Plane 6	Point 6	Point 5	Point 2

Two other planes are needed: Plane 7 Offset 250 from the *XY* plane and Plane 8 Offset 600 from the *XY* plane.

Now create conics. These take a start point, an end point and two tangent directions. You may also need to specify a plane for the curve, as mentioned above.

Conic 1	Point 1	Point 2	Line 1	Line 2	Plane 1
Conic 2	Point 2	Point 3	Line 2	Line 3	Plane 2
Conic 3	Point 3	Point 4	Line 3	Line 4	Plane 3
Conic 4	Point 4	Point 1	Line 4	Line 1	Plane 4
Conic 5	Point 6	Point 1	Line 5	Line 1	Plane 5
Conic 6	Point 6	Point 2	Line 5	Line 2	Plane 6
Conic 7	Point 6	Point 3	Line 5	Line 3	Plane 5
Conic 8	Point 6	Point 4	Line 5	Line 4	Plane 6

You now have a wireframe geometric framework with which to construct a solid. Make a new sketch on the *XY* plane, a circle radius 5 centred at point 1. Extrude this along conic 1 and conic 4. Create another sketch on the *XY* plane, a circle radius 5 centred around point 3. Extrude this along conic 2 and conic 3. Create a sketch on plane 8, a circle radius 5 centred at the origin and extrude this along conics 5, 6, 7 and 8. Finally create a circle on plane 7, again a circle radius 5 centred at the origin. Extrude this along lines 1, 2, 3, 4 and 5.

This exercise is intended to show you how to work with wireframe models to set up a geometric structure and then to use this to create a volume model.

4.14 Lofting

Lofting is an operation that creates an object from a series of sections. There is a pure surface technique, to be described in Chap. 5 and a volumetric version. The difference is that the volumetric version closes the end faces while the surface technique leaves them open.

Figure 4.104, from Stroud [1], shows a simple example of some sections and the lofted body in wireframe mode.

As with most geometric tools in CAD systems, the developers hide the truth from you! No, this is not because they are based on alien technology, it is simply that the mathematics and the details are complex. Instead you are presented with a set of indirect tools for shape control which are intended to be more natural than manipulating control points and parameters. This is, in fact, in keeping with some of the original work, by Pierre Bèzier, who created a geometric design system more intuitive for users than that based on equations. Actually, you should be able to find some tools to show the control polygons or move control points, but these are not generally encouraged. More of this in Chap. 5.

4.14.1 Parameters

Input:

Sections to be interpolated with start point and direction
Optional shape control parameters for each section

Output:

New solid or possibly modified solid if one or more sections belong to a solid

4.14.2 Potential Errors

Some conditions that should be checked for are:

- Self-intersecting interpolation curves or surfaces.
- Multipiece sections.
- Less than two sections.
- Mixed open and closed sections.

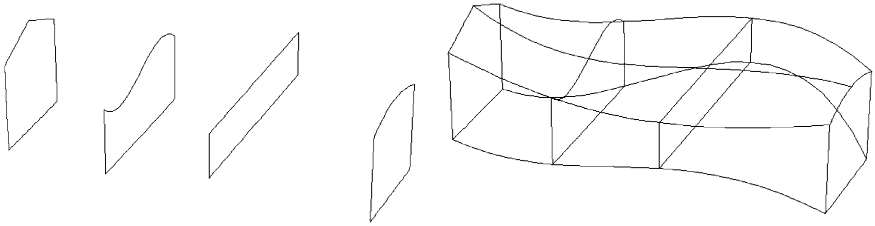


Fig. 4.104 Lofted sections (from Stroud [1])

4.14.3 The Lofting Algorithm

From a topological point of view the algorithm is simple. The sections are associated in the same topological framework and then sets of points, for each set one point from each section, are used to interpolate side curves. Edges are created between the points, creating also the side faces, and finally surfaces are calculated for these side faces.

4.14.3.1 The Topological Framework

Topologically, the sections are made into faces, one inside the other. Take the sections in Fig. 4.105.

These are made into flat sheet objects and then "glued" together, as shown in Fig. 4.106.

It should be stressed, though, that this is simply an organisational step, there is no change in position of the sections. Topologically they are as shown in the

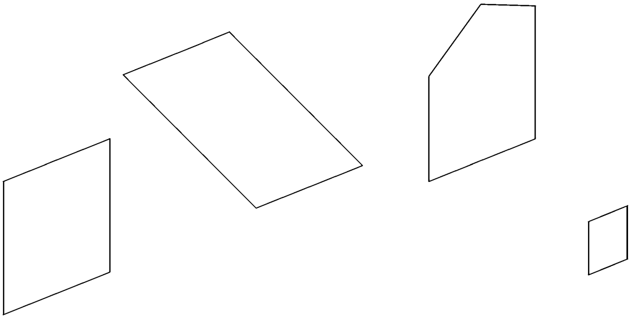
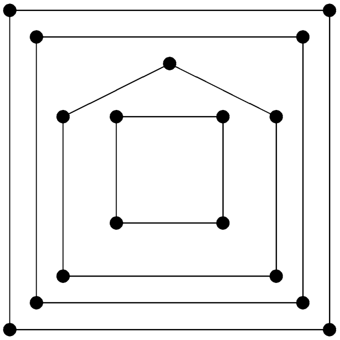


Fig. 4.105 Sections to be unified

Fig. 4.106 Unifying sections



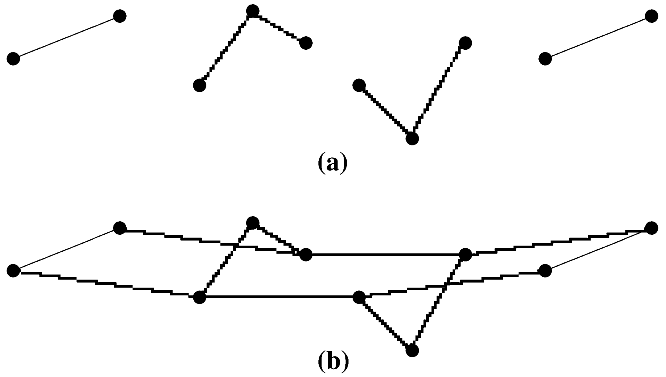


Fig. 4.107 Unifying open sections

figure, but spatially they are as defined. This is necessary for matching the sections and adding the side elements.

If the sections are open, as in Fig. 4.107a then there is a slightly different approach. The ends of each section are connected in a ladder-like structure before matching their internal topology.

4.14.3.2 Section Matching

If the sections have the same topology and the vertices can be matched then the lofted body construction is straightforward, the vertices are joined to create the body. However it is not always the case that section topology matches, so some sections will need to have extra vertices inserted.

In Fig. 4.106 the third section has five vertices, so it is necessary to insert an extra vertex in the other three sections, as shown on the left of Fig. 4.108.

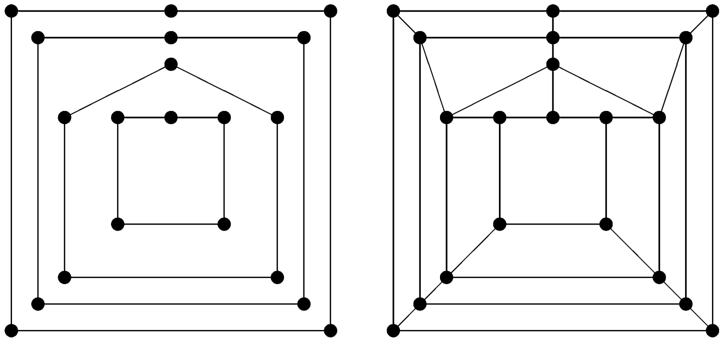


Fig. 4.108 Unifying sections

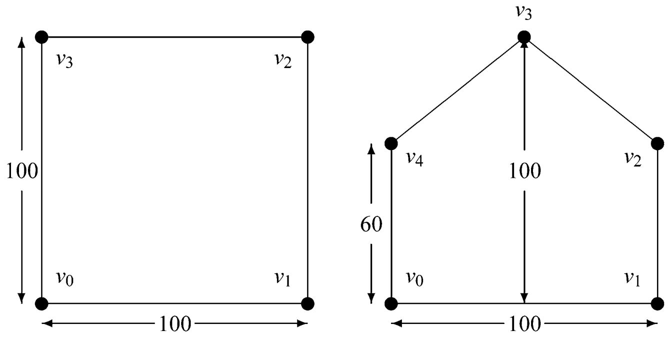


Fig. 4.109 Vertex mapping

The corresponding vertices are then joined with lines to complete the lofted body topology, as shown on the right of Fig. 4.108.

Obviously vertices should not be inserted arbitrarily. Comparing sections can be done by comparing vertex positions using the section length as a map.

Taking the two simple sections in Fig. 4.109, the length of the section on the left is 400, and the length of the section on the right is (approximately) 348. This gives the vertex maps:

v_0	v_1	v_2	v_3	v_0
0	0.25	0.5	0.75	1

v_0	v_1	v_2	v_3	v_4	v_0
0	0.29	0.46	0.64	0.83	1

Comparing the values, the largest discrepancy is with v_3 in the second section. Inserting an intermediate vertex halfway between v_2 and v_3 in the first section, at length 0.625 gives the required topology. An alternative is to create a single list with all the different parameter values, i.e.

v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_0
0	0.25	0.29	0.46	0.5	0.64	0.75	0.83	1

and subdivide each contour at these positions, joining the matching vertices.

Note that if the section is closed the first and last vertices (at 0 and 1) are the same, for open sections they will be different. In this way the open sections in Fig. 4.107 can also be compared.

Placing the vertices may not be optimal using this technique. It may be that if the sections are not too close in position it may be desirable to introduce new

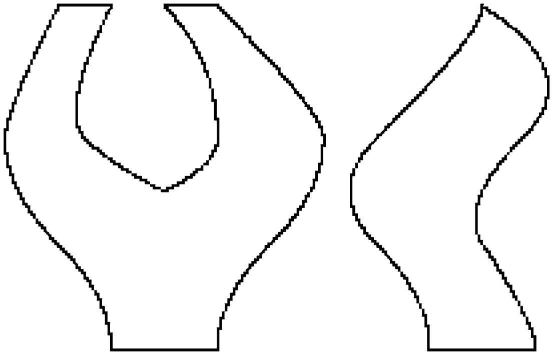


Fig. 4.110 Topology changes

vertices in both sections manually to try and get a matching set. Otherwise, though, it may be necessary to rethink the sections so that the topology matches and maybe to introduce supplementary sections.

Note, also, that the choice of start point and direction govern the values in the map. It is not possible to make the selection totally automatic because the technique is used to create strange shapes.

4.14.4 Other Comments

In lofting you cannot change the topology. Two shapes which cannot be created are shown in Fig. 4.110.

The figure on the left divides into two arms, the shape on the right ends in a point. Neither of these shapes is theoretically impossible to create with a modified technique, it is just that the implemented tools do not allow this.

The figure on the left could be produced by have one complex section where the section has an internal dividing curve, just before the multiple element sections. Joining them together might also be possible. At present, though, you can only make a shape like this using two overlapping lofts.

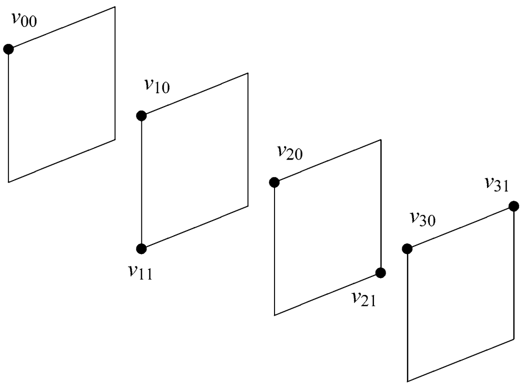
For the figure on the right, if the final section is a point, or even degenerate, you would generate degenerate side surface patches, where one side has zero length. This is possible, but creates an unwelcome singularity at the point, so other methods may be preferable for creating the end shape.

4.14.5 Experiments

4.14.5.1 Experiment with Sections

Make four square sections, 50 × 50 mm on planes spaced at 50 mm from each other. The square sections should be aligned, as shown in Fig. 4.111.

Fig. 4.111 Vertex mapping



Loft these sections in the "obvious" manner, in the order section 0, section 1, section 2, section 3, using v_{00}, v_{10}, v_{20} and v_{30} as alignment points and all sections having the same direction. You should get a straightforward, rectangular block shape.

Next, loft the sections but change the order, section 0, section 1, section 3, section 2, using v_{00}, v_{10}, v_{30} and v_{20} as alignment points and all sections having the same direction. If the system lets you do this you should get a strange bell shape. If the system objects you could try enlarging section 3 and reducing the size of section 2.

For the third step, loft the sections in the normal order, section 0, section 1, section 2, section 3, using v_{00}, v_{10}, v_{30} and v_{20} as alignment points but reverse the direction of section 2. If the CAD system allows this then you should get a twisted figure with a degenerate point somewhere.

Finally, loft these sections in the order section 0, section 1, section 2, section 3, using v_{00}, v_{11}, v_{21} and v_{31} as alignment points and all sections having the same direction. This should give you a perfectly valid figure, but there is no way that the CAD system could be expected to know that you want to align the sections in this way. This is why you have to be in control.

4.14.5.2 Shape Changing

One popular test is to change from circular to square cross-section. Create sections on three parallel planes, 100 mm apart, with a 100 × 100 square section, a circular section, radius 25 mm, and finally another 100 × 100 square section. Figure 4.112.

Create a lofted body with these three sections. The thing to note is how your system handles the transition. Is it automatic or do you have to break the circle into segments? You should also note what options you have for determining how the section mapping is done. Is it only by vertices? Do you have a proportional measure?

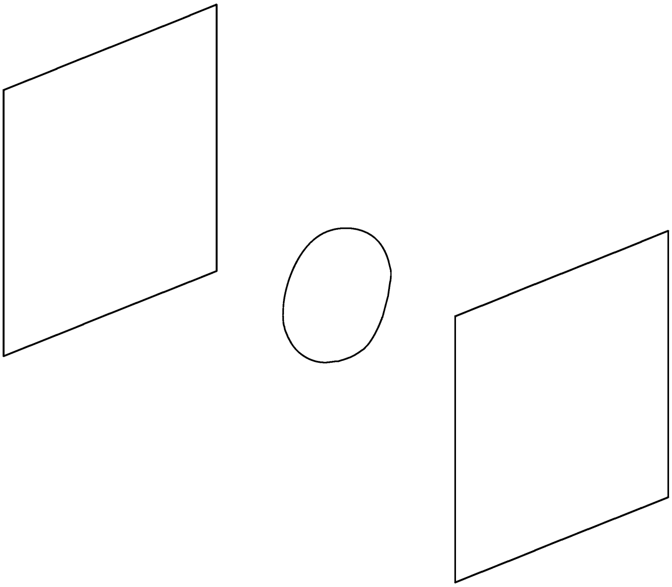
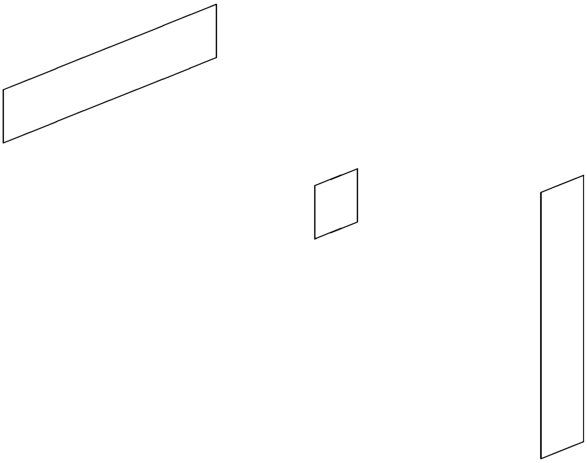


Fig. 4.112 Section shape changing

4.14.5.3 Proportion Changing

This experiment is to look at sections which have the same topology but different proportions. Figure 4.113 shows three section shapes on planes 100 mm apart. The first is 100 mm wide and 20 mm high. The second is square, 20 × 20 mm. The third is 20 mm wide and 100 mm high.

Fig. 4.113 Proportion changing



Try lofting between the first and third section, first of all. Again, check what types of connection the system allows. Do you only have a vertex connection? If so, then the proportion differences do not matter. If there is a proportional option then the vertices will not match and you should see extra vertices and connection lines. Repeat the loft, but using the second section as a bridge. If you have the proportion option then check how the connection lines change.

4.15 Patterns

The creation of a pattern is usually associated with either objects or with operations that create volumes and then combine these with the basic shape. This is not a necessity, though.

Normally, patterns come in two flavours: rectangular or circular. In this case, "pattern" is used to mean "repeated elements". Another type of pattern is described in Sect. 4.16.3 as one of the "Weird operations".

4.15.1 Parameters

For rectangular patterns

Input:

Number in *X* direction
Number in *Y* direction
X spacing, or total *X* distance
Y spacing, or total *Y* distance
Possibly elements to skip

Output:

Changes made in place

For circular patterns

Input:

Number of elements
Pattern centre
Pattern radius
Start angle
End angle

Output:

Changes made in place

4.15.2 Potential Errors

There are not many conditions to check for, these are mainly numerical.

- *X* or *Y* number less than 1.
- *X* or *Y* distance approximately zero.
- Circular pattern number less than 1.
- Circular pattern radius approximately zero.

4.15.3 Patterning Algorithm

Patterning is, at present, positioning and a Boolean operation. The positioning can be done on the sub-volumes created by some of the operations, such as positive or negative extrusions, or even the basic object. The sub-volume is copied and translated to each place in the pattern. The movement is either a translation, or a translation and rotation, depending on the type of pattern. Note, though, that you may not be informed if one of the pattern elements misses the target object completely and so makes no change.

Another way of creating a pattern is to use the parameters of the sub-operation with the pattern positions as parameters. There is a subtle difference between these two.

4.15.4 Experiments

4.15.4.1 Disappearing Holes

Make an object like that in Fig. 4.114.

Extrude the figure 40 mm. Make a hole on one part of one of the lower faces, as shown in Fig. 4.115. The hole should have depth 20. If your CAD system has a hole making function use that, if not, extrude a circle downwards 20 mm into the material.

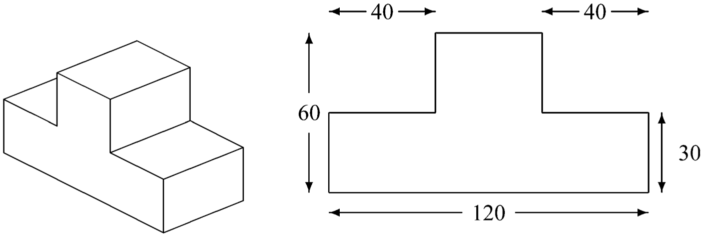


Fig. 4.114 Basic figure for disappearing holes

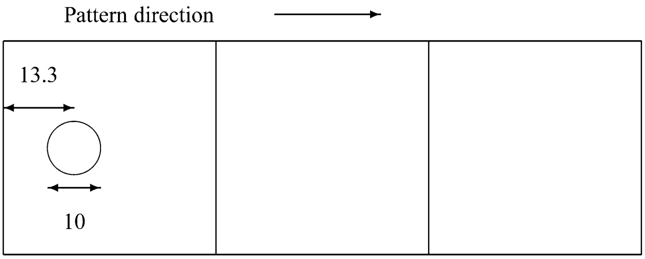


Fig. 4.115 Basic figure for disappearing holes

Put the graphics into "shaded image" mode. Make a rectangular pattern of six elements in the direction shown with a step length of 13.33.

Where are the middle two holes?

Put the graphics mode into wireframe to see them. If the CAD system has a hole making operation, then the middle two holes should really extrude to the outside of the object. More about this, though, in Chap. 10.

4.16 Weird Operations

This section just illustrates some other operations that are not found in CAD systems to show that there are more possibilities. In general there are many more possibilities for useful operations than have been realised. The problem is to identify what is needed and then to create a stable algorithm. This could be done by code developers or by users, if the users can frame their suggestions in an appropriate way. More in the next section.

4.16.1 Rebating Edges

The first extra operation is an operation to create shaped edges, shown in Fig. 4.116.

The algorithm for this operation is described in [1, 5]. It was developed for a stone product shaping application and is related to blending and chamfering. In Fig. 4.116a you see the original object. Figure 4.116b shows the edges around the top face with a simple cutout rebate. In Fig. 4.116c there is a quarter-round shape in the rebate. In Fig. 4.116d there is a square shape added into the rebate and in Fig. 4.116e a wiggly form has been added.

All four operations take the edge to be rebated. For the square rebate the other parameters are the offsets in each direction. For the quarter-round rebate the extra parameter is the radius of the quarter-round. For the square shape the extra

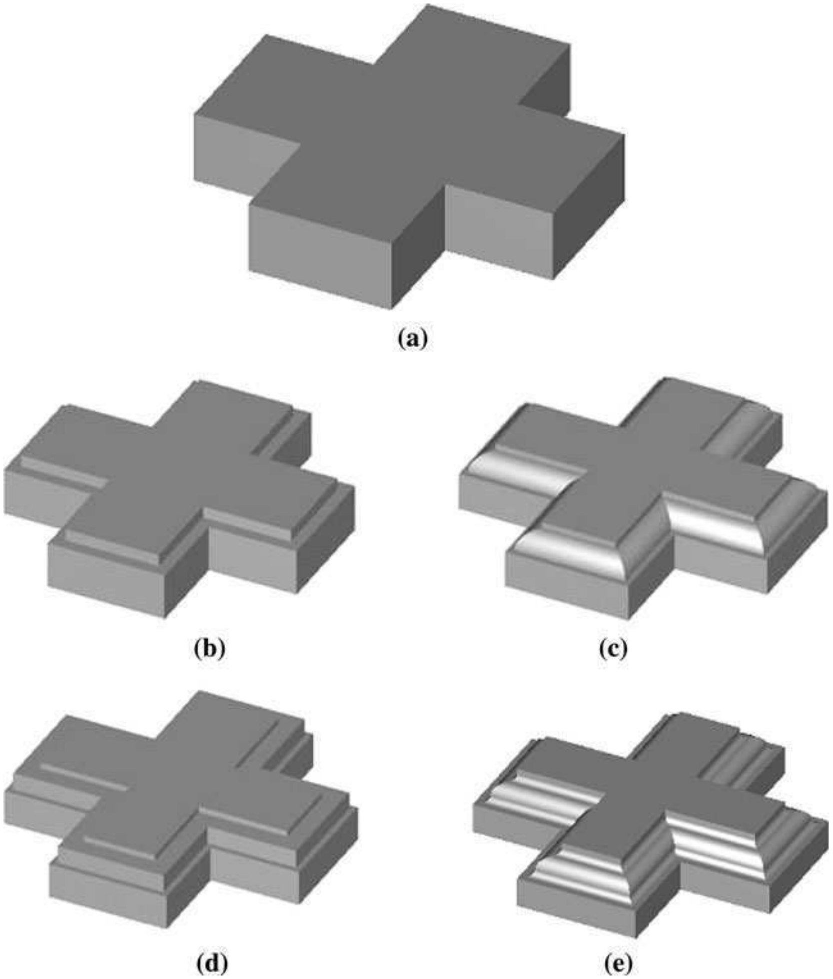


Fig. 4.116 Aesthetic cutout examples (from [5])

parameters would be the offsets back in each direction. Finally, for the wiggly rebate the extra parameters would be the radius of the positive cylinder and the radius of the two negative radii.

Creating such shapes is possible using CAD systems, but laborious and therefore error prone.

4.16.2 Bending Objects

Bending was developed to show the final state of a model of a paper clip, shown in Fig. 4.117a and b.

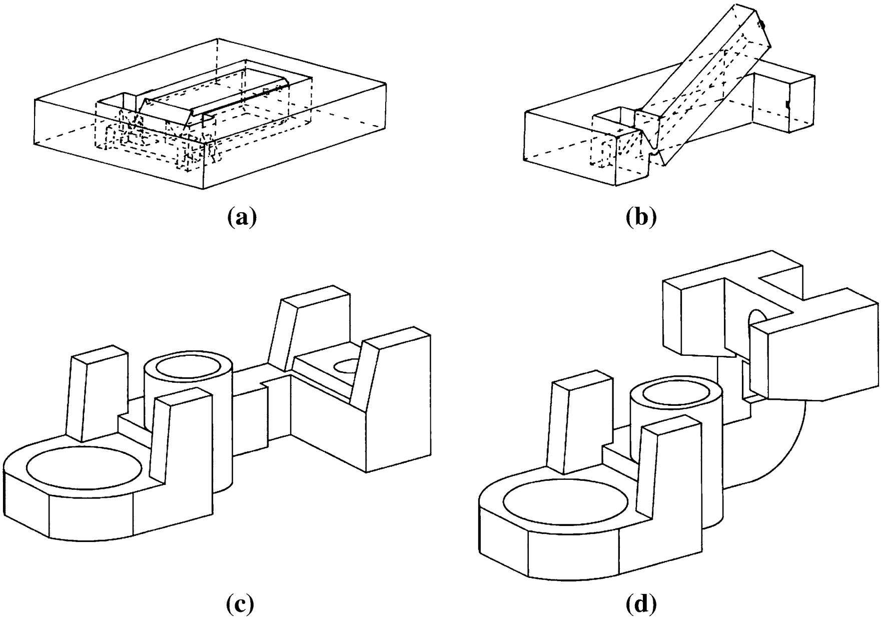


Fig. 4.117 Bent objects (from [6])

Figure 4.117c and d show bend applied to another object. The operation is an unusual application, developed around 1979, and has not been transferred into CAD systems.

The operation takes an axis, an angle, a face or edge in the object part to be moved.

4.16.3 Creating Simple Celtic Patterns

The algorithm for producing Celtic patterns is based on descriptions from a lovely book by Bain [7]. However, whereas Bain explains a wide range of patterns, only very simple patterns were implemented.

Celtic patterns are different from the patterns described earlier because the repeated elements are arranged alternately. There is a simple creation, shape filling method which can be used to create complex shapes. Figure 4.118 shows a planar example, Fig. 4.119 cylindrical and spherical examples and Fig. 4.120a ring design and a ball design using the same technique.

The shape definition is done interactively, defining an $m \times n$ grid with basic crossovers as the basic shape and then emptying sections, defining turns, etc. until the final shape has been defined. The conversion method then creates the three dimensional shape from the two dimensional flat pattern, with wrap arounds for the cylindrical or spherical patterns. The method is described in [1, 5].

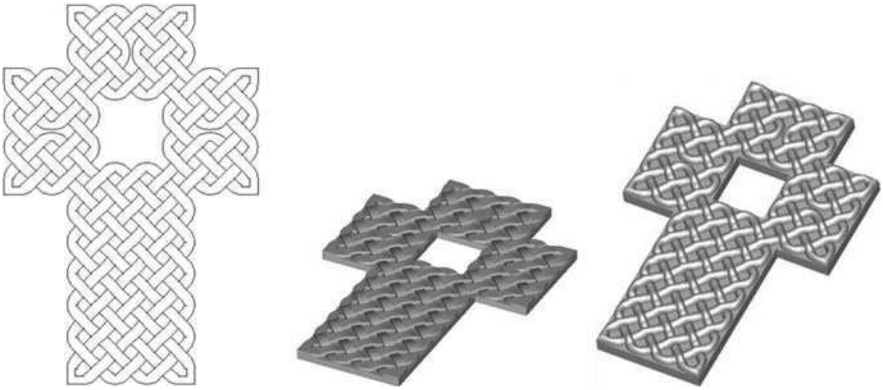


Fig. 4.118 Celtic cross-shape and 3D models (from [5])

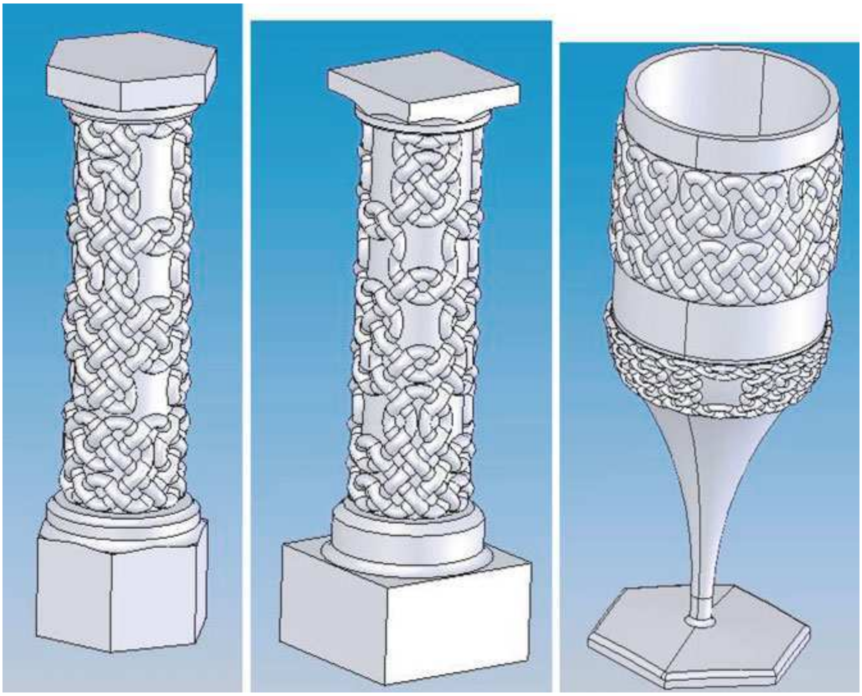


Fig. 4.119 Cylindrical and spherical Celtic patterns (from [5])

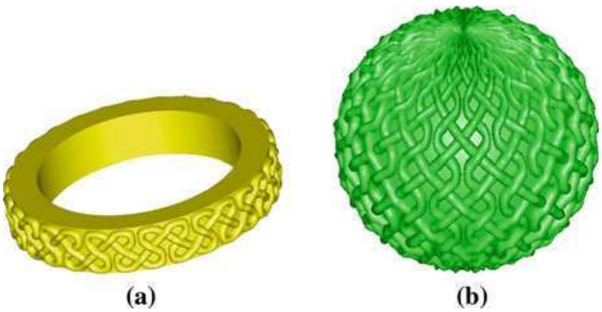


Fig. 4.120 Celtic style ring and ball

4.16.4 Sculpting

Sculpting applies a well-defined function to an object to warp or change the shape in controlled ways. The five images in Fig. 4.121 show the original object (Fig. 4.121a) and the object scaled twice along the *X*-axis (Fig. 4.121b). On the next row the object has one part enlarged (Fig. 4.121c) and after that, with a twisted rule applied (Fig. 4.121d). Finally the object with the central bridge part bulged (Fig. 4.121e).

This sculpting method is applied to facetted shapes, which are easy to manipulate. It would be relatively simple to develop an application to modify the geometry of general objects. Scaling in one direction sometimes exists, but twisting and local shape changing is not generally available.

4.17 Creating New Operations

This section is intended to give a guide to determining new operations. It is likely that you, the user of the system, could identify more useful operations than the software developers. This is because you know your application, and what would make it easier to develop your designs. The software developer knows how to create such operations, but not necessarily what operations would be desirable. The following are some suggestions to help develop operations.

1. Define the potential users, as this may help the software developer to evaluate the economic benefits of new operations.
2. Create “before” and “after” models to show the effect intended by the operation. For some operations this may be difficult, but it is important to be able to define clearly what is supposed to happen.
3. Define the expected parameters. The parameters may be topological elements, such as face, edge or vertex; geometric parameters, such as surfaces, curves or points; or other things, such as real values or vectors.

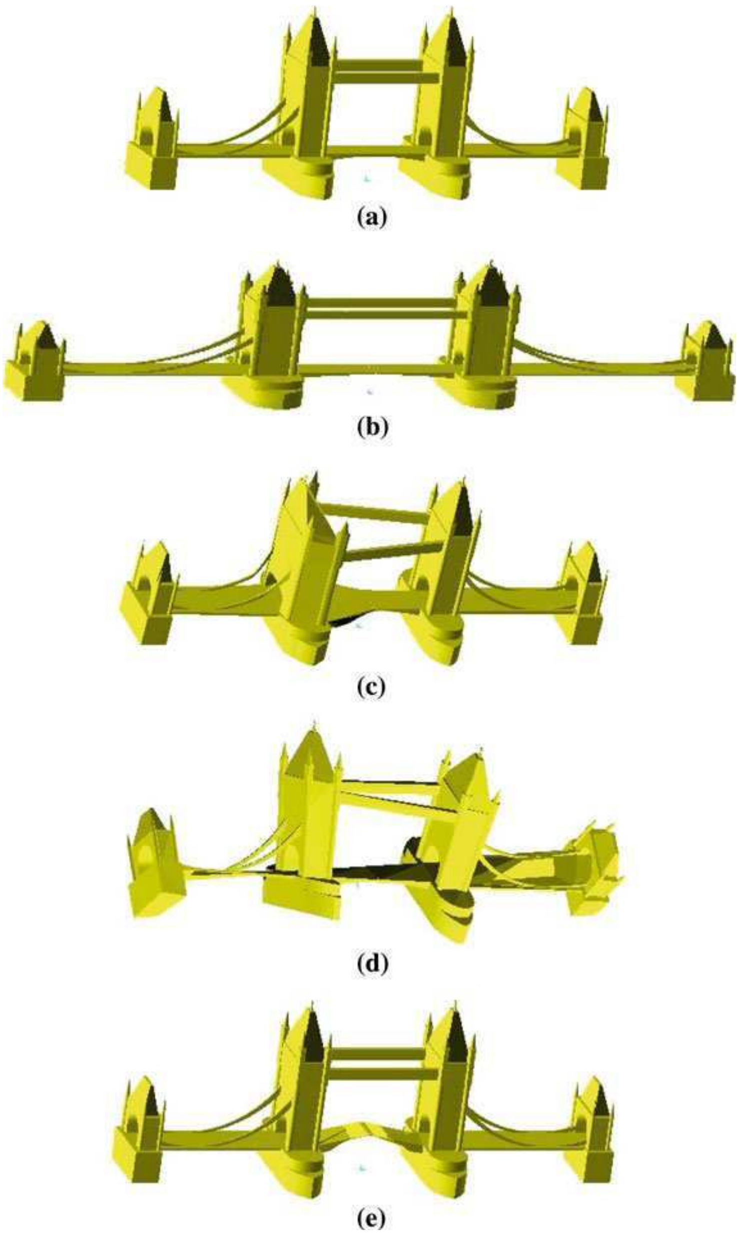


Fig. 4.121 Sculpted facetted objects

4. Define the possible error conditions, the critical cases which have to be avoided because they would lead to errors.

Existing operations, possible parameters and critical cases are described in this chapter as an aid to developing these. It is difficult to be precise about new operations because the nature of these new operations has to be determined on an individual basis.

The algorithms mentioned in this chapter fall roughly into two groups. On one side there are operations which add or subtract material by defining a new volumetric part and then applying a Boolean operation (e.g. extrusions, symmetry). The other group make changes to the structure locally to create the result (e.g. expanding sheet objects or shelling). It is useful if you can formulate an algorithm in one or both of these categories when you talk to a developer, although not essential. It may be easier for a developer to use an algorithm which creates a volumetric part and then adds, subtracts or intersects this with the main body. This can be easier for the system developer because the Boolean operations are standard operations and so it saves time. It may also make it easier to create patterns of your operation or a symmetrical operation. However, using Boolean operations hides potential problems. If you look back at Sect. 4.7 two alternative problem chamfers are shown in Fig. 4.74. Using Boolean operations it is easy to get an object which is technically correct geometrically, but the result may not make sense in the application world. This is why it is also important to be able to identify critical cases and say what should happen when these arise.

Finally, please note that the system you use does not necessarily have to be the way it is. Often system developers may make arbitrary decisions to solve problems without knowing that there may be a different option which would be preferable to the users. This is not intended as a criticism of software developers, simply an observation that the systems are complicated to develop. Development is probably best done by software engineers but they need to hear about what users want. For users, it is important to realise that there are alternatives and not to accept awkward manipulations when it might be simple to change the software tools. An example is the way in which blended edges are handled during drafting, the software solution may be correct as a solution, but should the operation really function that way? It is important to establish a dialogue with the system developers to suggest alternatives in order that CAD systems become tools function for the benefit of the user.

4.18 Chapter Summary

This chapter is intended to help users understand what is being done by CAD operations and to create their own test procedures to map out the functionality of a CAD system. Many “classical” operations are described, with a list of possible parameters, error conditions and a brief outline of algorithms. Possible experiments are described to clarify implementation details.

4.19 Operation Exercises

Many experiments for individual operations were described in the chapter.

4.19.1 Exercise 1: Sweeping

In many CAD systems modelling operations are a combination of an operation to create a solid and a Boolean operation to unite this solid with the original object. This exercise is to demonstrate some aspects of this.

Draw a sketch on the *xy* plane, as in Fig. 4.122.

Extrude the shape 200 mm. Now on the back face, the large face, 200 × 170 mm, sketch a rectangle 160 × 60 as shown in Fig. 4.123.

Extrude the rectangle to cut out a rectangular hole through the object. You can choose whether to extrude the rectangle 50 mm or "until last".

Change the visualization mode to wireframe.

On the top inside face, marked with an arrow in Fig. 4.124, draw a circle, radius 10.

Fig. 4.122 Basic shape

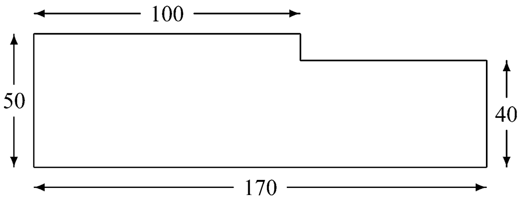
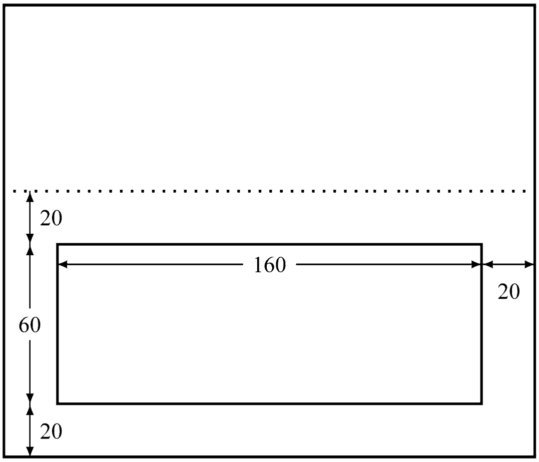


Fig. 4.123 Basic shape



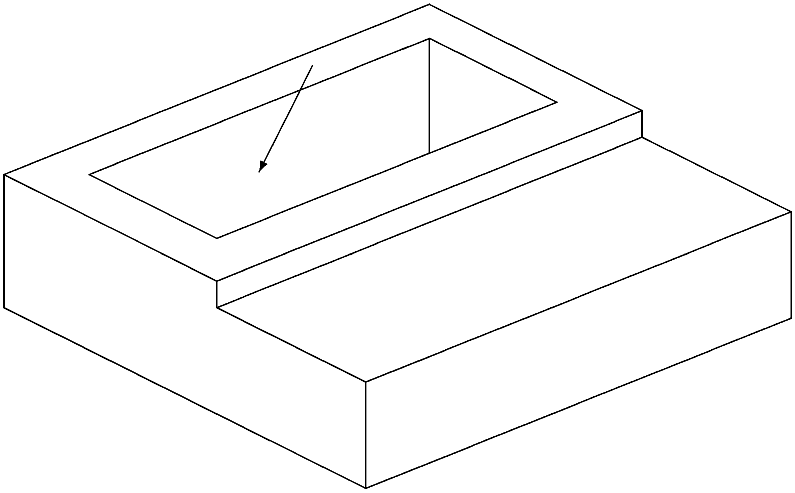


Fig. 4.124 Basic shape with first hole

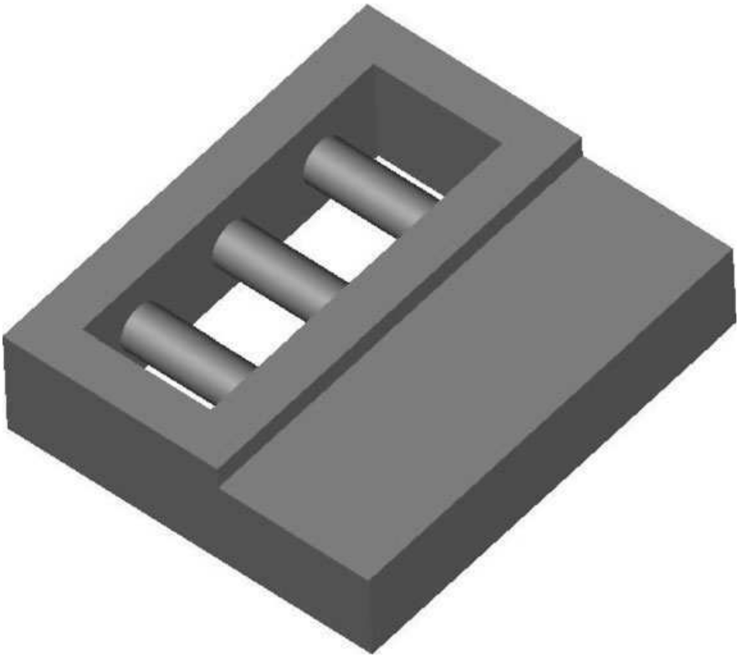
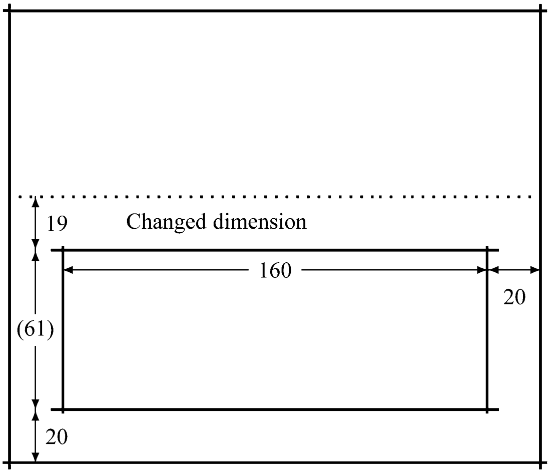


Fig. 4.125 Basic shape with cylinders

Fig. 4.126 Basic shape



Extrude the circle “until last”. Repeat the sketch to add a circle centred in the face (i.e the dimension 30 should be 80). Extrude the circle “until next”. Add a third circle, symmetrical with the first, that is the 30 should be 130. Extrude it 60 mm. The resulting shape should be as shown in Fig. 4.125.

Add a blend (or fillet), radius 5 to the cylindrical faces. Make sure that the blend is added to the face and not the individual edges around the face.

Edit the first sketch, as shown in Fig. 4.126.

Change the dimension 20 to 19 and exit the sketcher. Note the change to the blend in the column on the right. Now, the specific extrude is not long enough and so the edges round one blended end of the column are convex, hence the change in shape. The other two columns are adjusted automatically.

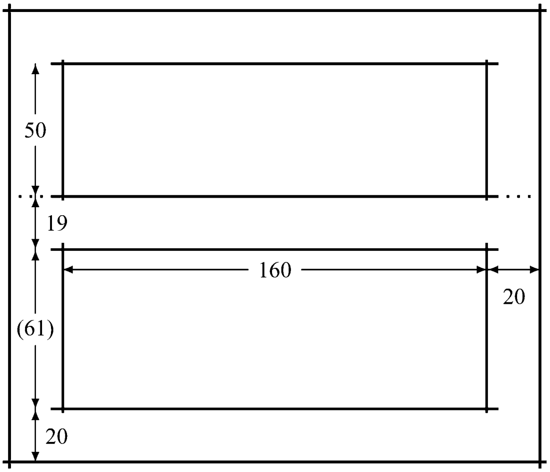
Now, go back and edit the sketch which created the rectangular hole through the object. Add a second rectangle to the face, as in Fig. 4.127.

When you update the operations you should get a shape like that shown in Fig. 4.128.

Note that the first column now extends through the extension. It is interesting to note whether the face blend has been transferred to the second part as well. If it has, as happens in CATIA v5 for example, this is due to a mechanism called “persistent naming”, which will be described in Chap. 7 and in Chap. 12. This mechanism enables the system to identify that the cylindrical part seen in the hole after the modified hole extrusion is also part of the face which was blended. It is a mechanism for trying to estimate what you intended to do, but will be described in more detail later.

The different options, length, until next, until last, until plane, define the sizes of the simple volumes added to the basic shape to give the result of the extrusion operation. It is difficult to be precise about how to use these, so this exercise is

Fig. 4.127 Basic shape



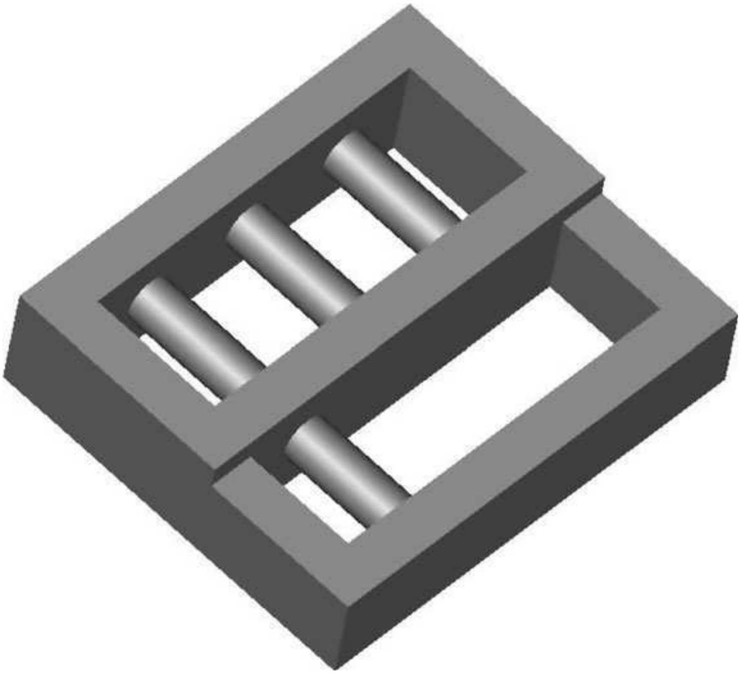


Fig. 4.128 Basic shape with cylinders after the second cutout

intended to show some effects. If parts are intended to be parametric then it can be dangerous to use explicit dimensions. “To next”, “To last”, “To surface” can be more useful for such cases.

A small extra task is to change the dimension “40” in Fig. 4.122 to 30 and to 25 and watch what happens to the cylinder extruded “until last”. In CATIA v5 the definition of “last” is the last face which totally intersects the shape being extruded, so that the end edge geometry can be calculated. Other definitions are possible, though. It is possible to define “last” as the last surface intersected by the shape.

4.19.2 Exercise 2: Complexity Testing

An important general test is to check the level of complexity supported by a CAD system on the hardware available. The tests described here involve two methods: symmetry or reflection; and pattern operations. Symmetry has an exponential build-up and is easier to create huge objects quickly, but both symmetry and patterns use Booleans to combine elements.

Make a shape such as that in Fig. 4.129. This shape is a 40 × 40 × 40 cube with three perpendicular 80 × 20 × 20 blocks added.

Make a pattern of 10 × 10 × 10 elements with a distance of 82 between each element. This is just to build a pattern with size which can be increased progressively. You keep doing this until the system cannot cope. Note that the first shape has 56 vertices, 84 edges, 36 faces and 6 hole-loops. With the first pattern

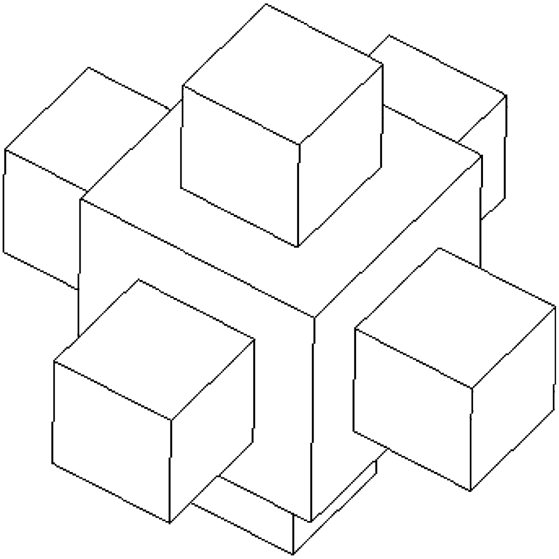


Fig. 4.129 Basic shape for complexity analysis

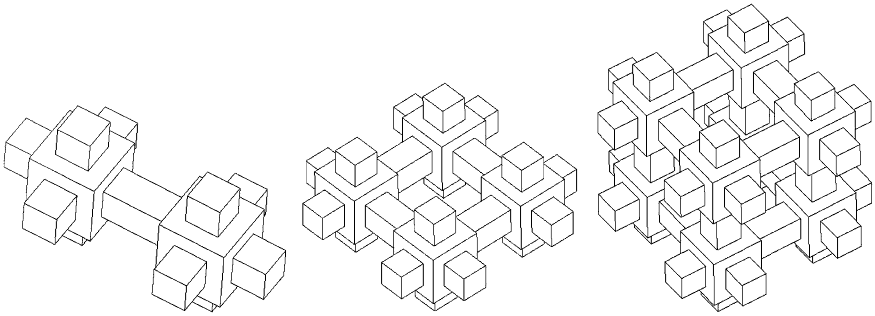


Fig. 4.130 Basic shape reflected

you should have 1,000 of these models, to calculate the number of elements in the models.

Another method is to use the symmetry operation. Reflect about each of the extrusions in turn, as shown in Fig. 4.130. This has a slightly more complex progression in the number of elements. The second shape has 104 vertices, 156 edges, 66 faces and 12 hole-loops. The third shape has 192 vertices, 288 edges, 120 faces, 24 hole-loops, it also has a genus of 1, but that is not recorded in the datastructure. The fourth shape has 352 vertices, 528 edges, 216 faces, 48 hole-loops, and a genus of 5. The numbers almost double each time but some elements are merged, hence the diminution. Note, if you use a pattern with distance of 80 instead of 82 you get the same kind of increase because the individual elements are merged. Count the number of times you can perform the symmetry operation to give a rough estimate of the number of elements allowed.

References

1. Stroud, I.A.: Boundary Representation Modelling Techniques. Springer, Heidelberg (2006)
2. Braid, I.C.: Designing with volumes. Ph.D. Dissertation, CANTAB Press, Cambridge Computer Laboratory, University of Cambridge (1974)
3. Braid, I.C.: Notes on a geometric modeller. CAD Group Document 101, Cambridge University Computer Laboratory (1979)
4. Mäntylä, M.: An Introduction to Solid Modeling. Computer Science Press, New York, ISBN 0-88175-108-1 (1988)
5. Stroud, I., Xirouchakis, P.C.: CAGD—computer-aided gravestone design. Adv. Eng. Software **37**(5): 277–286 (2006)
6. Jared, G.E.M., Stroud, I.A.: Local operators in the BUILD system, from advances in CAD/CAM. In: Proceedings of the 5th International IFIP/IFAC Conference on Programming Research and Operations Logistics in Advanced Manufacturing Technology PROLAMAT 82, pp. 55–65. North-Holland Publishing Company, Amsterdam (1982)
7. Bain, G.: Celtic Art the Methods of Construction. Constable, London (1951)

Chapter 5
Geometry

Geometry is one of the parameters of a modelling system. For curves and surfaces various "philosophies" have existed. Some early systems used planar approximations to the real geometry and so had facetted models. One system supplemented this facetted model with exact geometry so that real geometric calculations could be done as well to refine the approximation during modelling. Another philosophy was to use numerical forms for all the geometry, which complicates the algorithms but means that the system has a uniform approach. Perhaps the mainstream approach, though, is to have a mixture of simple analytic forms for common geometries and use numerical forms for everything else. This approach will be assumed here.

Note that there are many good books on different aspects of geometry. The first important one was by Faux and Pratt [1] on general geometry. Numerical geometry is a complex subject, though, and there exist other, more detailed books on different aspects of numerical geometry. One is by Farin [2] and another by Piegl and Tiller [3], which is on the important modern form, the NURBS form. Hoschek and Lasser [4] is yet another book which I find useful. It is not intended to go into great detail about numerical geometry in this chapter. The role of this chapter is to put geometry into a modelling and CAD perspective so other aspects are emphasised here.

5.1 Tolerances

Perhaps the first thing to mention is tolerances. These are not the same as design or manufacturing tolerances, these are numerical tolerances used for comparing values.

A list of tolerances, based on the work of Hans-Ulrich Pfeifer [5] is:

- Absolute tolerance—a measure of machine precision.
- Relative tolerance—a measure for comparing scaled values.

I. Stroud and H. Nagy, *Solid Modelling and CAD Systems*,
DOI: 10.1007/978-0-85729-259-9_5, © Springer-Verlag London Limited 2011

- Point tolerance—the minimum distance between distinct points.
- Angular tolerance—the minimum difference discernible between angular measures.
- Polynomial tolerance—A tolerance for polynomial evaluation.

You probably won't see all these, they may not all be present in the code, however you do sometimes need to know about tolerances and what they mean.

The problem arises with the representation of real numbers in computers. In general you should not write a line of code like:

$$\boldsymbol{if}\, a = b\ \boldsymbol{then}\ldots$$

where *a* and *b* are "real" numbers, that is, numbers which are not whole numbers or integers, and the "=" is a test for equality. In general, the test is done in this way:

$$\boldsymbol{if}\, abs(a - b) < tol\ \boldsymbol{then}\ldots$$

This is because real numbers are not exact, there might be a small error which is significant for a computer, even though it is not really large. It is usual to have statements like the following for comparing numbers in different ways.

```
$\boldsymbol{if}\,(a - b) > tol$ $\boldsymbol{then}$ $a$ is larger than $b$
$\boldsymbol{else\ if}\,(a - b) < -tol$ $\boldsymbol{then}$ $a$ is smaller than $b$
$\boldsymbol{else}$ $a$ is equal to $b$
```

The value of *tol* is the type of tolerance mentioned at the beginning of this section and is what you consider to be a reasonable difference, how to tell the computer what is "really" a significant difference. Work has been done on exact calculation using fractions, for example, but this is not the place to describe that.

In real terms it is to be expected that CAD systems will be able to work with differences of about 10^{-6}, though this depends on how many bits are assigned to representing a real value. If this is in millimetres then it is smaller than you can make, but it is still significant in computer terms. It is not possible just to increase the tolerance value arbitrarily because this will lose precision for the models. In early research this was called the "bolt-on-battleship" problem, because the size of a bolt with respect to the size of a battleship was small, but the bolt itself was not insignificant in size.

What you find, in several commands which use geometry, especially free-form geometry, let you specify the size of the tolerance used in the command. Tolerance values should certainly be less than 1, and probably not larger than 10^{-4} but it is difficult to be precise. Just be aware of the tolerances in commands and, if the system does not recognise geometric elements as being coincident that you think are coincident, it may be a tolerance problem.

5.2 Positions, Vectors and Transformations

This section describes some basic geometric elements.

5.2.1 Positions and Vectors

You are probably used to positions and vectors being simple three-element vectors: (x, y, z), but in modelling there is implicitly an extra element to make what is called "homogenous coordinates".

Homogenous coordinates have the form:

$$(x, y, z, w)$$

The "w" is a scaling factor dividing the vector elements, so this is equivalent to:

$$\left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}, 1\right)$$

This method of representing points allows them to be treated using transformation matrices, described in more detail in Chap. 13.

5.2.2 Transformations

Transformation matrices are 4 × 4 matrices which operate on homogenous coordinate systems. For the purposes of transformation matrices the homogenous coordinate element w can be considered to be 1.

The matrix has the form:

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix}$$

The sub-matrix:

$$\begin{bmatrix} a & b & c \\ e & f & g \\ i & j & k \end{bmatrix}$$

controls rotation and scaling. The column:

$$\begin{bmatrix} d \\ h \\ l \end{bmatrix}$$

controls translation. The bottom row:

$$[m \quad n \quad o]$$

may be all zeroes and, finally, the element:

$$[p]$$

is usually 1. Note, though, that if it is not 1 then this is equivalent to an implicit scaling because it changes the value of w in the homogenous coordinates.

Some common examples are:

Rotation by θ about the X-axis:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation by θ about the Y-axis:

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation by θ about the Z-axis:

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Translation by (d_1, d_2, d_3):

$$\begin{bmatrix} 1 & 0 & 0 & d_1 \\ 0 & 1 & 0 & d_2 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Uniform scaling by s:

$$\begin{bmatrix} s & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & s & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Unequal scaling by s_1, s_2, s_3 in the (x, y, z) directions:

$$\begin{bmatrix} s_1 & 0 & 0 & 0 \\ 0 & s_2 & 0 & 0 \\ 0 & 0 & s_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Note that it is common to allow only uniform scaling in CAD systems. This is because unequal scaling often changes the character of the geometric elements transformed. This is possible to do but the standard solution is to change the form of the geometry from simple, analytic forms to numerical geometry, so-called "geometric migration". This will be described later.

5.2.2.1 Transformation Combinations

Transformation matrices may be combined simply by multiplying them. In this way it is possible to decompose complex transformations into simple sub-steps. This has already been described in Sect. 4.3.

Suppose you have an object centred at (5, 10, 8) and you want to rotate the object about an axis through this centre in the Z-direction rather than the Z-axis through the origin. This can be done by moving the object to the origin, rotating it about the Z-axis through the global origin and then moving it back. The three matrices to do this are:

$$\begin{bmatrix} 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 10 \\ 0 & 0 & 1 & 8 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.866 & -0.5 & 0 & 0 \\ 0.5 & 0.866 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -5 \\ 0 & 1 & 0 & -10 \\ 0 & 0 & 1 & -8 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Instead of applying these separately, though these can simply be multiplied together to produce a single matrix:

$$\begin{bmatrix} 0.866 & -0.5 & 0 & 5.67 \\ 0.5 & 0.866 & 0 & -1.16 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This kind of transformation combination is used to accumulate a sequence of modifications for assemblies. They are also used for combining different graphics operations in a graphics "pipeline".

5.2.2.2 Transformation Blacklist

Obviously the values of the matrix elements are very general and could be anything, but certain combinations should be avoided.

Zero Scaling
The transformation matrix:

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

would “squash” all X values to be zero. A related matrix:

$$\begin{bmatrix} 0 & 0 & 0 & 50 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

would project the object onto the plane $X = 50$.

Projection matrices are useful for graphics purposes, but they are non-invertible. That is, you cannot recover the transformed geometry.

Shear Transformations
Shear transformations skew an object, creating changes to the coordinates which depend on original coordinate position. An example is the matrix:

$$\begin{bmatrix} 1 & 2 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

which would change the X position depending on the Y coordinate. This kind of matrix changes object geometry in strange ways.

5.2.2.3 Transformation Exercises

Use the shape in Fig. 5.1 for the exercises.

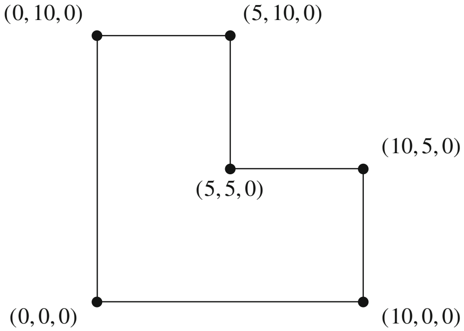


Fig. 5.1 Basic shape for transformation

5.2.2.4 Transformation 1

What does the following transformation do?

$$\begin{bmatrix} 0.866 & -0.5 & 0 & 0 \\ 0.5 & 0.866 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

5.2.2.5 Transformation 2

What does the following transformation do?

$$\begin{bmatrix} 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

5.2.2.6 Transformation 3

What does the following transformation combination do?

$$\begin{bmatrix} 0.866 & -0.5 & 0 & 5 \\ 0.5 & 0.866 & 0 & 2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

5.2.2.7 Transformation 4

What does the following transformation combination do and how is it related to the previous example?

$$\begin{bmatrix} 0.866 & -0.5 & 0 & 3.330 \\ 0.5 & 0.866 & 0 & 6.830 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

5.3 Analytic and Numerical (Free-Form) Geometry

With the mixture of analytic and numeric forms, the simple forms are used whenever possible and numerical, or free-form geometry is used for complex or unusual forms. This means that it is sometimes necessary to convert between geometric forms. “Geometric migration” is a term used to describe the change from analytic geometry to free-form, or numeric geometry.

A set of analytic curves might be:

Line
Circle
Ellipse

A set of analytic surfaces might be:

Plane
Cylinder
Cone
Sphere
Torus

There are several forms of free-form geometry, the most common are Bézier, BSpline and NURBS.

Exactly which type of geometry is used in the CAD system should be invisible to you. There have been different choices. One early implementation used only free-form geometry, a common philosophy now, though, is to mix analytic and free-form geometry. Analytic geometry is used for the common cases, free-form geometry is used for everything else. It is easy to give examples of uses of free-form geometry. One of these was given in the blending examples (Sect. 4.8.6.2, Fig. 4.83). If you intersect two cylinders of different radii, or if the axes do not intersect, then you get a complex intersection curve.

As stated above, the important thing is that you are not aware of what type of geometry is being used. About the only way I know of checking is to export a model using STEP AP203, for example, and read the resulting file. Free-form geometry “closes the gap”, that is, it provides a way of representing all geometry which cannot be represented by an explicit analytic form.

The notion of geometric “migration”, mentioned above, comes when an operation changes the form of the geometry. Consider a system which has only planes, circular cylindrical surfaces and free-form surfaces. If a cylinder, with the Z-axis as cylinder axis, is scaled in the X direction by 2, then the only way to represent this is to convert the analytic cylindrical surface into a free-form surface and scale the free-form surface unevenly. The analytic form becomes a free-form geometrical entity, hopefully without you realising it, but what happens if you now scale in the Y direction by 2? The cylindrical shape is re-established, but which geometry is now used?

The answer is probably that the free-form geometry is still used because it is more difficult to return and check that a free-form geometry corresponds to a simple form than to handle the free-form geometry. The disadvantage, though, is that analytic forms are more precise; they allow exact determination of some results and are less prone to small precision errors.

In general, be aware of the problem and avoid operations such as uneven scaling whenever possible.

5.4 Geometry and Modelling

As described in Stroud [6], as a general strategy in modelling, the topology and geometry are handled separately. Geometry is handled in terms of the general entities: point, curve, surface, and it is up to the geometric package to sort out the geometric types.

This means that the geometry is handled via a functional interface consisting of some general geometry handling functions. The main ones, from [6], are these:

1. Geometric intersection package.
2. Calculate a curve tangent.
3. Calculate a surface normal.
4. Calculate the parameter value of point on curve.
5. Calculate the parameter values of a point on a surface.
6. Calculate coordinates from a parameter value on a curve.
7. Calculate coordinates from parameter values on a surface.
8. Create a surface by sweeping an edge in a straight line, circular arc, or along a curve.
9. Modify geometry.
10. Reparametrise a curve.
11. Produce an offset curve from a given curve.
12. Produce an offset surface from a given surface.

If you have to develop extra code for a CAD system application, then what you expect to see is something like this list as part of what is called the "Application Programmer's Interface", or API. You do not need to know details of the data-structure, just be able to handle the general structures.

A widely used facility is the geometric intersection package. The six possible intersection combinations are as follows:

- Point–point intersection. This is a check to see whether two points are coincident.
- Point–curve intersection. This is a check to see whether a point lies on a curve.
- Point–surface intersection. Check if a point lies on a surface.
- Curve–curve intersection. Check (and return intersection results) if two curves cut each other at points, are coincident, or are partially coincident.
- Curve–surface intersection. Check (and return intersection results) if a curve lies on a surface, cuts through it, or is partially coincident with it.
- Surface–surface intersection. Check (and return intersection results) if two surfaces cut each other along a curve, touch at a point, are coincident, or partially coincident.

The intersection between the elements is then sorted out on the other side of the interface. This means that the geometry set can be changed without modifying code in many places.

The main geometric handling function, which you would use, would take two geometric entities of any type and, if you could see the code, might look like:

```
IF (point(g1) IS TRUE) THEN BEGIN
    IF (point(g2) IS TRUE) THEN point_X_point(g1, g2);
    ELSE IF (curve(g2) IS TRUE) THEN point_X_curve(g1, g2);
    ELSE IF (surface(g2) IS TRUE) THEN point_X_surf(g1, g2);
    END
ELSE IF (curve(g1) IS TRUE) THEN BEGIN
    IF (point(g2) IS TRUE) THEN point_X_curve(g2, g1);
    ELSE IF (curve(g2) IS TRUE) THEN curve_X_curve(g1, g2);
    ELSE IF (surface(g2) IS TRUE) THEN curve_X_surf(g1, g2);
    END
ELSE IF (surface(g1) IS TRUE) THEN BEGIN
    IF (point(g2) IS TRUE) THEN point_X_surf(g2, g1);
    ELSE IF (curve(g2) IS TRUE) THEN curve_X_surf(g2, g1);
    ELSE IF (surface(g2) IS TRUE) THEN point_X_surf(g1, g2);
    END
```

The curve_X_curve function would then handle the different types, breaking down the curve into sub-types, thus:

```
IF (straight(g1) IS TRUE) THEN BEGIN
    IF (straight(g2) IS TRUE) THEN straight_X_straight(g1, g2);
    ELSE IF (circle(g2) IS TRUE) THEN straight_X_circle(g1, g2);
    ELSE IF (ellipse(g2) IS TRUE) THEN straight_X_ellipse(g1, g2);
    ELSE IF (freeform(g2) IS TRUE) THEN
straight_X_freeform(g1, g2);
    END
ELSE IF (circle(g1) IS TRUE) THEN BEGIN
    IF (straight(g2) IS TRUE) THEN straight_X_circle(g2, g1);
    ELSE IF (circle(g2) IS TRUE) THEN circle_X_circle(g1, g2);
    ELSE IF (ellipse(g2) IS TRUE) THEN circle_X_ellipse(g1, g2);
    ELSE IF (freeform(g2) IS TRUE) THEN circle_X_freeform(g1, g2);
    END
ELSE IF (ellipse(g1) IS TRUE) THEN BEGIN
    IF (straight(g2) IS TRUE) THEN straight_X_ellipse(g2, g1);
    ELSE IF (circle(g2) IS TRUE) THEN circle_X_ellipse(g2, g1);
    ELSE IF (ellipse(g2) IS TRUE) THEN ellipse_X_ellipse(g1, g2);
    ELSE IF (freeform(g2) IS TRUE) THEN
ellipse_X_freeform(g1, g2);
    END
ELSE IF (freeform(g1) IS TRUE) THEN BEGIN
    IF (straight(g2) IS TRUE) THEN straight_X_freeform(g2, g1);
    ELSE IF (circle(g2) IS TRUE) THEN circle_X_freeform(g2, g1);
    ELSE IF (ellipse(g2) IS TRUE) THEN ellipse_X_freeform(g2, g1);
    ELSE IF (freeform(g2) IS TRUE) THEN freeform_X_freeform(g1, g2);
    END
```

Note the order of the geometric entities, g1 and g2, changes to make sure that they are in the correct order for the basic routines. Note, also, that the more analytic types there are, the more separate basic routines have to be implemented. This is a limiting factor which tends to keep the number of explicit, analytic geometric types to a minimum.

If a new curve type is introduced then the curve_X_curve only needs to be changed to called the new curve type functions. This usually happens during early development phases but by the time the CAD system comes to the market such changes are rarer. However, this should be invisible to the user.

The results have to be returned in a uniform set which has not only geometric entities, points, curves, or surfaces, but also extra information. The result set used in the pioneering BUILD system, was:

- COINCIDENCE (same orientation)
- COINCIDENCE (reverse orientation)
- POINT
- CURVE
- SELF-INTERSECTING CURVE
- SURFACE

The result set is a list of entities, which may be empty, have one element or several elements.

Similarly for the other functions. The curve tangent function takes a curve, you do not call "line_tangent", "circle_tangent", "ellipse_tangent", etc. functions yourself.

Exactly the same is done for surfaces, the programmer supplies a surface geometric entity to the interface and the interface breaks it down into types, plane, cylinder, cone, sphere, torus or free-form surface.

One thing to notice, though, is that all geometry is taken as parametric, not just the free-form geometry. Parameter values are very important for several algorithms that form the backbone of a CAD system. Drawing and Boolean operations, for example, both use parameters.

5.5 Working with Curves

The first thing to do is to extract the analytic curve types and handle these. The main interest comes, though, with free-form geometry, which is dealt with in more detail here. Note, though, that the presentation is based on work by other people and is an interpretation of that. If you are interested in the mathematical aspects and details then there are several good textbooks which can be consulted instead. See the list at the beginning of this chapter. The information presented here is intended to be enough to understand the CAD tools.

The principles for free-form curves and surfaces are very similar. It is easiest to explain the curves first and then extend this for surfaces. In this book the emphasis is on Bézier forms because these are easy to explain. The same types of operation are needed for other free-form curve types as well, but the explanation is more complicated.

5.5.1 Bézier Curves

Bézier geometry is an example of free-form geometry which has the advantage of having a simple mathematical background. These curves have been named after Professor Pierre Bézier, who developed them for Renault, but similar work was done by de Casteljau at Citroën and in the United States of America. Farin [2] gives a better account of this.

Bézier's idea was to provide designers with a more intuitive way for designing curves based on the end points and tangents at these end points.

The general form for a point on the curve is:

$$p(t) = \sum_{i=0}^{n} f(t) b_i$$

where the functions $f(t)$ are weighting functions, called the Bernstein polynomials, dependent on the parameter t. The terms b_i are a set of points, called "control points" which govern the shape of the resulting curve. The value of t usually varies between 0 and 1 (inclusive) for a curve, but the formula is valid for other values of t as well.

For Bézier curves the functions $f(t)$ are the expansion of:

$$((1-t)+t)^n$$

Giving the general formula for a Bézier curve as:

$$\sum_{i=0}^{n} \frac{n!}{i!(n-i)!}(1-t)^{n-i} t^i b_i$$

A quadratic Bézier curve has the form:

$$f(t) = (1-t)^2 b_0 + 2t(1-t)b_1 + t^2 b_2$$

A cubic Bézier curve (a common form) has the form:

$$f(t) = (1-t)^3 b_0 + 3t(1-t)^2 b_1 + 3t^2(1-t)b_2 + t^3 b_3$$

A quartic Bézier curve has the form:

$$f(t) = (1-t)^4 b_0 + 4t(1-t)^3 b_1 + 6t^2(1-t)^2 b_2 + 4t^3(1-t)b_3 + t^4 b_4$$

It is also possible to write these in matrix form. For example, for the cubic Bézier can be written as:

$$\begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

There are a number of important properties that can be observed.

1. From the function form, $((1 - t) + t)^n$ it is easy to see that the sum of the weighting functions, $f(t)$ must be 1 for any value of t.
2. When t = 0 only the first function is non-zero, so the curve passes through the first point. Similarly, when t = 1 only the last weighting function is non-zero, so the curve passes through the last point.
3. The tangent at the start of the curve is in the direction $b_1 - b_0$ and the tangent at the end of the curve is in the direction $b_3 - b_2$. This can be verified by differentiating the curve function with respect to t and substituting 0 and 1 into the differentiated version.
4. If t is between 0 and 1 then the curve lies within the convex hull of the control points.
5. The curve can be transformed (rotation, translation, scaling) by transforming its control points.
6. There are two descriptive elements for a curve the *degree* and the *order*. The *degree* is the power of the polynomials, the *order* is the number of points. *degree* = *order* − 1.

A simple example of a Bézier curve is shown in Fig. 5.2.

5.5.2 B-Spline Curves

B-spline curves are another common example of free-form geometry which has some advantages over the Bézier geometry described in the previous section. The“B” in the name stands for “Basis” and this works in a similar way to the Bézier curve, but with different functions.

Instead of having a simple set of basis functions, as with Bézier, the basis functions are derived using simple rules. This may seem a disadvantage but there are a number of benefits.

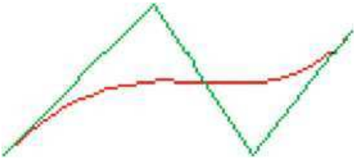


Fig. 5.2 Simple Bézier curve and control polygon

The basis functions for a cubic B-spline are:

$$\begin{gathered} t^3/6 \\ (1+3t+3t^2-3t^3)/6 \\ (4-6t^2+3t^3)/6 \\ (1-3t+3t^2-t^3)/6 \end{gathered}$$

There are different properties for B-splines to those for Bézier curves. For a start, only one of the equations is zero at t = 0 and at t = 1, which means that the curve does not pass through any of the control points. B-splines are useful for representing compound curves.

An example of a simple B-spline is shown in Fig. 5.3.

Stroud [6] gives an explanation of the basis function derivation, but you should consult dedicated geometric books, such as Faux and Pratt [1] or Farin [2].

These equations are shown plotted in Fig. 5.4.

In fact the conditions determine that the first basis function is a multiple of t^3 and the last a multiple of $(1-t)^3$.

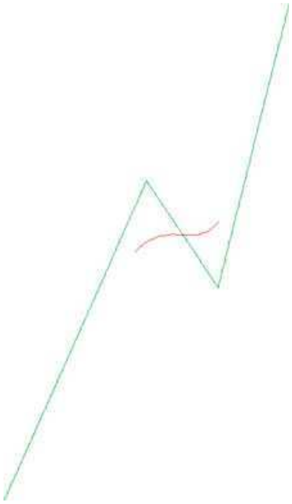
A simple conversion between Bézier and B-spline control points can be found by comparing the basis functions. Suppose the control points of two equal curves are $B_0^e, B_1^e, B_2^e, B_3^e$, and $B_0^s, B_1^s, B_2^s, B_3^s$, respectively.

Evaluating both sets of basis functions at $t = 0$ and at $t = 1$ gives:

$$\begin{aligned} B_0^e &= (B_0^s + 4B_1^s + B_2^s)/6 \\ B_3^e &= (B_1^s + 4B_2^s + B_3^s)/6 \end{aligned}$$

Differentiating the basis functions and evaluating these at t = 0 and t = 1 to find the tangents gives:

Fig. 5.3 Simple B-spline curve and control polygon



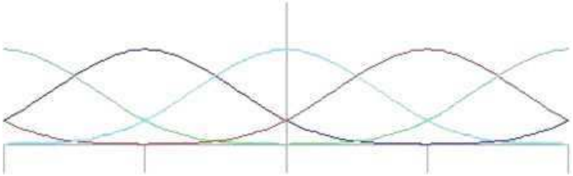


Fig. 5.4 Bézier basis functions

$$B_1^e = (2B_1^s + B_2^s)/3 \text{ and}$$
$$B_2^e = (B_1^s + 2B_2^s)/3$$

Solving the other way gives:

$$B_1^s = 2B_1^e - B_2^e$$
$$B_2^s = 2B_2^e - B_1^e$$

and:

$$B_0^s = 6B_0^e - 7B_1^e + 2B_2^e$$
$$B_3^s = 6B_3^e - 7B_2^e + 2B_1^e$$

Examining the control points for the two examples in Figs. 5.2 and 5.3, that is:

Bézier: (−1, −1, 0), (2, 2, 1), (4, −1, 5) and (6, 1.5, 7)

B-spline: (−12, −22, 3), (0, 5, −3), (6, −4, 9) and (12, 20, 9) which are the same curve, shows that these relationships hold.

The basis functions for a quadratic B-spline can be derived to give the functions:

$$t^2/2,$$
$$(-2t^2 + 2t + 1)/2,$$
$$(t^2 - 2t + 1)/2$$

For a quartic B-spline, you get the following basis functions:

$$t^4/24,$$
$$(-4t^4 + 4t^3 + 6t^2 + 4t + 1)/24,$$
$$(6t^4 - 12t^3 - 6t^2 + 12t + 11)/24,$$
$$(-4t^4 + 12t^3 - 6t^2 - 12t + 11)/24,$$
$$(t^4 - 4t^3 + 6t^2 - 4t + 1)/24$$

Substituting $(1 - t)$ for t in the B-spline basis equations and calculating the new equations gives the same equations as a result but in reverse order. This demonstrates that the equations are symmetric.

5.5.3 Rational Curve Forms

The geometric methods described thus far cannot be used to represent all shapes. As a simple example, a circle cannot be represented exactly. Take a Bézier form, for example, approximating a circle. You might consider a quadratic form with three control points at: $(0, r), (r, r), (r, 0)$. Evaluating this at $t = 0.5$ gives:

$$f(0.5) = 0.25(0, r) + 0.5(r, r) + 0.25(r, 0) = (0.75r, 0.75r)$$

For a circle the correct value would be: $\left(\frac{r\sqrt{2}}{2}, \frac{r\sqrt{2}}{2}\right)$.

The method that was developed to produce exact geometry is to assign weights to the control points. The rational Bézier form with weights looks like:

$$P(t) = \frac{(1-t)^n B_0 w_0 + nt(1-t)^{n-1} B_1 w_1 + \cdots + nt^{n-1}(1-t) B_{n-1} w_{n-1} + t^n B_n w_n}{(1-t)^n w_0 + nt(1-t)^{n-1} w_1 + \cdots + nt^{n-1}(1-t) w_{n-1} + t^n w_n}$$

For the quadratic example, above, you would have:

$$P(t) = \frac{(1-t)^2 (0, r) w_0 + 2t(1-t)(r, r) w_1 + t^2 (r, 0) w_2}{(1-t)^2 w_0 + 2t(1-t) w_1 + t^2 w_2}$$

According to Farin, w_0 and w_2 can be set to 1 without losing generality, which lets you solve for w_1 to find the weight for a circle. Doing this, with $t = 0.5$ gives:

$$P(t) = \frac{(1-t)^2 (0, r) + 2t(1-t)(r, r) w_1 + t^2 (r, 0)}{(1-t)^2 + 2t(1-t) w_1 + t^2}$$

$$\left(\frac{r\sqrt{2}}{2}, \frac{r\sqrt{2}}{2}\right) = \frac{0.25(0, r) + 0.5(r, r) w_1 + 0.25(r, 0)}{0.25 + 0.5 w_1 + 0.25}$$

$$\left(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}\right) = \frac{0.25(0, 1) + 0.5(1, 1) w_1 + 0.25(1, 0)}{0.25 + 0.5 w_1 + 0.25}$$

$$\left(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}\right) = \frac{(0.5 w_1 + 0.25, 0.5 w_1 + 0.25)}{0.5 + 0.5 w_1}$$

$$\frac{\sqrt{2}}{2} = \frac{0.5 w_1 + 0.25}{0.5 + 0.5 w_1}$$

$$\sqrt{2}(0.5 + 0.5 w_1) = w_1 + 0.5$$

$$\sqrt{2}(1 + w_1) = 2 w_1 + 1$$

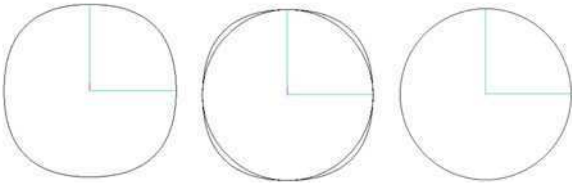


Fig. 5.5 Bézier and rational Bézier forms for a circle

$$\sqrt{2} - 1 = w_1(2 - \sqrt{2})$$

$$w_1 = \frac{\sqrt{2} - 1}{2 - \sqrt{2}} = \frac{(\sqrt{2} - 1)(2 + \sqrt{2})}{(2 - \sqrt{2})(2 + \sqrt{2})}$$

$$w_1 = \frac{\sqrt{2}}{2}$$

Figure 5.5 shows normal and rational Bézier forms corresponding to a circle. On the left is the standard Bézier form, on the right the same control points but with the weights set to $\frac{\sqrt{2}}{2}$ for the middle control point of each quarter circle. The two shapes are shown overlapping in the centre.

5.5.4 NURBS Curves

The geometric method which is most used at present is the NURBS method. The name NURBS stands for Non-Uniform Rational B-Splines. In this, the term “Non-Uniform” indicates that the basis functions need not be uniformly spaced, as with the other two forms, but are controlled by elements called “knots”.

Piegl and Tiller [3] define a NURBS curve as:

$$P(u) = \frac{\sum_{i=0}^{n} w_i * B_i * N_{i,k}(u)}{\sum_{i=0}^{n} w_i * N_{i,k}(u)}$$

and the B-spline basis functions $N_ik(t)$ are defined as:

$$N_{i,k}(u) = \frac{u - t_i}{t_{i+k} - t_i} * N_{i,k-1}(u) + \frac{t_{i+k+1} - t_{i+1}}{t_{i+k+1} - t_{i+1}} * N_{i+1,k-1}(u) \text{ with: } N_i, 0(u)$$
$$= 1 \text{ if } t_i < = u < t_{i+1}, 0 \text{ otherwise.}$$

Although NURBS geometry is the geometry used in advanced CAD systems I use Bézier geometry for the explanations because of its mathematical simplicity.

5.5.5 Multi-Piece and Closed Curves

It is sometimes necessary to use curves which are logically of one piece, but made up mathematically of a series of sub-portions. A related topic is that of closed curves where you need some degree of curve characteristic “similarity” at the ends. Note, this section talks about curves but the same discussions apply also to surfaces.

The term for similarity is “continuity”. Continuity comes in different levels:

- Continuity 0—Curves just touch.
- Continuity 1—Curves have the same tangent (direction).
- Continuity 2—Curves have the same curvature.
- Continuity 3—Curves have the same torsion.
- Curvature n—etc.

Depending on the degree of the curve you can go up and up. In fact, from a naive point of view, if you have curvature continuous quadratic curves then they will also be torsion continuous since both have torsion 0. However, this is not always so interesting.

Which is the best type of continuity to have between two curves?

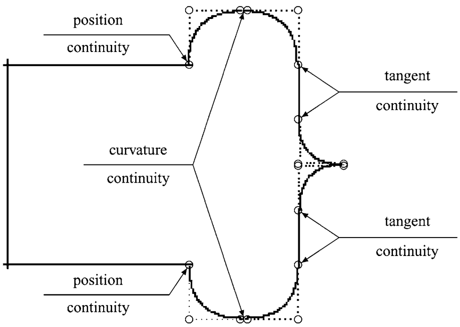
Wrong question. You need to decide what the curves are there for, then decide which type of continuity you need. Some CAD tools give you these options, some do not. For example, if you interpolate a set of points then the result may well be in terms of a set of curve segments rather than a single curve and so the curve may need a high level of continuity. If you are designing then maybe you only need tangent continuity.

As an illustration, consider the shape shown in Fig. 5.6. This shows a shape with various types of continuity. As should be obvious, all three types are useful even though the example shows a shape which is not typically an engineering shape.

Tangency continuity constrains two control points of a curve to be collinear with the constraining curve. Curvature continuity constrains the positions of three control points. Torsion continuity constrains four control points, and so on. This means that if you have a curve which you want to have particular continuity and you want to change the shape, then you have to add control points. This technique is called “raising the degree” and comes later. Figure 5.7 illustrates this.

Figure 5.7a shows the original arrangement, with a quadratic Bézier curve constrained to be tangent continuous with two lines. The control polygon is shown dotted. If you want to change the shape of this curve then you cannot without breaking the tangency constraints. In order to give more freedom you can change the curve into cubic Bézier, as shown in Fig. 5.7b. This lets you change the two internal control points. So long as they remain collinear with the constraining lines then you can move them, flattening or sharpening the curve. If you want still more freedom then you can raise the degree of the curve to a quartic, as shown in Fig. 5.7c. Here you have one totally free control point, shown as an open circle,

Fig. 5.6 Continuity types



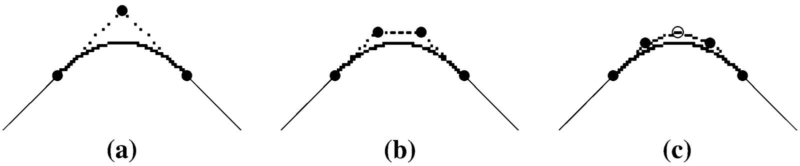


Fig. 5.7 Degree raising for manipulating constrained curves

which you can move as you want to change the shape of the curve. The four outer control points maintain the tangency constraints.

If you have a tool to show the control points and can constrain a curve to different extents then you will be able to see the growth of the control points as you impose the constraints. If, say, you were to impose a curvature constraint between the lines and the quadratic Bézier, so that the curve changes, then you would see an immediate increase in the number of control points.

5.5.6 *Nasty Curves*

Since free-form geometry is controlled by point positions it is possible to create many special cases. Figure 5.8 shows a simple two-dimensional Bézier curve with the control points: (−10, 0) (10, 10) (−10, 10) (10, 0).

What is the position at $t = 0.5$ and what is the tangent vector?

For convenience it is possible to use the vector form of the Bézier:

$$f(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} (-10,0) \\ (10,10) \\ (-10,10) \\ (10,0) \end{bmatrix}$$

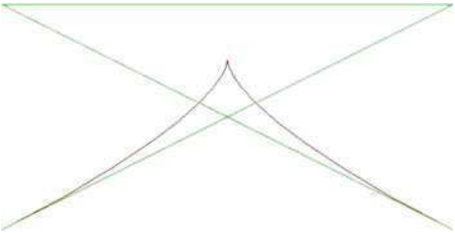


Fig. 5.8 Bézier curve with singularity

Multiplying the 4 × 4 matrix with the control points allows you to calculate easily the position and first derivative of the curve.

$$f(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} (80, 0) \\ (-120, -30) \\ (60, 30) \\ (-10, 0) \end{bmatrix}$$

For $t = 0.5$ you get:

$$f(0.5) = [0.125 \quad 0.25 \quad 0.5 \quad 1] \begin{bmatrix} (80, 0) \\ (-120, -30) \\ (60, 30) \\ (-10, 0) \end{bmatrix} = (0, 7.5)$$

For the derivative of the curve, which is the tangent, you use the derivative of the matrix with terms in t, thus:

$$f'(t) = \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix} \begin{bmatrix} (90, 0) \\ (-120, -30) \\ (60, 30) \\ (-10, 0) \end{bmatrix}$$

giving:

$$f'(0.5) = [0.75 \quad 1 \quad 1 \quad 0] \begin{bmatrix} (90, 0) \\ (-120, -30) \\ (60, 30) \\ (-10, 0) \end{bmatrix} = (0, 0)$$

meaning that the tangent has no direction at the point, which is a "singularity". Alternatively, differentiate the Bézier function with respect to t.

$$f(t) = (1-t)^3 b_0 + 3t(1-t)^2 b_1 + 3t^2(1-t) b_2 + t^3 b_3$$

$$f'(t) = -3(1-t)^2 b_0 + 3((1-t)^2 - 2t(1-t)) b_1 + 3(2t(1-t) - t^2) b_2 + 3t^2 b_3$$

Simplifying gives:

$$f'(t) = (-3 + 6t - 3t^2)b_0 + (3 - 12t + 9t^2)b_1 + (6t - 9t^2)b_2 + 3t^2b_3$$

or:

$$f'(t) = (3b_3 - 9b_2 + 9b_1 - 3b_0)t^2 + (6b_2 - 12b_1 + 6b_0)t + (3b_1 - 3b_0)$$

Substituting the values for the control points gives two equations, one for x and one for y.

$$f'_x(t) = (30 + 90 + 90 + 30)t^2 + (-60 - 120 - 60)t + (30 + 30) = 240t^2 - 240t + 60$$
$$f'_y(t) = (0 - 90 + 90 - 0)t^2 + (60 - 120 + 0)t + (30 - 0) = -60t + 30$$

Finding the roots of these gives a double root for x at 0.5 and a root for y, also at 0.5. If only x has a root, then the tangent is vertical. If only y has a root then the tangent is horizontal. When both have coincident roots there is a singularity. Note also that the curvature at $t = 0.5$ is infinite, as can be understood by looking at the formulae in Sect. 5.9.

However, despite all this, the curve might be useful anyway as part of a model if a portion of the curve without the singularity is used. If, say, an edge refers to the portion of the curve between $t = 0$ and $t = 0.4$ then the curve is defined along the whole portion of the edge using it. However, it is better to avoid such curves if possible because, if an operation needs to extend the edge for some reason then the singularity might become part of the edge, causing problems.

Figure 5.9 shows another simple two-dimensional Bézier curve with the control points: (−5, 0) (10, 10) (−10, 10) (5, 0).

This curve has no singularities, but has a self-intersection. The curve has the same position at the parameter values $t = 0.172673$ and $t = 0.827327$, approximately. This does not have the same significance as the singularity because the curve is fully defined at all places. As with the previous example, the curve can be used in part without problems but, as before, there is a risk that, if the edge is extended the self-intersecting part will fall within the edge.

Another possible cause of awkward geometry is if control points coincide. This causes discontinuities of various types and so should be avoided. This should not be a problem for most users, but if you do manipulate control points directly, avoid putting them in the same place.

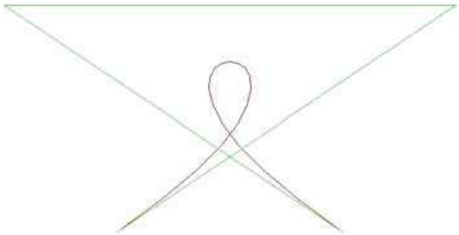


Fig. 5.9 Self-intersecting Bézier curve

5.5.7 Raising the Degree of a Bézier Curve

For the Bézier curve this is simple. Figure 5.10 shows a simple example.

Starting with a simple quadratic curve, shown at the top of the figure, new control points are added according to a simple rule. If the original control points are denoted p_0^0, p_1^0 and p_2^0, then two new points are added, p_1^1 and p_2^1. $p_1^1 = p_0^0 + \frac{2}{3}(p_1^0 - p_0^0)$. $p_2^1 = p_1^0 + \frac{1}{3}(p_2^0 - p_1^0)$. The reason for the value $\frac{2}{3}$ is that you have two spans (between p_0^0 and p_1^0; and between p_1^0 and p_2^0) which should become three spans in the cubic curve. The control points for the cubic Bézier curve with the same form as the original quadratic curve are: p_0^0, p_1^1, p_2^1 and p_2^0.

To convert this cubic curve to a quartic curve you follow the same procedure, but the ration is $\frac{3}{4}$ because you are going from three spans to four spans. The intermediate points are:

$$\begin{aligned} p_1^2 &= p_0^0 + \tfrac{3}{4}\left(p_1^1 - p_0^0\right) \\ p_2^2 &= p_1^1 + \tfrac{1}{2}\left(p_2^1 - p_1^1\right) \\ p_3^2 &= p_2^1 + \tfrac{1}{4}\left(p_2^0 - p_2^1\right). \end{aligned}$$

The final control points for the quartic curve are: $p_0^0, p_1^2, p_2^2, p_3^2$ and p_2^0. You may ask about the ratio of $\frac{3}{4}$, which is in evidence in the first equation but then seems to disappear. To understand this, think of the three spans as being numbered 0, 1 and 2. The span between p_0^0 and p_1^1 is span 0, the second, between p_1^1 and p_2^1 is span 1 and the span between p_2^1 and p_2^0 as being span 2. The first extra point is at $\frac{3}{4}$ of the spans, the second point is at $\frac{3}{4} + \frac{3}{4} = 1\frac{1}{2}$ and the third at $\frac{3}{4} + \frac{3}{4} + \frac{3}{4} = 2\frac{1}{4}$. So, the new points are at ratio $\frac{3}{4}$ of span 0, $\frac{1}{2}$ of span 1 and $\frac{1}{4}$ of span 2.

Finally, to go from a quartic curve to a quintic curve you have the ratio $\frac{4}{5}$. The new intermediate points are:

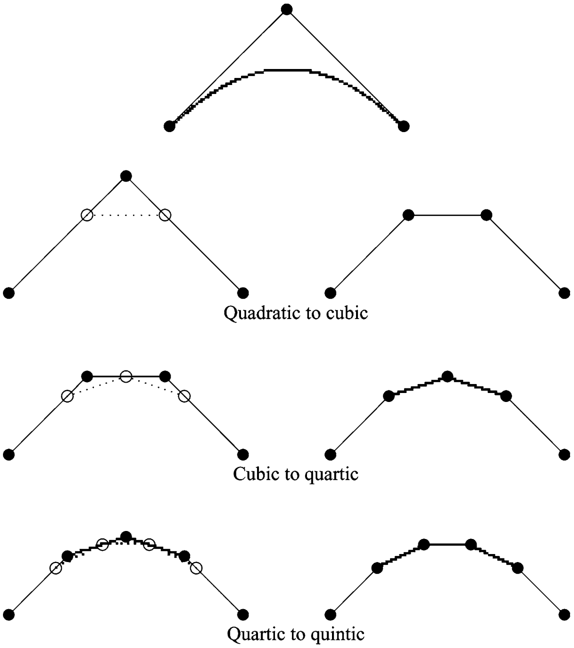
$$\begin{aligned} p_1^3 &= p_0^0 + \tfrac{4}{5}\left(p_1^2 - p_0^0\right) \\ p_2^3 &= p_1^2 + \tfrac{3}{5}\left(p_2^2 - p_1^2\right) \\ p_3^3 &= p_2^2 + \tfrac{2}{5}\left(p_3^2 - p_2^2\right) \\ p_4^3 &= p_3^2 + \tfrac{1}{5}\left(p_2^0 - p_3^2\right). \end{aligned}$$

The final control points are: $p_0^0, p_1^3, p_2^3, p_3^3, p_4^3$ and p_2^0.

Note how the control polygon gets closer to the shape of the curve as you raise the degree. You can also verify that the curve is the same by substituting the new control points into the appropriate formulae.

Raising the degree of B-spline curves is more difficult. This is dealt with by Farin [2], for example.

Fig. 5.10 Raising the degree of a Bézier curve



5.5.8 Subdividing Bézier Curves

De Casteljau's method can be used for subdividing Bézier curves. This is shown in Fig. 5.11.

To subdivide the curve at t, there is a recursive procedure. The first stage is to create a secondary control polygon based on the original polygon. New points are created at a ratio of t along each span. These points are then connected to create the new control polygon which has one span less than the original polygon. The process is repeated until there is only one span and the division point is a point at a ratio of t along the final span. Certain of the new points are then used to create the control polygons of the divided curve.

In Fig. 5.11 the value of t for subdivision is 0.6. Figure 5.11a shows the original cubic curve and control polygon with four control points: p_0^0, p_1^0, p_2^0 and p_3^0. In Fig. 5.11b the first-level secondary polygon is shown, connecting the three new points p_0^1, p_1^1 and p_2^1. $p_0^1 = (1-t)p_0^0 + tp_1^0$, where t = 0.6, so $p_0^1 = (1-0.6)p_0^0 +$ $0.6p_1^0$. $p_1^1 = (1-0.6)p_1^0 + 0.6p_2^0$ and $p_2^1 = (1-0.6)p_2^0 + 0.6p_3^0$. In Fig. 5.11c the second-level secondary polygon is shown, a line connecting two points: p_0^2 and p_1^2. $p_0^2 = (1-0.6)p_0^1 + 0.6p_1^1$ and $p_1^2 = (1-0.6)p_1^1 + 0.6p_2^1$. Finally, the separation point $p_0^3 = (1-0.6)p_0^2 + 0.6p_1^2$.

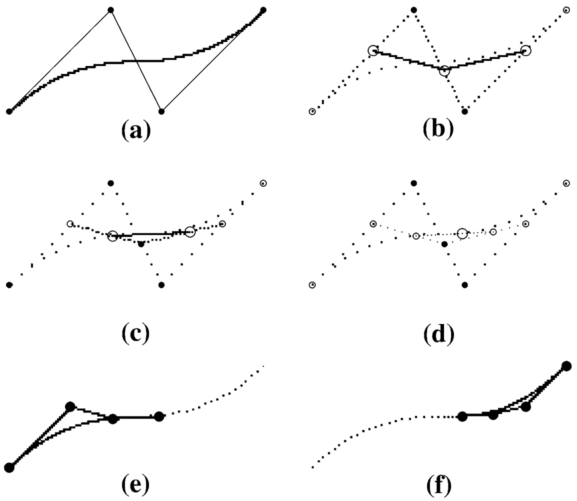


Fig. 5.11 Subdividing a Bézier curve using de Casteljau's method

The control points for the first part of the subdivided curve are: p_0^0, p_0^1, p_0^2 and p_0^3. The control points for the other half are: p_0^3, p_1^2, p_2^1 and p_3^0.

This can be conveniently written in pyramid form as:

$$\begin{array}{ccccccc}
 & & & p_0^3 & & & \\
 & & p_0^2 & & p_1^2 & & \\
 & p_0^1 & & p_1^1 & & p_2^1 & \\
p_0^0 & & p_1^0 & & p_2^0 & & p_3^0
\end{array}$$

For a simple, two dimensional Bézier example, with control points $(0,0), (50,50), (75,0)$ and $(125,50)$, and subdividing at $t = 0.6$ you have:

$$\begin{array}{ccccccc}
 & & & (73.8, 25.2) & & & \\
 & & (51,24) & & (89,26) & & \\
 & (30,30) & & (65,20) & & (105,30) & \\
(0,0) & & (50,50) & & (75,0) & & (125,50)
\end{array}$$

5.5.9 Interpolation

Curve "interpolation" is a technique for determining a curve passing through a given set of points. The problem to be solved is that you have a set of points and want to pass a curve through them. The problem is that there are an infinite number of solutions for the result. What creates the different solutions is the choice of the parameter values of the points on the solution curve.

Take, for example, a curve which spans four points:

$$(-2,-2)(1,2)(4,-2)(16,3)$$

Take the parameter values of the points as 0, t_1, t_2 and 1, respectively. It is possible to formulate the problem in terms of a matrix, which gives a method of solution more tractable for a computer. Formulating the problem for a cubic Bézier curve in matrix terms gives:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ (1-t_1)^3 & 3t_1(1-t_1)^2 & 3t_1^2(1-t_1) & t_1^3 \\ (1-t_2)^3 & 3t_2(1-t_2)^2 & 3t_2^2(1-t_2) & t_2^3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

where b_0, b_1, b_2, and b_3 are the unknown control points and p_0, p_1, p_2, and p_3 are the points to interpolate. In order to find the points of control it is necessary to multiply by the inverse, giving:

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

where the 4 × 4 matrix is the inverse of the matrix formed from the Bézier basis functions. For the Bézier curve case above it is possible to simplify the inverse directly because $a = 1, b = c = d = 0$ and also $m = n = o = 0$ and $p = 1$. Similarly f, g, j and k are simple to write down because they are just the inverse of the central 214:132 portion of the matrix, giving the simplified matrix:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ e & \frac{3}{q}t_2^2(1-t_2) & \frac{-3}{q}t_1^2(1-t_1) & h \\ i & \frac{-3}{q}t_2(1-t_2)^2 & \frac{3}{q}t_1(1-t_1)^2 & l \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where $q = 9(t_1t_2(1-t_1)(1-t_2))(t_2(1-t_1) - t_1(1-t_2))$.

The next step is to choose the values for t_1 and t_2. Two "obvious" methods are the equal step value, which would mean $t_1 = \frac{1}{3}$ and $t_2 = \frac{2}{3}$, and the arc-length method. The arc-length method seems to give better results than with even spacing and seems to be commonly used in CAD systems.

Figure 5.12 illustrates the principle. The total length of the curve is taken as approximately $l_1 + l_2 + l_3$ and the internal parameter values $t_1 = \frac{l_1}{l_1+l_2+l_3}$ and $t_2 = \frac{l_1+l_2}{l_1+l_2+l_3}$.

The final values can be determined by substituting the arc-length values, that is, $t_1 = 5/23$ and $t_2 = 10/23$ in the matrices above, which gives:

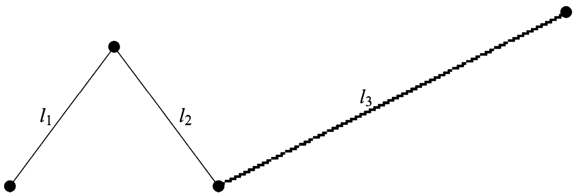


Fig. 5.12 Arc-length parameter estimation

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.479329 & 0.399441 & 0.110956 & 0.010274 \\ 0.180570 & 0.416701 & 0.320539 & 0.082190 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} (-2,-2) \\ (1,2) \\ (4,-2) \\ (16,3) \end{bmatrix}$$

or:

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1.6333 & 3.9185 & -1.3564 & 0.0712 \\ 1.56 & -5.0941 & 4.8831 & -0.3490 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} (-2,-2) \\ (1,2) \\ (4,-2) \\ (16,3) \end{bmatrix}$$

which gives the control points:

$$(-2,-2),(2.899146,14.030200),(5.734186,-24.121311),(16,3)$$

5.5.10 Curve Experiments

5.5.10.1 Interpolating a Circle

Create four points on the XY plane at positions: (−100, 0), (0, −100), (100, 0) and (0, 100). These lie on a circle, radius 100, centred at (0, 0). Interpolate a curve passing through the four points. Is it a circle?

The problem is that CAD systems interpolate as they go, giving you a result which is built up successively, rather than taking all the points at the end. The result of this exercise is not usually a circle. In any case, note that the curve at the start and end should be contangential. Create a line, through (−100, 0), direction (0, 1), to set the tangents for the start and end of the curve.

5.5.10.2 Make a Self-Intersecting Curve

Create five points:

$$(-10,-10,0),(2,5,0),(0,10,0),(-2,5,0) \text{ and } (10,-10,0).$$

Create a curve passing through these in the order shown. Does your CAD system warn you that you have created such a curve?

5.5.10.3 Make Another Self-Intersecting Curve

Create seven points:

$$(-10,-10,0),(-0.5,5,0),(-0.5,8,0),(0,12,0),$$
$$(0.5,8,0)(0.5,5,0)\,\text{and}\,(10,-10,0)$$

Create a curve passing through these in the order shown. Does your CAD system warn you this time that you have created a self-intersecting curve?

Something else to note about this curve is the direction of the curve at the start and end points. The curve is more complex, and hence a bit wilder than the first self-intersecting curve. If you want to control the shape of the curve at the start and end then you need to create extra curves at these points so that you can impose continuity constraints. Try this with straight lines from (−20, −10, 0) to (−10, −10, 0) and (10, −10, 0) to (20, −10, 0).

5.5.10.4 Continuity Experiment

Try making a circle and a line which is tangent. If your CAD system permits, create a constraint of curvature continuity between the line and circle. One of them has to deform, in I-DEAS it was the second curve selected. Try it in both directions and check the deformations. What happened in I-DEAS was that the curve which deformed was converted to a free-form curve and the control points of the curve placed accordingly. This is another example of the sort of geometric migration mentioned earlier where simple forms are converted to free-form curves.

5.6 Working with Surfaces

The most commonly used surfaces are a type called “tensor product surfaces”. Actually, there are other surface types with three sides, but in current systems there are usually four sides, so only four-sided surface patches are mentioned here. Each four-sided surface element may also be termed a “patch”, because multi-patch surfaces are sometimes generated to fill regions.

Some examples are given in Fig. 5.13. In Fig. 5.13a you have an example of a simple translational surface. In Fig. 5.13b there is an example of a ruled surface. In Fig. 5.13c you have a surface patch which is quadratic in one direction and cubic in the other. In Fig. 5.13d you have a patch which is cubic in both directions.

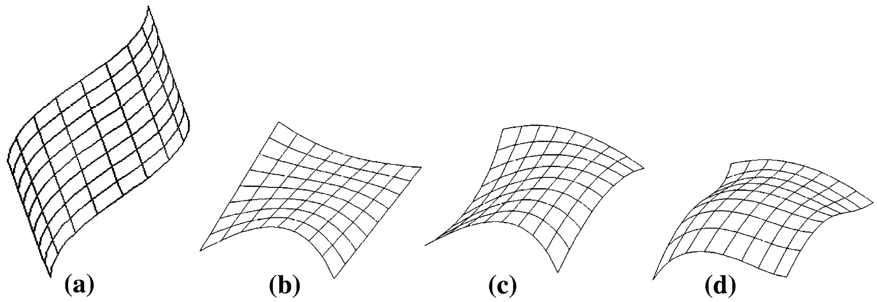


Fig. 5.13 Surface patch examples

5.6.1 Bézier Surfaces

The general form of a Bézier surface patch is:

$$\sum_{i=0}^{n}\sum_{j=0}^{m}\frac{n!}{i!(n-i)!}\frac{m!}{j!(m-j)!}(1-u)^{n-i}u^{i}(1-v)^{m-j}v^{j}b_{ij}$$

The two values m and n are the orders of the patch in the v and u directions, respectively. These need not be the same. In the examples in Fig. 5.13 the surfaces are of degree 3 in one direction, and degree one (Fig. 5.13a, b), two (Fig. 5.13c) and three (Fig. 5.13d) in the other direction.

The surface is defined over a parametric space of $0 \le u \le 1$ and $0 \le v \le 1$.

If you look at the description it can be seen that this is like multiplying two curve functions together, except that the control points are combined. A surface can be thought of as a continuous set of curves. Defining a point at parameter position u,v can be done by evaluating the surface equations at u, which gives the control points of a secondary curve. Evaluating this secondary curve at v gives the point on the surface.

5.6.2 Creating Surfaces by Extrusion

Extrusion is an easy way of creating surfaces. Taking any curve, the resulting patch has the original curve as one boundary, a translated copy of the curve as its matching boundary, and two linear sections.

Suppose you have a Bézier curve with the control points:

$$(0,0,0),(10,10,0),(20,0,0),(30,10,0)$$

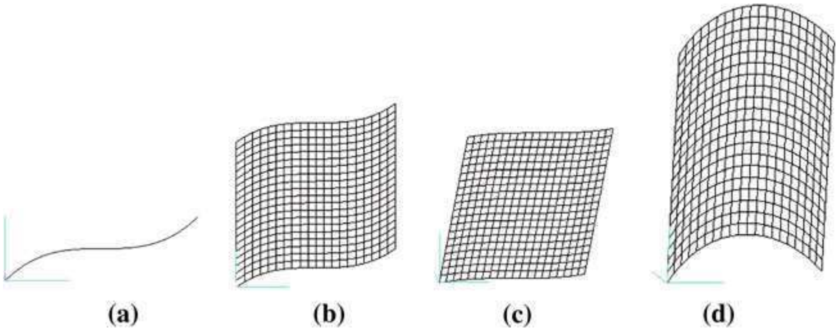


Fig. 5.14 Extruding a curve to obtain surfaces

Sweeping this curve with the vector (0, 0, 40) gives a cubic-linear patch with the control points:

$$\begin{bmatrix} (0,0,0) & (10,10,0) & (20,0,0) & (30,10,0) \\ (0,0,40) & (10,10,40) & (20,0,40) & (30,10,40) \end{bmatrix}$$

The curve in the example is a planar curve and the vector is normal to this plane. If the vector were: (5, −3, 30) then the surface would not be perpendicular to the curve plane. The curve does not need to be planar either, the same method applies whatever the original control points, although it is preferable that these are not linear and in the extrusion direction. These examples are shown in Fig. 5.14. The original curve is shown in Fig. 5.14a. In Fig. 5.14b you have the surface created when extruding by a vector (0, 0, 40). In Fig. 5.14c you have the surface extruded along the vector (5, −3, 30). Finally, in Fig. 5.14d you have a curve with the control points: (0, 0, 0), (10, 10, 10), (20, 0, 10), and (30, 10, 0) extruded along the vector (0, 0, 40).

5.6.3 Lofting

The lofting operation as a solid modelling operation has already been described in Sect. 4.14. This section describes how the geometry for the solid is created.

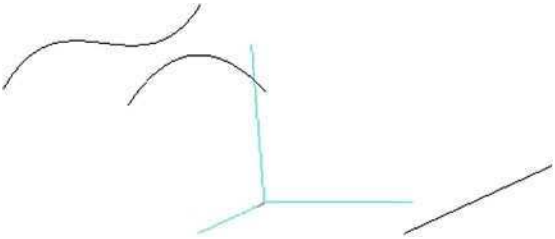
Figure 5.15 shows three arbitrary curves across which it is desired to create a surface. Suppose that the first curve is a cubic Bézier with control points:

$$(-10,-15,10),(-10,-5,5),(-10,5,15) \text{ and } (-10,15,10)$$

The second curve is a quadratic Bézier with control points:

$$(-4,-10,5),(-4,0,12) \text{ and } (-4,10,8)$$

Fig. 5.15 Curves for lofting



The final curve is a line from $(14, -12, 0)$ to $(14, 10, 0)$. For the interpolation the second section is considered to be at $t = 0.25$ for all points, just to simplify the explanation.

The first step is to make sure that all curves have the same degree. Since the highest degree is a cubic, then it is necessary to raise the degree of the quadratic curve once and the line twice.

The quadratic goes from:

$$(-4, -10, 5)(-4, 0, 12)(-4, 10, 8)$$
$$\text{to:}$$
$$(-4, -10, 5)(-4, -3.333, 9.667)(-4, 3.333, 10.667)(-4, 10, 8)$$

The line goes from:

$$(14, -12, 0)(14, 10, 0)$$
$$\text{to:}$$
$$(14, -12, 0)(14, -1, 0)(14, 10, 0)$$
$$\text{to:}$$
$$(14, -12, 0)(14, -4.667, 0)(14, 2.667, 0)(14, 10, 0)$$

The interpolation matrix is:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0.5625 & 0.375 & 0.0625 \\ 0 & 0 & 1 \end{bmatrix}$$

and its inverse is:

$$\begin{bmatrix} 1 & 0 & 0 \\ -1.5 & 2.667 & -0.1667 \\ 0 & 0 & 1 \end{bmatrix}$$

The two side curves are found using this matrix.

$$\begin{bmatrix} 1 & 0 & 0 \\ -1.5 & 2.667 & -0.1667 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} (-10, -15, 10) \\ (-4, -10, 5) \\ (14, -12, 0) \end{bmatrix} = \begin{bmatrix} (-10, -15, 10) \\ (2, -2.166667, -1.666667) \\ (14, -12, 0) \end{bmatrix}$$

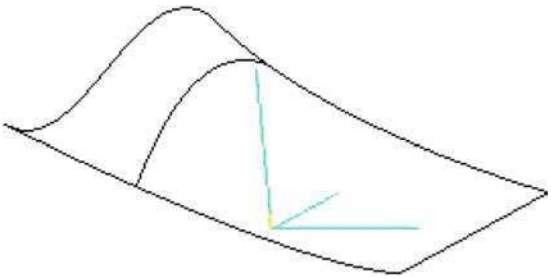


Fig. 5.16 Curves for lofting and side curves

and

$$\begin{bmatrix} 1 & 0 & 0 \\ -1.5 & 2.667 & -0.1667 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} (-10, 15, 10) \\ (-4, 10, 8) \\ (14, 10, 0) \end{bmatrix} = \begin{bmatrix} (-10, 15, 10) \\ (2, 2.5, 6.333333) \\ (14, 10, 0) \end{bmatrix}$$

The resulting curve set is shown in Fig. 5.16.

To find the internal surface control points it is necessary to interpolate the two sets of control points:

$$\begin{bmatrix} 1 & 0 & 0 \\ -1.5 & 2.667 & -0.1667 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} (-10, -5, 5) \\ (-4, -3.333, 9.667) \\ (14, -4.667, 0) \end{bmatrix} = \begin{bmatrix} (-10, -5, 5) \\ (2, -0.611, 18.2778) \\ (14, -4.667, 0) \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ -1.5 & 2.667 & -0.1667 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} (-10, 5, 15) \\ (-4, 3.333, 10.667) \\ (14, 2.667, 0) \end{bmatrix} = \begin{bmatrix} (-10, 5, 15) \\ (2, 0.944, 5.944) \\ (14, 2.667, 0) \end{bmatrix}$$

This gives the surface shown in Fig. 5.17, with the final set of control points as:

$$\begin{matrix} (-10, -15, 10) & (2, -2.167, -1.667) & (14, -12, 0) \\ (-10, -5, 5) & (2, -0.611, 18.278) & (14, -4.667, 0) \\ (-10, 5, 15) & (2, 0.944, 5.944) & (14, 2.667, 0) \\ (-10, 15, 10) & (2, 2.5, 6.333) & (14, 10, 0) \end{matrix}$$

5.6.4 N-Sided Patches

Mathematical surfaces are nice regular structures with four sides. The world is an awkward uneven place with surface regions with multiple sides. What do you do?

There has been a lot of work on this subject. The topic was introduced in Sect. 4.8 where one of the examples concerns a six-sided surface patch. The

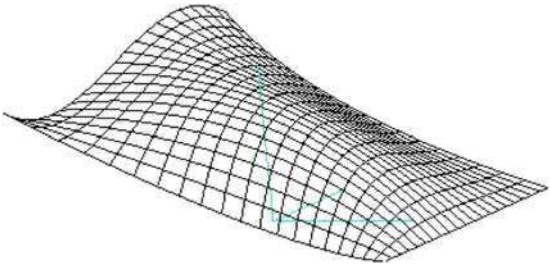


Fig. 5.17 Lofted surface

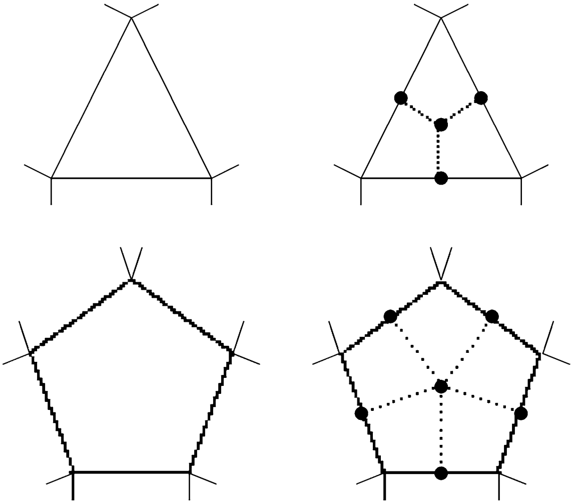


Fig. 5.18 Subdividing non four-sided areas

intention there was to perform an experiment to see the solution in the CAD system you are using.

A solution which seems to be in common use is that shown in Fig. 5.18. A point is created at the middle of each edge in the N-sided region and joined to the centre point. This creates N four-sided regions instead of one N-sided region.

The problem is that the position and surface normal at the centre point are not defined. This data has to be estimated, but the technique works quite well. Depending on how the system works, you may be able to see this result by blending a multi-edge vertex and looking at the result in wireframe graphics mode.

Note, though, that this is not the only solution. It is something that the CAD system developer chooses, it is another of the topics in the complex area of surface modelling.

5.6.5 Filling Areas

A common operation is filling boundary curves. Figure 5.19 shows a simple example of a four-sided filling problem.

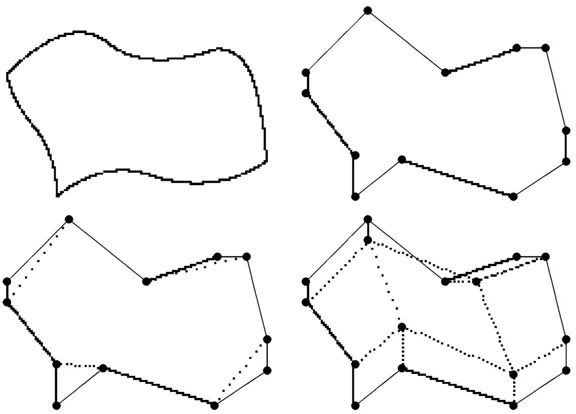


Fig. 5.19 Filling a square frame

The original curves are shown on the top left of the figure. These have control polygons, as shown in the image on the top right. The corner control points define planes, in this case, as shown on the bottom left of the figure. Control points lying on these corner planes preserve the necessary surface characteristics, as shown on the bottom right.

5.6.6 Surfaces on Curve Frameworks

Another common method is to create a set of curves and then add a surface. This is a method of working in the car industry, for example, where designers can set up a system of space curves representing "style lines". The car surfaces are then created between these curves.

This is related to the method of putting a surface in a boundary and lofting. This lets you take an arbitrary boundary and cover it, with the addition that you can define an internal structure with extra curves.

The extra information in the curve structures helps in setting the internal control points of the surface pieces. The positions of the curves, tangent and curvature continuity across the curves, all set up constraints on the positions of the control points.

However, it is not arbitrary how you arrange these curves. As an example, consider Fig. 5.20.

The corner points for the surfaces are shown, but note especially the points marked p_1 and p_2. Here, the tangent vectors of the side curves which meet there are collinear. This means that the surface normal disappears to zero and so the position of the control point is not well-defined. A possible solution may be to add extra curves, as shown in Fig. 5.21.

This, though, also brings problems. There are two three-sided surfaces. A three sided surface can be created by allowing the control points along one of the edges

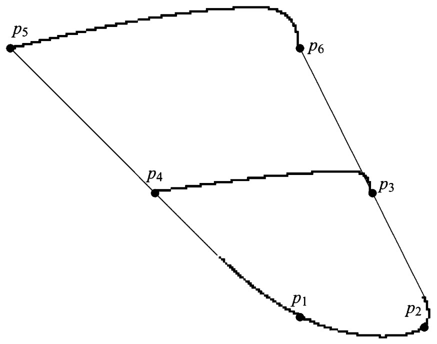


Fig. 5.20 Surface on curve framework

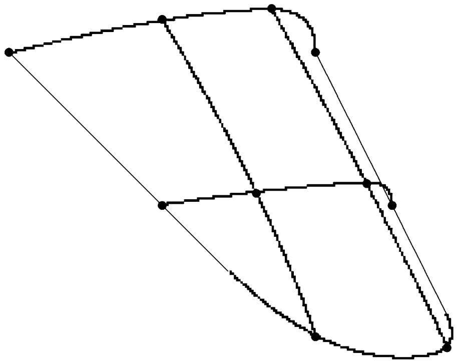


Fig. 5.21 Surface on curve framework with internal curves

to coincide. This means that there is, again, a problem with degenerate surface normals along the degenerate side. This can create continuity problems. The solution of inserting a central point and subdividing the three-sided patch into three four-sided patches has already been described. Yet another solution is to create a real four-sided patch which extends outside the region being filled, but only half of it is used.

5.6.7 Surfaces and Topology

This is less of a subsection than a comment. Surfaces in common use in CAD have a regular, four-sided structure. This is not absolutely necessary because there are three-sided patches, but three-sided patches do not seem to be in common use in CAD systems, so are ignored here. In surface modelling it is helpful if all the surface patches match exactly along one side. This is not always convenient for designing with surfaces. The method described for dealing with N-sided areas creates mismatched patches, for example. Várady and his team worked on general topology methods for surface modelling, but this will not be discussed here.

What is relevant, I think, is to know that mismatching surfaces mean that the control points do not match and so there is a risk that small geometric errors may creep in creating gaps. This is a limitation on numerical representations in computers, not in the modelling techniques themselves, and small geometric errors continue to dog the use of CAD in general, both in surfaces and solids.

Another, related topic concerns the order of the control points. If you look at the order given for the lofted surface in Sect. 5.6.3 you see that there are four rows of three points. This was done because of width restrictions, because three rows of four vectors is too wide for the page. However, you could specify the surface this way, without problems. You could give the last row, then the third row, then the second, finally the first, if you want. What this means is that the position corresponding to the parameter position $u = 0, v = 0$ could be any of the corners. So, if you traverse a multi-patch structure it is not certain that the patches are aligned in parameter space. If you want to generate a tool-path, for example, along two or more patches then, when you traverse the boundary, you have to check the parameter position of the common point and arrange the parameter traversal from there.

A final point about separate surface patches is that the orientations may be different. With a topological structure such as that described in Sect. 5.8.1 you can overcome this problem.

5.6.8 Reverse Engineering

A method which is sometimes used to create complex geometry is known as “reverse engineering”. There is a simple meaning of the term which indicates the simple disassembly of products and their remodelling in CAD, but this is not dealt with here. Reverse engineering, here, is taken to mean the reconstruction of surfaces from clouds of measured points. This is a very brief summary, more can be found in Marshall and Martin’s book [7] and surface techniques in Besl’s book [8]. The usual procedure is:

1. Measure an object.
2. Divide the measured data into point sets corresponding to surfaces.
3. Fit surfaces.
4. Possibly, recreate the topology.
5. Merge models.

There are several measuring methods, which can be grouped roughly into two groups: contact measurement and optical measurement. *Contact measurement* is more accurate than optical measurement, but is slower and cannot be used with delicate or deformable objects. It involves extending a probe to an object and recording the point where the probe stops. The method can also provide a surface normal approximation which helps in surface reconstruction. *Optical measurement* uses visual inspection to estimate 3D point position. One method involves shining a laser beam on the object, recording the image and using triangulation to gauge the point position in space. Another method involves projecting patterns onto an object and calculating the point positions from the pattern deflection.

Dividing the points into sets corresponding to surfaces is difficult. This process is called “segmentation”. Besl [8] describes one method based on curvature approximation. Points are segmented into sets with the same curvature characteristics. If using a CMM machine for inspection, say, it may be possible to use a manual method to measure critical points on a surface and fit a known surface to these, but this is not always convenient. If you do not segment the points then you can get distorted surfaces which are generally complex.

Calculating surfaces from point sets involves a technique called “surface fitting”. Unlike interpolation, where the surface passes through the points, surface fitting assumes that the surface passes close to the points being fitted. This is necessary because there can be measurement errors, generally smaller with contact measurement than optical measurement. Surface fitting will be dealt with further below, because it is an important topic.

The last two steps, recreating topology and joining partial models are for recreating complex models or solids. It is difficult to create a solid from measured points because there are usually holes in the measured data. Recreation can either be done by building a facetted model and then deleting edges between facets in the same surface or by creating faces from the segmented surface sets and making intersection edges between two neighbouring surface patches. It can be more realistic, though, to measure isolated surfaces, fit a surface and then use the surface as part of another model. Joining partial models is a technique for merging partial models created from measured point sets in different “views” of the object. The big difficulty with this, though, is to match the object parts.

For surface fitting, some important characteristics are the number of segments for the surface and the degree, or order, of each portion. If you have many segments and a high degree of surface then you approach, or maybe even achieve, interpolation. However, if there are errors in the surface then this may result in a wavy surface. If there are fewer segments and lower degree then the surface will be smoother, but may be less accurate in approximating the points. It is not easy to fit surfaces and it may take several experiments to get a satisfactory result.

5.6.9 Surface Experiments

5.6.9.1 Lofting Experiment

Make the self-intersecting curve in the curve examples, that is, create five points: $(-10,-10,0),(2,5,0),(0,10,0),(-2,5,0)$ and $(10,-10,0)$. Create a curve passing through these in the order shown. Extrude this curve 20 mm to create a self-intersecting surface. Does your CAD system warn you that you have created such a surface?

Try and thicken this surface, by a little, say 0.25 mm. Does the operation to create a solid by thickening a surface warn you?

Now try modifying the point positions to: $(-10,-10,0),(2,5,2),(0,10,4),$ $(-2,5,6)$ and $(10,-10,8)$. The curve is no longer self-intersecting, but repeat the extrusion and thickening steps to check whether or not you get a warning.

5.6.9.2 Surface Interpolation Experiment I

Create three lines:

1. From $(-10,-10,10)$ to $(-10,10,10)$
2. From $(0,10,10)$ to $(0,-10,10)$
3. From $(10,-10,10)$ to $(10,10,10)$

Note the direction of the second curve. You need not absolutely create a line in this direction because you control the direction at the lofting stage. Create a lofted surface through these with the second line in the opposite direction to the other two lines. The CAD system should warn you about this. If it does, try lowering the middle line by one in Z, so that it runs from (0, 10, 9) to $(0,-10,9)$ and try again. Keep lowering the line until the CAD system allows you to make the surface. This should give you an idea of how sensitive the system is to critical geometry.

5.6.9.3 Surface Interpolation Experiment II

Create three lines:

1. From $(-10,-10,10)$ to $(-10,10,10)$
2. From $(0,-10,10)$ to $(0,10,10)$
3. From $(10,-10,10)$ to $(10,10,10)$

Now the direction of the second curve is the same as for the other two. Create a lofted surface through these in the order: 1, 3, 2. The CAD system should again warn you about this. If it does, as before, lower the middle line by one in Z, so that it runs from $(0,10,9)$ to $(0,-10,9)$ and try again. Keep lowering the line until the CAD system allows you to make the surface. This is another test to see the sensitivity of the CAD system to degenerate surfaces.

5.7 Surface Analysis

There are several methods of visual analysis. Figure 5.22 shows some surface analysis figures from work by Várady (no reference).

Figure 5.22a shows the original surface as a shaded image. Figure 5.22b shows analysis basis on slicing through the surface in a direction chosen by the user, with

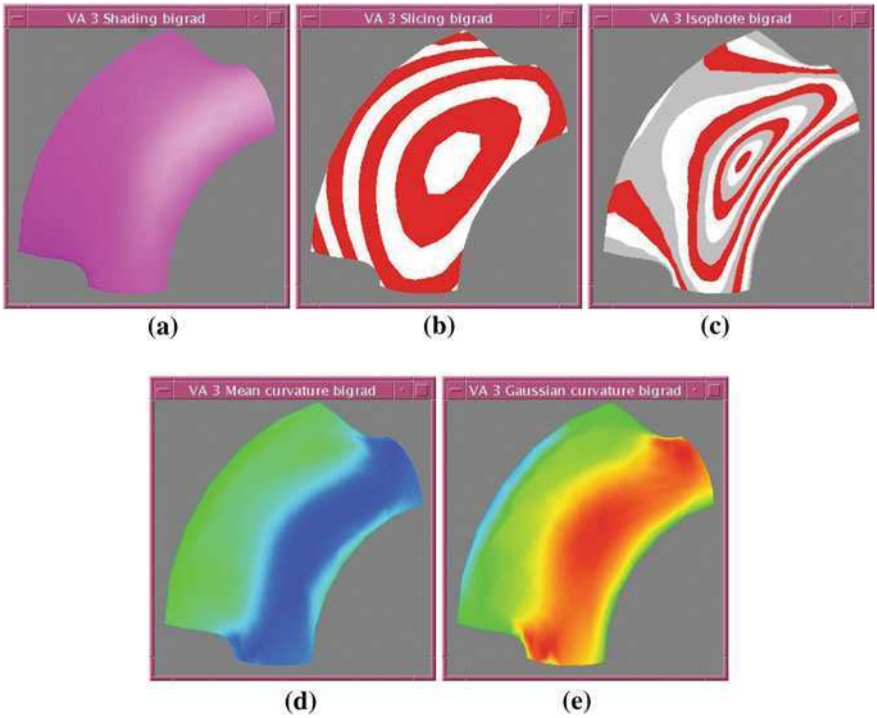


Fig. 5.22 Surface analysis methods

different sections coloured. Figure 5.22c shows what is called the isophote image, where the surface is divided into bands according to the angle between the surface normal and a light source. Figure 5.22d shows the surface coloured according to the mean curvature. Figure 5.22e shows the surface coloured according to the Gaussian curvature. Note that all these images are based on facetted approximations to the surface, which cause some visual artefacts.

The slicing and isophote images will show discontinuities in the bands if they cross surface patch boundaries which are not curvature continuous. The mean curvature analysis is based on the sum of the two principle curvatures. The Gaussian curvature is the product of the two principle curvatures. Both of these techniques will show up wavy surfaces as being stripy.

There are other techniques which are not illustrated here. Reflection lines are similar to the isophote method. There are also “hedgehog” displays showing the normal vectors on the surfaces.

Surface analysis is intended to provide a quick visual check of the quality of a surface. It can be very useful for checking fitted surfaces, for example, to gauge the effect of different parameter settings from the point of view of smoothness.

It is worth exploring these tools if you are going to work with free-form surfaces. Personally I found Várady’s methods and the I-DEAS implementation straightforward to use. By contrast, the CATIA v5 implementation seems to me to be unnecessarily clumsy.

5.7.1 Surface Analysis Experiments

5.7.1.1 Cylindrical Surface Experiment

Create a cylinder and analyse the cylindrical surface using the mean curvature analysis tool and the Gaussian analysis tool. The mean curvature shows you the sum of the principal curvatures while the Gaussian curvature analysis tool shows you the product. What can you conclude about the result from the Gaussian analysis?

5.7.1.2 Wavy surface experiment

Create five lines:

1. From $(-20, -10, 10)$ to $(-20, 10, 10)$
2. From $(-10, -10, 8)$ to $(-10, 10, 8)$
3. From $(0, -10, 10)$ to $(0, 10, 10)$
4. From $(10, -10, 8)$ to $(10, 10, 8)$
5. From $(20, -10, 10)$ to $(20, 10, 10)$

Interpolate a surface passing through these lines in the same order. Analysis the curvature to see the ripples.

5.7.1.3 Doubly Curved Surface Experiment

Create three curves interpolating the points:

1. $(-10, -20, 20)(-10, 0, 10)(-10, 20, 20)$
2. $(0, -20, 0)(0, 0, 10)(0, 20, 0)$
3. $(10, -20, 20)(10, 0, 10)(10, 20, 20)$

Interpolate these three curves to give a doubly curved surface as in Fig. 5.23.

Along the centre line of the surface there is a straight portion. How is does this appear when analysing the surface for Gaussian curvature?

5.7.1.4 Discontinuity Analysis

Now create two surfaces adjacent to each other to see what happens with the discontinuity analysis.

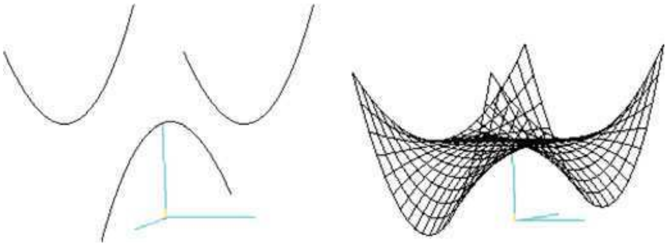


Fig. 5.23 Doubly curved surface

Create five lines:

1. $(-50, -50, 50)$ to $(-50, 50, 50)$
2. $(-25\sqrt{2}, -50, 50 - 25\sqrt{2})$ to $(-25\sqrt{2}, 50, 50 - 25\sqrt{2})$
3. $(0, -50, 0)$ to $(0, 50, 0)$
4. $(25\sqrt{2}, -50, 50 - 25\sqrt{2})$ to $(25\sqrt{2}, 50, 50 - 25\sqrt{2})$
5. $(50, -50, 50)$ to $(50, 50, 50)$

Create one surface interpolating lines 1, 2 and 3. Create the second surface interpolating the lines 3, 4 and 5. Analyse the surface with isophote, or by projecting lines onto the surface. Now change line 5 to run from $(50, -50, 50)$ to $(50, 50, 100)$ and redo the analysis. Note the changes in the results.

The intention of this experiment is to get you used to using different types of analysis and interpreting the results. By all means try other surfaces, this type of analysis is useful for examining properties of designed surface sets.

5.8 Integration of Geometry

OK, once you have your geometry, what do you do with it? Techniques for thickening surfaces to produce solids have already been described in Sect. 4.11. Filling closed sheet objects has also been described, in Sect. 4.12 if you have defined a complete set of surfaces bounding an object. This section presents some of the methods for integrating geometry.

5.8.1 Standalone Geometry

The boundary representation can be extended to provide a unified modelling environment for geometric entities as well as solids. A surface can be thought of as being like a piece of paper. For the purists, I know that a piece of paper has thickness but a surface does not, but it is a simple an analogy. Something like that

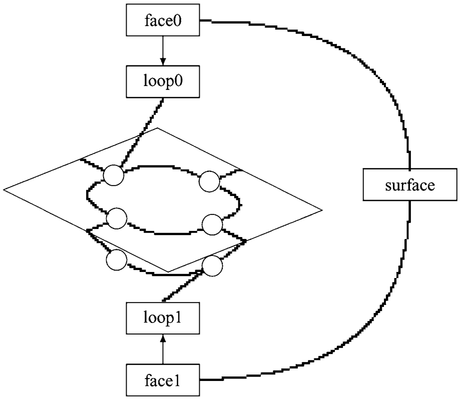


Fig. 5.24 A degenerate form for integrating surfaces into CAD

is shown in Fig. 5.24. Note that face 1 might have a flag to say that the geometry is negated to be consistent with the topological normal.

Free-form surfaces have a natural topology made from the four edges which consist of bounded curves. Spheres and toruses are limited surfaces, but planes, cones and cylinders need to be delimited somehow. This can be done by using the degenerate forms shown in Fig. 5.24. This is the same structure as is used for modelling solids and will be described in Chap. 6. Curves also need to be limited, which can be done by assigning them to edges which are attached to vertices. These are so-called “wireframe models”, which will also be described more fully in Chap. 6.

Joining two surfaces is illustrated in Fig. 5.25 If the edges of two surfaces match, as at the top of the figure, then joining is simple. The edges just swap one face pointer and you end up with a structure such as that shown on the top right of the figure. This double edge can be collapsed back to a single edge, but this is not strictly necessary. It is slightly more natural to maintain these edges separate.

If the edges do not match, as at the bottom of the figure, then it is necessary to break one or both edges so that there are matching edges. These edges are then joined in the same way, as shown at the bottom of the figure.

This is the nice case. What can happen all too frequently, though, is that there is a mismatch between the edges and they are not joined. Near misses, such as those shown in Fig. 5.26 can easily happen.

The difference between points may be 0.0001 mm, less than you can manufacture, but the software is able to distinguish between 0.000001 mm. This difference is usually controlled using a parameter of the system, a “tolerance” value. Sometimes you can persuade the system to join edges by increasing the tolerance value, but it is not always easy to find a good value. If the tolerance is too large then this risks causing errors by uniting points which should be distinct.

As already mentioned, the joined sheet objects can be thickened or, if closed, converted to a solid by ripping out the inside. Once a solid they can be developed further with standard volumetric functions.

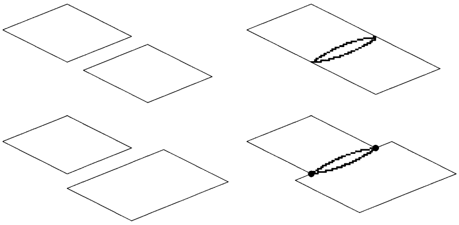


Fig. 5.25 Joining degenerate surface models

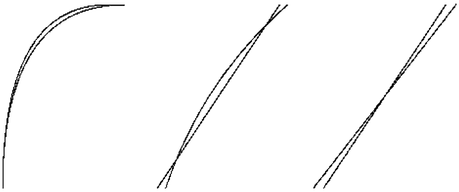


Fig. 5.26 Non-matching curve examples

Note that an advantage of having this double-sided method for representation is that you don't have to worry about surface orientation. A common effect in surface modelling systems was that the surfaces designed separately had different orientations. This means that there was no consistent definition of where the material of a body was. Graphically the object appears correct. For machining the toolpaths could be calculated on the surface while the user indicated manually the approach side. For rapid prototyping, though, the facets produced from the surface set would have different orientations and there could also be mismatches from neighbouring, non-joined surfaces. This caused problems because the contours used for rapid prototyping had badly oriented portions and/or gaps. Joining surface patches in the manner outlined above is important to avoid such problems.

5.8.2 Adding Surfaces to a Solid

How do you get isolated surfaces into a model? Two early techniques were the SETSURF technique, by Jared, and sectioning, by Smith, for the BUILD system. Neither of these is ideal for creating a sculpted shape. They can be used for small modifications, but are not intended for creating a global smooth shape.

SETSURF was described in Sect. 4.4 and is illustrated in Fig. 4.45. Requirements mentioned there are that the new surface is at least as big as the surface in which it is being set and that the new surface intersects all the surfaces surrounding the face in which the new surface is being set. SETSURF is a natural method to use for inserting geometry into an object but is potentially awkward if there are several neighbouring faces to be changed. This is because you might need to create intermediate topology and geometry, which is awkward.

Sectioning is a sort of Boolean operation that has already been described in Sect. 4.1.5. It acts as though there is infinite material behind the sectioning surface. A requirement is, though, that the surface is at least as big as the object being sectioned. Otherwise the object would only be partially cut.

5.8.3 Object Smoothing

An elegant technique for creating smoothed models from planar polyhedra was published around 1983 by Chiyokura and Kimura [9–11]. Some work was also done on the same topic by the commercial company Shape Data who produced the Romulus, and then the Parasolid kernel modelling systems (their smoothing package was called “Remus”).

Chiyokura and Kimura’s system, MODIF, required the user to build a polyhedral model of an object and then indicate which edges were static, which were rounded. The conversion operation then recalculated the topology and geometry to produce a smoothed object. The technique was published, in a book as well as papers, but does not seem to have been incorporated into modern CAD systems, unfortunately.

Remus was once described as working by continuously chamfering edges to be smoothed, and then the resulting edges of these, and so on, until some limit was reached. This may be an analogy of what happens rather than the actual algorithm. This technique, too, seems to have disappeared and is not currently available in CAD systems.

5.8.4 Rounding Example

The following is a small worked example of creating surfaces to round off a vertex. Figure 5.27 shows the geometric basis for the example.

The task is to round off the vertex, or corner, at (10, −10, 10). Since there are three edges meeting at the vertex the result is a three-sided region. Since it is necessary to use four-sided patches it is necessary to divide the region into three four-sided patches meeting at a common point. In order to do this it is necessary to decide on the position of the central point and the common normal at this point. For this exercise the central point is taken as being at (9, −9, 9) and the normal vector as (1, 1, 1), which will be normalised during the calculations.

The figure is symmetric in this case, so to shorten the exercise the control points for one surface patch will be calculated and the others derived from this.

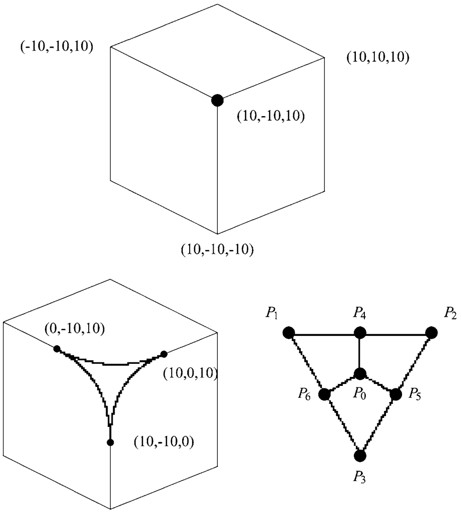


Fig. 5.27 Rounding off the corner of a cube

The first thing to do is to split the curve P_1P_2 to find the point P_4. To do this it is possible to use the de Casteljau method.

The original curve has the control points: (0, −10, 10), (10, −10, 10) and (10, 0, 10). With the value of "t as 0.5 this gives the triangular form of de Casteljau as:

$$\begin{array}{ccccc} & & (7.5,-7.5,10) & & \\ & (5,-10,10) & & (10,-5,10) & \\ (0,-10,10) & & (10,-10,10) & & (10,0,10) \end{array}$$

This gives the first control points of the surface patch. The patch structure is as shown in Fig. 5.28. From what has been determined:

$$\begin{aligned} P_1 &= (0,-10,10) \\ P_{14} &= (5,-10,10) \\ P_4 &= (7.5,-7.5,10) \\ P_0 &= (9,-9,9) \end{aligned}$$

It is also possible, because of the symmetry, to determine that:

$$\begin{aligned} P_{14} &= (5,-10,10) \\ P_6 &= (7.5,-10,7.5) \end{aligned}$$

In order to determine P_{40} it is necessary to perform a line-plane intersection of a line through P_4, perpendicular to the curve and the plane through P_0 with the given plane normal. This is illustrated in Fig. 5.29.

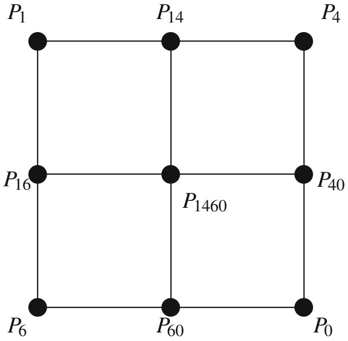


Fig. 5.28 Surface patch control point grid

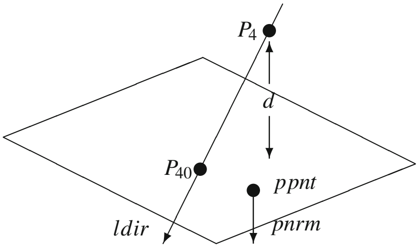


Fig. 5.29 Intersecting a line and a plane

$$P_{40} = P_4 + ldir * (d/(pnrm))d = (ppnt - P_4) \cdot pnrm\ ldir = \left(\frac{1}{\sqrt{2}}, \frac{-1}{\sqrt{2}}, 0\right),$$

$$pnrm = \left(\frac{1}{\sqrt{3}}, \frac{-1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\right)\ ppnt = (9, -9, 9)$$

This gives $P_{40} = (8.5, -8.5, 10)$ as the value of the intersection point. By symmetry it is possible to say that $P_{60} = (8.5, -10, 8.5)$.

This leaves only the central point, P_{1460} to be determined. Here, this point must lie in the two side planes, $Z = 10$ and $Y = -10$ to preserve the tangent conditions with these surfaces, and it must lie in the plane through P_0 with normal $(1, 1, 1)$. Solving these constraints gives you the position: $(7, -10, 10)$. The control points of the grid are shown in Fig. 5.30.

Note the values of P_{14} and P_{16} are the same. What does this imply for the surface normal at P_1? (Fig. 5.30).

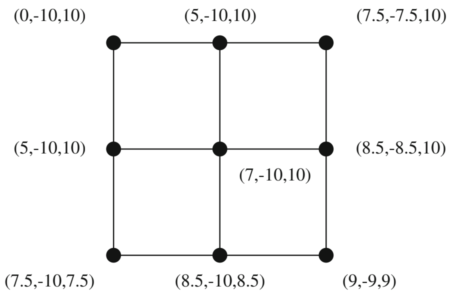
The final geometric entities are shown in Fig. 5.31.

5.9 Some Formulae

5.9.1 Tangent

The tangent to a curve can be found by differentiating the equation to find $f'(t)$ and evaluating this for the value of t required. You can use the matrix form, which is useful if you have a lot of calculation. If you use the de Casteljau method to find a

Fig. 5.30 Surface patch control point grid



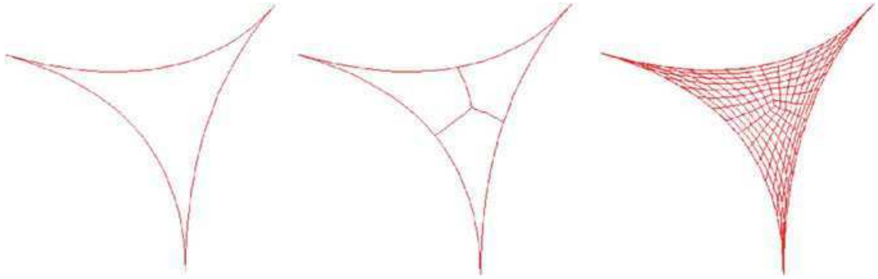


Fig. 5.31 Boundary curves, internal curves and surfaces for rounded corner

point, the last line gives the tangent direction and the size of a degree n curve tangent is n times the length of the line.

5.9.2 Curvature

$$\kappa(t) = \frac{|f'(t) \times f''(t)|}{|f'(t)|^3}$$

5.9.3 Surface Normal

For analytic surfaces

$$\frac{dS(u,v)}{du} \times \frac{dS(u,v)}{dv}$$

5.9.4 Principal Curvatures

$$E = X_u \cdot X_u \; F = X_u \cdot X_v \; G = X_v \cdot X_v \; L = X_{uu} \cdot norm \; M = X_{uv} \cdot norm \; L = X_{vv} \cdot norm$$

$$K = \frac{LN - M^2}{EG - F^2} 2H = \frac{EN - 2FM + GL}{EG - F^2}$$

5.10 Chapter Summary

This chapter is a very thin overview of the complex subject of geometry. It is not meant to give a full treatment, but to teach a little of the functions and how they are used for complex operations. The chapter describes important functions, such as curve and surface interpolation, curve extrusion, surface analysis. The chapter also describes the integration of geometry into the solid modelling environment.

5.11 Geometry Exercises

5.11.1 Interpolation Exercise

The first task is to define a cubic Bézier curve which passes through the points:

$$(0,0)(4,3)(14,3)(10,6)$$

You should define the control points for the curve, although the first and last are identical to the first and last points above, so in effect the points of control are:

$$(0,0)(a,b)(c,d)(10,6)$$

Where it is necessary to find a, b, c et d so that the curve passes through the two points (4, 3) and (14, 3).

The formula to calculate a point P(t) on a cubic Bézier curve is:

$$P(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3$$

Where P_0, P_1, P_2 and P_3 are the points of control. For the curve above you have the two equations:

$$(4,3) = (1-t_1)^3(0,0) + 3t_1(1-t_1)^2(a,b) + 3t_1^2(1-t_1)(c,d) + t_1^3(10,6)$$

and

$$(14,3) = (1-t_2)^3(0,0) + 3t_2(1-t_2)^2(a,b) + 3t_2^2(1-t_2)(c,d) + t_2^3(10,6)$$

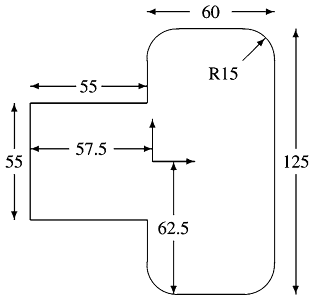


Fig. 5.32 Simple extrusion shape

In order to be able to resolve these it is necessary to estimate the values of t_1 and t_2, first calculate the values of a,b, c and d with t_1 and t_2 calculated from the arc length (the proportion of the distance between the points).

Now create the points and the curve using your CAD system. Try and find a function for displaying the curve control points. These are usually editing functions for manipulating the shape. Do the control points lie at the place you calculated for them?

In the CAD system create a sketch on the XY plane such as that shown in Fig. 5.32.

The part should be symmetric about the horizontal axis. Cover the shape so as to make a portion of surface. Extrude the same sketch a distance of 57.5 mm. Now join the part from the first operation (when you covered the surface) with the extrusion. This now forms an open structure, now give the part a thickness, say 1 on the interior and 2 on the exterior.

This method of working is useful for, say, creating thin aesthetic shapes such as car bodies, ship hulls, vacuum cleaner covers or other aesthetic shapes. You create one surface and then the system can create the solid for further working directly. You can also create objects as solids and then convert them to thin, hollow shapes using the command to create a shell (Fig. 5.32).

5.11.2 Unequal Surface Transformation

Not all CAD systems allow uneven scaling so you may not be able to do this in your CAD system.

Make an object by circular extrusion and then use unequal scaling to change the form of the surface.

Sketch a simple square on a plane and rotate this about one of the edges of the square. After creating the initial object, save the object in STEP format, which is the easiest format to look at. Open this original object with a text editor. Now scale the object with uneven scaling, if your CAD system allows it, and save it to a second file using STEP format. Open this new file and compare the geometry. Now scale the object back using uneven scaling and save the object to a third file.

In the first file you should find planes and a cylindrical surface. In the second file, you should find that much of the simple geometry has been converted to B-spline geometry in order to be able to perform the uneven scaling. In the third file it is likely that you still find B-spline geometry, even though the shape is cylindrical. This is because it is much harder to convert complex geometry back to simple forms.

5.11.3 Linear Exercises

Use a cubic Bézier to interpolate the points:

$$(0,0),(5,0),(7.5,0),(10,0)$$

What happens?

Write out the cubic Bézier form in full and collect the terms of t^3, t^2, t and constants together in terms of the control points: b_0, b_1, b_2 and b_3. Calculate the terms for t^3 and t^2 when you have the control points:

$$(0,0),(5,0),(10,0),(15,0)$$

and also for the control points:

$$(0,0),(10,0),(15,0),(20,0)$$

5.11.4 Using a Solid to Guide Geometry

Make the object in Fig. 5.33 as one object in an assembly. Making the object in an assembly means that the cover, or shell, surrounding this object is created as a separate object.

The object represents a volumetric representation of half the space occupied by an internal mechanism around which a cover is to be created. Figure 5.34 shows the faces and the base points used for a set of offset points for curve creation. Create a new object and in this create the points shown in the figure.

The offsets from the nearby vertices are:

Now create splines for each point set. The first spline should pass through the points P1, P2 and P3. Add a tangent at the start and end of the spline so that the curve has a vertical direction at P1 and a horizontal direction at P3. The horizontal direction at P3 means that the curve and its mirrored image will have the same tangent direction after the symmetry operation in the last step.

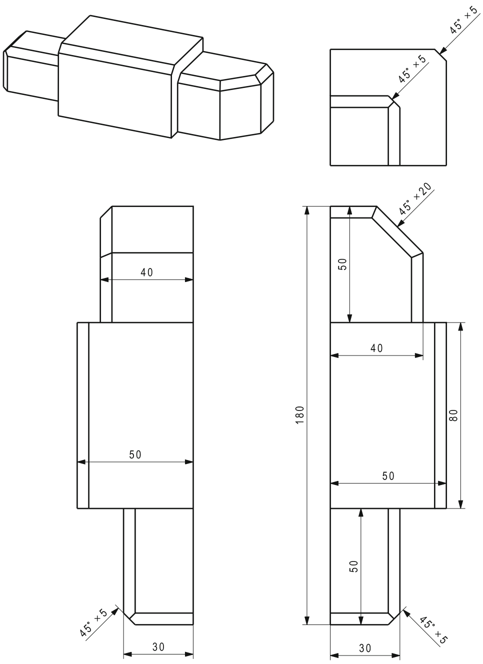


Fig. 5.33 Solid for surface creation

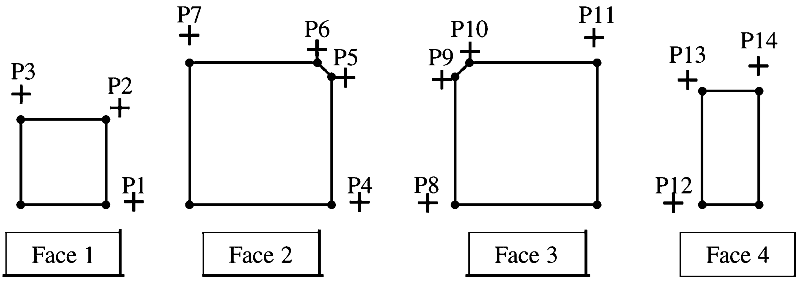


Fig. 5.34 Faces and offset points for creating surfaces

Repeat the process making splines through:

P4, P5, P6 and P7 (set the tangent directions for P4 and P7).
P8, P9, P10 and P11 (set the tangent directions for P8 and P11).
P12, P13, P14 (set the tangent directions for P12 and P14).

After the process you should have a set of curves like those shown in Fig. 5.35. Interpolating a surface through these gives the shape in Fig. 5.36.

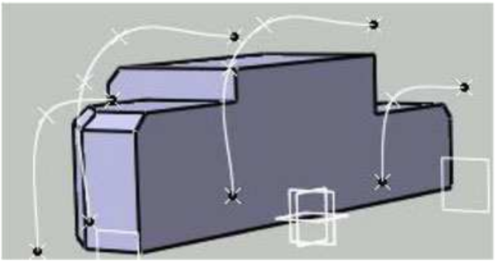


Fig. 5.35 Splines for surface interpolation

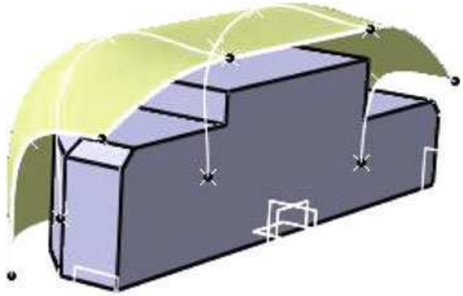


Fig. 5.36 Surface interpolated through splines

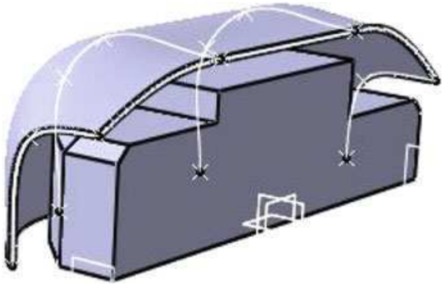


Fig. 5.37 Solid created by giving surface thickness

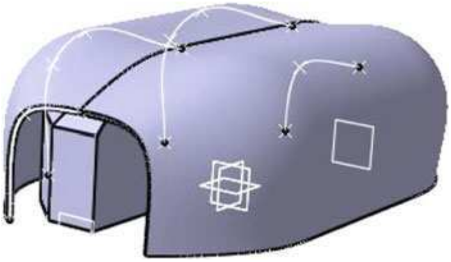


Fig. 5.38 Shell finished by reflection

Thicken this surface to give the half shell shown in Fig. 5.37.

Finally, complete the cover by reflecting the thickened shape in the appropriate plane, Fig. 5.38.

Depending on how the system works, the points used to create the splines, and hence the whole object may or may not be linked to the original solid. If they are linked, then changing the original solid will create a new shell automatically.

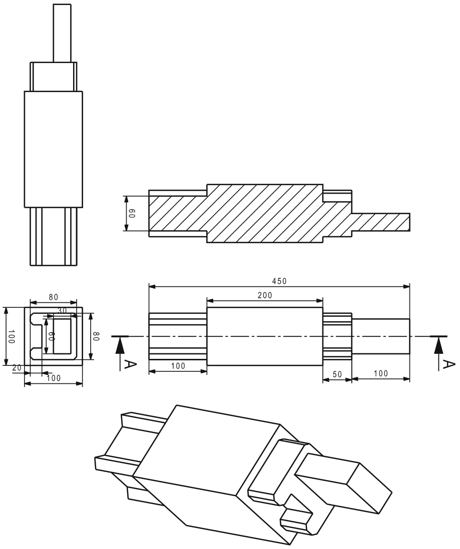


Fig. 5.39 Solid for surface creation

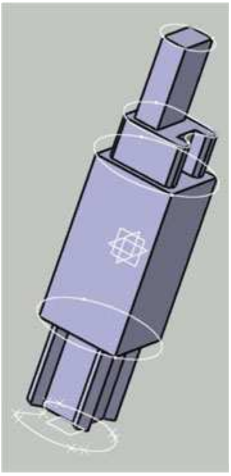


Fig. 5.40 Closed splines created around a solid

Point	Vertical	Horizontal
P1	10	0
P2	5	5
P3	0	10
P4	10	0
P5	5	0
P6	0	5
P7	0	10
P8	10	0
P9	5	0
P10	0	5
P11	0	10
P12	10	0
P13	5	5
P14	0	10

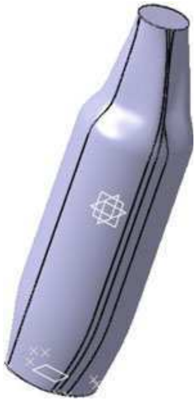


Fig. 5.41 Solid created from splines

5.11.5 *Creating a Solid Guided by a Solid*

Now make the object in Fig. 5.39.

Use the vertices to create a series of sections in the form of spline curves and the tool for creating a lofted surface to create the cover shape. It doesn't matter exactly how you create the splines, the shape is arbitrary to demonstrate the principle. However, each section should be closed. Figure 5.40 shows the splines used in this example, but they are arbitrary.

The shape is made as a solid lofted part over these sections. If your CAD system doesn't allow you to make a solid then you can fill in the first and last sections to

Fig. 5.42 First variant

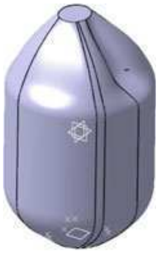


Fig. 5.43 Second variant

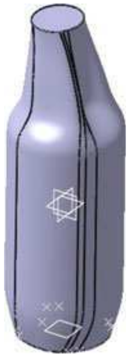
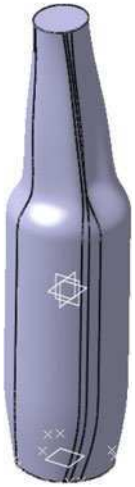


Fig. 5.44 Third variant



close the surface model, however, making a solid is probably available. Figure 5.41 shows the lofted solid.

What the lofted solid does is to gobble up the internal solid around which it was built. The original model is no longer part of the model, but it is still there and can be modified. Since the splines defining the lofted shape are defined using points created with reference to the original model then changing the original model should change the splines and hence the lofted shape. This means that parametrising the original simple shape The final lofted shape can be turned into a shell to define a closed shape, like a type of bottle.

Figure 5.42 shows a variant where the middle part has been enlarged giving a fatter shape.

Figure 5.43 shows a variant where the bottom part has been lengthened.

Figure 5.44 shows a variant where the top part has been lengthened.

5.11.6 A Silly Parametric Aeroplane

The final exercise in the parametric shape vein is a silly aircraft model. The shape is not very aerodynamic, but was produced quite quickly as an illustration. The first step is to define the internal shapes which are used to guide the final complex shape. Since the aircraft is symmetrical, only half shapes were created and the final shape created by reflection, in this case about the XZ plane.

The guide shape is shown in Fig. 5.45. It consists of a cabin box, in front of this a nose shape and to the rear a tail shape.

The final shape and the half splines used to create the aircraft geometry are shown in Fig. 5.46. Note that there are two sets of spline sections, one set for the length of the fuselage and one for the front wing. These are controlled separately and the resulting volume objects added together. The rear wing is not controlled and so has fixed size.

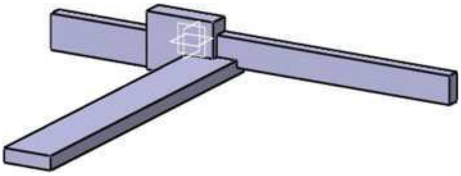


Fig. 5.45 Basic aeroplane interior

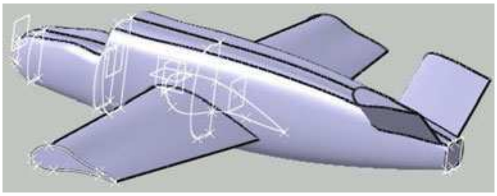


Fig. 5.46 Aeroplane with spline sections

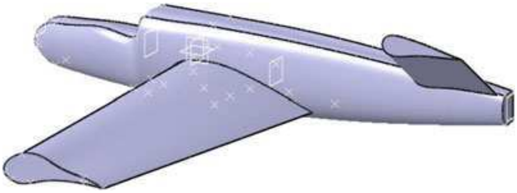


Fig. 5.47 Half aeroplane

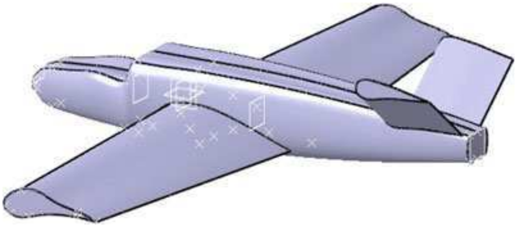


Fig. 5.48 Aeroplane variant 1

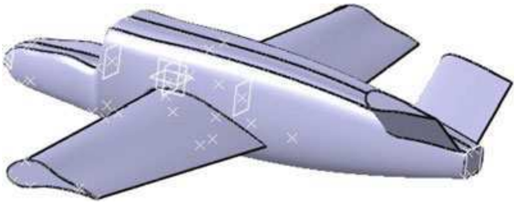


Fig. 5.49 Aeroplane variant 2—heightened cabin

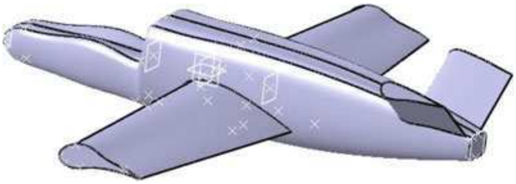


Fig. 5.50 Aeroplane variant 3—lengthened nose

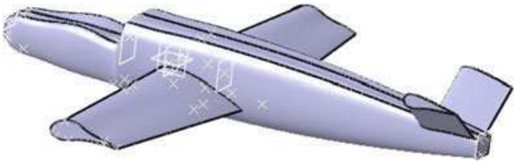


Fig. 5.51 Aeroplane variant 4—lengthened nose and tail section

Fig. 5.52 Aeroplane variant 5—Lengthened wings

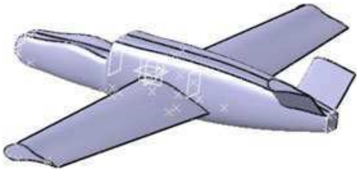


Figure 5.47 shows the half shape created from the spline sections. Note that there is, for each section, a straight part lying in the XZ plane. This is to create a planar surface of symmetry.

The initial aeroplane shape is shown in Fig. 5.48.

For the first variant, the cabin has been heightened, as in Fig. 5.49.

For the second variant the nose section has been lengthened (Fig. 5.50).

The fourth variant has a lengthened tail section in addition to the lengthened nose section Fig. 5.51.

The final variant has extended wings. Figure 5.52.

This is just a simple example which you can try with your own system. There are alternative construction methods, such as defining style lines along the length of the object, using the interior solid as a guide, and creating section points by intersecting the style lines with transversal planes.

References

1. Faux, I., Pratt, M.J.: Computational Geometry for Design and Manufacture. Ellis-Horwood, Chichester (1978)
2. Farin, G.: Curves and Surfaces for Computer-Aided Geometric Design. Academic Press, London, ISBN 0-12-249054-1 (1988)
3. Piegl, L., Tiller, W.: The NURBS Book. Springer, Heidelberg, ISBN 3-540-61545-8 (1997)
4. Hoschek, J., Lasser, D.: Fundamentals of Computer Aided Geometric Design, translated and published by A. K. Peters, ISBN 1-56881-007-5 (1993)
5. Pfeifer, H.-U.: Methods used for intersection geometrical entities in the GPM module for volume geometry. CAD J. **17**(7), 311–318 (1985)
6. Stroud, I.A.: Boundary Representation Modelling Techniques. Springer, Heidelberg (2006)
7. Marshall, A.D., Martin, R.R.: Computer Vision, Models and Inspection. World Scientific Publishing, Singapore, ISBN 9810207727 (1992)
8. Besl, P.J.: Surfaces in Range Image Understanding. Springer, Heidelberg, ISBN 0-387-96773-7 (1986)
9. Chiyokura, H., Kimura, F.: Design of solids with free-form surfaces. Comput. Graph. (SIGGRAPH ’83 Proc.) **17**, 289–298 (1983)
10. Chiyokura, H., Kimura, F.: A method of representing the solid design process. IEEE Comput. Graph. Appl. **5**, 32–41 (1985)
11. Chiyokura, H.: Solid Modelling with DESIGNBASE. Addison-Wesley Publishing Company, Reading, ISBN 0-201-19245-4 (1988)

Chapter 6
Non-Manifold Models

The normal objects that you meet in everyday life are called "manifold" objects. Which means, putting it glibly, that at every point on the surface the neighbourhood around the point is homeomorphic to a disc. You may, or may not want to know that. Figure 6.1, shows an alternative way of understanding this, from Braid, that, at every point on the outside of the object, a small enough sphere will be cut into two pieces, one inside the object and one outside.

The following potted history is what I believe to be true, but if someone ever writes a definitive history of CAD then there may be other factors of which I am not aware.

In the original Boundary Representation modelling systems only valid solids were considered. In the BUILD research system, as an intermediate step, it was possible to represent flat objects, but these were usually only shapes which were to be swept. Towards the end of the 1970s an internordic project, GPM, was set up to develop methods for "Geometric Product Modelling" was set up incorporating a number of modelling methods. In Denmark the user system was developed. In Norway an "Assembled Plate Construction" (APC) module (SINTEF) and a surface module were developed (SI). In Sweden and Finland the volumetric modelling module was developed. The APC module was a specialised, advanced module for modelling constructions made from thin plates. As part of the volume module it was intended to be able to interface with both this module and the surface module, so thin plate models were introduced as part of the volumetric modelling system [1].

One of the Swedish ideas was that, by mixing different representations in the same modelling framework, you could represent different stages and levels of models. In the beginning you might have a simple sketch. This might then be fleshed out into partial models, idealisations of the volumetric shape. For production needs these might need to be expanded into full volumetric models. As the lifetime of a product develops it may prove useful to go back to idealisations and maybe even sketches. This is described by Kjellberg [2], who, as far as I know, pioneered this method, but his dissertation is in Swedish so is not so accessible. One of the examples he used is illustrated in Fig. 6.2, a simplified model of an excavator.

I. Stroud and H. Nagy, *Solid Modelling and CAD Systems*,
DOI: 10.1007/978-0-85729-259-9_6, © Springer-Verlag London Limited 2011

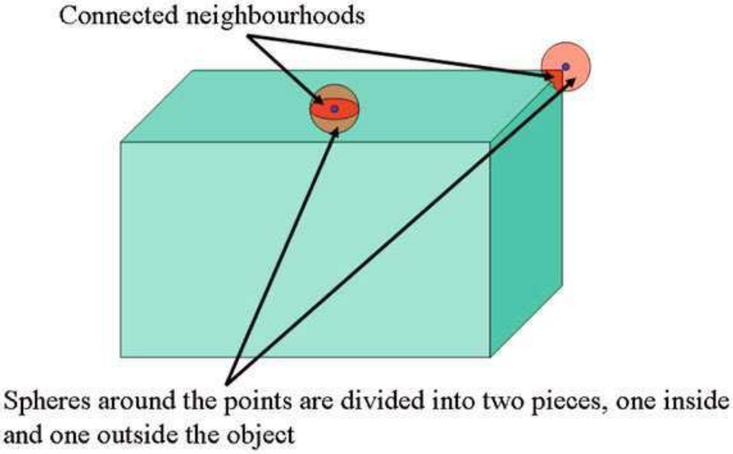
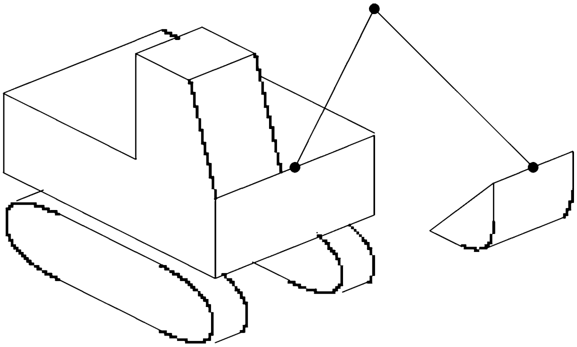

Connected neighbourhoods
Spheres around the points are divided into two pieces, one inside and one outside the object

Fig. 6.1 Manifold object definition

Fig. 6.2 Kjellberg's "Grävskopa", or exacavator (from Kjellberg [2])



The whole excavator is an assembly of models with different characteristics. The body of the excavator and the tracks are solid models, the arm is a wireframe model and the scoop is a compound sheet model.

So, for non-manifold models in CAD, it is necessary to distinguish between three types of special and non-manifold model:

1. Wireframe models.
2. Sheet models.
3. Non-manifold solid models.

These can be integrated into the same datastructure for easy transition between applications. For wireframe models the loop and face information is "ignored", that is, set to NULL. Sheet models are like degenerate solid models, with the limit edges corresponding to thin faces. Non-manifold solids have coinciding portions.

Some systems keep solids apart from sheet models and wireframe models, others integrate them. This is a strategic question which, of course, affects the user

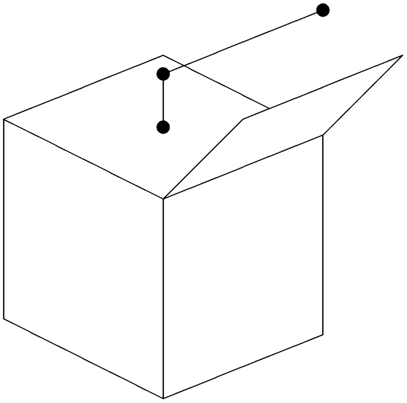


Fig. 6.3 Chimæra model with volumetric-, sheet- and wireframe parts

but is up to the CAD developer. Neither strategy is particularly bad, they both have advantages and disadvantages. Integrating them means that the user has a fluent design environment and that common functionality is shared. Keeping them apart means that there is less chance of error, in creating sheet models instead of solids. One thing that should never be done, though, is to have solids with sheet and/or wireframe elements integrated into the same model, such as the one shown in Fig. 6.3. This is possible, but sheet and wireframe models are idealisations of something, where a solid model is a full solid. Integrating them would mean that you have a model which has to be interpreted differently in different places, which is not particularly a good idea. Nobody does this, as far as I know, so it should not be a problem.

6.1 Datastructure Needs

A common method for implementing non-manifold modelling is to use the so-called "STAR" representation. The development that allowed this was to introduce the loop-edge links so that edges could refer to more than two loops (loops are face boundaries). In the original Boundary Representation (Brep) datastructure, in BUILD, there was a fixed restriction that there were two loops at every edge. They could be the same loop, but there were never more, because such objects were unrealisable. The addition of loop-edge links to the datastructure, which appeared in the GPM Volume Module, allowed rings of links around the edge and, hence, any number. Figure 6.4 shows this.

A requirement for this method of representation is that the links are ordered around the edge. Figure 6.5 illustrates this. On the left of the figure you see a normal case. The large black dot represents the edge, seen in cross-section. The lines represent faces and the small dots are just to indicate where there is material. Turning around the edge counter-clockwise, as indicated by the arrow, the links between the edge and the faces are classified as "enters" or "leaves" depending on

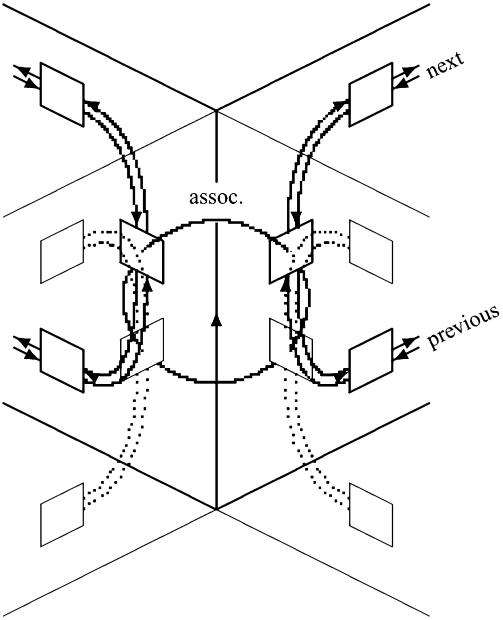


Fig. 6.4 Edge links in a non-manifold model (from Stroud [3])

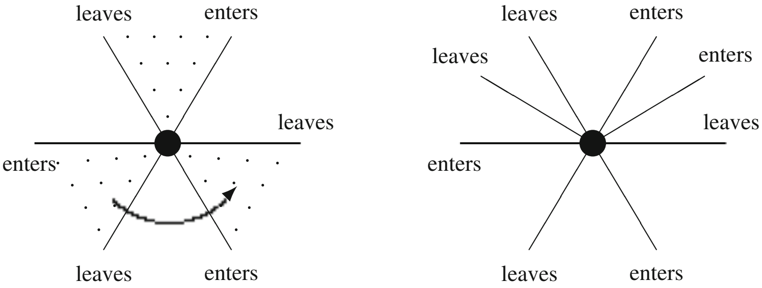


Fig. 6.5 Ordering edge links around the edge

whether you enter the material or leave it. There is a sequence of alternating pairs in a correct figure. On the right of Fig. 6.5 is an incorrect case. The sequence is "enters–leaves–enters–leaves–enters–enters–leaves–leaves". The double "enters" and "leaves" indicate that there is material within material, i.e. that there is a self-intersecting object. Note that this ordering procedure can be done for volume objects, but for sheet objects the enters–leaves classifications with coincide, and ordering is difficult.

The star representation is not necessary to implement non-manifold models, the non-manifold condition is a geometric condition, not a topological one. In the GPM project edges were allowed to refer to only two faces, even though loop-edge links were part of the datastructure. It was felt that it was more natural to duplicate edges. Edge duplication also has an advantage in that the meaning of the

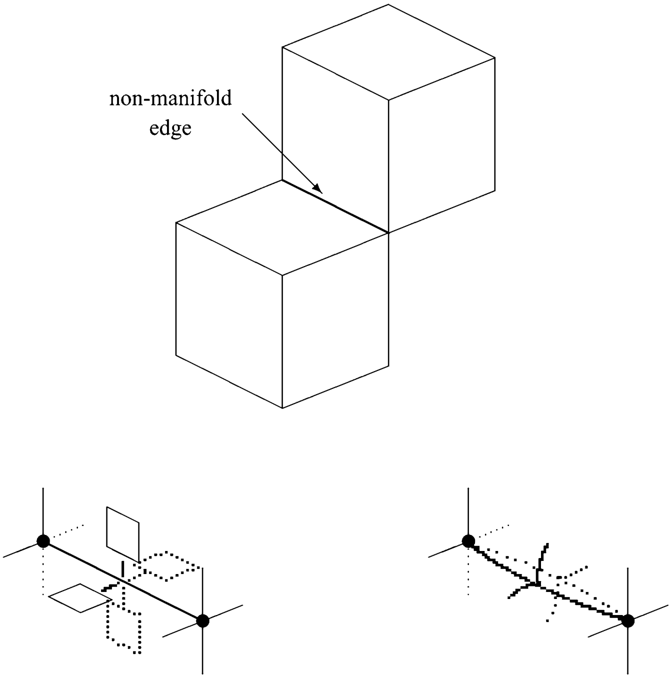


Fig. 6.6 Star and degenerate model comparison

datastructure entity connections is unambiguous. With the star representation it is not clear whether the object parts are just connected at the non-manifold edge, or whether they just miss each other. It is important to know this because performing an operation on a non-manifold edge should entail a conversion before the operation. This conversion involves pairing up the loop-edge links and duplicating edges. A visual comparison is shown in Fig. 6.6. For a non-manifold edge, as shown at the top of the figure, the star version is shown at the bottom left and the degenerate version on the bottom right. For the star version an advantage is that the links are associated, it is clear that the edge is a special case. Unless there is a link between the duplicated elements this connection is not explicit.

The ambiguity problem is illustrated graphically in Fig. 6.7. The star arrangement is shown at the top of the figure. If you group edge link 1 with edge link 2 and links 3 and 4 then you get the double edge joined case at the bottom left. If you group link 2 with link 3 and 4 with 1 then you get the arrangement on the bottom right, where the objects are separate, only being joined at the vertices of the edges.

The problem is that there is no way for the CAD system to know which one you mean. This means that operations, such as chamfering, could have two interpretations, as illustrated in Fig. 6.8, as already mentioned in Sect. 4.7. Of course, what you would like is the CAD system to ask you which you mean, but at the moment the trend is to ignore these edges.

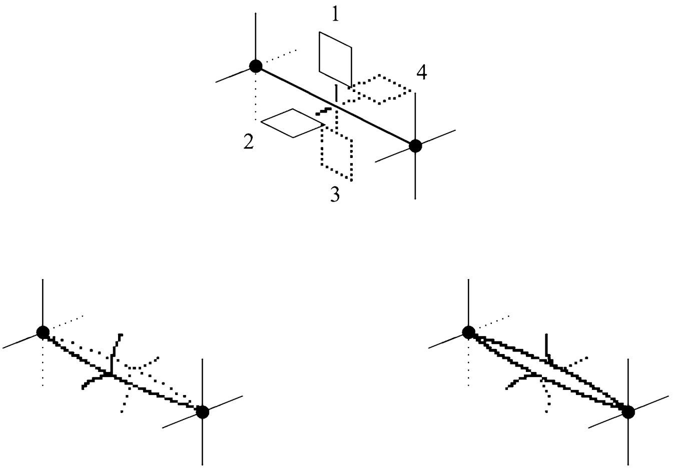


Fig. 6.7 Star ambiguity illustration

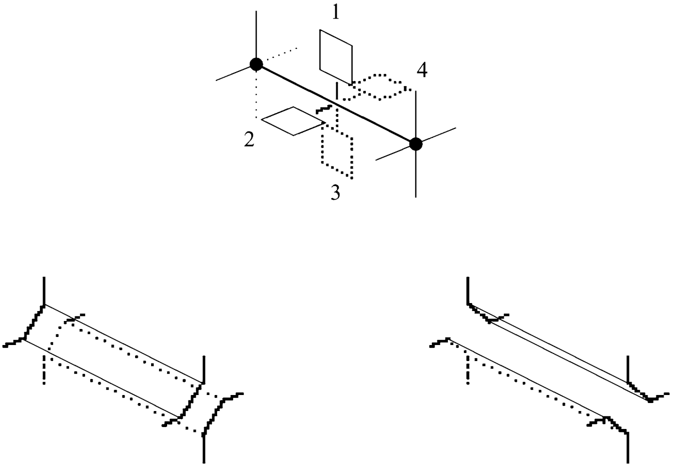


Fig. 6.8 Ambiguous chamfer operation

The point of explaining this is two-fold. The first is to explain why, sometimes, you get discrimination between model elements to which you would like to apply an operation. Secondly, to explain the notion of edge duplication and edge-link pairing, which is how to interpret the multi-link edges.

6.2 Wireframe Models

Wireframe models are another example of partial models which can be useful for special purposes, such as sketching the centre-lines of pipework. At one time CAD systems used wireframe models exclusively.

6.2.1 Wireframe Datastructure

The datastructure of wireframe models is very simple, consisting of “nodes” and “links”, as shown in Fig. 6.9. The nodes are represented by vertices and the links by edges to use the same elements as for sheet objects and solids.

While these are enough for simple shapes, the lack of surface information is a handicap for many functions, from drawing to manufacturing. A wireframe graphics view of an object is shown in Fig. 6.10 to emphasis this.

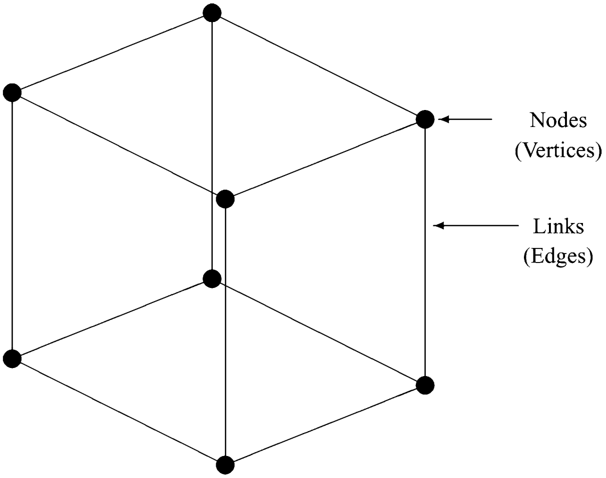
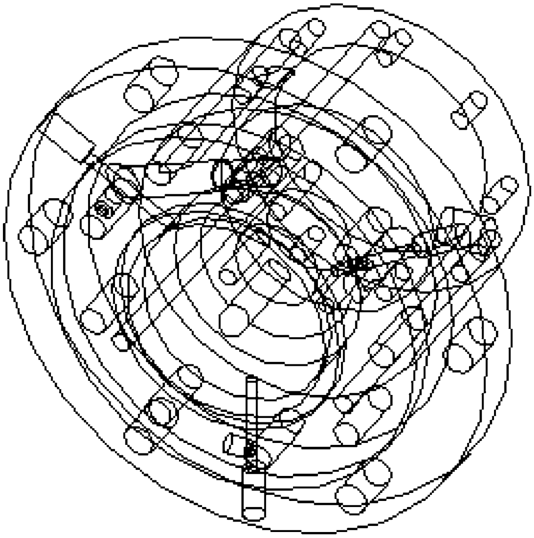


Fig. 6.9 Wireframe model elements

Fig. 6.10 Complex wireframe image



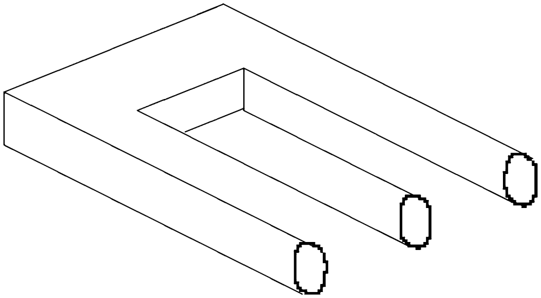


Fig. 6.11 Illusory figure

6.2.2 *Impossible Wireframes*

It is also possible to make objects which are not realisable, as in Fig. 6.11. Sheet objects and volume objects are *Eulerian* objects, which means that they follow the formula described in Sect. 2.7.3. This means that there are restrictions on how you build models, which precludes models such as that in the figure. Other models, such as Möbius strips, or Klein bottles can also be created using wireframe techniques, but are excluded using volumetric techniques. More usual than these recreational objects, though, is that it is possible to create erroneous objects.

6.2.3 *Wireframes and Modelling*

An old research topic was the automatic conversion of wireframe models to solids. It is not possible to guarantee a conversion and some counter-examples exist of objects which cannot be converted. A feasible use for wireframe models is as a support for sketching or they can be used as idealisations and then converted, as with the operation described in Sect. 4.13.

One current use for wireframes is for defining two-dimensional shapes to be set into surfaces, as has already been described in Sect. 3.7. They can also be used for modelling curves in a geometric package and, for example, swept to create surfaces, as will be described in Sect. 6.3.1.

6.2.4 *Wireframe Experiments*

6.2.4.1 Creating Pipework

Make a shape like that on the left of Fig. 6.12. Extrude a circle along this path to create a simple pipe. You can finish the pipe using the shelling operation to create the interior.

If you have a shape like that on the right of the figure you cannot create it in one piece with the extrusion along path operation. You can create the basic shape as

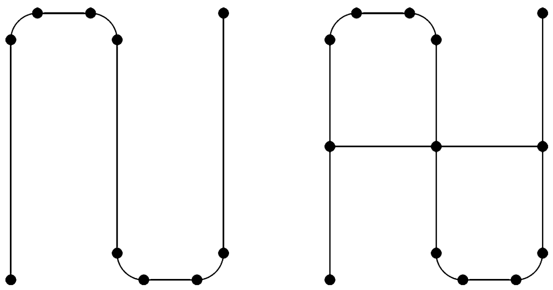


Fig. 6.12 Pipework centrelines

with the figure on left and then add the additional shape with a second extrusion along a path.

The questions concern how to perform the various parts of the operation. If, after the first extrusion, the basic shape is turned into a extruded shape then the second operation will create interior elements rather than the desired shape. The second shape should be added before the shelling operation to create the interior. However, this creates the final object in one piece, but it would normally be created in several shaped pieces which would be welded together. This can be done by creating the outer shape as a solid model, separating it into elementary parts, and then creating the individual shelled pieces to be made.

The purpose of this long explanation is to say that, while the facilities may exist to create simple models, real applications need to be based around correct interpretations rather than using standard tools.

6.3 Sheet Models

Sheet models are a useful tool for representing idealisations of thin-plate models or for representing surfaces, as described in Sect. 5.8.1. Sheet models are non-manifold because they are infinitely thin, but in some applications it is quite natural to use them rather than volumetric models. The GPM APC module was mentioned at the beginning of this chapter, and there was a successful oil rig platform design application based on it. Other applications, such as layouts, for modelling shapes to be cut from cloth or shapes to be cut from thin metal sheets, do not need volumetric models. It is more efficient and more natural to use sheet models.

An important part of the use of sheet models is their interpretation as idealisations of thin-plate models. In this respect, the duplication of edges is more natural than using a star representation for coincident edges. The duplicated edges lie on different sides of the sheet objects and would be slightly apart if the sheet object were expanded to produce a volumetric model. This was one of the reasons that this method was used in the GPM volume module.

6.3.1 Extruding Wireframe Models

A common operation is to extrude curves to produce surface models. This means that edges go from being non-Eulerian to being Eulerian as part of the extrusion process. The edges go from being linked through vertices to being linked into chains as borders of faces. This was also discussed briefly in Sect. 4.2. It is important to know how wire extrusion is integrated into the CAD system, whether sheet models are separate from volumetric models or coexist.

Branching wire objects have already been mentioned in Sect. 4.11. These are usually excluded from extrusion operations by CAD systems, so will not be dealt with further here.

Figure 6.13 shows an object that was used for comparison of solid modelling systems for a seminar organised by the CAM-I organisation in 1983. It is an object, reportedly of a gun-platform, which is composed of thin-walled parts.

As part of the GPM Volume module demonstration, this part was shown both as a sheet model and as a set of flattened shapes to be cut from plate material, shown in Fig. 6.14. The method for doing this is described in Stroud [3]. The operation has not appeared in commercial CAD systems, as far as I know, but it shows what could be done as part of a special application.

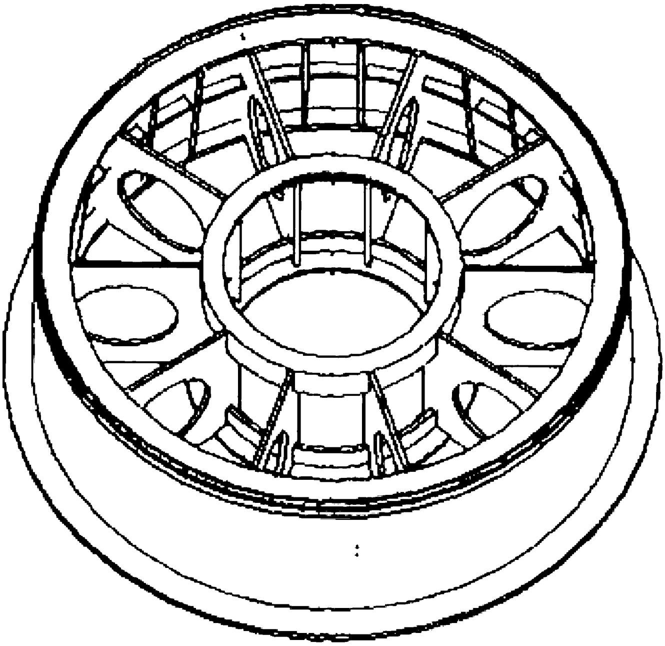


Fig. 6.13 The Contraves example

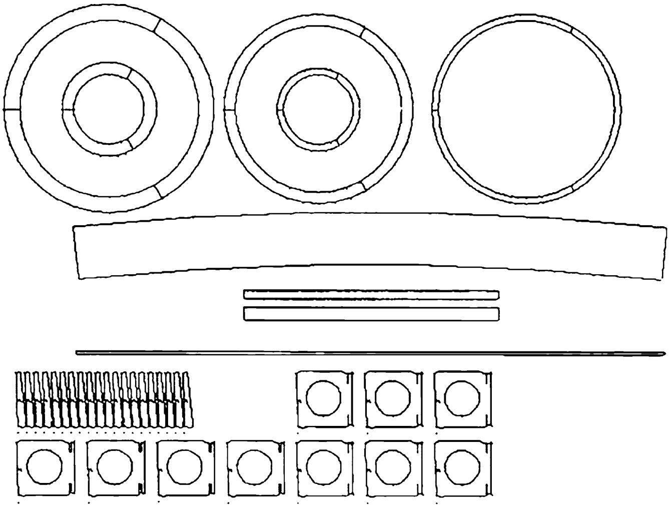
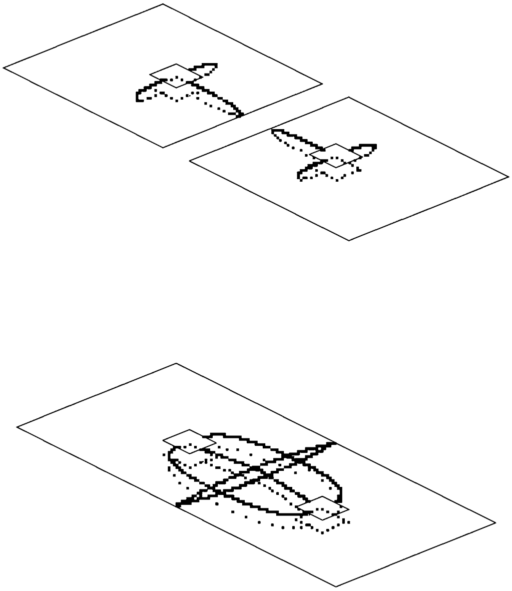


Fig. 6.14 The Contraves object extruded as flat shapes

6.3.2 Joining Sheet Objects

Joining sheet objects representing surface portions has already been described in Sect. 5.8.1. The procedure is shown in Fig. 6.15. Each edge has two loop-edge links, linked in a chain. The edge-link pairs are regrouped to form two chains, in the figure, or one if the edges are merged.

Fig. 6.15 Joining sheet objects



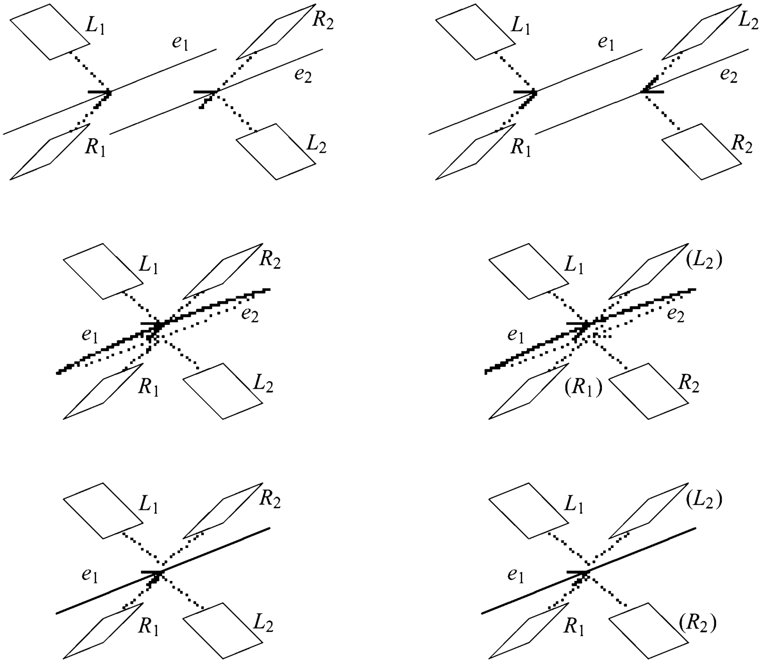


Fig. 6.16 Relinking edge links when joining sheet objects

Figure 6.16 shows how the loop-edge links are rearranged. The case where the edges are in the same direction is shown in the left hand column of the figure. The case where the edges are in the opposite direction is shown in the right-hand column. The original arrangement is shown at the top. Edge e_1 has left link L_1 and right link R_1. The edge to which it is to be joined is e_2, with left link L_2 and right link R_2.

If the edges are in the same direction and both are kept, middle left, then link L_1 is paired with link R_2 and link L_2 is paired with link R_1. If the edges are merged, bottom left, then the links are arranged in the circular sequence L_1, R_2, L_2, R_1.

If the edges are in opposite directions, and both are kept, middle right, then L_1 is paired with L_2, which becomes a right link, and link R_1 is paired with R_2, which becomes a left link. If the edges are merged, bottom right, then the links are merged into the sequence: L_1, L_2 (which becomes a right link), R_2 (which becomes a left link), R_1.

6.3.3 Volume Models to Sheet Models

Section 4.9 describes the simple way of converting volume models to sheet models as one step in the shelling process.

An operation that you find in CAD systems is to unfold, or flatten, volume models. This was described in Sect. 4.10.

These methods allow a designer to create a part as a solid and then convert it to a sheet model to be made from thin material. Converting a volume model to a sheet models is a first step in at least one flattening algorithm. This conversion allows the concave edges along which the object is to be bent to be marked as grooved for finishing operations after cutting.

6.3.4 Sheet Model Experiments

6.3.4.1 Extruding Wires

First of all, create an open shape as a sketch in the volume modelling part and extrude it in a straight line or circular arc, as in Fig. 6.17. This is the same experiment as for extrusion and is intended to show whether or not sheet models and volume models are integrated or separate. The question is not whether the CAD system can do it or not, these are simple shapes, but whether they are allowed to coexist or not.

Another extrusion experiment to try on simple shapes involves wires with one edge in the extrusion direction, as shown in Fig. 6.18. The shape on the top left of the figure should probably cause an error, as the extruded shape, bottom left,

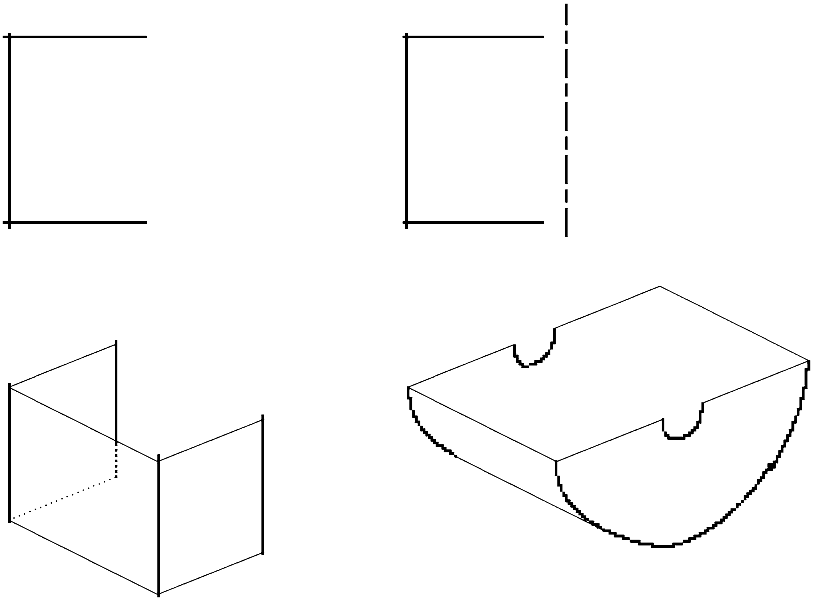


Fig. 6.17 Open sheet objects from extruded wires

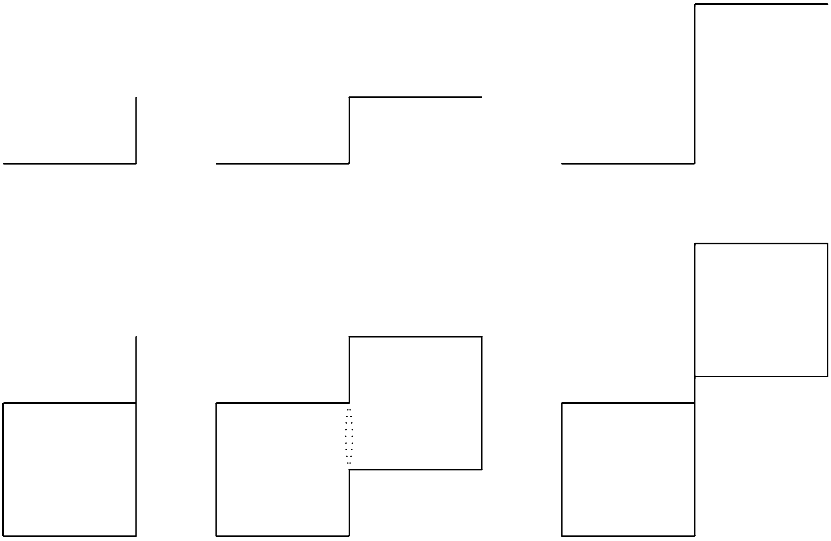


Fig. 6.18 Wire objects with critical edges

would have a dangling edge, which is not a good idea. The shape shown top middle, though, is more debatable. The result of an extrusion would be a valid shape, bottom middle, though the internal edges, shown dotted, would have to be handled properly. The third shape, top right, would cause problems, because the extrusion would leave a single wire edge in the middle, or the shape would have degenerate parts.

Test these three shapes to see if the CAD system allows them or not. For the shape on the right, make sure that the middle edge is longer than the extrusion distance.

6.3.4.2 Extruding Branching Wires

This is another experiment that has been suggested before, in the section on extrusion. Extruding branching edges should not be a problem, except for critical cases where one or more edges are in the extrusion direction or there are coincident edges. The reason for not implementing this is a strategic decision by CAD implementers about the complexity allowed. The problem is to work out the connections at the complex branch-points. If there are at most two edges at every vertex then there is no problem. If there are more then a geometric test is needed to sort out pairings.

One of the edges is taken as a base edge, the zero degree edge and the other edges are ordered using their tangent vectors, projected onto the plane defined by the common vertex and the extrusion direction (Fig. 6.19). This method was developed by Müller [4]. Note that edges 3 and 4 have the same tangent direction,

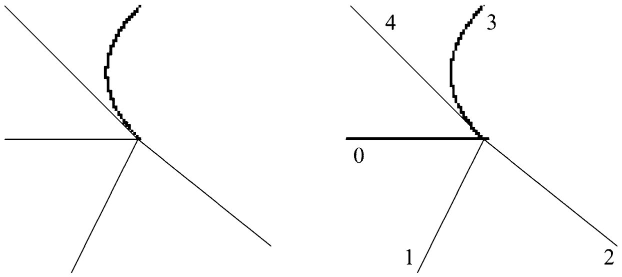


Fig. 6.19 Ordering wire branches

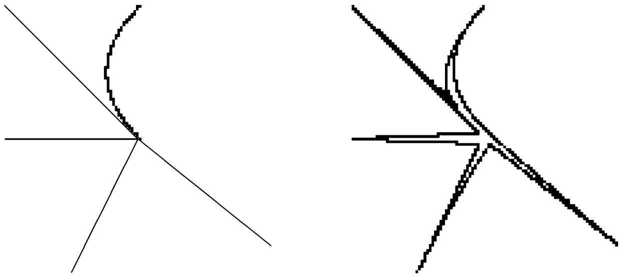


Fig. 6.20 Slicing wire branches

but this is sorted out in Müller's method by using the curvature. The method will not work, though, if there are coincident edges or edges parallel to the extrusion direction.

An alternative is to slice the edges to create a degenerate face, which is then extruded. See Fig. 6.20. This can then be extruded using a normal face extrusion and then the edges collapsed back as a finishing process.

The actual details of how this is done in a particular CAD system are not really important, the question is whether or not the system allows you to extrude branching wires.

6.3.4.3 Joining Three or More Sheet Objects

Dedicated CAD systems for thin plate modelling should be able to handle this case, though as part of a general CAD system this may not be allowed. A simple test is shown in Fig. 6.21. Create a line on the Z = 0 plane, say from $(-50, 0)$ to $(0, 0)$ and extrude it 60. Create a second line, from $(30, -50)$ to $(0, 0)$ and extrude this 60, as well. Finally, create a third line, from $(30, 50)$ to $(0, 0)$ and extrude this 60, the same distance for all three sheet objects. Now try joining them. In the same way that branching wires can be handled, sheet merging can be handled if the CAD system developer is prepared to invest the effort.

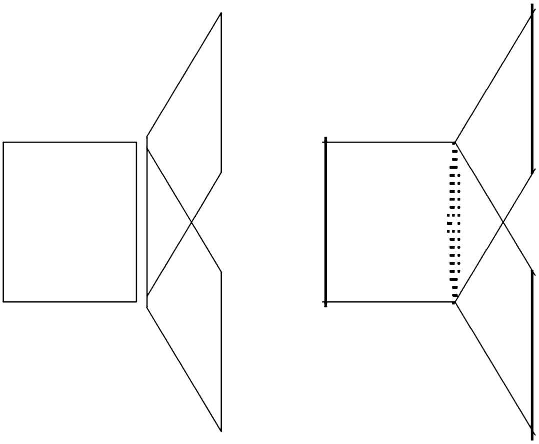


Fig. 6.21 Slicing wire branches

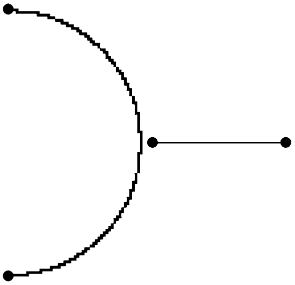


Fig. 6.22 Joining non-matching sheet models

6.3.4.4 Joining Sheet Models Without Matching Edges

This experiment is to test how much effort has been put into sheet modelling in the CAD system. This is equivalent to a Boolean operation on sheet models and is not impossible, technically, just requires some effort from the CAD system implementer.

On the Z = 0 plane, draw a semicircle, or, if you wish, a full circle, radius 25, say. Extrude this 50 to create the first sheet. Now make a line just touching the middle of the semicircle, from (25,0) to (50,0), say, as illustrated in Fig. 6.22. Try to join them and check whether or not the system allows this.

6.4 Partial Models

Partial models are a special type of sheet model which are useful as mechanisms for applications. They have faces and surfaces on one side but nothing, or maybe one unsurfaced face, on the other side, as was done in the BUILD system. If there is nothing on the back of the partial model, as is normal, then the boundary edges are only partially defined, with only one loop-edge link.

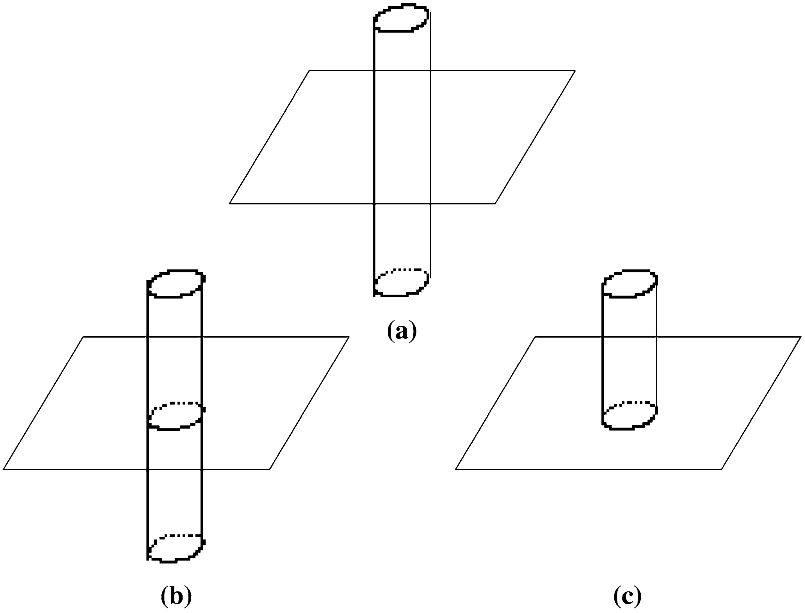


Fig. 6.23 Joining non-matching sheet models

You would not expect to see these as part of the CAD system, they are tools for applications or intermediate results, perhaps. Partial models act as though they have unlimited material behind them. Provided any operation does not go beyond the boundaries of the partial model they act as normal volumes. An application using partial models will be described in Sect. 10.3.2.

A simple illustration of the differences between sheet objects and partial objects is shown in Fig. 6.23. If you were to add a cylinder to a square shape, shown in Fig. 6.23a you would get the result in Fig. 6.23b if the square shape were a sheet object and the result in Fig. 6.23c if the square shape were a partial object. The reason is that the sheet object would have two intersections and both top and bottom of the cylinder would be classified as outside the sheet. On the other hand, with a partial object, there would be only one intersection, with the defined face, and only the top of the cylinder would be classified as outside the object. In addition, the addition of the cylinder to the sheet object should really be classified as invalid, since this would create a mixed object with volumetric and sheet parts.

6.5 Non-Manifold Volume Models

Beware of non-manifold portions in volumetric models.

Although creating such model parts is technically correct, integration in all operations seems patchy, at least at the time of writing. The problem of handling star representations and pairing loop-edge links has already been mentioned. As with sheet models, this comes back to having a clear method of interpretation of what the non-manifold edge means: is it where there is material or is it where objects touch without being joined? Similar considerations exist for objects just touching at vertices.

Note, also, that there are two kinds of non-manifoldness that you might create. The first kind is where there are edges with more than two faces or vertices with multiple edge sets, the second is where two elements touch without having common topological elements. An example of the second type is shown in Fig. 6.24. The object was created using an extrusion along a path, described in Sect. 4.2.8.

The sort of object shown in the figure is hard for a modelling system to resolve because the normal point-set considerations cannot be used to sort out the object. Points have neighbourhoods which are not homeomorphic to discs, if you prefer to have it that way. An attempt to subtract the object in Fig. 6.24 from a block failed because, the system said, it could not sort out the tangential relationships.

Sorting out this type of non-manifold object would require a special type of Boolean operation, for evaluating self-intersecting objects, which, instead of taking two objects and comparing their faces, compares the faces of a single object with each other. It is not impossible to implement, but I have not yet seen such an operation in a CAD system. Such an operation is linked with another topic, called "model checking" or "model healing", which will be described briefly in Sect. 14.3.

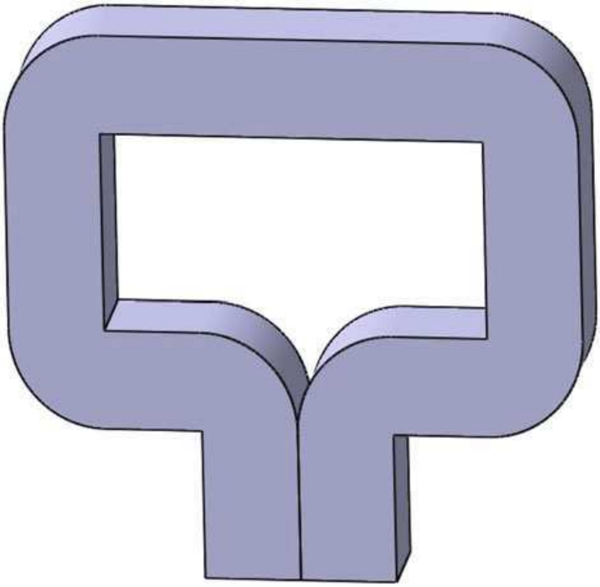


Fig. 6.24 Non-manifold solid with touching elements

6.5.1 Datastructure Improvements

The inherent ambiguity of the star representation as commonly implemented is a drawback, but there are ways of improving it. The best work I know in this area was by Luo and Lukacs [5, 6]. They introduced elements called "bundles" for handling non-manifold vertices and "wedges" for non-manifold edges. Bundles and wedges can be thought of as being equivalent to loops for faces in that they allow multiple associations between datastructure elements. A full treatment of this, though, lies outside the scope of this chapter, which is intended to deal with practical aspects of the use of non-manifold solids.

The reason for mentioning this is that the current structure might change to avoid this ambiguity. If it does, in the future, you may be prompted to tell the system whether you mean objects to "touch-and-join" or "touch-but-miss". If you are asked, then it means that the system can distinguish between the cases and operations such as chamfer or blend on non-manifold edges and vertices may work. If you get this kind of message, check what happens by blending the non-manifold model part.

6.5.2 Non-Manifold Volume Applications

To some extent, non-manifold modelling is a solution in search of a problem. The use of sheet- and wireframe-models as idealisations is a clear and useful facility that has proved its usefulness. The uses for non-manifold solid models is less clear. One suggestion has been to use them for finite-element modelling. However, finite element models tend to have a large number of elements and the overheads associated with a volume would make these difficult to handle.

One more appropriate application area is that described by Luo and Lukacs and involves the use of non-manifold volume models for process planning. This is a development building on an idea proposed by Malcolm Sabin, one of the most influential figures in CAD/CAM. At a conference in 1983 Sabin suggested building back a CAD model to its stock as a way of manufacturing planning. Lukacs and Luo's idea was to keep the built-back volumes separate and to link them with the CAD model to create a compound, non-manifold model. The separate parts could then be easily identified for manufacturing planning, I know of no CAD systems using such methods, though.

6.5.3 Volumetric Experiments

The first three of these come from previous exercises. They are simple ways of creating non-manifold solids.

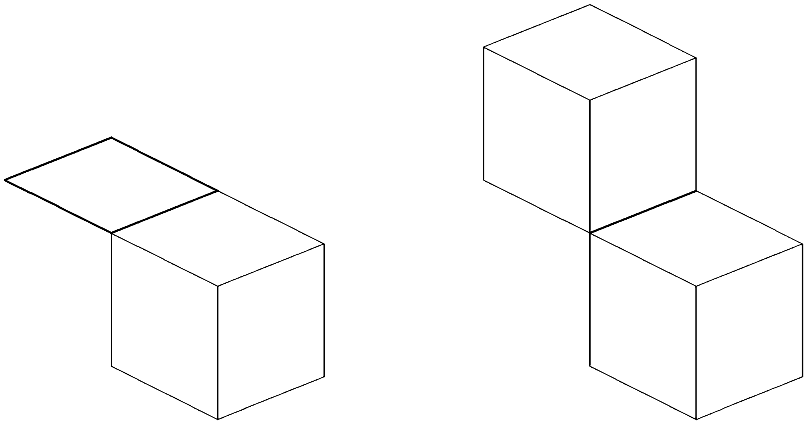


Fig. 6.25 Creating a non-manifold edge

6.5.3.1 Touching Edges by Extrusion

This exercise was described in Sect. 4.2. Make a square shape 100 × 100. Extrude it 100 units. On the top face, create a new square, 100 × 100, which touches one of the edges of the top face and extrude this 100 units to create the object. See Fig. 6.25.

6.5.3.2 Touching Vertices by Extrusion

Another exercise from Sect. 4.2. This is a similar exercise to the previous one, except that the shapes touch only at a vertex. Make a square shape 100 × 100. Extrude it 100 units. On the top face, create a new square, 100 × 100, which touches one of the edges of the top face and extrude this 100 units to create the object. See Fig. 6.26.

6.5.3.3 Extruding Touching Shapes

Yet another exercise from Sect. 4.2. Make a pair of triangles touching at a vertex, as shown in Fig. 6.27. The order of definition shown be p1-p2-p3-p1 and p4-p5-p2-p5. You should not try making this as p1-p5-p4-p3-p1, which will not create the important vertex in the middle.

The actual size of the triangles is not important, just that they touch at one vertex. Extrude the touching triangles about the length of the side to give a reasonable thickness. The question is what happens with the common vertex, is it one edge or two?

6.5.3.4 Touching Faces

This creates the shape shown in Fig. 6.24. In the YZ plane, Y = 0, create the shape shown in Fig. 6.28. On the XY plane, Z = 0, create a 25 × 25 square shape

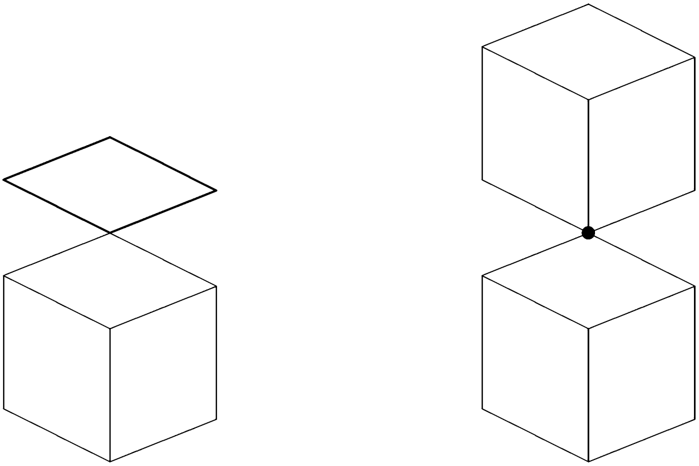


Fig. 6.26 Creating a non-manifold vertex

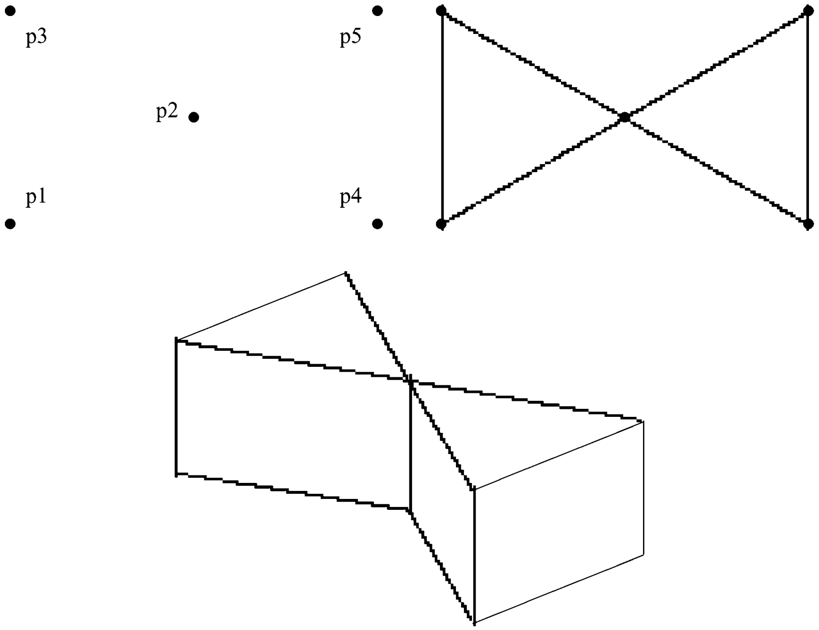


Fig. 6.27 Creating touching triangles

centred about one of the end vertices of the shape shown, marked A and B in the figure, and extrude it along the path to create the shape.

This method of creating touching faces may not work in some systems which use Boolean operations to join sub-extrusion elements.

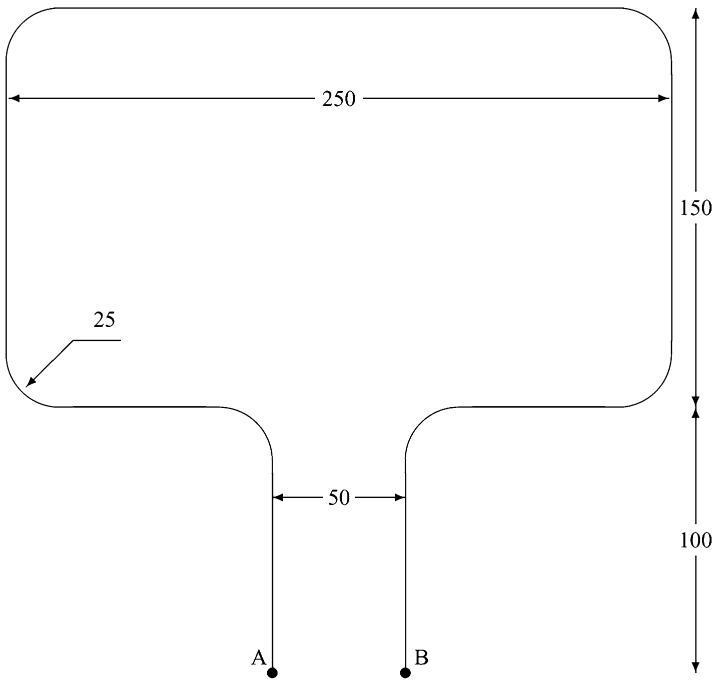


Fig. 6.28 Path shape for touching faces

6.5.3.5 Chamfering to Create Touching Elements

Create a thin shape such as that shown in Fig. 6.29a. The arms might be 75 units long and 10 units wide. Extrude the shape 75 to give a volumetric model like that shown in Fig. 6.29c. Now chamfer the convex edge as shown in two dimensions in Fig6.29b, and in three dimensions in Fig. 6.29d.

If the depth of the chamfer is correct, $armthickness \times \sqrt{2}$, then you get a non-manifold edge, shown dotted in Fig. 6.29d. If Boolean operations are used to create the chamfer then the edge might be a star non-manifold edge. If a simple local operation is used then the edge may touch the face, but not be associated with it. You can test this by attempting to blend the edge.

6.6 Chapter Summary

This chapter deals with the subject of non-manifold and idealised models. While idealised models have a clear use as design sketches, non-manifold volume models have a less clear role in modelling. Being able to work with idealisations as part of the design process adds fluency to CAD use and may also help by providing clearer design intent.

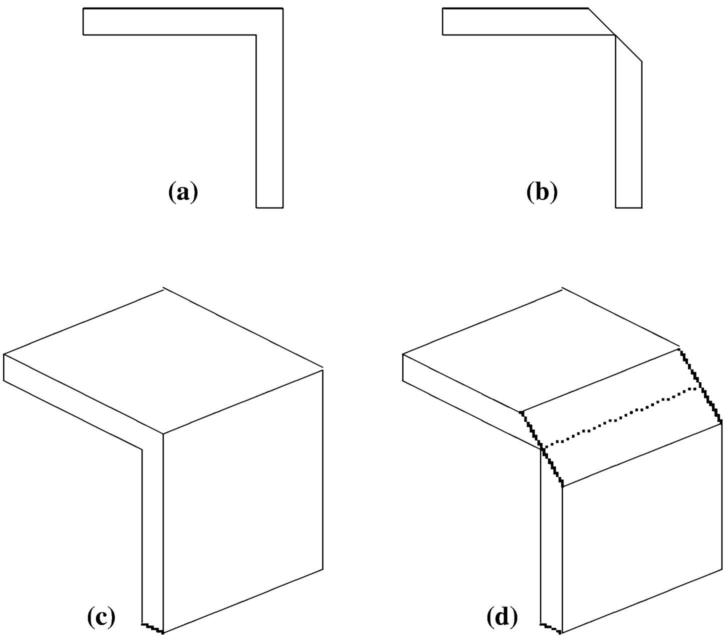


Fig. 6.29 Chamfering to create touching elements

6.7 Non-Manifold Exercises

6.7.1 Idealisation Matching

To practice identifying cases where idealisations are useful, identify whether idealisations might be useful for the following design tasks and, if so, which type, or types, of idealisation you would use. Write short notes to justify your answer.

1. A drilling platform structure.
2. Factory piping layout.
3. A building design.
4. A ship design.
5. Car body design.
6. The Gehause Rohteil, shown in Fig. 1.73.
7. The Eiffel Tower.

6.7.2 Building the Excavator

Create an assembly model of the excavator shown in Fig. 6.2 use your own judgement as to which elements to create unified and which to keep separate.

Write down a list of the different models in your assembly, together with the type of model they are. How would you convert the idealisations into volumetric objects? Which operations would you use?

References

1. Kjellberg, T.A., Lindholm, G., Sorgen, A., Haglund, G.: GPM Report 10: GPM—Specifikation Volymgeometri. Department of Manufacturing Systems. KTH, Stockholm, Sweden. Confidential document, ISBN 91-85212-54-7 (1980)
2. Kjellberg, T.A.: Integrerad Datorstöd for Mänsklig Problemlösning och Mänsklig Kommunikation inom Verkstadsteknisk Produktion begränsat till Produkt-utveckling, Produktionsberedning, Kon-struktion och Till-verknings-beredning. En systemansats baserad paa Produkt-modeller. Ph.D. Dissertation, Department of Manufacturing Systems, KTH, Stockholm, Sweden (in Swedish) (1982)
3. Stroud, I.A.: Boundary Representation Modelling Techniques. Springer, Heidelberg (2006)
4. Müller, M., Stroud, I., Xirouchakis, P.: Preprocessing of degenerate slices in layered manufacturing. In: Topping, B.H.V. (ed.) Developments in Engineering Computational Technology, pp. 63–68. Civil-Comp Press, Scotland, ISBN 0-948749-70-9 (2000)
5. Luo, Y.: Solid modelling for regular objects Renewed theory, data structure and Euler operators. Ph.D. Dissertation, Computer and Automation Institute (1991)
6. Luo, Y., Lukacs, G.: A boundary representation for form features and non-manifold solid objects. In: The First ACM/SIGGRAPH Symposium on Solid Modeling Foundations and CAD/CAM Applications, Austin, TX, June (1991)

Chapter 7
The CAD Interface and Graphical Output

This chapter deals with graphics, which is how you know what's been done, Fig. 7.1. As stated earlier, graphics can be divided into two categories: dynamic graphics and static graphics, corresponding to the CAD image and engineering drawings. They differ in their roles, but many of the principles are the same.

Note, also, that there are some good books on graphics that give an in-depth coverage of the topic. It is not the intention to cover all the topics in graphics, but to explain their interaction with CAD. Good books for graphics are Newman and Sproull [1], Foley et al. [2] and Szirmay-Kalos et al. [3], for example. The whole topic, though, is very extensive. The increasing use of computer graphics in simulators, films and general image processing in films and so on has ensured the vitality of the subject. Some of the advances are more widely useful and have come through to CAD.

7.1 Graphics Transformations

The same transformation mechanism as has been described in Sect. 5.2.2 are also used heavily in graphics.

CAD models are three dimensional and graphics is two dimensional, at least for the moment. This means that there is a projection onto the graphics plane. This can be understood relatively easily by considering the graphics coordinate system, as shown in Fig. 7.2.

It is necessary to have an viewing position, a centre point for the projection, called the view origin and a vector defining the up direction of the image to define a system of coordinates. The vector from the viewing position to the view origin defines the "into-display" vector. The vector product of the up direction and the into-display vector gives the "display right" vector. The vector product of the display-right vector with the into-display vector gives the "display-up" vector. Note that this is a left-hand coordinate system.

I. Stroud and H. Nagy, *Solid Modelling and CAD Systems*,
DOI: 10.1007/978-0-85729-259-9_7, © Springer-Verlag London Limited 2011

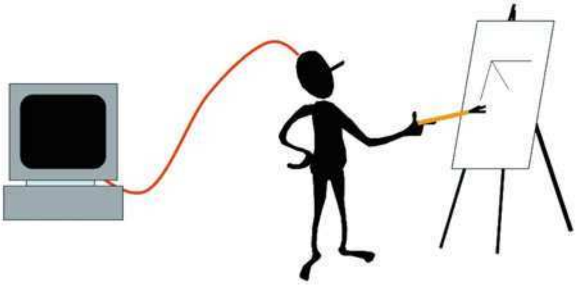


Fig. 7.1 Finding out what's been done

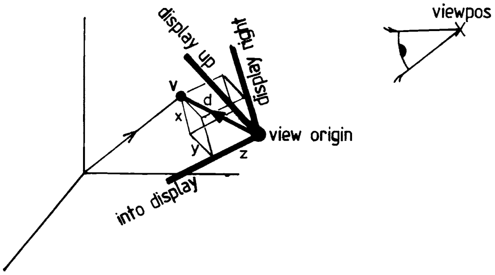


Fig. 7.2 Graphics coordinate system (from [4])

Any point, *v*, in normal three-dimensional space can be converted to the two-dimensional graphics space coordinates by subtracting the graphics origin and calculating the scalar products of the display-right and display-up vectors to produce the graphics X-, and Y-coordinates. Thus:

$$v_G = v - view\_origin$$

$$v_X = v_G \cdot display\_right$$

$$v_Y = v_G \cdot display\_up$$

This can be replaced by matrix multiplications. The first operation can be replaced by the matrix:

$$\begin{bmatrix} 1 & 0 & 0 & -O_x \\ 0 & 1 & 0 & -O_y \\ 0 & 0 & 1 & -O_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The operations to produce the two-dimensional coordinates give the matrix:

$$\begin{bmatrix} DR_x & DR_y & DR_z & 0 \\ DU_x & DU_y & DU_z & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where *DR* stands for Display_Right and *DU* for Display_Up.

These two matrices can, of course, be combined as:

$$\begin{bmatrix} DR_x & DR_y & DR_z & 0 \\ DU_x & DU_y & DU_z & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -O_x \\ 0 & 1 & 0 & -O_y \\ 0 & 0 & 1 & -O_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} DR_x & DR_y & DR_z & T_x \\ DU_x & DU_y & DU_z & T_y \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where $T_x = -O_x DR_x - O_y DR_y - O_z DR_z$ and $T_y = -O_x DU_x - O_y DU_y - O_z DU_z$. Note, though, the third row of the second matrix, which is:

$$\begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}$$

This means that the graphics transformation is non-invertible. It is preferable to retain the Z-coordinate as well, even though it is not used as a screen position. In fact, the Z-coordinate can be used for various purposes. The modified projection matrix would be, then:

$$\begin{bmatrix} DR_x & DR_y & DR_z & 0 \\ DU_x & DU_y & DU_z & 0 \\ ID_x & ID_y & ID_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where ID stands for Into_Display.

Transformation matrices in graphics have been used for a long time, now. Their importance has increased, though, as special graphics devices, including graphics cards, have come onto the market. A classical architecture for such a graphics device is shown in Fig. 7.3. There was an old system for vector graphics, from the Evans and Sutherland company, in which the information in the graphics memory was line segments and which worked this way. Later systems used triangles and could produce shaded images, but worked in the same way.

Another revolutionary system was developed by Silicon Graphics, and contained what was called a "pipeline" of transformations to perform various tasks. At the start were transformations which performed rotation, translation and scaling, at the end were perspective and clipping. Clipping is the removal of what lies outside graphics space. The graphics space is in the form of a rectangular, as shown on the left of Fig. 7.4. If the image is being drawn in perspective space then the graphics space is like a slightly truncated pyramid, on the right of Fig. 7.4.

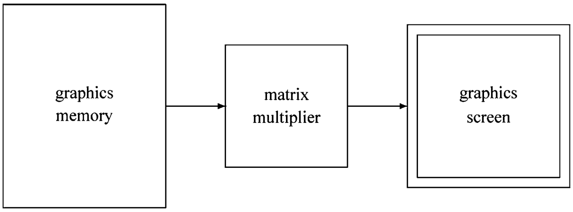


Fig. 7.3 Graphics hardware architecture - simplified

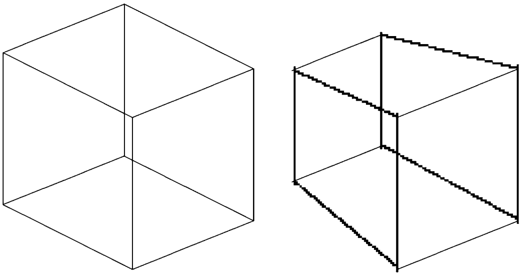


Fig. 7.4 Graphics space, after Foley et al. [2]

The side faces around the block correspond to the sides of the display region. Anything that lies outside these, lies outside the displayed region and so is removed. The front and back have different functions. Anything lying in front of the front clipping plane can be thought of as lying behind the eye position, so is invisible. Anything lying behind the back clipping plane can be thought of as too far away to be seen. The clipping due to the front and back plane can seem strange, at times. These can also be used to enhance the image, as will be explained later.

The advantage of the use of matrices is that the graphics model being shown does not change, only the matrices. Every time you rotate, translate or scale the picture on the screen, you are changing only a transformation matrix. This matrix is then placed into the architecture and, at the next update, a new picture is generated. The application of transformations is done by special hardware so that you do not overload the CAD system. The graphics information has to be kept simple, though, to let it be transformed in this way.

7.2 Displaying Objects

The first thing to note is that models are usually now exact whereas graphics is approximative. When you look at a curve on the screen it is represented by a lot of short straight line segments, but the real curve, in model space is usually a smooth curve. The picture you get is a "reasonable approximation" to the model, don't be perturbed by the approximative nature of the image.

Objects can be displayed in various ways, wireframe, shaded, hidden-line etc. These are used in different ways. Figure 7.5 illustrates three principle ones, with images generated by the ACIS viewer. In Fig. 7.5a the image is a wireframe image, showing only the edges of the object. Figure 7.5b shows a hidden-line image, an image where the edges behind the faces are not shown at all. Finally, in Fig. 7.5c there is a shaded image of the object.

This can all be done with the same graphical model made up of suitable approximating elements. For high quality images this would be done using exact methods, and high quality methods such as ray tracing will not be dealt with here. The classical method uses graphical models made of triangles, which are general stable forms for handling by hardware.

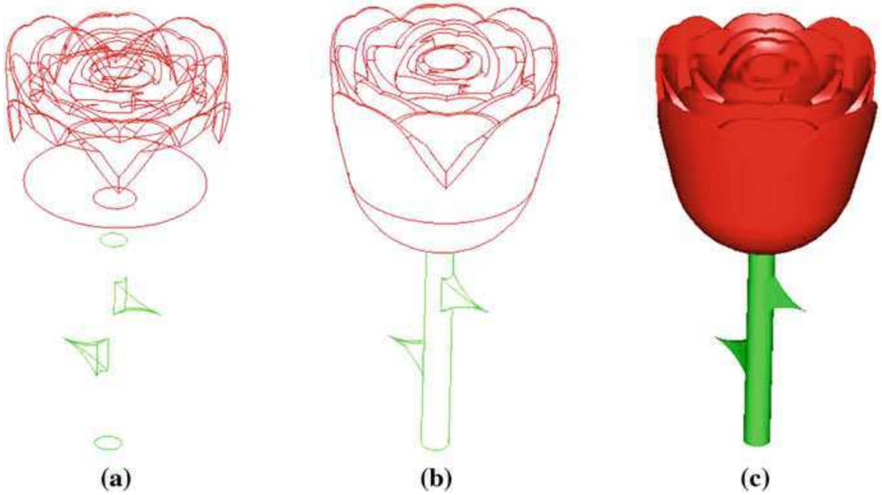
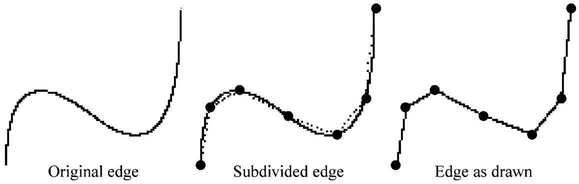


Fig. 7.5 Wireframe-, hidden line- and shaded-images

Fig. 7.6 Approximating a curved edge



7.2.1 Approximating Objects

The first step in producing the graphics model which is used for display is to approximate the curved edges with polygonal edges. For regular curved edges, such as circles, it may be enough to calculate a regular series of points because the edge has uniform curvature. Approximating general edges is shown in Fig. 7.6.

To approximate a curved edge, a number of points are calculated along the edge, Fig. 7.6 middle. These points are then connected by line segments, Fig. 7.6 right. It is usual to use adaptive subdivision with a tolerance, as illustrated in Fig. 7.7.

The middle point of each line segment is checked to see how far it lies from the curve. If the distance is more than some predefined limit then a new intermediate point is calculated, Fig. 7.7 middle. This step is repeated as often as necessary. The old and new points are then connected to produce the edge image, Fig. 7.7 right.

Once the edges have been subdivided, the faces can be divided into triangles. This is shown in Fig. 7.8. The original face is shown in Fig. 7.8a. The face is L-shaped with a square hole. The first step in triangulation is to join all the concave vertices to neighbouring vertices. A concave vertex is one around which

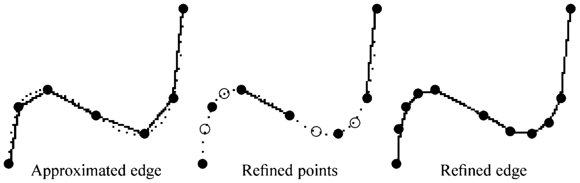


Fig. 7.7 Refining an approximation

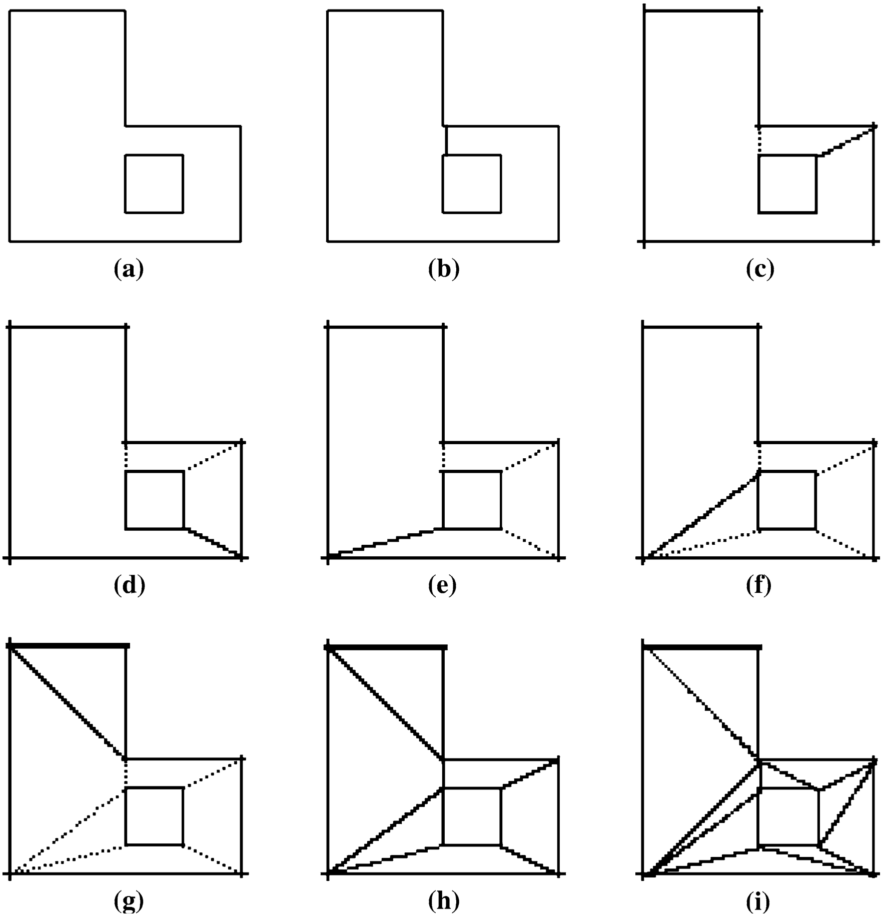


Fig. 7.8 Triangulating a face

the face makes an angle of 180 degrees or more. There are five in this example, one vertex in the external boundary of the face where the two arms of the L-shape meet and the four surrounding the hole.

In Fig. 7.8b one of the hole vertices is joined to the concave vertex of the external boundary, which happens to be the closest pair to join. Note that this leaves both vertices still concave and they will again have to be joined in later steps, in Fig. 7.8f, g. In the next join, in Fig 7.8c, the next closest pair are joined. Previously added new edges are shown dotted in the figures to make it easier to see which is the new edge. The joins continue until all vertices are convex and the face has been divided into a set of convex regions, as shown in Fig. 7.8h. The final step is to divide these convex subfaces so that all subfaces are triangular, as shown in Fig. 7.8i.

One reason for explaining this method is that triangulation is used for other purposes, notably calculation of volume, area and so on and also for exporting models in STL format. Look carefully at the final set of triangles. For graphics and for other purposes the shape of the triangles does not seem to be taken much into consideration. The control of the facetting is often done using a single parameter, the chord height tolerance. There are two other ones which are used, the minimal internal angle and the maximum edge length. These are illustrated in Fig. 7.9.

The chord height is the measure from the straight line to the exact edge portion which is approximated by the line. If this value is greater than the maximum allow chord height difference then the line is split and edge approximated by two pieces of the range. This is recursively applied until the lines approximating the edge are sufficiently close. The chord height is also used to check the discrepancy between a surface and its approximating triangles. If the centre of the triangle is further from the surface than the chord height tolerance then the triangle, and possible its neighbours, is subdivided and the subdivided triangles moved closer to the surface. Figure 7.10 shows two subdivision schemes. The original facet is shown in Fig. 7.10a. If a new vertex is inserted in the middle of the facet then you get the subdivision in Fig. 7.10b. This doesn't affect neighbouring facets but the new subfacets are more elongated than the original. In Fig. 7.10c each of the facet edges is subdivided, creating more regular sub-facet shapes, but the neighbouring facets also have to be split.

The minimal internal angle is a value used for regularising triangles. If any of the angles, α, β or γ is less than the minimum angle, the triangle is redefined to increase the proportions of the sides.

The edge length parameter is just to break long edges into multiple pieces, which can also serve to make the triangles more regular.

Note, finally, that if the same facetted graphics model is to be used for drawing wireframe-, hidden line- and shaded images then it is necessary to be able to distinguish between the external facet edges and internal facet edges. The external facet edges are the ones corresponding to the boundary of the face. These correspond also

Fig. 7.9 Triangulation tolerance types

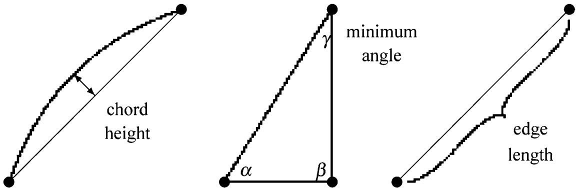
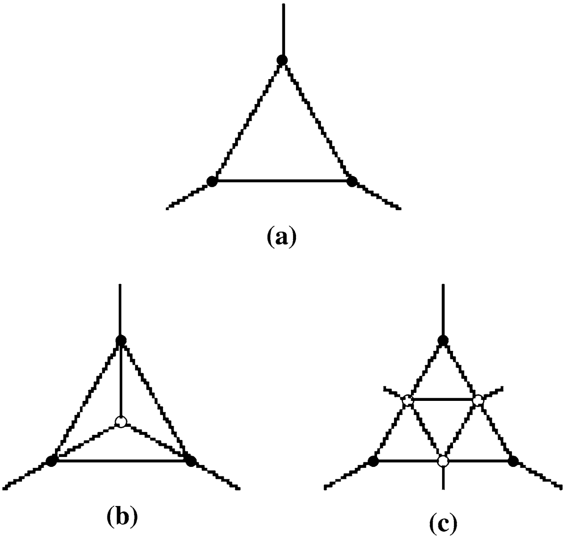


Fig. 7.10 Facet subdivision strategies



to the edges bounding the face. If the external facet edges cannot be distinguished then it is necessary to have at least two models in the graphic memory.

7.2.2 Wireframe Drawing

Wireframe drawing is the simplest type of display. If done by software it can be described simply as the following procedural method.

for_every_edge_in_body(b, draw_edge)

where the "draw_edge" function steps along the edge drawing straight line segments to approximate the edge, as described in the previous section.

However, using the graphics model, this would involve looking at all facets generated to approximate an object, and draw those facet edges which correspond to the edges of the face.

If the graphics model is to be used for hit-testing directly then it would be necessary to label all the straight line sections corresponding to an edge with some sort of internal label. More, though, about this topic in Sect. 7.4.

7.2.3 Silhouette Lines

One feature of curved images is drawing what are termed the "silhouette lines". These are illustrated on the simple example of a cylinder in Fig. 7.11, from [4].

A cylinder may be represented with one or more side edges, as in Fig. 7.11a. If all these edges are drawn then you get the image in Fig. 7.11a. However, the side

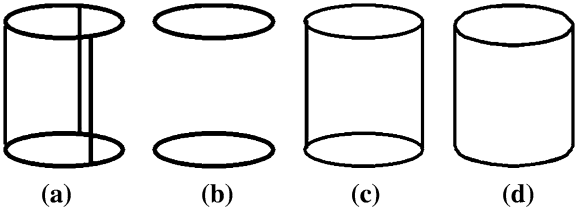


Fig. 7.11 Fake edges and silhouette edges in the model (from [4])

edges, also known as "fake" edges, are not really part of a minimal representation because the faces to the left and right of each edge lie in the same surface. Simply ignoring these when drawing gives you an image such as that in Fig. 7.11b. This lacks cohesion between the top and bottom circles, so drawing the silhouette curves, where the normal vector to the surface is perpendicular to the view direction, gives a slightly more realistic edges, Fig. 7.11c. Where the silhouette lines meet the boundary curves of the face the edge is divided into a visible and invisible portion. The top circle, however, is visible because it bounds the top face, a face whose surface normal is pointing at least slightly towards the viewing point, so is drawn whole. Only half of the bottom circle is drawn, giving the pseudo-hidden line image in Fig. 7.11d. This is only possible, though, with convex objects and normally a realistic image is more complex to produce.

7.2.4 Shaded Images

The best quality shaded images are produced by computationally expensive techniques such as ray tracing. These can be quite realistic and can produce advertising pictures. However these techniques take too long for interactive display so working images are produced using triangular approximations. Note that some graphics packages allow general planar shapes to be transferred, but the triangle is the simplest form, and general, so that is used here. Bodies are triangulated by preprocessing the edges to subdivide curved edges and long edges and then the faces are divided into triangles. The faces are first subdivided to remove "concave vertices" to leave convex sub-faces and then the sub-faces are divided up to produce triangles. The subdivision is illustrated in Fig. 7.8.

Each facet was, originally, defined by three points and a normal vector. A more modern version is to have a normal vector for each vertex point, as shown in Fig. 7.12. This allows a better shaded image than with one normal vector, where the triangle would have the same basic colour, resulting in visual discontinuities.

The colour of a point in an image depends on the surface normal, the position of the point relative to the light source and the position of the eye, as shown in Fig. 7.13.

In Fig. 7.13a the vector from the light source to the point gives the incident vector. The reflected ray is symmetric about the surface normal at that point. The size of the reflected ray depends on the reflectivity index of the object, which

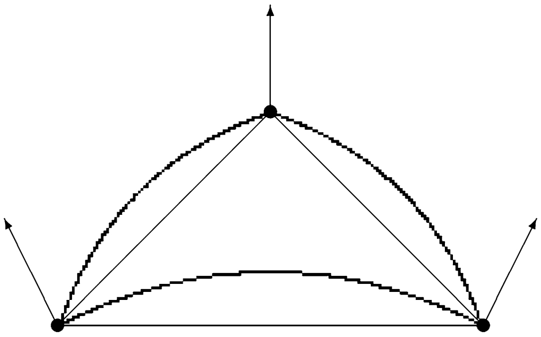


Fig. 7.12 Triangle and normals

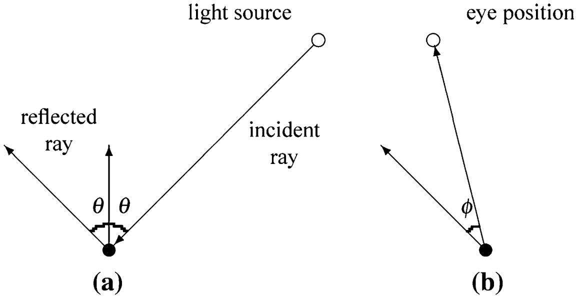


Fig. 7.13 Triangle and normals

depends on the material so for general graphics without material properties some sort of default has to be used. The angle between the vector from the surface point to the eye and the reflected ray (Fig. 7.13b) determines the amount of light arriving at the eye, and hence the final colour intensity of the point. The actual colour is an assigned property, defined by the CAD system either as a default or for some purpose. In I-DEAS, for example, object parts were coloured according to the operation which made them.

For a facet, the colour intensity of the corner points can be calculated and then any point within the triangle can be coloured according to its position relative to these corner points. This topic, though, is a graphics topic and so will not be pursued here. Interested readers should look at any good graphics book, such as Newman and Sproull [1], Foley et al. [2] or Szirmay-Kalos et al. [3], although this is by no means an exclusive list.

There are, therefore, several options for displaying a facetted object:

- smooth shaded
- flat shaded
- unshaded hidden line (providing that the facets belonging to different faces can be identified).

The graphics options in a CAD system should be examined and identified. These are "convenience" tools, not critical for object creation but useful at different times and different ways.

7.2.5 Exact Hidden-Line Images

Exact hidden line images are produced from the exact geometry of the model without using the facetting approximation step. Exact hidden line images produced static images, that is, from a fixed viewpoint.

Suppose you have the object shown in Fig. 7.14. Drawing just the edges and silhouette lines would give you the view on the left. What you would like is to remove the edges that would normally be hidden by the material to produce a line drawing or shaded image such as that on the right of the figure.

One way of doing this is to treat each face as a separate polygon, project these polygons onto the view plane and then intersect the resulting patchwork, removing those parts that are hidden. If the face is planar then it is easy to test if the face might be visible by checking the face normal with the view direction. If the scalar product of the face normal and the "into_display" vector is negative then the face is potentially visible. For curved faces it is a little more complicated.

Curved faces fall into three categories:

1. Wholly visible.
2. Wholly invisible.
3. Partially visible and partially hidden.

The silhouette lines, mentioned earlier, are used to divide faces into visible and hidden portions. Hidden portions are ignored, visible portions are projected onto the view plane.

When all the polygons have been projected they are intersected, as shown in Fig. 7.15. For each intersection, one polygon portion lies behind the other and that portion is removed, trimming the hidden polygon or removing it altogether. When all polygon pairs have been intersected the remaining polygons are drawn, optionally shaded. Another option is to keep the hidden polygon elements and draw them with dotted boundaries.

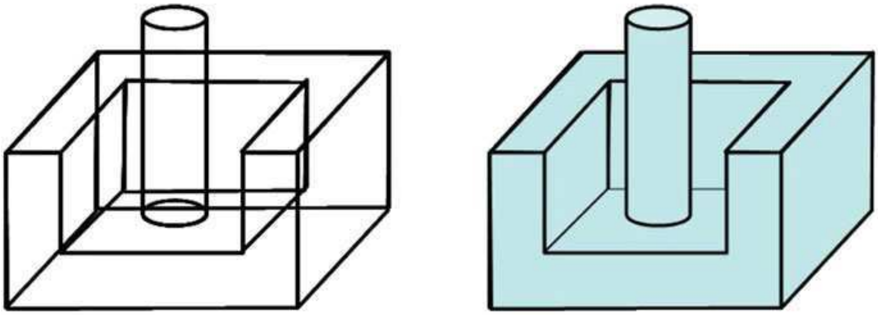


Fig. 7.14 Object to be drawn with hidden lines removed

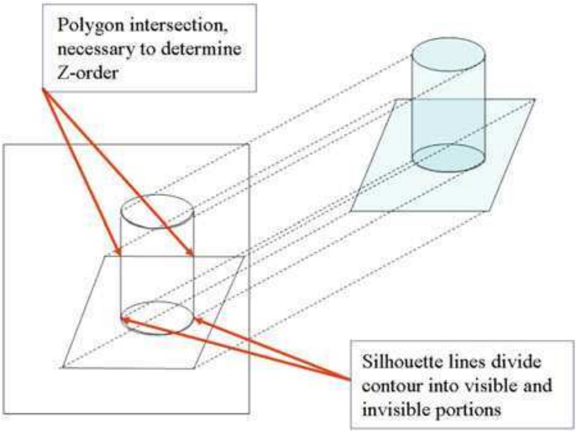

Polygon intersection, necessary to determine Z-order
Silhouette lines divide contour into visible and invisible portions

Fig. 7.15 Polygon projection

7.2.6 Patterned Objects

Patterning objects is related to shading, but imposes a defined pattern on the object rather than a single colour. This can be done by treating the pattern as an infinite sheet and then fitting it onto individual facets. The territory of the facet has, therefore, different basic colours. Instead of having a single basic colour for each facet the colour has to be determined from the pattern. The intensity, though, is calculated in the same way as for normal shading.

7.2.7 3D 'Tricks'

Note that there are a few "tricks" which have been employed to enhance the impression of three dimensionality. Hillyard in the Cambridge BUILD group developed an option to plot a pair of superimposed images of an object, one red and one green, which could be viewed using special glasses. Another trick is to make objects fainter the further away they are. This is termed depth cueing and I first saw this in a graphics machine developed by Evans and Sutherland in the early 1980s. Another line of work from the early 1980s was to develop rapid prototyping as a technique to produce physical models of parts. Use of glasses, polarising or with synchronised shutters is quite common. Virtual reality, with spectacles and an immersive environment is another area which has become popular in research. Work is still going on to develop real three-dimensional displays where an object can be viewed without special glasses. However, while this would be useful for understanding object shape this can be considered as a bonus and so will not be dealt with further here.

7.3 Engineering Drawings

Engineering drawings are static, two dimensional images of three dimensional objects. The use of these is part of the history of engineering and CAD. Engineering drawings were, of course, part of the production process as a means of communicating shape. They were produced by skilled designers using pen and paper. Some early commercial systems were electronic drawing boards for creating and copying drawings with no 3D model behind it. Since copying and correcting were important tasks these helped improve efficiency, but they were still prone to human errors. Nowadays better communication methods are possible, but engineering drawings are still not dead and communication is still far from perfect. Engineering drawings, therefore, are still part and parcel of CAD systems.

Engineering drawings are a formalised method of communicating shape from one person to another, say from a designer to a manufacturer. The formalisation includes elements such as tolerances to achieve functionality, surface finishes and fillets aimed at communicating functionality. This section is not about the formalisation but about the methods to create and manipulate the drawings.

7.3.1 Model-Derived Drawings

An example of an engineering drawing is shown in Fig. 7.16.

The different views are all produced in the same way, by drawing the object with hidden lines removed or dotted from different viewpoints. The central image is drawn into-display vector $(0, 0, -1)$ and up vector $(0, 1, 0)$. The top-middle view is drawn with into-display vector $(0, -1, 0)$ and up vector $(0, 0, 1)$. The left-hand view is drawn with into-display vector $(-1, 0, 0)$ and up vector $(0, 1, 0)$. The top right view is from an oblique view point with a non-aligned into-display vector. The view at the bottom is a sectioned view created by sectioning the original object and drawing the section with into-display vector $(0, 1, 0)$ and up vector $(0, 0, -1)$. The view origin should, in all cases, be the centre of the box surrounding the part. Each sub-image is, in fact, a rectangular graphics window, the size of which should just include the whole object.

The choice of views is under user control and varies according to the complexity of the object. This is not described here because, as mentioned before, this is connected with design presentation whereas this section is about the application of modelling techniques to CAD. The drawing can be of a single object or also an assembly, the basic techniques applied are the same. The views may be produced from a facetted model or as an exact hidden-line image of the solid model(s). An exact hidden-line model gives a better quality image and the static image requirement is fulfilled in engineering drawings. Producing such an image, though, is more complex than using facetted models.

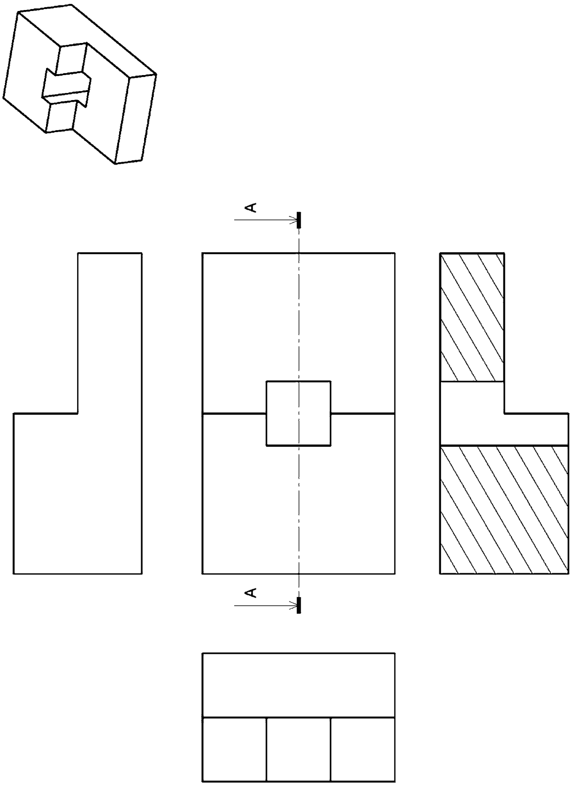


Fig. 7.16 Example of an undimensioned engineering drawing

Two options that should be noted are: dotted or invisible hidden edges and the number of segments used to produce curved edges. Invisible hidden edges give a realistic image but sometimes it is desirable to show them dotted for better understanding. The number of segments used for curved edges controls the image quality. If too low the curved edge will appear to be polygonal. This may or may not be available as a parameter, so check.

The section view is not a standard view and is produced by a designer to illustrate detail. The user defines a line or set of lines in two dimensions over an image window. These must cut the object completely, otherwise the section would only be partial, so the lines may be extended to cut the object bounding box. Figure 7.17 shows the section example from Fig. 7.16.

The dotted lines round the image mark the window boundaries, which are slightly larger than the object. The central dotted line, marked A–A, is the section line, extending to the window boundaries. This section line, together with the object bounding box on one side creates a cube figure, shown shaded in Fig. 7.17b. The extension of this shape upwards and downwards to the bounding box gives a solid block which is subtracted from a copy of the original object and drawn. New faces created by the Boolean operation are drawn hatched, as shown in Fig. 7.16.

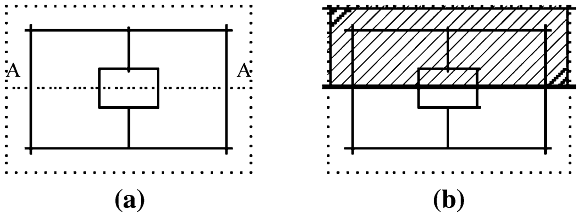


Fig. 7.17 Producing a section

7.3.2 Adding Dimensions and Symbols

The first stage produces a two-dimensional layout which is then used to add dimensions and other information. The dimensions added to the drawing have to be separated from the object shape so that the dimension lines are not confused with the lines drawn from the model. You do not want to be able to set dimensions between dimension lines, for example. One technique that was once used was to put graphical elements in an image in terms of "layers". It was certainly present in a system called CAMAX in the early 1980s. The notion is analogous to having a basic drawing and then adding details on transparent sheets to overlay that drawing.

7.3.2.1 Linear Dimensions

Some examples of dimension types are shown in Fig. 7.18. If two lines are parallel (Fig. 7.18a) then a simple dimension is added perpendicular to these (Fig. 7.18d) with the distance. If two lines are not parallel (Fig. 7.18b), then an angular dimension is added because there is no unique distance between the two, Fig. 7.18e. The dimension value is the angle. If the two lines do not overlap, as in Fig. 7.18c then an extension line is normally added, Fig. 7.18f. Note that all lines are treated as infinite and that the placement of any dimension may cause extension lines if the dimension is placed outside the range of an image line.

The engineering drawing in Fig. 7.16 with dimensions is shown in Fig. 7.19.

Tolerances can be added as extra elements in the dimension. They can be implemented simply as two extra number fields, the values being added by the user. A small experiment you can try is to set stupid tolerance values. This is not a serious error, but you can get an idea of whether or not the system is sensitive to the application or more-or-less a visual editor. Make a figure like that shown in Fig. 7.20, extrude it, say 20 units and create from it an engineering drawing.

Now play with adding tolerances to the dimensions marked A, B and C. Try adding a normal tolerance, such as ±0.02 to dimension A. Can you set tolerances of +50 (lower tolerance) and +40 (upper tolerance) to dimension B? The lower limit should always be less than the upper limit. Now try adding tolerances of ±50 to dimension C. This would mean that the whole dimension could have a value of −20, which is also ridiculous, the dimension plus the lower tolerance should be greater than zero.

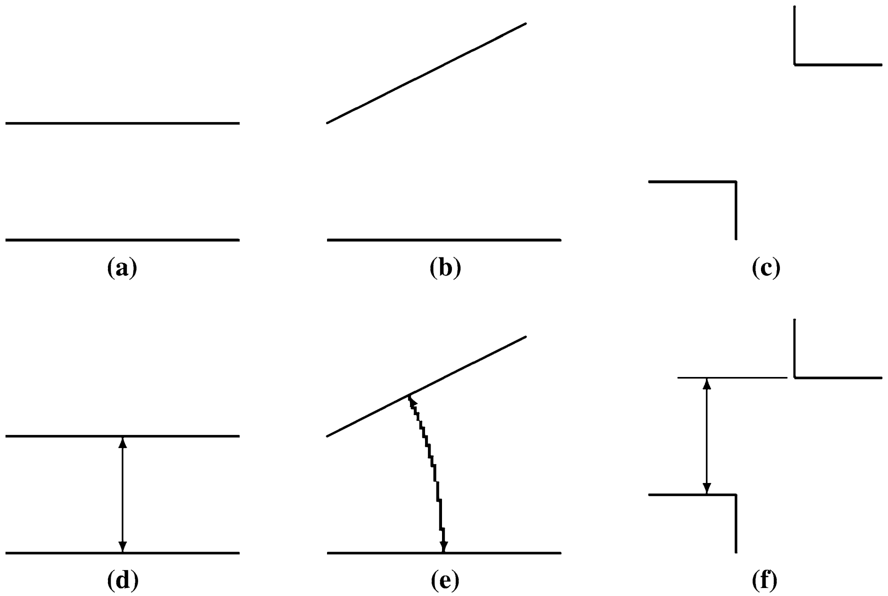


Fig. 7.18 Adding dimensions

It is not easy to have an automatic filter for strange or unrealistic values for dimensions and tolerances. However, if the system does not at least warn for some simple cases then it means that the onus is on you, the user, to get the values right. It also means that you should check that what is displayed is actually what you want, and not a typing error.

7.3.2.2 Circles, Circular Arcs and Threads

Circles and circular arcs are dimensioned with a radius or diameter, depending on the option. It is less clear how to dimension other entities, though. Figure 7.21 shows some examples of circles with dimensions. The two complete circles are dimensioned with diameter dimensions, the two quarter circles with radial dimensions. A CAD system can produce either for a circle because the geometry is well-defined. A radial dimension runs to, or in the direction of the centre of the circle, although it could be positioned outside the circle. The diameter dimension runs through the centre of the circle, but could also run between tangent vectors on opposite sides of the circle. These are relatively easy to produce and it is necessary to check each CAD system to see which it provides.

Note the two complete circles corresponding to the through holes. Both of these holes are threaded, one implicitly and one explicitly. There is an important difference in that the implicit thread is easily identifiable and produces a stylised notation whereas the explicit threaded shape produces a realistic image.

Fig. 7.19 Dimensioned simple block

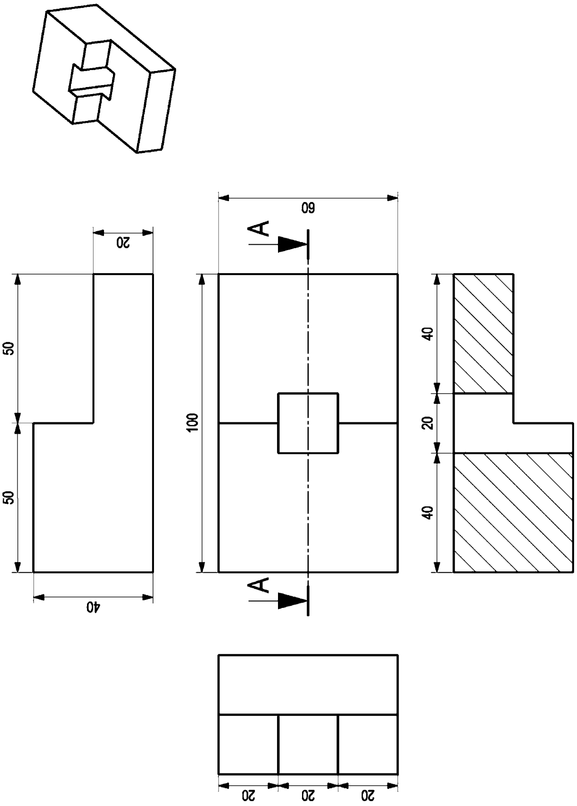
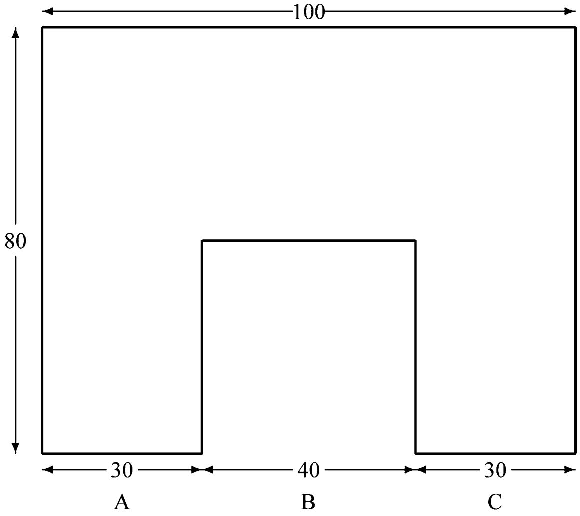


Fig. 7.20 Figure for adding tolerances to dimensions



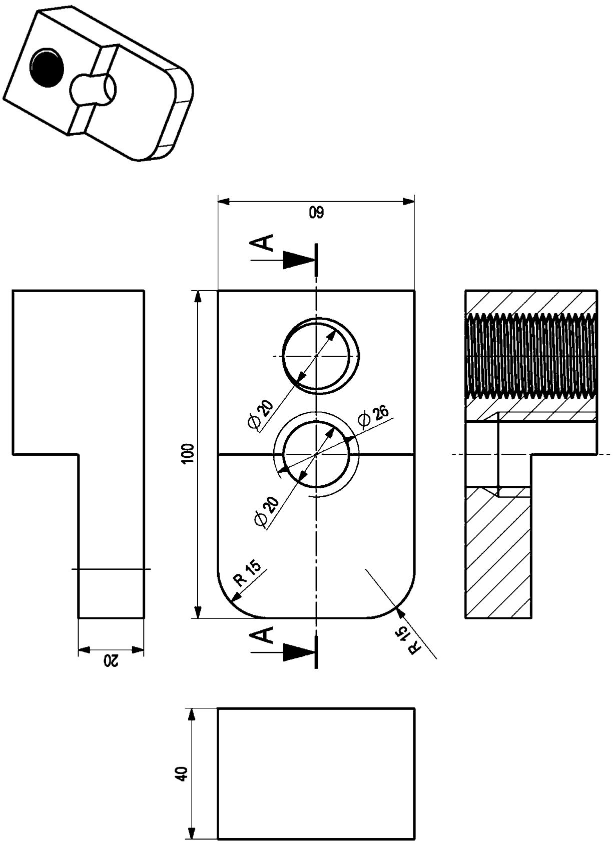


Fig. 7.21 Dimensions on circles and arcs

Although it is easy for you to identify the realistic image as a thread, it is very difficult to do this in a computer. This means that downstream applications would need complex software to recognise the form as a threaded hole or human intervention. This is a topic which will be covered in more detail in Chap. 10. Creating complex shapes with implicit geometry using information tags is covered in Chap. 8. Note, also, that the hole itself is strangely placed to have a thread, because of the cutaway. This does not deter the drawing creation, though, because the drawing rules are applied without considering the application. It is up to the user to make sure that the geometry makes sense.

Regarding the thread stylisation more closely, there are two indicators of the thread, a three-quarter circle in the top view, indicating the limit of the thread, and the vertical lines indicating the thread profile in the side view. Stylised threads are shown in more detail in Fig. 7.22.

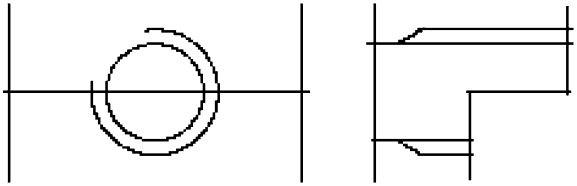


Fig. 7.22 Stylised thread drawing

7.3.2.3 Dimensioning Chamfers and Blends

Threads are an old example of the use of information in modelling, where a complex shape need not be modelled exactly because it can be produced directly. There is no need to have the exact geometry for manufacturing planning. Similar potentially useful examples of implicit shapes are implicit blends (or fillets) and implicit chamfers. These are usually produced explicitly, losing their identity. If they were implicit then this would allow automatic addition of chamfer symbols and blend radii in drawings. An alternative is to label the model elements resulting from the chamfer or blend with the origin so that these can be identified easily.

Examples of dimensioned chamfers are shown in Fig. 7.23. Of the dimensioned chamfers, only that noted "5 × 45°" was produced as a chamfer. That marked "10 × 45°" and that marked "28.22 × 45°" were both part of the original extruded shape. The drawing was produced using CATIA v5 and the chamfer data marked seems to be calculated from the 2D-image. It appears that if an edge is indicated as a chamfer then the angles with the neighbouring edges are calculated. These should add up to less than 180 degrees. The fact that the long edge, marked "28.22 × 45°", is probably too long to be considered as a chamfer is ignored. Note also that this edge and a neighbouring edge are both marked as being chamfers at a 45° angle, which does not make sense. However, it is too much to expect automatic error checking to help this problem, so the onus is on the user to use such labelling correctly.

Note that there is one blended edge marked with a radius in the figure. This, too, could be produced automatically if the model preserves the information that the shape was produced by blending. There is a nuance, though, that some blended objects have a default blend for the whole body, which would be marked as information on a drawing, and only certain different blends marked explicitly. This would require a "blend object" command, or some such, for all edges or as a note on the body. At any rate, it is better to use the chamfer command for chamfered edges and the blend command for blended edges and not introduce them as parts of a two-dimensional shape.

Note that the edge marked as "chamfer?" was actually produced by the chamfer command, but perhaps because a neighbouring edge is curved this cannot be noted as a chamfer automatically by the CAD system.

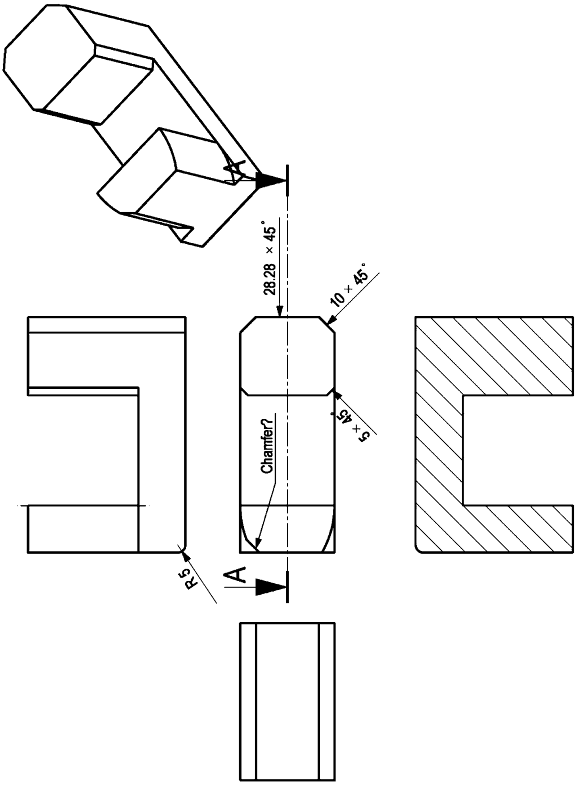


Fig. 7.23 Dimensioned chamfers and blend

7.3.2.4 Dimensioning Symbols

It seems likely from other developments that more information will be placed in models in order to make them usable directly in downstream applications. Examples are dimensions, tolerances and surface finish information. If this is so, then more of the drawing information could be added automatically. The symbols used are part of design practice, so are not discussed in detail here. In a CAD system they take the form of two-dimensional compound shapes which are edited into the drawing in the same way as for dimensions.

7.3.2.5 Automatic Dimensioning

Another possible automation is to dimension parts automatically. This seems to be based on the way that two dimensional shapes were dimensioned and the operations used to make the part. Such facilities need to be used with care because the dimensioning used to control a two-dimensional shape and the dimensioning shown in a drawing have different purposes. The dimensioning for a two dimensional shape is normally just to control shape and the shape is exact. Dimensioning in a drawing is used to indicate not just distances but critical distances. The choice of dimensions is to do with functionality requirements, not just the way that a shape has been made. Hence, care should be taken about accepting automatic dimensioning schemes based on the way a part has been made.

7.4 Interacting with Models

The method of referring to model parts in command files will be described in more detail in Sect. 12.2. This section is mainly about interactive graphics picking, but the other methods are mentioned here for completeness.

Identifying elements in models has a long history. The current method, picking elements interactively using a mouse is convenient but not sufficient. It is necessary to identify elements also as part of history trees. Before describing the methods, though, a little bit of history. An old, but quite good description of the methods is given by Várady et al. in [5].

In early Boundary Representation solid modelling research commands were given by typing or by creating command scripts. Each element used as input to a command had a number so that a chamfer command might be:

chamfer edge 15 by 4

All edges in the target body would then be scanned looking for one with the identity number 15. A pointer to that edge would then be passed to the chamfer operation procedure to be modified, as shown in Fig. 7.24.

Each element in the model received a number automatically when it was created and so had its own identity. While this made it possible to identify elements uniquely, the numbers were not stable. If an extra operation was applied before edge 15 was created then a different edge would be assigned the number 15. For this reason a young gentleman by the name of Chris Cary developed a topological navigation method to define edges, vertices using a relational sequence. The relations were:

LEFT RIGHT BACK FRONT TOP BOTTOM

In Fig. 7.24, the command to chamfer edge 15 would become:

chamfer top edge of front face of left face by 4

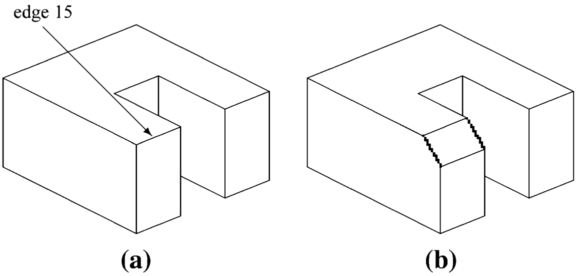


Fig. 7.24 Chamfering edge 15

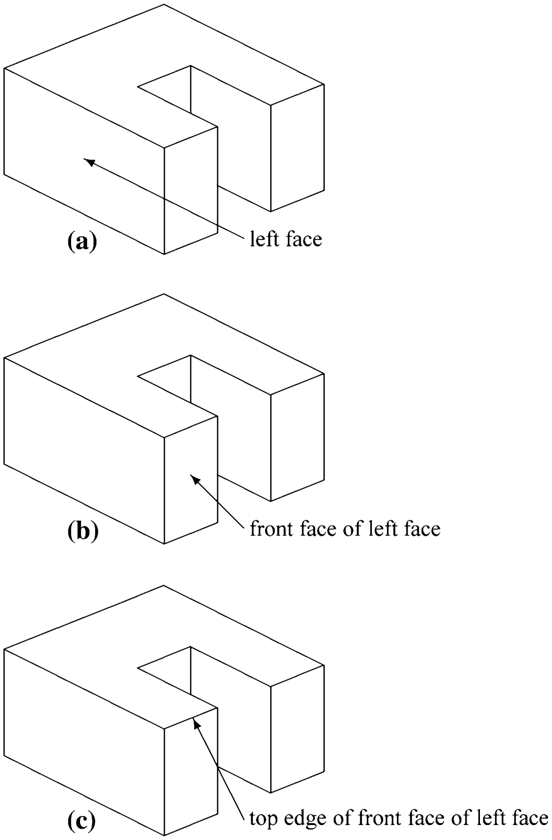
Figure 7.25 shows the sequence for identifying the edge. Note that the edge is not the "front edge of top face", because there are two edges which could be found. Similarly the edge is not the "top edge of front face" because there are two faces which could be identified as the front face. The edge could also be "right edge of front vertex of left edge of top face".

This method assumes a particular orientation of the object so as to be able to identify front, back, top, bottom, left and right. However, when the method was developed it was intended for solid modelling research with models in static positions, so this was not a problem.

For modern CAD, though, textual commands such as the ones above are not used. Current methods for identifying model elements use hit-testing, where the user moves the cursor over the image of the element to be selected and then issues a mouse click to activate identification.

If the graphical model is a true communications model then the graphics device may return some sort of key associated with a graphics element to identify elements "hit" by a ray test. This is how Carleberg [6] implemented an early communications model for the GPM Volume Module. Carleberg's method was based

Fig. 7.25 Chamfering top edge of front face of left face

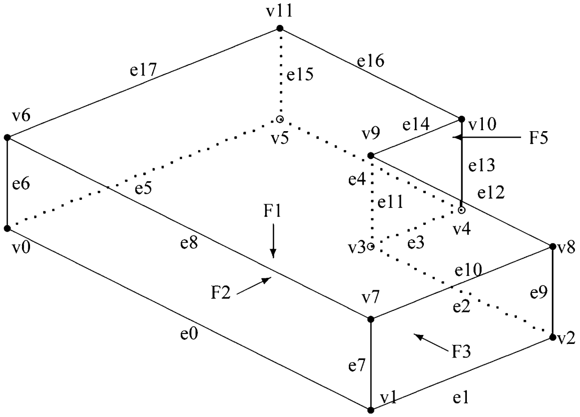


on an Evans and Sutherland vector drawing device. This device had a memory containing vectors which were drawn in a continuous loop. When a mouse click was detected any line drawn within a certain distance from the mouse position was flagged as a "hit" and the segment name returned. Each vector, or line segment, was named with a pointer to the model address, so that the model edges corresponding to the flagged vectors could be identified. A natural extension is that any triangular facet containing the mouse position would communicate the address of the face from which the facet was generated. This is a "graphical-space" hit-testing method because the identification is done by the graphics device.

An alternative is the "model-space" method. In this method, a mouse click identifies a screen position and the view orientation, defined by a transformation matrix, defines a view direction. The position and the direction together define a line which can be intersected back with the model to identify an element. Any vertex lying within a certain distance of the line is "hit". For an edge, the closest point between the line and the curve of the edge is checked. If the point is less than the certain distance and it lies on the curve portion defined by the edge then the edge is "hit". For a face, the line is intersected with the surface of the face. If the line is coincident with the surface, or intersects the surface at a point lying within the face then the face is "hit". A list of all vertices, edges and faces hit is returned to determine which is the desired parameter. Figures 7.26 and 7.27 illustrate these. The original object is shown in Fig. 7.26.

Consider the object oriented as in Fig. 7.27a. The user clicks on the screen near vertex v7, creating a ray. In the graphics space version the screen point coordinates are enough and the projected triangular facets colouring that screen point are "hit". In Fig. 7.27b the top and front facet triangles are shown. The point lies inside the triangular facet (v7, v8, v9), corresponding to face 1. None of the other triangles are hit. Although two facets of face F5 are visible, as shown in Fig. 7.27c, the screen point lies outside these. Finally, the ray cuts through one of the facets of the bottom face. However, the normal of these facets points away from the view direction and hence these may be ignored in some implementations. Note that if

Fig. 7.26 Hit-test object



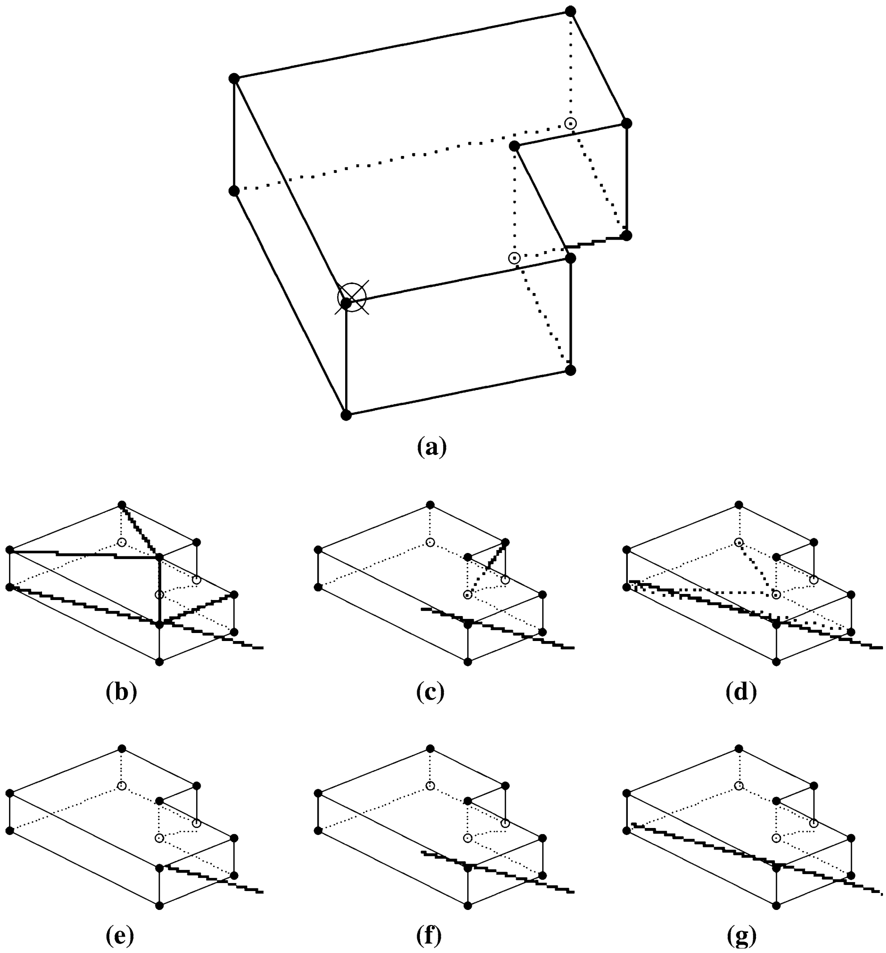


Fig. 7.27 Hit-testing

you have a wire frame image then the image is ambiguous and you may need to turn on shading to understand which faces are towards the eye point.

In order to find edges and vertices it is necessary to have a second and may be a third communications model for these. Line segments colouring screen points within the vicinity of the screen coordinates of the hit denote an edge hit. Vertex points projected into the vicinity of the screen hit point should also be noted as hits.

The model space version works in a similar way to a Boolean operation. The hit point and view direction are used to create a straight line. This line is intersected with the surface of every face in the body. If this results in a point or set of points then these are tested to see if they lie in the face, or, using a tolerance, within a small distance of the face boundary. If they do, then the face is hit. In this way, face 1 is a direct hit while faces 2 and 3 in Fig. 7.26 would be hit because their intersection points lie within tolerance of the boundary. The surface of face 5 gives an intersection point which lies outside the face. There is an intersection point with face 0, not shown marked in Fig. 7.26 which may be considered to be “invisible”. The curve is also checked against every edge. If the closest distance between the line and the edge is less than the tolerance then the edge is hit. Similarly, if the distance between a vertex and the line is less than tolerance then the vertex is hit.

Which of these two methods is used should be transparent to the user.

What happens, though, when you record an interactive sequence or want to store the selection as part of a history tree for later replay?

In this case it is not so obvious how to proceed. One method which is used is to store the screen point and model orientation and redo the hit test. Here the model-space method is easier to use because there is no need to regenerate an image or store the graphics model. Another method is to store an identifier of the element selected and to re-identify it from the model. The identifier needs, therefore, to be relatively stable, unlike the simple numbers mentioned earlier. The identification technique is known, currently, as “persistent naming”.

With persistent naming, model elements receive an initial identifier. As modelling proceeds, new model elements derived from the basic elements receive a subsidiary name defining their origin. This has already been mentioned, in Sect. 4.19 and is illustrated in Fig. 7.28.

In the original model, shown in Fig. 7.28a an edge might get the identity “e1”. If this edge is now split, as in Fig. 7.28b the partial edges might be renamed “e1.1” and “e1.2”, denoting that they both derive from the original edge, “e1”. If one of them is split again, as in Fig. 7.28c, then you have three derivative edges: “e1.1”, “e1.2.1” and “e1.2.2”. Completing the operation, to cut out a slot, as shown in Fig. 7.28d, “e1.2.1” is now deleted. If edge “e1” had been filleted, say, after the original object was made, and then the slot operation inserted prior to this, by editing the history tree, the model has lost edge “e1”. However, with persistent naming edges “e1.1” and “e1.2.2” would be identified as derivatives and so the filleting operation would be applied to them.

This is, of course, a very simplified version of what happens. However, the aim here is not to write a treatise on persistent naming but to explain briefly the purpose and how it works.

7.5 Chapter Summary

This chapter covers the graphics output and input aspects of a CAD system. Graphical rendering methods were described as well as hidden line methods. Hidden line methods fall into two categories: static and dynamic. The dynamic method is commonly done using facetted graphical models. The static methods are useful for engineering drawings when a fixed viewpoint is used.

Engineering drawings are really a communications medium, and the formal use is part of engineering practice and not the subject of this chapter. This chapter

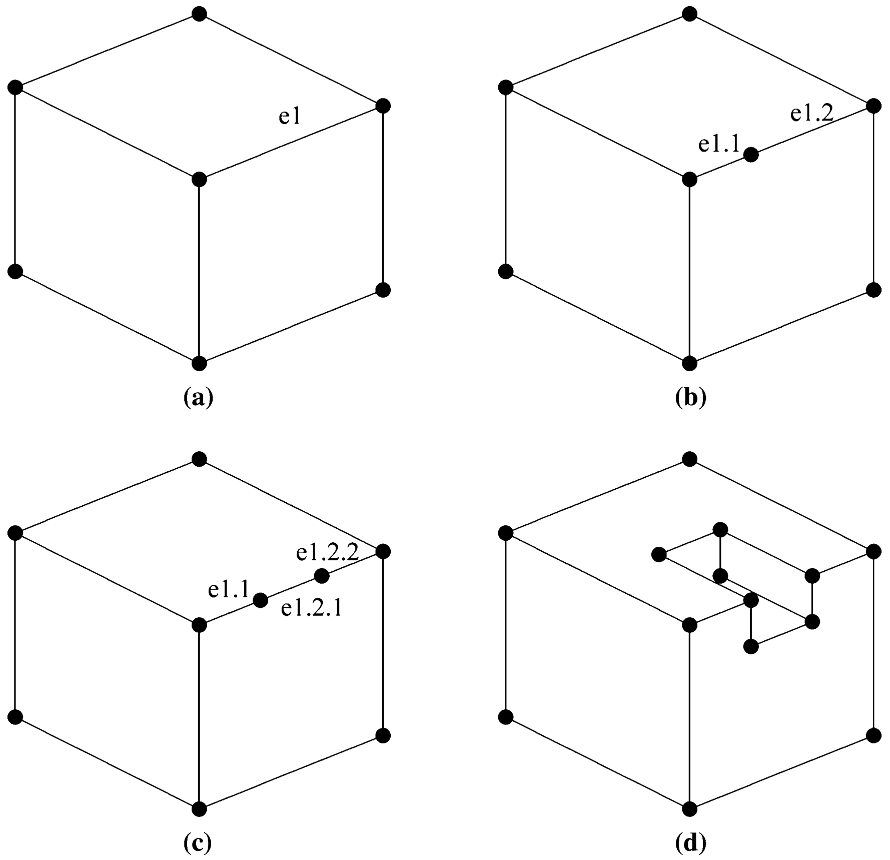


Fig. 7.28 Persistent naming example

deals with how they are made and some of the graphic editing tools used to manipulate them.

Finally, this chapter deals with interaction between the user and the solid model. This can be done directly by pointing at elements, or indirectly using naming techniques.

7.6 Drawing Exercises

7.6.1 Matrix Identification

What does the following matrix do?

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

What does this matrix do?

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0.01 & 1 \end{bmatrix}$$

In order to understand this you should remember the description of homogeneous coordinates, from Chap. 5. A set of homogeneous coordinates is:

$$(x, y, z, w)$$

This corresponds to the set of coordinates:

$$(x/w, y/w, z/w, 1)$$

References

1. Newman, W.M., Sproull, R.F.: Principles of Interactive Computer Graphics. McGraw-Hill, New York (1979)
2. Foley, J.D., van Damm, A., Feiner, S.K., Hughes, J.F.: Computer Graphics Principles and Practice. Addison-Wesley, Reading (1984)
3. Szirmay-Kalos, L., Màrton, G., Dobos, B., Horvàth, T., Risztics, P., Kovàcs, E.: Theory of Three Dimensional Computer Graphics. Akadèmia Publishers, Budapest (1994)
4. Stroud, I.A.: Boundary Representation Modelling Techniques. Springer, Heidelberg (2006)
5. Várady, T., Gàal, B., Jared, G. E. M.: Identifying features in solid modelling. Comput. Ind. **14**(1–3), 43–50 (1989)
6. Carleberg, P.: STU report. KTH, Industriell Produktion (1985)

Chapter 8
Information and Properties

At the end of the 1970s major effort was devoted to being able to create solid shape models. However, the shape is only part of the product and so information needs to be added to complete what is termed a "Product model". Examples of such information are: surface finish, material, tolerances. Although the BUILD system, the original B-rep modelling research system, had NOTES, the main thrust in this direction was, as far as I know, started in Sweden by Kjellberg et al. in the GPM project. In the GPM volume module different categories of information were identified: INFORMATION, MODIFICATIONS and CONSTRAINTS, as well as other extra design supports like HELP GEOMETRY. However, information handling is not easy. There is no convenient information mathematics as there is for shape so handling it tends to be a little ad hoc. A basic information set, though, should be treatable in a more-or-less consistent manner.

This chapter explains how the information is implemented, information types as well as strategies for using information. Some of the information types mentioned below are speculative in that they do not seem to be part of any CAD system. However this is done to illustrate information types and also because the use of information in CAD is expected to increase in the future, maybe to include some of these types.

8.1 Methods

The first question concerns how to add the information. It could be added as a separate "layer", after the shape has been made, there was at least one system which did this. The preferred method now, though, is to add it directly to the model.

The advantage of adding the information as a separate layer is that the information is kept separate and there is less risk of corrupting it. A disadvantage, though, is that the information is not associated with the model and hence may lose relevance.

I. Stroud and H. Nagy, *Solid Modelling and CAD Systems*,
DOI: 10.1007/978-0-85729-259-9_8, © Springer-Verlag London Limited 2011

Associating the information with the model puts it in the relevant place and lets the user add it at any time but risks corruption as modelling goes on. Nevertheless, this seems to be better than keeping the information separate. This section describes the techniques used to add information.

Theoretically information could be added to any model element, but the main elements for information are:

- Assembly
- Instance
- Single object
- Face
- Edge
- Vertex

Secondary items are:

- Surface
- Curve
- Point

It is a good idea to allow information to be attached anywhere in case somebody thinks of a new use, but these are the elements that you see and can easily identify.

The simple method is to declare all information types as members of a general superclass, "INFORMATION", and to have an extra pointer field in all entities of this type. The individual information types are then declared as examples of this superclass but with their own characteristics. Typing information is important because methods for handling it currently work on a case-by-case basis.

8.2 Identifiers and Names

These are special information elements for identifying elements in the model. Identifiers may be simple numbers assigned to elements as they are created and are used for debugging, etc. Names are for identifying elements of the model for operations. This is the usage of the two terms here.

One disadvantage of identifiers is that they are dependent on the object creation method, as has already been mentioned in Sect. 7.4. Suppose you start with a block, as in Fig. 8.1a. Now subtract a rectangular shape to create a cutout (Fig. 8.1b), one edge of which is identified as edge 12. Edge 12 is now chamfered to create the object in Fig. 8.1c. Now suppose the original base shape is modified to add a circular hole in the original shape. When extruded to create the basic shape in Fig. 8.1d, the hole edge is now edge 4, what was edge 4 in Fig. 8.1a becomes edge 5, edge 5 becomes edge 6, and so on, until what was edge 11 becomes edge 12 and there is a new edge 13 as the extruded hole edge in the top face. When the cutout is added in Fig. 8.1e, what was edge 12 in Fig. 8.1b

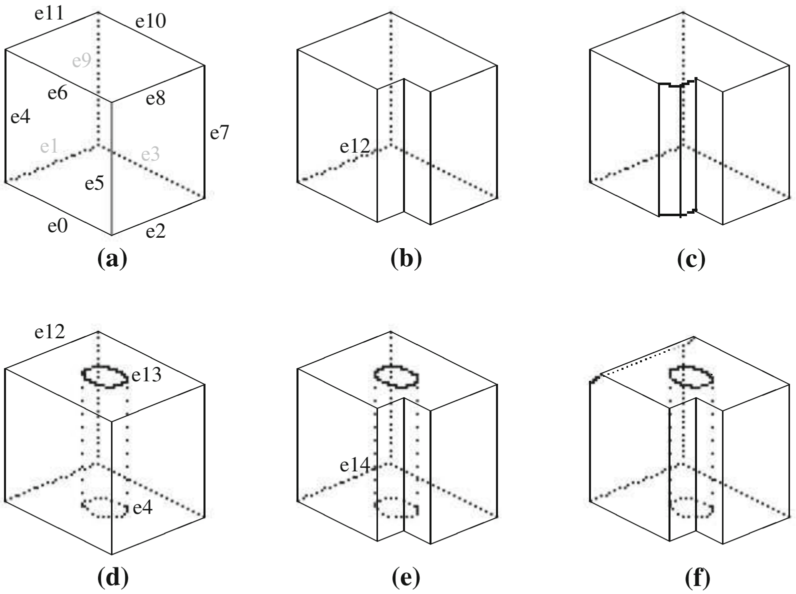


Fig. 8.1 Changing edge identifiers. **a** Simple block shape. **b** Cutout added. **c** Edge e12 chamfered. **d** Block with hole. **e** Cutout added. **f** Edge e12 chamfered

becomes edge 14 in Fig. 8.1e. If you now chamfer edge 12 then the back edge of the top face is chamfer, Fig. 8.1f.

What this is supposed to show is that identity numbers are volatile and so cannot be relied on. If you write the data structure to disc and read it back then the edge ordering may change if the edges are accessed via faces, say, as in the ACIS kernel, and not in the order that they were created. The entity identification numbers are useful for debugging but are not stable enough to be used as part of a history tree.

The other type of entity access mechanism is via names. Two basic types of name can be distinguished: user-defined names and system defined names.

User defined names are most useful after the model has been created. If named entities are modified then it is uncertain how the changed model is named. This is shown in Fig. 8.2. The top face is named “tface”, as shown in Fig. 8.2a. If this face is now split with a new edge, as shown in Fig. 8.2b, there are four variants as to how the changed model is named. The first variant, not shown, is that the name disappears altogether. The other three variants are shown in Fig. 8.2c–e.

In an early modelling system, the GPM Volume Module, names were not allowed for entities involved in variations because of this. User assigned names are for allowing high-level reference to models in applications.

The other use of names, mentioned above, is to have names generated and supported by the system. This is now common and is called “persistent naming”.

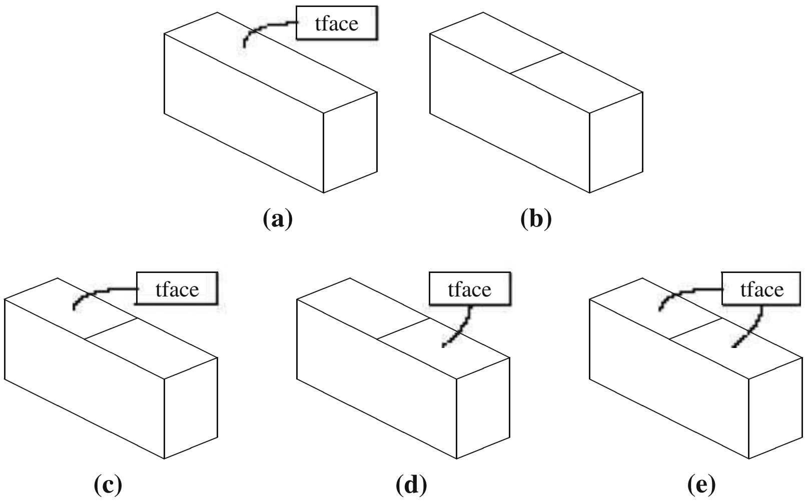


Fig. 8.2 Modifying a named face. **a** Named face. **b** Face split. **c** Variant A. **d** Variant B. **e** Variant C

The trick for using this is to assign names and modify the names so the names evolve to reflect how the model was created. This gives an element of traceability which can be used to provide more stability when referring to elements in modelling command parameter lists. This topic will be dealt with in more detail in Chap. 12.

8.3 Engineering Information

Generally speaking, the extra engineering information can be divided into four classes. This is a subset of the categories identified in [1] which are relevant for CAD. The others are mentioned elsewhere.

- Pure information—e.g. colour information which is to be interpreted by a human.
- Shape modifiers—e.g. screw threads or implicit blends which may be complex to model and which are better noted as information.
- Shape constraints—e.g. surface finish.
- Feature information—e.g. holes, slots, etc.

Which information elements are present in the CAD system is an implementation matter, there is no standard, easily identifiable set. As with the choice of operations, CAD system developers may have come to have a common set, but individual sector and application needs will throw up new examples. This is not an easy problem to solve. The CAD code needs to be stable, it cannot stop working if some new information element is added by a user. There is no convenient information mathematics to allow standard algorithms to be developed to handle information classes. For me, the best solution I have seen is that in the ACIS modelling kernel. Maybe there is something similar in Parasolid, but I have not looked at that in detail. In ACIS, an implementor can define his or her own information "attributes" with certain default routines which can be called to handle the attributes in specific ways. The implementor should develop special routines for when entities are joined, split, deleted and so on. Whether or not CAD implementations based on ACIS make use of this to create a wider set of information in a model is another question, but at least the mechanism exists.

8.3.1 Pure Information

Pure information is used for appending information with no geometric influence to elements of the model. This can be carried over to drawings or, if these are replaced, to electronic communication.

Examples of pure information are:

- Colour—For rendering of images or perhaps painting in downstream applications. I-DEAS used colour to indicate model elements created by different operations. This can be implemented by attaching a colour note to elements which have been added and then rendering the items in that colour when drawing the object. Colouring objects is necessary to produce shaded images. This information could be recorded centrally or added as notes, which is a more flexible solution.
- Price—For calculation of product component costs. This would allow a bill-of-materials to added costs for components.
- Residual value—For calculation of end-of-life value. This can also be relevant for planning layouts so that valuable, reusable components are protected.
- Material—For rendering or for downstream applications such as process planning and machining. Material notes on objects are commonplace and you should be able to find them in your CAD system.

8.3.2 Shape Modifiers

A shape modifier is a note attached to an element to indicate a complex or routine shape that can be made easily. A classic example has been the screw thread. This is a geometrically complex shape which is awkward to model but relatively easy to make. It is more efficient to record it with a note attached to faces or features. This makes it easy to find in the model, rather than having to try to identify certain surfaces as screw thread surfaces. See the example in Sect. 7.3.2.2, illustrated in Figs. 7.21 and 7.22. It also means that the information about the thread can be recorded directly rather than trying to retrieve it from geometric characteristics. The screw thread information can be seen in drawings using formal drawing techniques and Autocad can render the model to make the thread appear as though it has been made.

Examples of shape modifier information are:

- Screw thread—The classic example, noted above. The screw thread may be on a hole or on an extrusion, such as the shaft of a bolt.
- Implicit blend – This was implemented in BUILD by Ian Braid and later implemented in Romulus, a geometric kernel developed by Shape Data. In the Romulus implementation there was a "thumb-weight" parameter because, as was explained, a mould maker would blend edges by running a thumb around them. Nowadays most systems seem to have explicit blending, whereby new topology and geometry is introduced. It is useful, however, to be able to identify blended elements as being due to blending so information notes may be present to note these. Note, though, that normally blending is done by recording the elements to be blended in a history tree and changing the command to add or subtract elements if needed. The elements do not seem to be marked specially.
- Bolt cutting—I do not know that this has ever been implemented, but this is a valid example. Someone once told me of an assembly in which there were a number of standard bolts, but one needed to be shorter than the others, for assembly reasons. In practice this was done by cutting the standard bolt. If the standard bolts are imported and one is changed then it has to be made into a new model. A bill of materials would list it as a separate object. Having a shape-changing note, which would be attached to an instance of the bolt, would mean that all bolts would remain standard and make it easy to identify the extra operation for assembly instructions. This is a particular case, but modifying any standard part, by drilling an extra hole or cutting off some part, for example, could be handled in this way.
- Gear wheels—Gear wheels are examples of mechanical parts which are often standard. Although the geometry of a gear wheel may appear to be simpler than screw threads, they can, in fact, be quite complex. If these can be imported as finished shapes from a standard catalogue then this seems the ideal solution. If not, however, it is reasonable to approximate them with cylinders, cones, or other simple geometry, with information attached.

8.3.3 Shape Constraints

A shape constraint is something which imposes conditions on parts of a model. It might be imposed on one element or on two.

Examples of shape constraint information are:

- Surface finish—This allows the designer to identify functionally important elements for manufacturing. It is currently more common to add this information when producing engineering drawings, but the increasing use of 3D models as a communications medium requires this information to be added to a model.
- 3D dimensions and tolerances—Dimensioning and tolerancing is usually done in engineering drawings. In models dimensioning is done in two dimensions, for sketches for extrusion, and the three dimensional dimensions are given as parameters for operations. Dimensions and tolerances are critical for downstream applications and so should be incorporated as part of a model. How this can be done is not obvious, but 3D dimensions and tolerances are also candidates for new shape constraint tags.

8.3.4 Feature Information

Features are another growth area in CAD and will be dealt with further in Chap. 10. Features are isolated shape sub-elements in a model. Features can be created explicitly by using feature-making operations or by sewing feature models into a model, or implicitly as side effects of operations. Marking these, or recording them in some way, means that they are available as input to applications such as manufacturing. This is a complex subject, though, which is why there's a separate chapter on it. Here, only the information elements are introduced.

A few examples of feature information are:

- Pocket—Position of pocket, profile of pocket, depth, bottom conditions, side-wall conditions, for example.
- Hole—Position of hole, orientation of hole, radius of hole, thread information
- Slot—Position of slot, path of slot, depth of slot, width of slot.
- Profile—position of profile, path of profile, depth of profile.
- Chamfer—The face or faces resulting from a chamfer operation.

There are a lot of variants on these, and there are many more possibilities. Arguably, a planar face with a surface finish condition could be considered to be a feature, in manufacturing terms, but more later. There are many application areas which need different feature definitions and different information sets. The feature information above is what might be expected to come from CAD operations, but it should be possible to add feature information for other application areas as well. In Chap. 10 examples of different features will be given with the information associated with them. Note that feature recognition in Chap. 10 is a sort of dynamic calculation of features.

8.4 Functional Elements

There is also a class of element which can be represented more practically with information than by shape. Examples are:

- Screw threads (again)
- Gear wheels (again)
- Springs
- Cams and followers

These are generally used for kinematic simulations and so will be dealt with in Sect. 13.5.

8.5 Mechatronics

Mechatronics, in a broad sense, involves the use of electronic components in mechanical systems. This has become increasingly common because the electronics components can perform work previously done by mechanical linkages, provide more flexibility and greater reliability. It is easy to find examples, most modern cars include electronics components, so do washing machines, for example. Any product which includes both mechanical parts and electronics components should be able to represent the functionality of the electronics components in a single unified model with the traditional CAD shapes.

So, the use of mechatronics also entails extension of CAD. The shape of electronics components, the casings in which they are housed, is important for the overall design. However, the functional characteristics of the component and the connections are also relevant. With these, the geometry is not necessarily the main point of interest. The designer probably does not want to design the actual electronics but may well want to have a representation of what the component does. For a full simulation this is also necessary. Additionally, as with other components, the electronics components can have a residual value which makes them interesting at the end-of-life of a product. As well as protecting them it may also be necessary to consider access and disassembly options around them in a complete CAD model.

8.6 Modelling and Information

In general, CAD systems handle shape well but are less consistent with the use of information. A demonstration of this is given in one of the exercises, in Sect. 8.10.1. An example of how information might have to be handled by

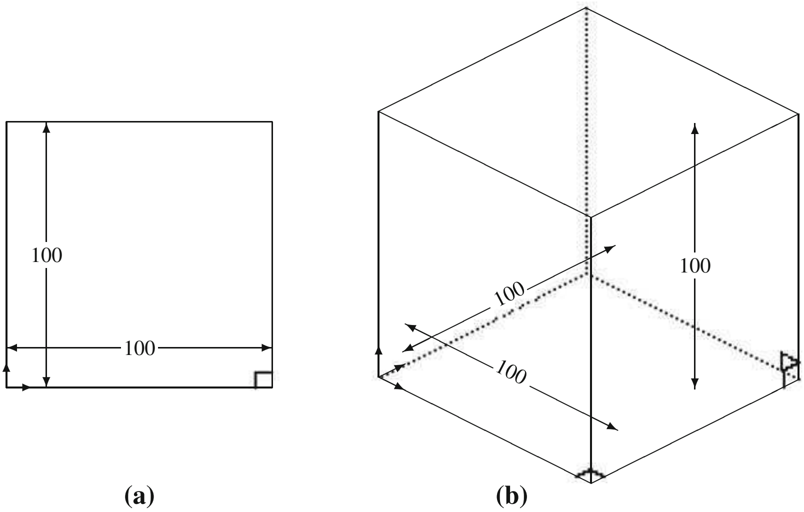


Fig. 8.3 Dimension transitions for extrusion

modelling operations concerns dimensions, if you want to have 3D dimensions in a model

Nobody, as far as I know, handles dimensions in the way that is described here. I am using it, therefore, because it shouldn't reveal a mechanism in an existing system. Most systems I know avoid having 3D dimensions because they are difficult to handle and use operation parameters instead as a way of changing shapes. This is described later, in Chap. 12.

Take the example shown in Fig. 8.3. On the left of the figure is a square, with dimensions between the opposite sides, a constraint of perpendicularity between one pair of neighbouring edges and the square has one corner at the origin. If the shape is extruded, and the system developer wants to handle the dimensions, then it is necessary to have a "data migration" to reattach and reconfigure the information entities attached to the square. When the 2D shape is extruded, the edges generate faces while the vertices generate edges. This means that the dimensions between the edges should be transferred to the faces, together with the implicit constraint of parallelism. The constraint of perpendicularity stays the same, while new perpendicularity constraints appear between the base edges and the new side edges because the extrusion is normal to the 2D shape definition plane.

If you think about the above description you should get an idea of the complications involved in handling information. You have to examine the information and the operation to know how to modify it and this can be very time consuming. As new information entities are added it is necessary to work out new methods and update algorithms. Information is difficult to generalise. It is a key element in new developments but is still a research topic.

One of the first methods for handling information comes from the BUILD system. Implicit blend information was attached to edges and, when those edges were split, the edge blends were shared between the original and the new edge so that they both had an implicit blend. This may sound simple but it established a basic principle about splitting elements with shape modifiers.

Other suggestions for being more rigorous about information handling are contained in [1]. These suggestions are based on division of information into different categories, already mentioned, and classification of operations. One rule of thumb, though, is that there should not be two or more information elements of the same type attached to an entity in the system.

8.7 Using Information in CAD

Up until now this chapter has dealt with the how and what of modelling information. In this section some practical considerations are given.

As well as the earlier classification, product information can be classified as belonging to one of two classes:

1. System defined information—Information defined by the CAD system during model building.
2. User defined information—Information defined by the user to annotate a model.

An example of the first type of information might be feature information. If the system builds up a description of a model dynamically as part of the model, then this has to be handled by the system software. Permanent names are another example of this category.

For user defined information, such as material properties, these should be added as late as possible, preferably after all shape modelling has been completed. There is also information which might be associated with some operations, such as threaded holes. The same strategy applies, that is, to add these features as late as possible, also to avoid interactions with other elements in the model.

Generally, you will be able to find a way of putting stupid or conflicting information into models. Some exercises to do this are given at the end of this chapter. What is more awkward is when you do this but don't mean it. As stated in the previous section, you shouldn't really have two pieces of information of the same type attached to any entity. Even if the information is the same when the information entities are attached there is still the possibility of changing one piece of information later.

However conflicts can arise unexpectedly. Figure 8.4 shows some examples of information conflicts with threads, shown in stylised drawing form.

In Fig. 8.4a there is a simple mismatch between the thread sizes in objects at a matching hole and insertion. You should spot this type of error fairly easily, but it illustrates the point that matching elements are often designed separately and the information added separately, possibly by different people at different times.

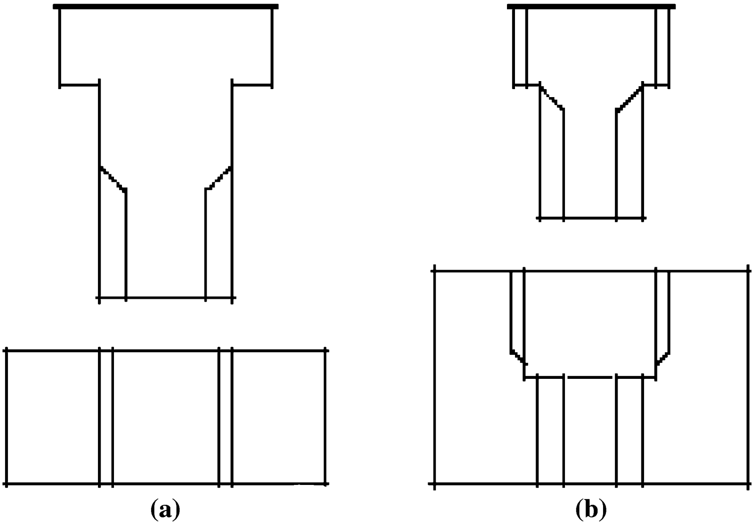


Fig. 8.4 Simple information conflicts

This is an example of features which span objects, to be dealt with in Chaps. 10 and 11. You should really be able to define the information once and have it established automatically in both parts.

The second example, in Fig. 8.4b, is more subtle. The threads match but the objects make no sense. Neither object is necessarily wrong in itself, but because the objects have two different thread sizes, and both are rigid objects, then there is a conflict. This is sort-of analogous to a constraint loop in an assembly, to be described in Chap. 13.

It is not reasonable to expect a CAD system to detect all such errors automatically. You should be aware that there are potential problems with information and that checking is difficult because the semantic content is hard to interpret automatically. If you use information in any form, then you should check yourself that it makes sense. Note again, though, that it is easier to detect matches with implicit threads than if the geometric form is evaluated. It is harder to perform a geometric mismatch check.

8.8 Volume and Area Calculation

Mass properties are information elements which are not recorded explicitly but calculated dynamically. However, these are important for engineering and so the calculation methods are described briefly here.

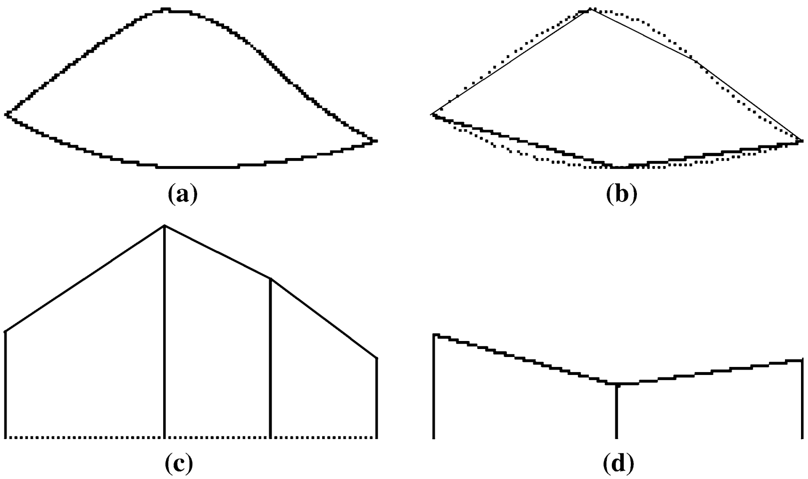


Fig. 8.5 Calculating volumes. **a** Original shape. **b** Approximated shape. **c** Positive areas. **d** Negative areas

There is a simple method for calculating the volume and surface area of a product based on triangular facets. As far as I know this method is due to Ian Braid or his colleagues at Cambridge University, UK.

The method subdivides each face in a model into triangular facets, in the same way as is done for producing shaded images. Each facet is then taken as the top of a triangular column, from some base plane below the object whose volume is being calculated. If the facet normal is directed away from the base plane then the column volume is added to the sum. If the facet normal is directed toward the plane then the column volume is subtracted from the sum. This is illustrated for two dimensions in Fig. 8.5.

Figure 8.5a shows the original shape. This is approximated with straight lines in Fig. 8.5b, although these would be planar facets in three dimensions. The top three lines lead to positive areas, Fig. 8.5c, while the two bottom lines give rise to negative areas. The area of the complete figure is the sum of the positive and negative areas. There is an exercise to calculate the area of this shape, with given coordinates.

For a volume, surface area is calculated by summing the area of all the triangular facets which approximate an object. The equivalent in the two dimensional Fig. 8.5 would be to sum the length of the lines approximating the shape to estimate the boundary length.

The method for volume calculation can be applied to partial objects, although the result may not be entirely accurate. For this reason, CAD systems may not allow you to calculate the volume of sheet models, only of closed volumetric models.

8.9 Chapter Summary

This chapter describes the information elements which add an extra dimension to the CAD system. The chapter describes different types of static information element found in a CAD system. The chapter also describes dynamic information, in terms of volume and surface area, calculated from the model when required.

8.10 Information Exercises

8.10.1 Conflicting Material Properties

Create a cube, 100 × 100 × 100, and a cylinder, radius 25 and height 500, so that the cylinder and cube overlap. Label the cube as being of some sort of wood and the cylinder as being of steel. Add the two objects and see what the system says.

These have to be separate objects so it is not necessarily obvious how to create them. Some systems allow you to create different objects easily, but there is a tendency, unfortunate in my view, to constrain the user to have one object per file. This means that it may be necessary to create the objects as parts of an assembly and then add them. CATIA v5, though, which has the one-object-per-file philosophy, allows you to insert a new object into an object being made and then apply a Boolean operation.

In my view, the wrong result is just to add the objects and say nothing. If this happens then check to see which material information tag remains or whether both have been kept. If both are there then the error is obvious. If just one remains it is likely to be the material tag associated with the first object.

A more correct approach is to warn the user that there is a conflict. The two objects have non-accumulative information tags of the same type but with different information. This should be enough to provoke a warning to alert the user of the conflict. The system could then either ask the user which one to choose or adopt a default strategy (take the material tag from the first object) and let the user sort it out manually if necessary.

A variant on this is to subtract one object from the other. In this case there should be no problem, even though body parts from one object appear in the other.

8.10.2 Modifying Threaded Holes

Make a cube 100 × 100 × 50, that is, 100 in X, 100 in Y and 50 in Z. In the top face, which is 100 × 100 make a threaded hole, radius 20.

The following sub-exercises are intended to show how the CAD system reacts to modifications of this threaded-hole element. If the system performs the operation without comment then this is an example of information corruption, when the information element is in conflict with the geometric information. If this happens, then it is even more important to add information elements as late as possible, when the main shape changes have been finished.

8.10.2.1 Drafting the Threaded Hole

Apply a draft angle to the threaded hole. Use, say, 5 degrees as a draft angle to show a change. A drafted threaded hole makes little engineering sense, since adding a draft angle is for mould making, and threaded holes are not generally made by moulding.

8.10.2.2 Scaling the Threaded Hole

If your CAD system permits, apply an uneven scaling to the cube, scaling it by 2 in the X-direction. In one system I have used it is possible to create an elliptic threaded hole. In the drafting package it is possible to see that the hole is threaded, even after the change in scale. An elliptic threaded hole is also something which is difficult to produce.

8.10.2.3 Corrupting the Threaded Hole

Create a bar down the middle of the hole. This destroys the reason for having a thread down the hole.

8.10.2.4 Splitting the Threaded Hole

Cut away half of the cube, through the centre of the hole. This, too, destroys the reason for having the threaded hole.

8.10.3 Area Calculation 1

Calculate the area of the shape in Fig. 8.6, given the coordinates shown in the figure.

8.10.4 Area Calculation 2

Now try calculating the volume of a quarter-circle, radius 50, at different degrees of approximation.

Fig. 8.6 Calculating volumes

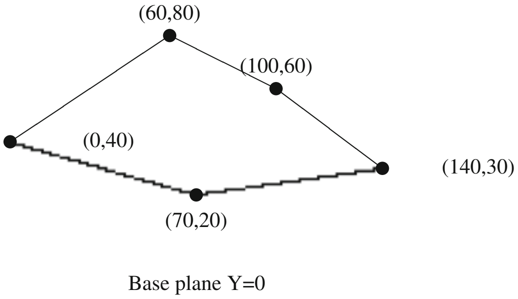
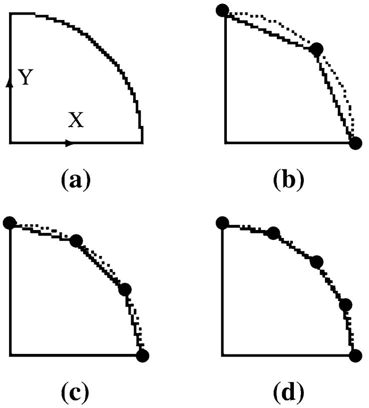


Fig. 8.7 Calculating area of a quarter-circle



The area of the quarter circle is: $\frac{\pi 50^2}{4} \cong 1963.4954$. Calculate the area approximations in Fig. 8.7b–d.

Reference

1. Stroud, I.A.: Boundary Representation Modelling Techniques. Springer, Heidelberg (2006)

Chapter 9
Databases and Data Exchange

This chapter describes some of the main aspects of communication between modelling systems. First of all, local disc formats are described, since this is a basis for other data exchange. Data exchange can be divided into roughly two categories: standardised data exchange; and non-standardised data exchange. Standardised data exchange is formal, done once, but sometimes bulky. Non-standardised data exchange often means that code has to be specially written or adapted.

9.1 Local Disc Formats

In simple terms, the disc format used for model storage is simply a “file image” of the internal data structure. In these terms the file contains a representation of all the fields of all the entities associated with the object. To illustrate this, let us examine a disc file which might result from the data structure used for the course. The logical diagram is as in Fig. 9.1.

The data structure definitions are as follows:

```
class body
int number;
shell *pshell;
edge *pedge;
vertex *pvert;
body *next;

class shell
int number;
face *pface;
shell *next;
body *pbody;
```

I. Stroud and H. Nagy, *Solid Modelling and CAD Systems*,
DOI: 10.1007/978-0-85729-259-9_9, © Springer-Verlag London Limited 2011

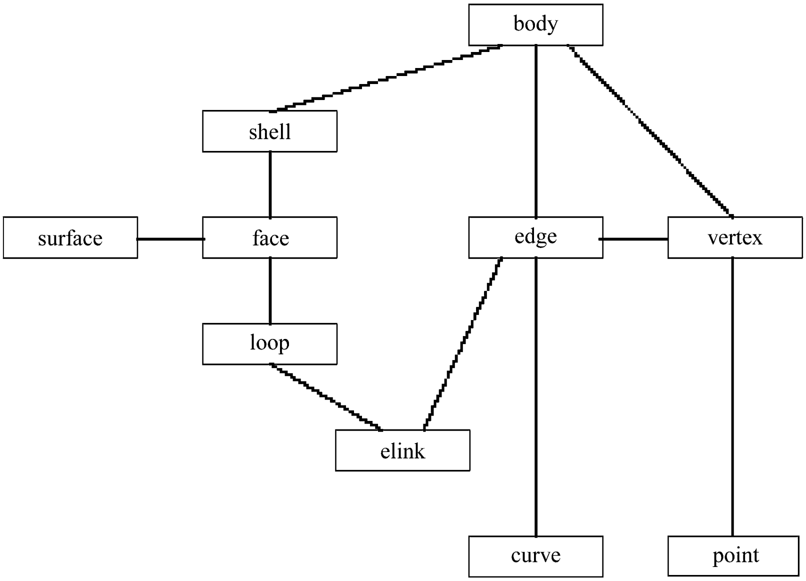


Fig. 9.1 Simple boundary representation data structure

```
class face
int number;
shell *pshell;
loop *ploop;
surface *surf;
face *next;

class loop
int number;
elink *eref;
loop *next;
face *pface

class elink
int number;
elink *cclink;
elink *cwlink;
loop *ploop;
edge *pedge;

class edge
int number;
elink *rlink;
elink *llink;
curve *pcurve;
```

vertex *svert;
vertex *evert;
edge *next;

class vertex
int number;
union(loop,edge) *pref;
point *pos;
vertex *next

The geometry formats are as follows:

plane: point, normal
cylinder: 2 axis points, point on surface
cone: apex, point on axis, point on surface
sphere: centre, point on surface
general quadric: 4 × 4 matrix
torus: centre, point on axis, point on major axis, point on surface
free-form surface: point matrix, weights, knots.

straight: Two points
circle: centre, normal, point on circle
ellipse: points on major and minor axes
free-form curve: vectors of points, weights, knots

9.1.1 Reading and Writing Disc-Files

This method is based on the first such disc format from the BUILD research system. There are two main reasons for describing this method. The first is that commercial system output may be considered to be the property of a company and, therefore, companies may not be happy if they are published. In any case, most system disc-files are written in binary format and you cannot read them anyway. The second main reason for using the BUILD method as a basis is that the use of logical pointers instead of memory locations is a method that is stable and is used in other systems. Therefore, if you understand the method then you should be able to understand other disc-files if you have to read one. In addition, the main modern data exchange standard, STEP, which will be described in Sect. 9.2.1.3, uses a similar method with line numbers doubling as pointers.

The first thing written is a header. This will contain various information, such as the version number of the writing system, system parameters such as tolerances and so on. In this method, the first line of model information consists of a line containing the number of entities in the file. This information is used to set up the arrays for reading. Note that the array indices start from 0, so the first element in the array is at *array*[0].

The method of reading and writing is similar to the method of copying objects, using entity numbers as array pointers. First of all, all entities are numbered consecutively from 0 to n so that the numbers can be used as logical pointers for the file. Real memory pointers are volatile and change when using the system which is why these are replaced by simple entity numbers which become logical pointers. The method is summarised for a face in Fig. 9.2. The values "1234567", "8834586" and "554396" represent real memory addresses in the example to distinguish these from the numbers written in the file.

The first thing written is the keyword "*face*" so that the reader can tell which type of entity is to be read. The first number written is the face identity, in this case 5. The face's original number, 27 in the figure, is replaced temporarily by the address of the face in the list of all faces in the object. The second number is the identity of the shell containing the face. The third number is the number of the loop of the face. The fourth number is the number of the surface of the face. The fifth and final value is the number of the next face. Here, though, the next face is a NULL pointer. In this case, the value "-1" is used as a number, since valid entities always have positive values. The final line, written to the file would be:

$$\text{face } 5\,0\,5\,5\,-1$$

The reverse process, the reading procedure, is illustrated in Fig. 9.3.

When reading the file, the first line of the file specifies the number of each type of entity to be read. Using this, a set of arrays are created containing empty entities of each type. When reading, the entities are filled up, using the memory addresses

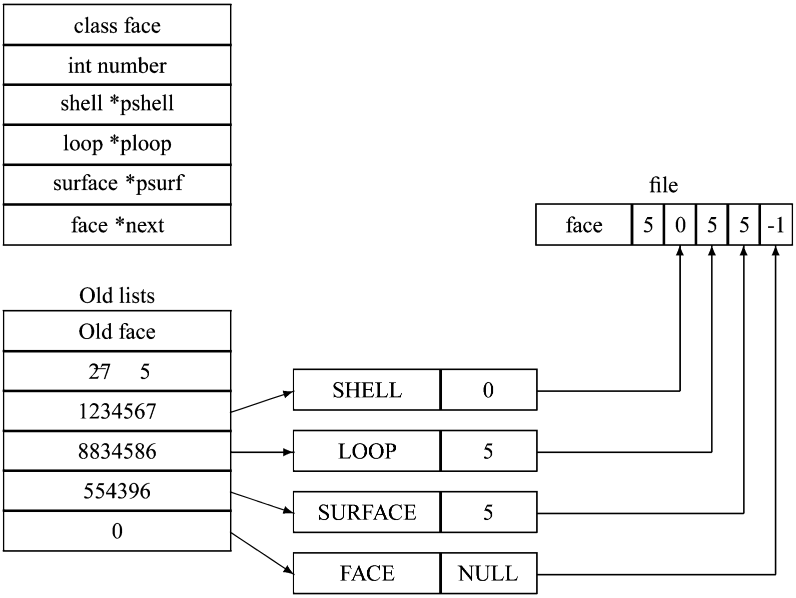


Fig. 9.2 Writing a face to disk

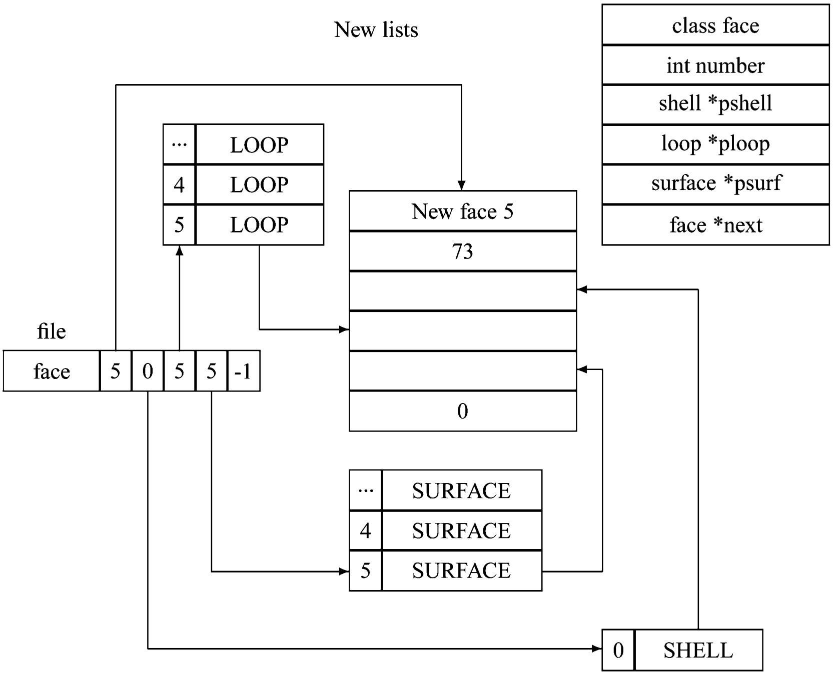


Fig. 9.3 Reading a face from disk

of the entities in the list instead of the logical pointers. For the face example, the reader reads the word "*face*" and the value, 5, and starts to fill element 5 in the list of faces. The actual number of the face, in this case 73, is not changed in the reading process because the numbers corresponding to pointers are interpreted as list addresses. Using the datastructure definition, the next value read, 0, is interpreted as a reference to a shell. The pointer in element 0 of the shell list is inserted into field 2 of the face record. The next value, 5, is used to find the loop in element 5 of the loop list and the pointer is written into field 3 of the face record. Then a pointer to surface 5 in the surface list is written into field 4 of the face record. Finally, the −1 is read. This should be a face record but is a NULL definition, so field 5 of the face record is set to 0, the NULL pointer.

For a cube, this works as follows. The first line of the file is:

0 0 1 1 6 6 24 12 8 6 12 8

This specifies the number of elements contained in the file. This is illustrated in Fig. 9.4.

0 assemblies
0 instances
1 body
1 shell

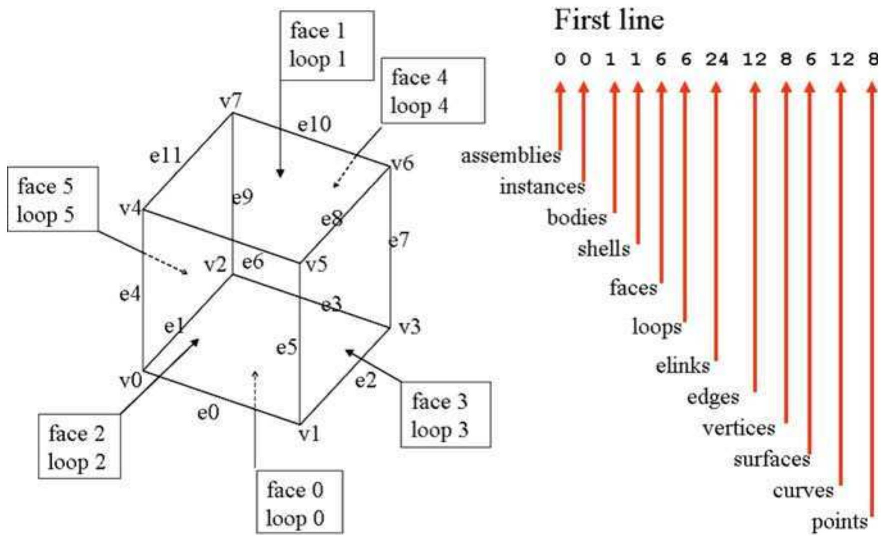


Fig. 9.4 The element number line

6 faces
6 loops
24 elinks
12 edges
8 vertices
6 surfaces
12 curves
8 points

This information is used to set up a number of arrays containing the elements to link, as shown in Figs. 9.5 and 9.6.

As described above, the values corresponding to pointers are interpreted according to the type of the field. The second line in the file is: “body 0 0 0 0 −1” (Fig. 9.7). The first value, 0, is interpreted as an integer, the number of the body. The second integer, 0, corresponds to a shell type field, and so is interpreted as the number of the shell from the shell entity array. Similarly the third integer, 0, is interpreted as a pointer to edge 0 in the edge array and the fourth integer in interpreted as a pointer to vertex 0 in the vertex array. The final integer, −1, corresponds to the next field of the body and indicates that the next pointer is NULL. By convention, here, a negative integer is interpreted as a NULL pointer.

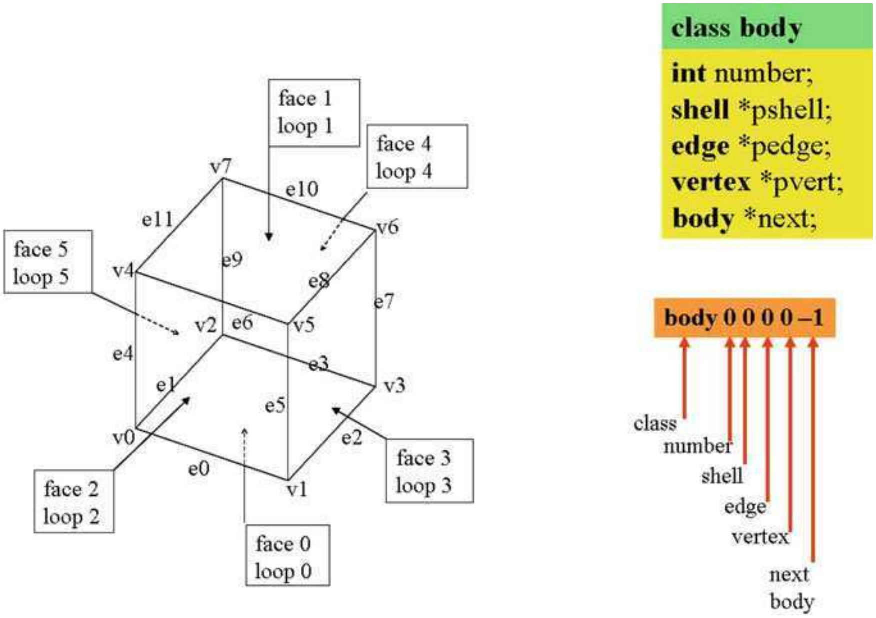
For an edge, with the line “edge 3 6 7 3 2 3 4”, you have the interpretation shown in Fig. 9.8. The “3” is the number of the edge to be defined. The “6” and “7” and the numbers of the edge-links. The next “3” is the curve number. The “2” and “3” are the array addresses of the start and end vertex and the final “4” is the address of the next edge. The connections are made locally, so the pointers are set

bodies	shells	faces	loops	elinks
b0	sh0	f0	l0	el0
		f1	l1	el1
		f2	l2	el2
		f3	l3	el3
		f4	l4	el4
		f5	l5	el5
				el6
				el7
				el8
				el9
				el10
				el11
				el12
				el13
				el14
				el15
				el16
				el17
				el18
				el19
				el20
				el21
				el22
				el23

Fig. 9.5 Data-structure tables (1)

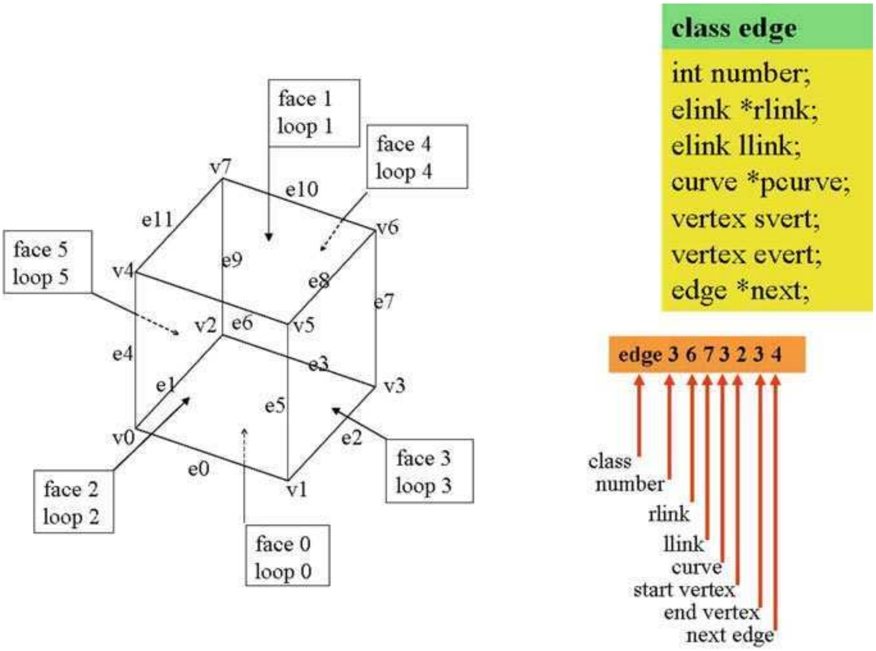
edges	vertices	surfaces	curves	points
e0	v0	s0	c0	p0
e1	v1	s1	c1	p1
e2	v2	s2	c2	p2
e3	v3	s3	c3	p3
e4	v4	s4	c4	p4
e5	v5	s5	c5	p5
e6	v6		c6	p6
e7	v7		c7	p7
e8			c8	
e9			c9	
e10			c10	
e11			c11	

Fig. 9.6 Data-structure tables (2)



```
class body
int number;
shell *pshell;
edge *pedge;
vertex *pvert;
body *next;
```

Fig. 9.7 The element number line



```
class edge
int number;
elink *rlink;
elink llink;
curve *pcurve;
vertex svert;
vertex evert;
edge *next;
```

Fig. 9.8 The element number line

to point from the edge to different entities, but the back pointers are only set when the records for those entities are read.

The whole file might look like:

```
SIMPLE version 1.0 0 0 1 1 6 6 24 12 8 6 12 8
BODY 0 0 0 0 −1
SHELL 0 0 0 0 0
FACE 0 0 0 0 1
LOOP 0 0 −1 0
LELINK 0 1 3 0 3
LELINK 1 2 0 0 2
LELINK 2 3 1 0 1
LELINK 3 0 2 0 0
FACE 1 0 1 1 2
LOOP 1 4 −1 1
LELINK 4 0 11 1 5 7 −1
LELINK 5 0 6 1 6 4 −1
LELINK 6 0 8 1 7 5 −1
LELINK 7 0 10 1 4 6 −1
FACE 2 0 2 2 3
LOOP 2 8 −1 2
LELINK 8 0 6 2 9 11 −1
LELINK 9 0 4 2 10 8 −1
LELINK 10 0 0 2 11 9 −1
LELINK 11 0 5 2 8 10 −1
FACE 3 0 3 3 4
LOOP 3 12 −1 3
LELINK 12 0 8 3 13 15 −1
LELINK 13 0 5 3 14 12 −1
LELINK 14 0 1 3 15 13 −1
LELINK 15 0 7 3 12 14 −1
FACE 4 0 4 4 5
LOOP 4 16 −1 4
LELINK 16 0 10 4 17 19 −1
LELINK 17 0 7 4 18 16 −1
LELINK 18 0 2 4 19 17 −1
LELINK 19 0 9 4 16 18 −1
FACE 5 0 5 5 −1
LOOP 5 20 −1 5
LELINK 20 0 11 5 21 23 −1
LELINK 21 0 9 5 22 20 −1
LELINK 22 0 3 5 23 21 −1
LELINK 23 0 4 5 20 22 −1
EDGE 0 0 1 10 3 1 11 −1 0
EDGE 1 1 2 14 2 2 0 −1 1
EDGE 2 2 3 18 1 3 1 −1 2
EDGE 3 3 0 22 0 4 2 −1 3
EDGE 4 0 4 23 9 5 3 −1 4
EDGE 5 1 5 11 13 6 4 −1 5
EDGE 6 4 5 5 8 7 5 −1 6
EDGE 7 2 6 15 17 8 6 −1 7
EDGE 8 5 6 6 12 9 7 −1 8
EDGE 9 3 7 19 21 10 8 −1 9
EDGE 10 6 7 7 16 11 9 −1 10
EDGE 11 7 4 4 20 0 10 −1 11
VERTEX 0 EDGE 0 0 1
VERTEX 1 EDGE 0 1 2
VERTEX 2 EDGE 1 2 3
VERTEX 3 EDGE 2 3 4
VERTEX 4 EDGE 4 4 5
VERTEX 5 EDGE 5 5 6
VERTEX 6 EDGE 7 6 7
VERTEX 7 EDGE 9 7 0
SURFACE 0 0 1 PLANE 0.0000 0.0000 −1.0000 5.0000
SURFACE 1 0 1 PLANE 0.0000 0.0000 1.0000 5.0000
SURFACE 2 0 1 PLANE 0.0000 −1.0000 0.0000 5.0000
SURFACE 3 0 1 PLANE 1.0000 0.0000 0.0000 5.0000
SURFACE 4 0 1 PLANE 0.0000 1.0000 0.0000 5.0000
SURFACE 5 0 1 PLANE −1.0000 0.0000 0.0000 5.0000
CURVE 0 0 1 STRAIGHT −5.0000 −5.0000 −5.0000 5.0000 −5.0000 −5.0000
CURVE 1 0 1 STRAIGHT 5.0000 −5.0000 −5.0000 5.0000 5.0000 −5.0000
CURVE 2 0 1 STRAIGHT 5.0000 5.0000 −5.0000 −5.0000 5.0000 −5.0000
CURVE 3 0 1 STRAIGHT −5.0000 5.0000 −5.0000 −5.0000 −5.0000 −5.0000
CURVE 4 0 1 STRAIGHT −5.0000 −5.0000 −5.0000 −5.0000 −5.0000 5.0000
CURVE 5 0 1 STRAIGHT 5.0000 −5.0000 −5.0000 5.0000 −5.0000 5.0000
CURVE 6 0 1 STRAIGHT −5.0000 −5.0000 5.0000 5.0000 −5.0000 5.0000
CURVE 7 0 1 STRAIGHT 5.0000 5.0000 −5.0000 5.0000 5.0000 5.0000
CURVE 8 0 1 STRAIGHT 5.0000 −5.0000 5.0000 5.0000 5.0000 5.0000
CURVE 9 0 1 STRAIGHT −5.0000 5.0000 −5.0000 −5.0000 5.0000 5.0000
CURVE 10 0 1 STRAIGHT 5.0000 5.0000 5.0000 −5.0000 5.0000 5.0000
CURVE 11 0 1 STRAIGHT −5.0000 5.0000 5.0000 −5.0000 −5.0000 5.0000
POINT 0 −5.0000 −5.0000 −5.0000
POINT 1 5.0000 −5.0000 −5.0000
POINT 2 5.0000 5.0000 −5.0000
POINT 3 −5.0000 5.0000 −5.0000
POINT 4 −5.0000 −5.0000 5.0000
POINT 5 5.0000 −5.0000 5.0000
POINT 6 5.0000 5.0000 5.0000
POINT 7 −5.0000 5.0000 5.0000
```

9.1.2 Versions and Old Disc-Files

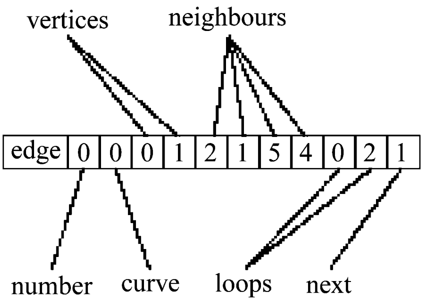
Unlike standardised data exchange (see Sect. 9.2), disc-file formats in modelling systems change relatively frequently, especially at the beginning of system development as new facilities are added. It is, therefore, necessary to be able to recover old disc-files in new versions of the CAD system.

What the CAD system has to do is to adopt a strategy for introducing the new information. Suppose, for example, that the system implementers decide to introduce shells, to group faces, as in Fig. 9.9.

Supposing the original object is a rectangular block with a rectangular cavity, that is, an empty area totally enclosed in the block, as shown in Fig. 9.9a. The original save would have created a simple list of all faces in the object, as shown in Fig. 9.9b. The way that the CAD system could read the old file is to put all the faces into a simple list in a new shell, as in Fig. 9.9c. A subsequent step would then use face adjacency to separate the face list into separate shells, to create the final structure shown in Fig. 9.9d.

This is a simple change to manage. A more complicated change would be to change the edge connectivity from direct pointers to loop-edge links. For this kind of change it is necessary to introduce new topological elements, as shown in Fig. 9.10.

In the original structure an edge has direct pointers to neighbouring edges while in the new structure there are loop-edge links which have to be added. For *e*0 in the figure, the input line might be:



edge	0	0	0	1	2	1	5	4	0	2	1

What changes is that instead of having direct pointers to the neighbouring edges and the loops, the edge will have pointers to two loop-edge links. One way that this might be done is to create all the edges without loop-edge links and then to add these progressively as the edges are read. So, when edge 0 is read, with the structure shown above, two loop-edge links are created between the edge and loop 0, on the right, and loop 2 on the left. The four neighbouring edges are edges 2 (rcw), 1 (rcc), 5 (lcw) and 4 (lcc). Here, "rcw" stands for Right ClockWise edge, "rcc" for Right Counter-Clockwise, "lcw" for Left ClockWise and "lcc" for Left

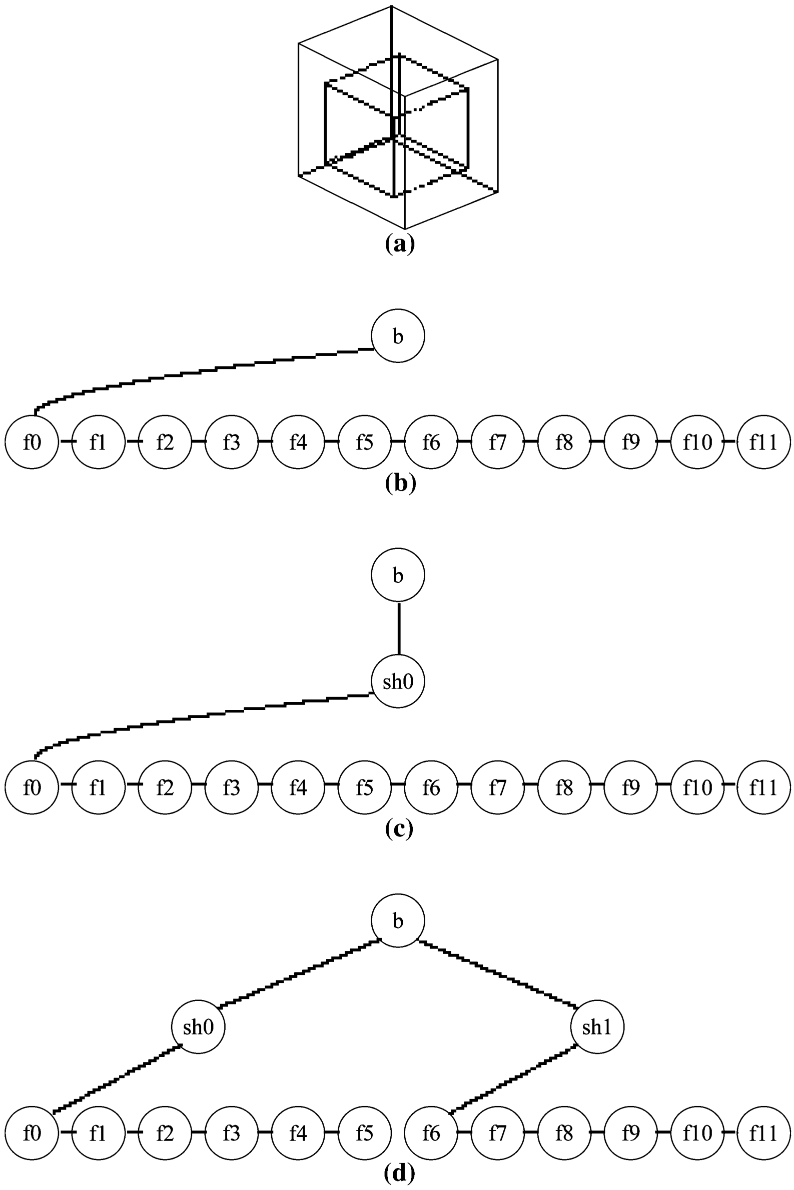
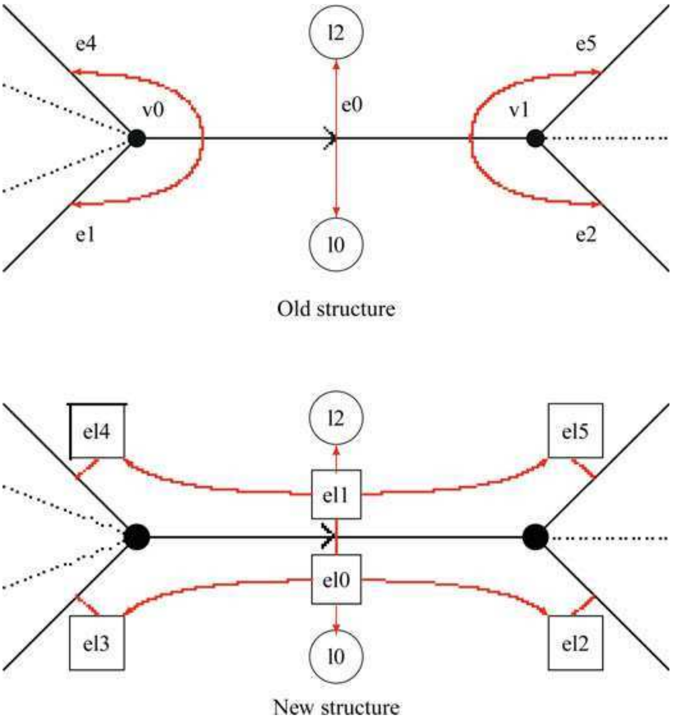


Fig. 9.9 Reading old disc files—adding shells

Counter-Clockwise. Edge 2 has no loop-edge links, so one is created, referring to loop 0, the right loop of edge 0. The clockwise link pointer of loop-edge link 0 is set to point at edge-link 2 and the counter-clockwise link pointer of loop-edge link



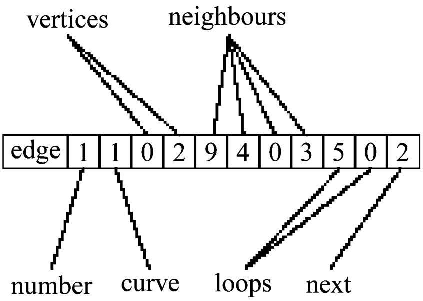
Old structure

New structure

Fig. 9.10 Reading old disc files—adding loop-edge links

2 set to point to loop-edge link 0. The process is repeated for the other three relationships, rcc, lcw and lcc.

The next edge to be read, edge 1, has the input structure as follows:



edge	1	1	0	2	9	4	0	3	5	0	2

The same process is repeated. However, edge 1 already has one loop-edge link, loop-edge link 3 which refers to loop 0, the left loop of edge 1. In this case loop-edge link 3 becomes a “left” loop-edge link. And so on, the new structure being created, “on-the-fly”. When the model is next saved, the new elements will be saved to make the new model file compatible with the new version.

A third type of change might be to change geometry. Suppose the model developer decides to replace Bèzier geometry with NURBS geometry. Since Bèzier geometry is a special case of NURBS geometry this simply means creating a NURBS surface with the appropriate parameters to replace the original geometry.

Another type of geometric change might be to add or suppress a particular type. Suppose an ELLIPSE type is added to the CAD system instead of using, say, a NURBS representation. In this case, probably nothing would be done during reading, the old numerical ellipses would remain numerical and ellipses would be added only to new models. Going the other way, if the new CAD version has suppressed ELLIPSE as a specific curve type, then whenever an ellipse-type curve is read then the curve created is a numerical curve, say a NURBS curve, representing the same shape.

Any such changes should be completely invisible to the user, though. This section is just to indicate how reading old geometry works. It can be a good idea, as general practice, to migrate old CAD files to new versions by opening them with the new system and then saving them with the new system elements. New versions of CAD systems do not always introduce new datastructure elements, but just in case...

9.1.3 Reading Newer Disc-Files

If you are using, say, version 10 of a CAD system and you want to receive a file from someone using version 11, then you should not expect to be able to use the CAD system native form. Instead, it is necessary to use a standardised exchange format, although you may lose CAD-system specific information. In any case, whenever exchanging CAD files with someone who has the same CAD system, check the version numbers.

9.1.4 Reading Other CAD System Models

This section describes importing other CAD system models into your own. This may be done partly because writing specific translators between systems is often simpler than writing translators for standards because there are fewer cases to cope with. Also, in the early days there were no standards for CAD solid model data exchange.

One point to note is that specific translators are most effective if the structures are similar, if they are too different then it is difficult or impossible to write an automatic translator. This is illustrated in Fig. 9.11. In the early days of solid modelling there were several solid representation schemes that existed. Since then, CAD systems generally use Boundary Representation, so this is less of a problem. However, it is part of the history, which is why it is presented here. Enjoy.

Another important consideration is the compatibility of different systems in terms of their equivalence of information. Once information is lost by converting it then it is difficult or impossible to recreate, so sometimes exchanging models to and then from another system results in a different model. See Fig. 9.12.

Suppose system A creates a cylinder and then passes it to system B which has only planar surfaces and straight lines. System B has to convert the cylinder into an approximation, at the centre of Fig. 9.12. If system B then re-exports the cylinder to system A, system A will only get the approximation, because the original geometric information has been lost. Naturally this is a very simple example. Systems are nowadays, in general, largely compatible in terms of structure and geometry. Even now, though, there are some special surface representations which require conversion. However, there are more variations in things like information and history which may be lost in any such conversion.

One way of converting a model from another CAD system is illustrated in Fig. 9.13. The datastructure of the foreign CAD system is reproduced inside the importing system, the foreign CAD file is read in creating a separate CAD model and then this model is interrogated to create the final model.

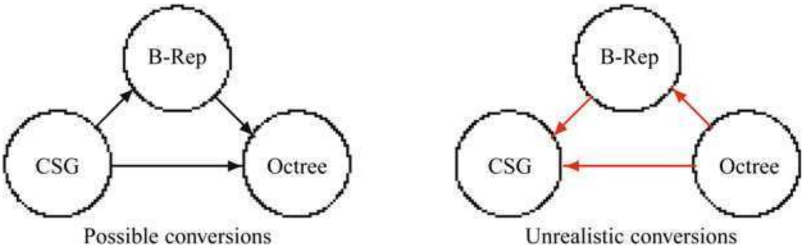


Fig. 9.11 Reading other CAD disc files—representation conversions

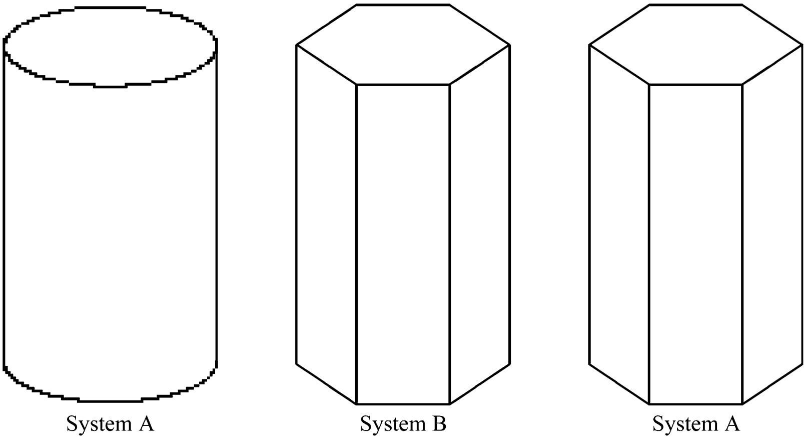


Fig. 9.12 Reading other CAD disc files—information loss

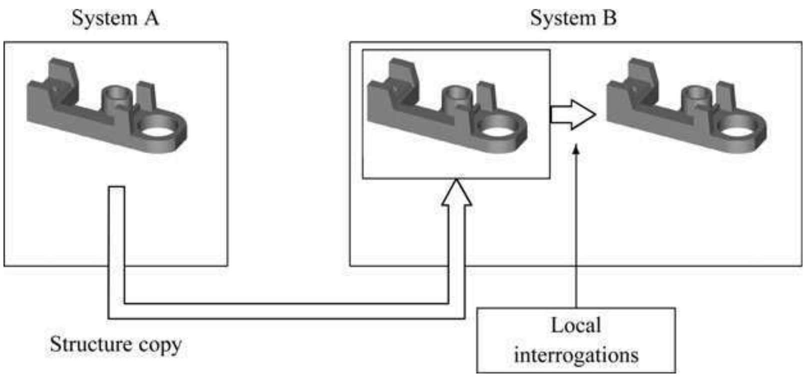


Fig. 9.13 Reading other CAD disc files—information loss

This is a little similar to what happens when reading STEP files, see Sect. 9.2, since STEP defines a datastructure for exchange.

For CAD system vendors it is advantageous to be able to read CAD files from other systems so that a company can import its old files into a new system, should the company change their CAD vendor. CAD systems have different advantages and disadvantages and so it is not unreasonable to change systems or use multiple systems, hence exchanging CAD files is not uncommon. However, exchanges can be haphazard, depending on how much of the other CAD system's structure is imported. With data exchange using standards the data available is well-defined. Another point to note is that CAD system file structures change more frequently than standard formats which need to stay stable. Also, for archiving, it can be more useful to retain models in a standard form than in the local CAD system form. Standard files tend to be bulkier than CAD files, but have a longer life.

A final point to be made in this section concerns CAD system families. Originally CAD system developers created their own software and there were more differences between systems. One company, Shape Data Ltd. of Cambridge, UK pioneered the use of Boundary Representation in the commercial world. They developed a package, Romulus, which could be used as a kernel for system developers. See Fig. 1.67 for how this kernel is placed. Since modelling software development is time-consuming and costly, using a package has become popular. This means that there are several systems which have a common interior and, hence, internal datastructure. Currently, the two main commercial kernels are Parasolid, the descendent of Romulus from Shape Data, and ACIS, developed by some of the original directors of Shape Data. There is also a kernel system available for CSG, SvLis, which can be used for building a different type of CAD system. Because of this common basis several systems can read each other's CAD model files simply because they use the same interior.

9.2 Shape Data Exchange

The purpose of data exchange is to be able to communicate shape information between different systems. Traditionally this has concerned exchange between CAD and CAD systems or between CAD and CAM systems. There are two ways to perform data exchange between systems:

1. Specific system–system interchange
2. Interchange via a neutral file format

The role is illustrated in Fig. 9.14, a figure which I didn't invent, but I do not know the original source to quote.

It is clear that if communication is needed between a new system SYSTEM F and the other systems then ten new translators need to be developed (five for system F and one each for systems A–E). With communication via a neutral file format only two translators need to be developed, one to read and one to write the neutral format. Communication using a standard works something like a telephone exchange (invented by the Hungarian Tivadar Puskas apparently).

Although it seems clear that using a neutral format is preferable there are, almost inevitably, problems, and there has been mixed success in introducing and gaining acceptance for a neutral exchange format.

Another problem concerns incompatibility between modelling systems. Exchange between similar systems, say between B-rep systems, is much more straightforward than communication between CSG systems and B-rep systems, as already mentioned and illustrated in Fig. 9.11. These technical problems dogged the CAD system market for many years. The current attempt at introducing a globally acceptable exchange format is the STEP standard.

Some examples of CAD data exchange files are given in Appendix B. It should be said that data exchange files are not really meant to be read by people, but the examples are included so that you can see how they look. It can be helpful to understand a little of the contents.

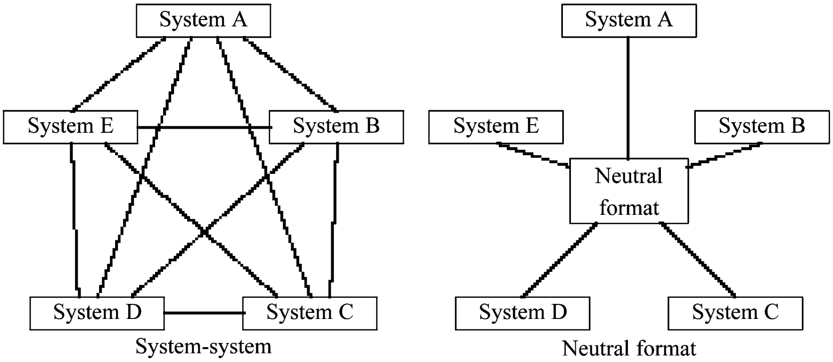


Fig. 9.14 A commonly used picture for illustrating communication methods

9.2.1 Standards

The following are some common standards.

9.2.1.1 IGES

The following is what I think is true. IGES started off as a standard for exchanging drawings. In about 1979, at a CAM-I conference on solid modelling abilities, there was a discussion about exchanging solid models and IGES was proposed as a way of doing this. Extensions were needed to IGES and the CAM-I group of experts produced a proposal for solid exchange, called XBF. This was used to exchange solids within the CAM-I group but was seen mainly as an input to the development of IGES. When the new extended version of IGES appeared it largely ignored the XBF suggestions and contained geometric descriptions. In later versions primitive solids, as used in CSG representations, were included even though the main solid modelling emphasis had moved away from CSG. One type of entity, the trimmed surface patch, approaches a little the notion of Boundary Representation, but IGES lacks the topological structure needed for proper exchange. Many systems can exchange IGES data. However, joining the model, which is optional in some systems, requires identifying matching elements to glue surface patches together.

One personal criticism of IGES is that it seems to contain lots of different representation types instead of requiring that users conform. This seems to have meant that implementers used only part of the IGES standard, which is very large, instead of the whole. This has meant in the past that two applications using IGES were not able to communicate because they had implemented different parts of IGES.

Another criticism of IGES is that it is verbose. If you look at the example of IGES output in Appendix B you will see that it is 80 columns wide and consists of fixed fields. I used to ask my students why should the lines be exactly 80 columns wide. After a brief puzzled pause I would announce triumphantly that it was because the original standard was intended to be stored on punched cards. I then used to ask if any of the students had ever seen a punched card. One year a student asked: “What’s a punched card?” I was forced into the realisation that I am older than I think. For the record, a punched card is shown in Fig. 9.15, it is a piece of thick paper, about 18.75 mm × 8.25 mm. The reason for asking about punched cards is to show how standards tend to gather history because they have to be stable. Punched cards are no longer used by computers but they still have an implicit dinosaurial existence through IGES.

Another point to make is that one manufacturer, if memory serves me well, Aerospatiale, developed the SET version of IGES to be more compact than IGES. The SET format has free format fields and removes the white spaces. What this shows is how some people adapt standards because they see improvements, which means that there are standards variants. It is certain that the fixed format of IGES is now well outdated and makes traditional IGES files very large, so there was a good

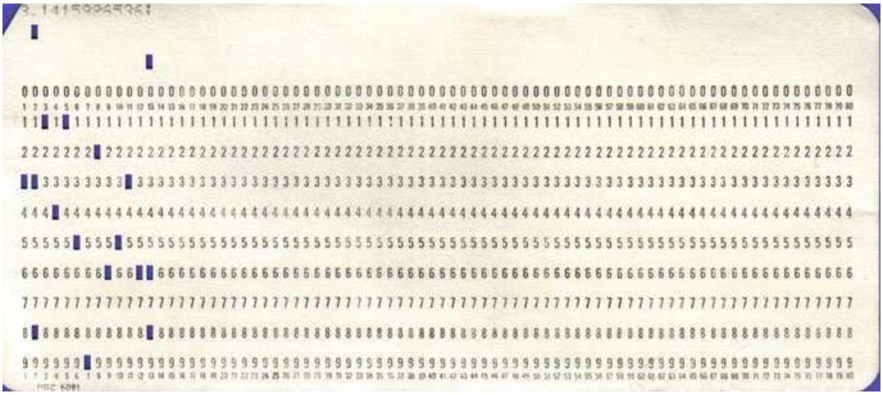


Fig. 9.15 A punched card

reason to adapt the standard. However, what is shown in the Appendix is the ASCII clear version. There is also a compressed version in binary form which reduces the size of the files somewhat.

A further criticism of IGES is the use of code numbers instead of type keywords. If you look at the IGES example in the Appendix, you will not find helpful types, like "straight line", "plane", "point", etc. Instead there are code numbers which are explained in the IGES documentation. The format is explained in more detail below. For the full standard, though, and explanations, see the official documentation [1], or later versions.

IGES has five sections:

- S section—the Start section, one line at the beginning of the file.
- G section—the Global section containing information about the sending system, delimiter characters and so on.
- D section—the Directory entry section, a list of entities in the file with various parameters and a reference to the geometric data in the parameter section.
- P section—the Parameter data section, which contains the geometric data which is interpreted according to the entity.
- T section—the Terminate section.

The S-section consists of one line, so is not reproduced here. The G-section is shown below.

```
1H,,1H;,20HCNEXT - IGES PRODUCT,13HBLK102030.igs,44HIBM CATIA IGES - CATG      1
IA Version 5 Release 15 ,27HCATIA Version 5 Release 15 ,32,75,6,75,15,5HG      2
PART4,1.0,2,2HMM,1000,1.0,15H20071114.070402,0.001,10000.0,6Hstroud,3HSTG      3
I,11,0,15H20071114.070402,;                                                G      4
```

In several places you can see a number followed by "H". This stands for a sequence of characters, so "1H," means ",", which indicates a separation character. As mentioned above, there is information about the system which produced the file and other source information as well. This is so that the file has some sort of traceability. Note, also, that the last characters in the line are "G" followed by some spaces and a number (1–4). The "G" indicates the section and the numbers are line sequence numbers, used for sorting when information was contained on punched cards rather than in sequential files.

The first four lines from the directory entry section are:

```
108     1      0      0      0      0      0      001010001D      1
108     0      0      1      0                           0D      2
110     2      0      0      0      0      0      001010001D      3
110     0      0      1      0                           0D      4
```

In fact, this corresponds to two entities, each having two lines specifying different data aspects. The first number in the two rows is the entity type code, 108 for the first entity and 110 for the second. In order to decode these it is necessary to look at the standard documentation. 108 means a plane and 110 means a straight line. The second number of the first line of each entity is a reference to a line in the parameter data section which contains the geometric information associated with the entity.

Other element codes in the file are:

102—composite curve
142—curve on parametric surface
144—trimmed parametric surface

These are interesting because these are a way of passing topological information using IGES. IGES does not have explicit topological information, this is a way of providing the information indirectly. A trimmed parametric surface is approximately equivalent to a face. The composite curve entity is used to create a curve sequence. This is roughly equivalent to a loop, or contour, which is then associated with a parametric surface and used to define the trimmed parametric equivalent to a face. To be closer to B-rep topology it would be necessary to reuse the same curves. Since the curves are repeated it would be necessary to use geometric tests to associate the elements if a closed volume is to be created.

The two lines from the parameter data section corresponding to the lines above, and the following lines showing some other entities are:

```
108,-1.0,0.0,0.0,5.0,0,-5.0,10.0,0.0,1.0,0,0;                        1P      1
110,-5.0,10.0,15.0,-5.0,-10.0,15.0,0,0;                              3P      2
110,-5.0,-10.0,15.0,-5.0,-10.0,-15.0,0,0;                            5P      3
110,-5.0,-10.0,-15.0,-5.0,10.0,-15.0,0,0;                            7P      4
110,-5.0,10.0,-15.0,-5.0,10.0,15.0,0,0;                              9P      5
102,4,3,5,7,9,0,0;                                                  11P      6
142,0,1,0,11,2,0,0;                                                 13P      7
144,1,1,0,13,0,0;                                                   15P      8
```

The first number in the line is the entity type code so that the rest of the data can be interpreted directly using the standard description. Just before the line sequence number is the line sequence number of the directory entry section.

Included is one “cycle” of information, with the geometric information for the eight entities corresponding to a face. The first is the geometric information for a plane. After the code, 108, the next four numbers define the plane. They are in the form: $ax + by + cz - d = 0$. Given are the four values: a, b, c, d and the points of the plane (x, y, z) are any points satisfying the equation. This means that the plane has normal vector $(-1, 0, 0)$ and is at distance 5 from the origin, so the definition of the plane is $x = -5$. After this definition is a zero pointer, because this is an unbounded plane, three values specifying the position of a display symbol and a size parameter.

Following the plane definition are four lines defining the straight lines bounding the face. Each straight line is defined by its start and end points.

Then comes the geometric data specifying the composite curve, type 102. This consists of the number of curves and the directory entry references for the four bounding curves, which are the four preceding curves.

The next line specifies a curve on a parametric surface, type 142. This specifies the directory entry of the surface, 1 in this case, and the directory entry of the curve on the surface, 11. There are some other organisational parameters as well.

Finally comes the specification of the trimmed parametric surface, type 144. This specifies the directory entry of the surface being trimmed, 1, and then the boundary or boundaries. The second 1 indicates that the boundary trims the surface, the 0 means that there are no inner boundaries, and the 13, a reference to the directory entry of the curve on parametric surface, defines the boundary.

Finally, the termination line is:

```
S      1G      4D     96P     48                                         T      1
```

This contains information about the number of lines in each section, i.e.

S: 1
G: 4
D: 96
P: 48

This states that there was one start record, four lines in the global section, 96 lines in the directory entry section and 48 lines in the parameter data section.

9.2.1.2 VDA-FS

VDA-FS stands for Verband der Automobilindustrie – Flächen Schnittstelle. The standard was developed for data exchange by the German car industry. This description is based on documents kindly provided by Claudia Rainfurth and Meinolf Gröpper of VDMA. For the full definition of VDA-FS, the reader should refer to the original documents. However, some of the information is given here because of the difficulty of obtaining the definition.

VDA-FS is a relatively simple standard, in terms of the number of elements defined, yet is capable of communicating complex shapes. The geometric elements in the standard are:

- POINT
- PSET
- MDI (Master dimension)
- CIRCLE
- CURVE
- SURF
- CONS (Curve on surface)
- FACE
- TOP

In addition there are some non-geometric items:

- HEADER
- $$ (Comment)
- BEGINSET
- ENDSET
- GROUP
- TMAT (Transformation matrix)
- TLIST (Transformation list)
- END

Concentrating on the geometric items, the formats, from the VDA definition document, are:

- POINT / x,y,z
  The data is just the three coordinates of the point.

- PSET / n,(n)*[x,y,z]
  The data is the number of points in the set followed by that number of coordinate triples.

- MDI / n,(n)*[x,y,z,vx,vy,vz]
  The data is the number of points followed by that number of sextuples of values, three for the point and three for the vector. The vector does not need to be normalised.

- CIRCLE / x,y,z,r,vx,vy,vz,wx,wy,wz,α,β
  The first triple gives the coordinates of the centre point. This is followed by the radius of the circle. The next two triples define two axes in the plane of the circle, which can be thought of as the *X*- and *Y*-axes of the local coordinate system. The triples are vectors relative to the centre and should be perpendicular to each other. The final two values are the angles of the start and end of the circle, in degrees.

- CURVE / n,(n+1)*[par], (n)*[ord,(ord)*[ax],(ord)*[ay],(ord)*[az]]
  The curve is a piecewise curve consisting of *n* segments. After *n* has been given there are $n + 1$ parameter values defining the starts and ends of the segments. There then follows the curve data. Each curve segment is defined by its order (the number of control points) followed by that number of *x*-values, that number

of y-values and that number of z-values. The parameter values defining the limits of the curve segments are global parameters, each curve segment is parametrised locally from 0 to 1 for point calculation. The values of ax, ay and az are coefficients of a polynomial in u, where u is the curve parameter. So, point calculation is done in the same way, for x this would be: $x(u) = \sum_{j=0}^{iord-1} ax_j * u^j$.

- SURF / nps, npt, (nps+1)*[pars], (npt+1)*[part], (nps*npt)*[iordu, iordv, (iordu* iordv)*[ax], (iordu*iordv)*[ay], (iordu*iordv)*[az]]
  This is similar in style to the curve definition. This defines a set of joined surface patches, *nps* in the u-direction and *npt* in the v-direction. The global parameters are given, but each surface segment is parametrised locally from 0 to 1. Each surface patch is of order *iordu* in the u-direction and *iordv* in the v-direction. There then follows the list of polynomial coefficients, organised in rows in the u-direction. Point calculation is done in a similar way to that for curves, for x this would be: $x(u,v) = \sum_{k=0}^{iordv-1} \sum_{j=0}^{iordu-1} ax_{j,k} * u^j * v^k$.

- CONS / surfname, curvename, s1, s2, np, (np+1)*[parp], (np)*[iordp, (iordp)*[as], (iordp)*[at]]
  CONS stands for "Curve on surface" and associates a curve with a surface. The surface and curve are given by name. The s1 and s2 are the two parameter limits of the curve. The last part of the data defines the 2D curve in the surface parameter space corresponding to the named curve. *np* defines the number of segments, this is followed by the set of $np + 1$ parameter values defining the curve limits and then the *np* sets of curve parameters. Each 2D curve segment is given by the order and then that number of coefficients of the u polynomial followed by the coefficients of the v polynomial. Note that there is duplicate information in that a parameter value of the specified curve gives a point in 3D space. The u, v coordinates also lead to a point in 3D space via the surface, so these two should tally.

- FACE / surfname, m, (m)*[n, (n)*[consname, w1, w2]]
  FACE is a record to associate groups of CONS elements into surface limits. *surfname* is the name of the surface. *m* gives the number of boundary sets of the face. This can be one, if the face has only an outer boundary, but can be more. There then follow the *m* sets of boundaries. The first element, *n* specifies the number of curves in the boundary and this is followed by that number of named CONS together with the lower and upper parameter limits. This is a "trimmed patch" definition.

- TOP / m, (m)*[(2)*[fsname, n, (n)*[consname, w1, w2]], icont]
  The TOP item is for linking faces and/or surfaces into connected groups. The first element, *m* gives the number of face–surface pairs to link and there then follows that number of pair definitions. Each pair consists of a pair of records, a face or surface name followed by the number *n* of CONS entities in the boundary followed by that number of CONS names and the lower and upper parameter values. Finally, the integer *icont* specifies the continuity between the

surfaces, 0 for position, 1 for tangency and 2 for curvature continuity. Note that this can join two FACE entities, two SURF entities or a FACE entity to a SURF entity.

- TMAT / c11, c12, c13, c21, c22, c23, c31, c32, c33, c41, c42, c43 The main elements of a 4 × 4 transformation matrix of the form:

$$\begin{bmatrix} c11 & c12 & c13 & c41 \\ c21 & c22 & c23 & c42 \\ c31 & c32 & c33 & c43 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  There is also a condition that the matrix is invertible.

- TLIST / tmatname, n, (n)*ename The name of a transformation, the number of elements in the list, *n*, followed by a list of *n* element names to be transformed by the matrix.

The first part of the file in Appendix B is a heading identifying the origin of the data. This is to provide data traceability.

```
SLDWORKS = HEADER / 20                                                  00000001
************************************************************************00000002
VDAFS VERSION     :  2.0                                                00000003
----------------------------- SENDER DATA ------------------------------00000004
SENDER COMPANY    :   EPFL                                              00000005
CONTACT PERSON    :  Ian Stroud                                         00000006
TELEPHONE NO.     :  UNKNOWN                                            00000007
ADDRESS           :  UNKNOWN                                            00000008
SENDING SYSTEM    :  SolidWorks-2007078                                 00000009
SENDING DATE      :  11/15/2007                                         00000010
FILE NAME         :  blk102030vda.vda                                   00000011
------------------------------ PART DATA -------------------------------00000012
PROJECT NAME      :  UNKNOWN                                            00000013
OBJECT CODE       :  NONE                                               00000014
VARIANT           :  NONE                                               00000015
CONFIDENTIALITY   :  UNCLASSIFIED                                       00000016
DATE EFFECTIVE    :  11/15/2007                                         00000017
---------------------------- RECEIVER DATA -----------------------------00000018
COMPANY NAME      :  UNKNOWN                                            00000019
RECEIVER NAME     :  UNKNOWN                                            00000020
************************************************************************00000021
```

As stated above, an interesting aspect of VDAFS is its relative simplicity. It was developed to allow German car manufacturers to exchange complex shapes, so is more suited to freeform shapes than to the simple cube example given. However, since the aim here is to compare the same example with different exchange methods, the simple example is used here, too.

In the cube example there are several curve definitions similar to the following:

```
CV1 = CURVE / 1, 0.0000000000000000E+00, 2.0000000000000000E-02,        00000022
  2,                                                                    00000023
    -5.0000000000000000E+00, 3.4694469519536142E-15,                    00000024
    1.0000000000000000E+01, -2.0000000000000000E+01,                    00000025
    1.5000000000000000E+01, 0.0000000000000000E+00                      00000026
```

To interpret this you have the first integer which specifies that there is one curve in the sequence. The next two values, 0 and 0.02, give the global curve parameters. However, each curve is parametrised from 0 to 1 locally. The curve data is given as the order, 2, indicating that the curve is linear, followed by the polynomial coefficients. In fact, here, the first column of coefficients can be interpreted as a point and the second column as a vector, so the line is defined geometrically as: $(-5, 10, 15) + t * (0, -20, 0)$. Since this is parametrised from 0 to 1, the end points of the line are $(-5, 10, 15)$ and $(-5, -10, 15)$.

```
SR13 = SURF / 1, 1, 0.0000000000000000E+00, 1.0000000000000000E+00,       00000082
 0.0000000000000000E+00, 1.0000000000000000E+00,                          00000083
4, 4, -4.9999999999999964E+00, -2.6020852139652106E-15,                   00000084
 -2.6020852139652106E-15, 2.6020852139652106E-15,                         00000085
 0.0000000000000000E+00, 0.0000000000000000E+00,                          00000086
 0.0000000000000000E+00, 0.0000000000000000E+00,                          00000087
 0.0000000000000000E+00, 0.0000000000000000E+00,                          00000088
 0.0000000000000000E+00, 0.0000000000000000E+00,                          00000089
 8.6736173798840355E-16, 0.0000000000000000E+00,                          00000090
 0.0000000000000000E+00, 0.0000000000000000E+00,                          00000091
 -1.0000000000000000E+01, 1.9999999999999996E+01,                         00000092
 0.0000000000000000E+00, -1.7347234759768071E-15,                         00000093
 0.0000000000000000E+00, 0.0000000000000000E+00,                          00000094
 0.0000000000000000E+00, 0.0000000000000000E+00,                          00000095
 0.0000000000000000E+00, 0.0000000000000000E+00,                          00000096
 0.0000000000000000E+00, 0.0000000000000000E+00,                          00000097
 1.7347234759768071E-15, 0.0000000000000000E+00,                          00000098
 0.0000000000000000E+00, 0.0000000000000000E+00,                          00000099
 -1.5000000000000000E+01, 0.0000000000000000E+00,                         00000100
 0.0000000000000000E+00, 0.0000000000000000E+00,                          00000101
 2.9999999999999996E+01, 0.0000000000000000E+00,                          00000102
 0.0000000000000000E+00, 0.0000000000000000E+00,                          00000103
 0.0000000000000000E+00, 0.0000000000000000E+00,                          00000104
 0.0000000000000000E+00, 0.0000000000000000E+00,                          00000105
 0.0000000000000000E+00, 0.0000000000000000E+00,                          00000106
 0.0000000000000000E+00, 0.0000000000000000E+00                           00000107
```

This is a surface definition. The first two numbers, 1 and 1, indicate that this is a single surface element, not a collection of patches. The next four values indicate the global parameters in the u and v directions, in this case 0 and 1 for both. Since there is only a single patch all the rest of the data concerns this surface element. The first two numbers are the order for this patch. Interestingly enough, these are both 4, indicating that the planar surface has been converted to a cubic surface. However, from the data it can be seen that many of the polynomial coefficients are zero, indicating that the higher order terms of the polynomial play no role. The numbers are arranged in three groups of 16 values, the first for the X coefficients, the second for the Y coefficients, the third for Z. This means that $a_{0,0} = (-5, -10, -15)$, $a_{1,0} = (0, 20, 0)$, $a_{0,1} = (0, 0, 30)$, $a_{1,1} = (0, 0, 0)$. All other a values are zero. The formula for calculating a point on this surface is $x(u, v) = \sum_{k=0}^{3} \sum_{j=0}^{3} ax_{j,k} * u^j * v^k$, or from the data, $x(u, v) = (-5, -10, -15) + (0, 20, 0) * u + (0, 0, 30) * v$. All other terms are zero. By inspection it is possible to say that this surface lies in the plane $X = -5$, since there are no terms which vary X. The limits of the surface, from the parameters, are $(-5, -10, -15)$, $(-5, 10, -15)$, $(-5, -10, 15)$ and $(-5, 10, 15)$.

An alternative formulation of the surface data might have been:

SR13 = SURF / 1, 1, 0.0, 1.0, 0.0, 1.0, 2, 2, −5.0, 0.0, 0.0, 0.0, −10.0, 20.0, 0.0, 0.0, −15.0, 0.0, 30.0, 0.0

Note, also, the way that the planar geometry has "migrated" to become a cubic. Even if the values are zero there are a few minor numerical errors which could, theoretically, destabilise the surface. However, as pointed out above, this data exchange format is really meant for freeform geometry.

```
CN19 = CONS / SR13, CV1, 2.0000000000000000E-02,                  00000238
 0.0000000000000000E+00,                                          00000239
1,                                                                00000240
-2.0000000000000000E-02, 0.0000000000000000E+00,                  00000241
  2,                                                              00000242
    0.0000000000000000E+00, 1.0000000000000000E+00,               00000243
    1.0000000000000000E+00, 0.0000000000000000E+00                00000244
```

This is a "curve on surface" definition which identifies that curve CV1 lies on surface SR13. Note that CV1 is involved in another CONS entity, CN41, lying on surface SF16. This is because the curves correspond to edges which lie between two faces.

The first two values, 0.02 and 0.0, are the global parameter limits of the curve. The next value, 1, defines how many curve segments there are in the definition. The next two values are the parameter values which limit the curve segments, in this case −0.02 and 0.0. The value 2 defines the order of the curve segment, that is, it is a linear segment, and the final four values are the coefficients of the (*u*, *v*) space polynomial corresponding to the curve. These are (0, 1) and (1, 0), which means that the curve runs from (0, 1) to (1, 1) in the surface parameter space, or (−5, −10, 15) to (−5, 10, 15), if you use the surface data definition.

```
FC43 = FACE / SR16, 1,                                            00000406
  4,                                                              00000407
    CN41, 0.0000000000000000E+00, 2.0000000000000000E-02,         00000408
    CN42, 0.0000000000000000E+00, 9.9999999999999967E-03,         00000409
    CN40, -2.0000000000000000E-02, 0.0000000000000000E+00,        00000410
    CN39, -1.0000000000000000E-02, 0.0000000000000000E+00         00000411
```

This is another interesting aspect of VDA-FS, the association of elements into face-like elements. This defines that the face lies on surface SR16, it has 1 boundary consisting of four elements. The elements are CN41, CN42, CN40 and CN39, together with their parameter values.

9.2.1.3 STEP: ISO 10303

STEP is the most modern of current data exchange standards and, my personal view, is the best choice of standard exchange at the moment. It is the most complete of the standards mentioned here because it contains experience from the

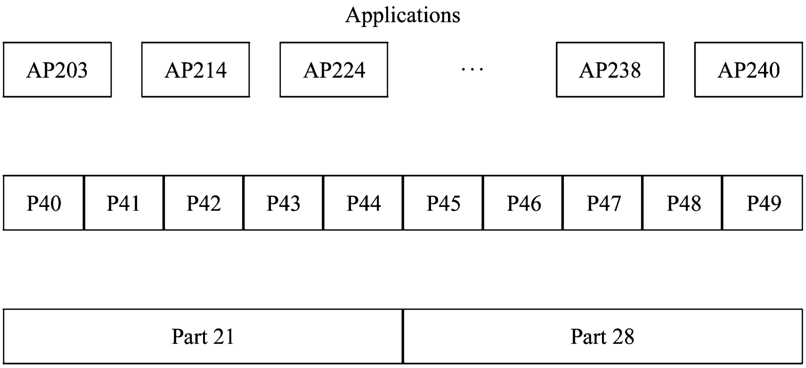


Fig. 9.16 STEP structure

others. It is also partly based on another modern standard, developed as part of a European project, CAD*I, see Schlechtendahl et al. [2].

The STEP effort is a layered structure, something like that shown in Fig. 9.16. For a considered explanation see Owen [3], for example. What follows is a personal interpretation.

At the bottom of STEP you find a number of basic modules, perhaps the two most important for CAD data exchange are Parts 21 and 28, which define the physical file structure and the XML version. If you look at the STEP example in Appendix B you see on the first line: “ISO-10303-21;” which means that this is a STEP file—ISO 10303 is the official ISO code for STEP—in Part 21 physical file format.

On the next layer up are a number of what are called “Integrated Resources” for different areas. For CAD data exchange, the geometrical and topological definitions are in Part 42, for example. Other parts define information and other common elements, such as the information about the origin of the file. The idea behind the integrated resources is to have a limited number of conceptual elements into which all high-level application concepts are decomposed. The intention is that these form common building blocks so that if you can read or write these, then you can read or write any STEP application.

At the top there are the application areas. These define the high-level elements needed for different purposes. Some of these are shown in the figure but in reality there are many more application areas and many more elements in STEP. STEP is a huge effort aimed at providing a comprehensive interface for data exchange. What is shown here is a subset because only a portion is relevant here. Some of the application protocols (APs), those shown in Fig. 9.16, are:

- AP 203—configuration controlled 3D designs of mechanical parts and assemblies
- AP 214—core data for automotive mechanical design processes
- AP 224—mechanical product definition for process planning using machining features

- AP 238—computer numerical controllers
- AP 240—process plans for machined products

The Application Protocols define what is called an ARM, Application Reference Model, which is a definition of what is needed in the application. This is then defined formally in terms of the Integrated Resources which are what are output.

The aim here is not to provide a complete and accurate overview of STEP but to present a brief outline to help understand STEP file output. It is worth understanding STEP because it covers such a wide area. There are many gaps which are being slowly filled. One of these is the history trees, which has recently been defined in STEP but will take time to come into CAD file exchange. Assemblies are passed across, but assembly constraints seem to be lost.

Looking at the file examples in Appendix B, you have at the start:

Example 1

```
ISO-10303-21;
HEADER;
FILE_DESCRIPTION(('CATIA V5 STEP Exchange'),'2;1');

FILE_NAME('\\\\Icap2ksrv\\licp\\LICP-staff\\stroud\\doxtrans\\book2
\\BLK102030.stp','2007-11-14T06:04:41+00:00',('none'),('none'),'CATIA Version 5
Release 15 (IN-10)','CATIA V5 STEP AP203','none');

FILE_SCHEMA(('CONFIG_CONTROL_DESIGN'));

ENDSEC;
/* file written by CATIA V5R15 */
DATA;
```

Example 2

```
ISO-10303-21;
HEADER;
FILE_DESCRIPTION (( 'STEP AP203' ),
    '1' );
FILE_NAME ('blk102030b.STEP',
    '2007-11-14T07:10:46',
    ( 'LICPPC78' ),
    ( 'EPFL' ),
    'SwSTEP 2.0',
    'SolidWorks 2006104',
    '' );
FILE_SCHEMA (( 'CONFIG_CONTROL_DESIGN' ));
ENDSEC;

DATA;
```

The reason for giving two examples is so that you can compare different styles of outputting the same information. As you can see, the layout is different, although the content is basically the same. In the CATIA file, the elements are bundled together by type, whereas in the Solidworks file the elements are arranged in line-number order. For the computer this should not make a difference, of course, but for a human reader these two layouts have different advantages and disadvantages. In the CATIA file it is easy to find related elements of the same type, whereas in the Solidworks file it is easy to find given line numbers. The files are not, though, intended for humans to read directly, but it helps to be able to identify some of the information.

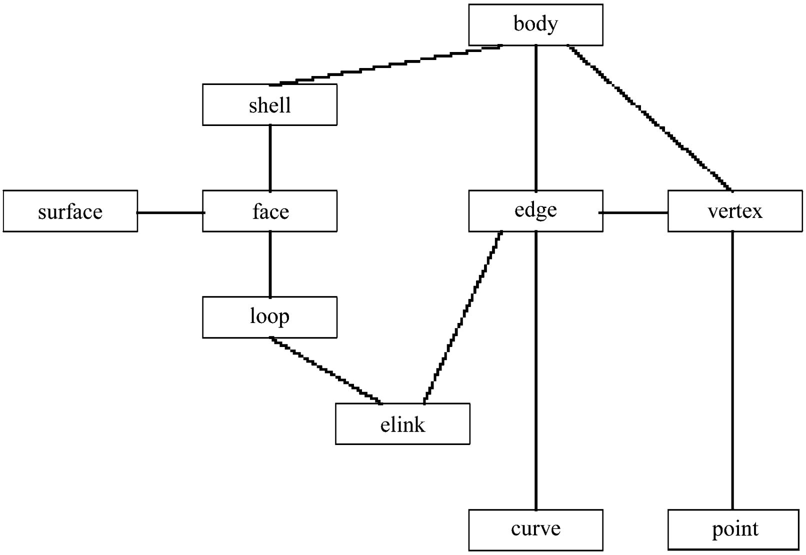


Fig. 9.17 Simple boundary representation data structure

The elements in the file are like the elements of a general Boundary Representation data structure. This means that the “trick” of reading and writing STEP files is to match those elements with the elements in your data structure. This is similar to what was suggested in Fig. 9.13. Examine the data structure from Fig. 9.1, reproduced in Fig. 9.17.

Compare this with a similar diagram based on the STEP file data structure from Part 42, as shown in Fig. 9.18. Of course, the STEP file structure shown in the figure is a gross over-simplification, because there are many more entities not shown, but there is a match between these main elements and the data structure elements of a normal B-Rep structure.

The correspondences can be classified roughly as in Table 9.1

Looking at the file, you have the line:

```
#91 = MANIFOLD_SOLID_BREP ( 'NONE', #88 ) ;
```

which defines a body. Actually, there are other entities associated with the body at a higher level, but these are ignored here. From the body line you see a reference to line number 88, (#88), which is a “closed_shell” entity:

```
#88 = CLOSED_SHELL ( 'NONE', ( #101, #20, #106, #23, #93, #85 ) ) ;
```

If you continue following the links, you find that the shell refers to a list of faces. Note, though, that this list is dynamic. The usual practice when creating discfiles is to keep the entities of fixed length. Another complication is that STEP, as with some other files, contains forward references. This means that one entity may refer to others which have not yet been created. In the shell example, above,

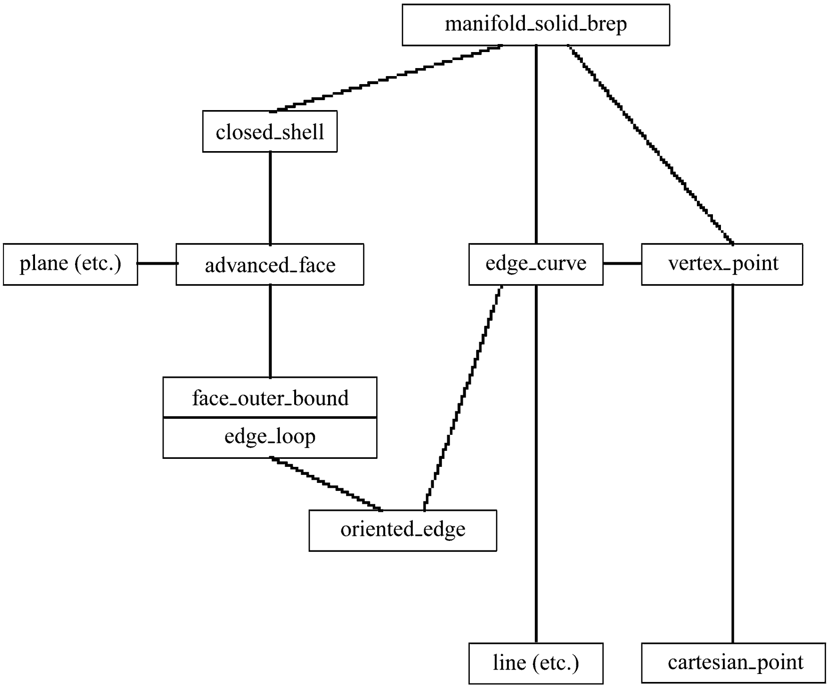


Fig. 9.18 Simple boundary representation data structure

Table 9.1 Comparison between STEP entities and B-rep structure

STEP entity	B-rep entity	Comments
manifold_solid_brep	body	There are more types of "body" discriminated in STEP. These can all be mapped to body
closed_shell	shell	Again, several types of STEP shell can be mapped to shell
advanced_face	face	This is a more-or-less straightforward correspondence
face_outer_bound, edge_loop	loop	The loop information is spread over two entities in STEP. The first defines that this is an outer bound, the second that it is an edge sequence and gives the edges
oriented_edge	elink	The STEP entity gives the orientation within the edge_loop
edge_curve	edge	The "edge_curve" name indicates that the edge is a portion of a curve
vertex_point	vertex	As with the edge, the name indicates that the vertex lies at a point in space
?	surface	There is no direct surface entity but rather a "superclass" of different elementary types
?	curve	Again, curves are divided into different types which have to be read and interpreted

the closed_shell on line 88 refers to line numbers 101, 106 and 93, which are face references which have to be created as empty slots.

This presentation is very simplified, but space does not permit a very detailed analysis of STEP. In reality, when building a STEP interface, it is necessary to work through the detailed descriptions of all the integrated resources and match them to elements in the CAD system structure. Where there is no correspondence, new entities should be created to preserve the information. The STEP integrated resource descriptions are good and comprehensive, giving a clear picture of the entities in the files. They are not reproduced here because of copyright restrictions, but are mainly because the principle relevance would be for producing applications, in which case it is worth buying the official documents. The intention here is to produce a short overview to give an idea of the information and its meaning.

It is necessary, though, to comment on some shortcomings of the STEP standard. First, the history tree is not (yet) communicated. Actually, a STEP standard for the "construction history" does exist, although it is very new, see Sect. 9.3. However, it takes time for these to become part of CAD systems and it may turn out that there have to be modifications to accommodate all the facilities in CAD systems. Another shortcoming is the lack of constraint information in assemblies. This, too, will probably appear at some point.

On the positive side it should be said that the STEP standard is the best data exchange standard that exists at the moment. The optimum is to exchange data between CAD systems of the same company and with the same version, but this is not always possible. STEP offers a solution acceptable to many CAD systems and allows transfer between systems of the same type but with different versions.

9.2.2 Common Non-Standards

These non-standard exchange methods are widely used and are what is termed "de facto" standards. In general, beware of these because the standardisation process involves comprehensive reviews and so standard exchange methods are more stable. However, since these non-standards are used then it is necessary to explain what they do and their properties.

9.2.2.1 STL

The STL information is summarised in Fig. 9.19. The representation is made up of a number of triangular facets. Each facet has three corner points and the face normal.

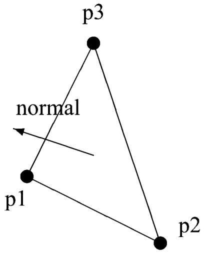
The complete file starts with the declaration:

solid simple

and ends with the declaration:

endsolid simple

Fig. 9.19 STL facet information



In between there is a series of facet definitions which define the geometry. Each facet has the form:

```
facet normal −1.000000 0.000000 0.000000
    outer loop
       vertex −30.000000 25.000000 40.000000
       vertex −30.000000 25.000000 0.000000
       vertex −30.000000 −25.000000 40.000000
    endloop
endfacet
```

Strictly speaking the vertex coordinates are supposed to be all positive, but this restriction, possibly originally for computing reasons, has been relaxed.

STL seems to be a graphics format that was used for communicating with stand-alone graphics systems and has since been used for Rapid Prototyping and communicating measured point data. Beware, as a communications format STL is a disaster. To be fair, it was useful for a period when geometric processing power and knowledge were limited. For rapid prototyping it is necessary to intersect a model with a plane. Intersecting an STL model with a plane is trivial, computationally, and hence STL was useful for a while. However, times have changed and it would be easy now to use an exact model instead of an STL model. Also, STL is an approximative representation and so it is necessary to set the approximation parameters correctly to get the desired precision. Because of the fragmentation of curved surfaces, STL files get large because the format is verbose. Note, several suggestions have been made to replace STL and there is also a modern standard in preparation for STEP-NC (ISO 14649).

The triangulation method tends to be arbitrary. Because of the historical links with graphics it is typical to use graphics triangulation for output in STL form. Figure 7.8 shows a face triangulation. If you examine the figure you can see that some of the triangles are long and thin and, in general, there is little control over the shape of the triangles. Figure 7.9 shows the types of STL creation parameter. The most common is the chord height tolerance, which determines the maximum allowable discrepancy between the real geometry and the facetted approximation. Another control parameter is the minimum angle parameter, which is for creating more uniform triangles. This should, of course, be 60° or less, though 60° is an ideal. The edge length parameter determines the maximum edge length, and also has an effect of making the triangles more uniform. However, you may not find all these if you want to output STL. Note, also, the difficulty in knowing what the parameter settings should be.

Another problem with the STL format is that the facets contain no information about neighbouring facets; every facet is independent. This not only creates redundant information, because the corner points are repeated, but means that is more difficult to recreate a solid by joining facets. See Mäkelä and Dolenc [4] or Stroud [5] if you really want to know how to recreate a solid. If your CAD system can recreate a solid then you can check for holes. Incomplete STL files, or files with other mismatch problems have caused a lot of problems for rapid prototyping. Ideally, if you want to communicate using STL then you should write a file and then read it back to check for problems before sending the STL file.

STL is sometimes used to communicate measured data. This is data generated by measuring physical objects. This consists of a sequence of points, but can be delivered as STL by associating neighbouring points as facets. However, note again the multiplicity of the point data communicated using STL. If measured data is communicated using STL it is necessary to join the facets in order to have measured points once for surface fitting.

9.2.2.2 VRML

VRML is better than STL in the sense that it can communicate a unique set of corner points. However, it is sometimes used to communicate separated facets, in the same way that STL does.

There are several pieces of information concerning graphics properties such as colour and transparency. The geometric information in the file looks something like:

```
coord Coordinate {
    point [
        −5 −10 15,
        −5 −10 −15,
        −5 10 −15,
        −5 10 15,
        ]
    }
coordIndex [
2,1,3,-1,
1,0,3,-1,
]
```

The first elements are the corner "point" data. Actually these would correspond to vertices in a boundary representation model. Each line has a triple of *x*, *y* and *z* coordinates. These are implicitly numbered as 0 to 3, in the case shown.

The faces are given in the "coordIndex" part as a series of vertex references, ending with "−1". In the example above there are two triangular faces, one with the corner points at (−5, 10, −15) (−5, −10, −15) (−5, 10, 15) and the second with corner points (−5, −10, −15) (−5, −10, 15) (−5, 10, 15). Edges are created between neighbouring vertex points. When creating an object it is necessary to check whether an edge has already been created between the vertices. If so, this creates a link between two neighbouring faces.

9.2.2.3 Point Formats

Point data is, perhaps, the hardest type of data to handle because the elements are not linked into units. Point data often comes from measured objects for so-called "Reverse Engineering", for recreating CAD models from physical objects. Sometimes this data is communicated using STL, which does give a structure, but at other times the point data is communicated in raw form.

There are several types of point data format. The simplest type is as *X*, *Y*, *Z* triples in ASCII form. Another type of data format, the RIS format, involves a start and end point in (*X*, *Y*, *Z*) format and a series of *Z* values between these to create profiles.

The problem with point data files is, as stated above, to associate the points with each other. It is important in reverse engineering to separate these points into groups corresponding to surface portions (called "segmentation") and then to fit surfaces to these. It is not intended to describe this in detail here, see Besl [6] or Martin and Marshall [7], for example, for basic references. This is far from easy, so it can be worth examining the data to see if it has common elements, such as constant *X*, *Y* or *Z* value. This indicates a scanning orientation and means that points can be associated into rows. One trick that was done for some students at the EPFL (Ecole Polytechnique Fédérale de Lausanne) was to preprocess a points data file so that the points in a row created a spline. This was done by creating fake macro files for the system used (I-DEAS). These splines were then used to create surfaces manually using lofting. This, though, does not get round the problem of segmentation but made it easier to deal with the points.

9.2.3 Other Formats

There are a number of other formats for data exchange which are used, but which are not covered above. These are formats which come from commercial systems and hence can be considered proprietary. To avoid infringing any copyright or intellectual property regulations there is no attempt to explain them here. Remember that, while the formats are often good, they can change as new elements are added to the data structure or as existing items are changed.

This makes these formats more volatile than standard formats, as already explained above.

9.2.3.1 DXF Files

DXF is a format that comes from Autocad. It is another format which is good for communicating drawings but has some three dimensional information as well.

9.2.3.2 SAT Files

SAT files are disc files of the system ACIS. There is also a binary format file with the extension “SAB”. The first line of the file contains a version number which can be checked to see if the file is readable. The entities of the object represented by the file are written in a long list. Each line has a keyword, representing the type of entity in the file, followed by a set of numbers representing the pointer and numeric fields of each particular entity.

9.2.3.3 Parasolid Files

The extension for Parasolid files is “x_t” or “x_b” depending on whether they are text or binary files. Parasolid, like ACIS, is a modelling kernel which is used by several commercial companies as a basis for their software. This makes the Parasolid files readable within a family of software system, but the files are again sensitive to version control.

9.2.3.4 CGR Files

CGR stands for Computer GRaphics files which, while there may be three dimensional data, are intended for computer graphics, not for geometric objects.

9.3 Functional Interfaces

Functional interfaces do not communicate the final shape but the way it is made.

9.3.1 DJINN

Another method of communication is to communicate operations directly. A way of doing this is to use the DJINN standard, which is a standard for defining the functional interface for modelling systems. What this means in effect is that implementors write interfaces which use modelling kernels in standard ways using standard calls, then the calls themselves become a potential means of communication.

DJINN was written by Cecil Armstrong, Adrian Bowyer, Stephen Cameron, Jonathan Corney, Graham Jared, Ralph Martin, Alan Middleditch, Malcolm Sabin, Jonathan Salmon, a sort-of All-Stars team of talent in British Geometric Modelling. Between these people there is a vast amount of experience of geometric and solid modelling, of various types, which means that the work is relevant and of high quality. Because of the breadth of knowledge, DJINN is an interface which is not technology dependent but provides access to the functionality of a model. The aim was to create a standard interface which could be used to share applications between researchers. This would also mean that it might be possible to demonstrate research applications in commercial environments, provided that software suppliers adopt the DJINN way of working. It might also mean that application software could be distributed directly to companies and mean closer collaboration between researchers and industry, but these developments, as far as I know, have not yet been realised.

DJINN consists of a proposed set of basic calls to a modelling system on top of which an application can be built. It also has a standardised set of objects which can be handled by the interface.

The calls can be used as a part creation history, but could equally well be used as the basis of a macro-language or command interpreter interface. DJINN itself is more comprehensive than a simple part creation history, containing many extra facilities needed for programming standardisation.

There is, of course, a conflict between the aims of DJINN and commercial considerations. For a commercial developer it is more advantageous if code is tied closely to their system. Application code migration does occur, although more rarely, but system independence is not really a main requirement for commercial developers. DJINN is of much more interest to researchers in the academic world. Having a DJINN interface in a CAD system would allow researchers to add new functions to a CAD system, such as feature recognition, analysis or manufacturing code. However, since DJINN has not (yet?) been taken up by commercial companies it is only mentioned here rather than given the fuller treatment it deserves.

9.3.2 The STEP Construction History

The STEP construction history standard is aimed at communicating the way that a designer has created an object. This means that the functionality and the corresponding parameters of CAD systems are standardised and recorded in a standardised way. You then communicate the lists of these used to create an object so that another CAD system can use the list to create an object.

The advantage of doing this is obvious. Just writing out a complex part using STEP 203 or IGES and then reading it back shows you the problem. All the convenient operation parameters that are accessible for modification in the history tree of a particular CAD system are missing, with only the final shape communicated. A problem, though, concerns what happens when new operations appear that are not covered by the standard. Another problem is when one CAD system wants to have different parameters to those of another. This is a little like what happened with earlier attempts to define shape data exchange methods, where different modelling methods, different geometric methods and different interests competed. Modern shape data exchange has become possible because shape modelling matured. The question is whether construction history is sufficiently mature to allow for successful construction history exchange. For a number of basic operations, yes it is, but possibly not for newer application-based operations. This is a topic to watch, not necessarily a solved topic.

9.4 Chapter Summary

This chapter describes ways for communicating model data via disc files. Local disc files are usually more efficient in storage terms but are more volatile than standards. Local discfiles are more-or-less simple copies of the memory data-structure for a model, with logical pointers instead of real pointers.

The alternative to commercial system discfiles are standards, or neutral files. Standards are good not only for communicating between different systems but also between different versions of the same CAD system. Standards are also more useful for archiving, a somewhat neglected subject, but important for products which may still be active after 50 years.

Several “de facto” standards also exist. The term “de facto” meaning that, though they are widely used, they are not standards. These should also be labelled “Beware”, not because you shouldn’t use them but that you should be wary of their shortcomings.

Finally, two approaches to functional standardisations were mentioned, DJINN and the STEP construction history standard.

9.5 Data Exchange Exercises

9.5.1 Simple Save 1

Create a cylinder and save it. Identify the different possibilities offered. Note the formats which are local to the CAD system and which alternatives are offered. Note, also, whether the formats are offered in TEXT or BINARY mode.

9.5.2 Simple Save 2

Create and save in STEP format a cylinder, a sphere and a torus. Open the files with a text editor and compare the different geometric entities.

9.5.3 STEP Export and Import

Create a 100 × 100 × 50 block centred about the origin and create a cylindrical hole, radius 10, through it. Export it with STEP and then reimport it to your CAD system. Note if the construction history is also imported or whether the object appears simply as a final shape. This is just to check the stage of implementation of the construction history files.

9.5.4 STL Save 1

Create a cylinder, radius 20, and height 80, say, and subtract from it a cylinder with the same axis, radius 10 and height 100, say, to create a tube. Export this using STL. Export this in STL format with chord height tolerance 0.1, 0.01, 0.001. Note the differences in file sizes. Does your system allow you any other control parameters? Read the STL file back into the CAD system, if you can. Note how the facets approximating the outer side of the cylinder are all inside the real object while the facets surrounding the hole are all outside the real object. This is to do with the fact that only the corner point positions lie on the object.

9.5.5 STL Save 2

Repeat the previous exercise but using a sphere, radius 20, instead of a cylinder. Note the way that the file sizes increase and explain the difference.

9.5.6 Chained Save

This is better if you can save using different CAD systems but is worth trying even with one system.

Create an object with simple geometry, like that shown in Fig. 9.20. This object has planar, cylindrical and spherical surfaces. Save the object in STEP format. Reopen the STEP file and export it, or save it, in IGES format. Reopen the IGES

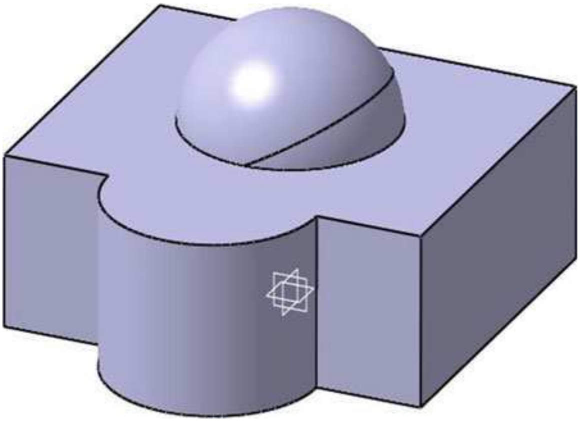


Fig. 9.20 Simple object modelled for chained data exchange

file and save the object in a second STEP file. Now compare the first and second STEP files with a file comparison utility and examine the differences.

The intention is not to criticise any particular implementation, just to illustrate that you cannot expect the model to stay exactly the same through multiple data exchange steps. This also means that techniques such as feature recognition, see Chap. 10 are important to relate shapes.

References

1. US PRO: IGES—An American National Standard, Initial Graphics Exchange Specification, IGES 5.3. ANS US PRO/IPO-100-1996, ANSI Approved, 23 September 1996
2. Schlechtendahl, E.G. (ed.): Specification of a CAD*I Neutral File for CAD Geometry Wireframes, Surfaces, Solids, version 3.3. Springer, Heidelberg, ISBN 3-540-50392-7 (1988)
3. Owen, J.: STEP An Introduction. Information Geometers, Winchester (1993)
4. Mäkelä, I., Dolenc, A.: Some efficient procedures for correcting triangulated models. In: Solid Freeform Fabrication Proceedings, pp. 126–134, September (1993)
5. Stroud, I.A.: Boundary Representation Modelling Techniques. Springer, Heidelberg (2006)
6. Besl, P.J.: Surfaces in Range Image Understanding. Springer, Heidelberg, ISBN 0-387-96773-7 (1986)
7. Marshall, A.D., Martin, R.R.: Computer Vision, Models and Inspection. World Scientific Publishing, Singapore, ISBN 9810207727 (1992)

Chapter 10
Features

Features are localised elements of a model which are interesting for some reason. This is a personal definition, but there is no clear definition of features that everyone can agree on. Features became interesting as a topic during the 1980s and are still the subject of research. Features are important because they transcend the representation method and have a high-level meaning which is useful for applications. Some examples of features are shown in Fig. 10.1.

An object is then made up of a basic shape with a lot of features, as in the example in Fig. 10.2.

Note that there are two general categories of feature: intrusive features and extrusive features. Intrusive features are where material is removed, extrusive features are where material has been added. The features from Fig. 10.1 are classified into groups in Table 10.1.

These are only a few feature examples, not an exhaustive list, just to illustrate the categories.

HEALTH WARNING

If someone tells you that they understand
features and can tell you about them, the
first thing to do is go into sceptical mode.

I understand features and can tell you about them.

You should now be in sceptical mode. What follows in this chapter is an opinion, more-or-less informed, about some aspects of features. However, remember that many people talk glibly about features and not all of these people know what they are talking about. Figure 10.3 illustrates one credibility criterion if someone, especially a salesperson, informs you that their CAD system is a feature-based system.

I. Stroud and H. Nagy, *Solid Modelling and CAD Systems*,
DOI: 10.1007/978-0-85729-259-9_10, © Springer-Verlag London Limited 2011

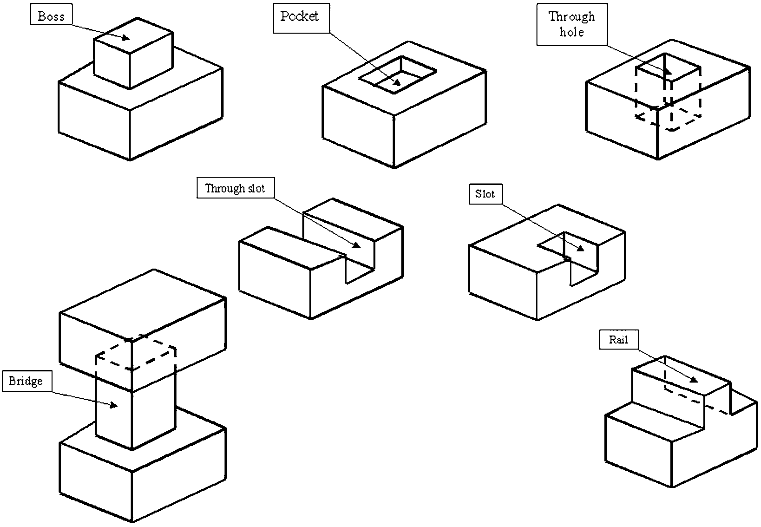


Fig. 10.1 Some examples of features

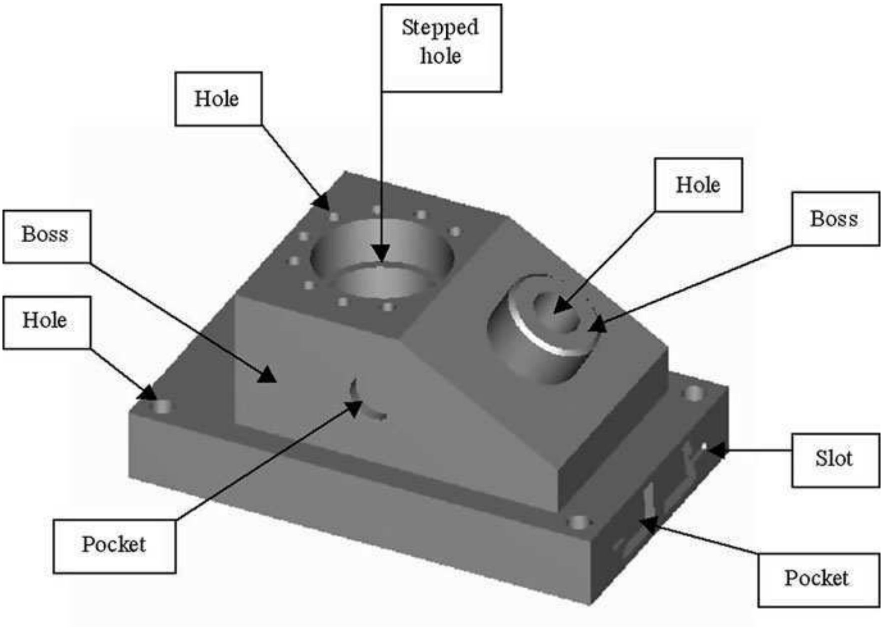


Fig. 10.2 Some examples of features in an object

In case you do not know the idiom, in English there is a way of expressing disbelief by saying "And pigs might fly". So, if someone does tell you that a system is feature-based, look around for flying pigs. It may help if the person

Table 10.1 Intrusive and extrusive feature categories

Intrusive features	Extrusive features
Pocket	Boss
Through slot	Rail
Slot	
Through hole	
Bridge	

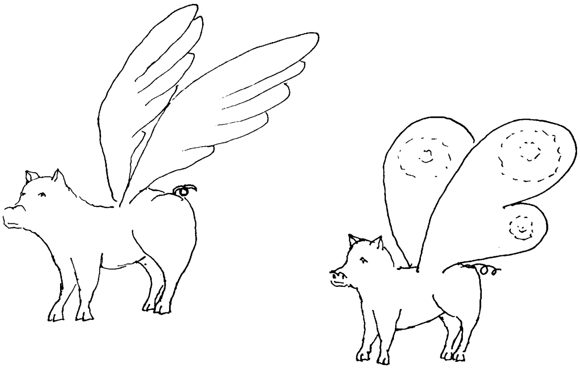


Fig. 10.3 Credibility criterion

telling you is a flying pig. If not, check the system. This is not to say that feature-based systems do not, or will not, exist, just that the term has been used too loosely in the past to be confident as to what someone actually means.

My personal criteria for a system to be feature-based is that the operations produce the feature corresponding to the operation name, and that the identity and integrity of these features are preserved during modelling. The operation names are preserved in a list and this list should describe the part, providing the system is, by my definition, feature-based. Note, though, that this is difficult to do.

I can illustrate this using CATIA v5, although you can find similar things in other systems. In CATIA there are three operations which are named: “Pocket”, “Groove” and “Hole”. “Pocket” is actually used to mean “linear extrusion and subtraction”.“Groove” means “circular extrusion and subtraction”. The most appropriately named is the “Hole” operation, which, for example, might rotate a profile about an axis and then subtract it from an object. To illustrate the problem of disparity between the description of an object from the history tree and the shape of the part, consider the following features:

In Fig. 10.4 the hole is made by sweeping a multiple contour, a square outer shape with a round inner contour. The “history” says that there is only an extrusion, but, as is clear, the object is a block with circular hole.

In Fig. 10.5 the hole is made by creating a rectangular block, sketching a circular shape on the top face and then sweeping this downwards as a “pocket”. Although, in some sense, this may be considered as a pocket, it is more appropriate to think of it as a circular hole.

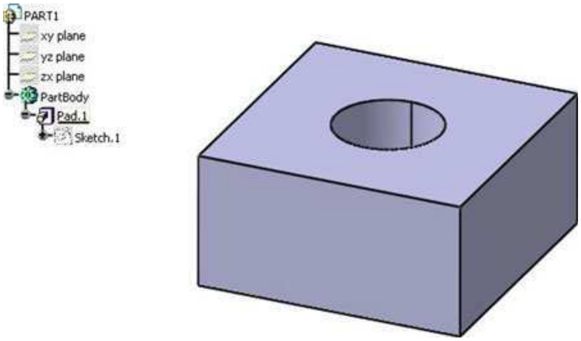


Fig. 10.4 Hole made from sweeping a multiple contour

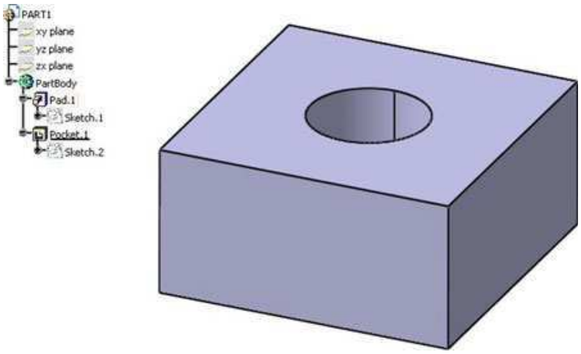


Fig. 10.5 Hole made from sweeping a circular contour as a pocket

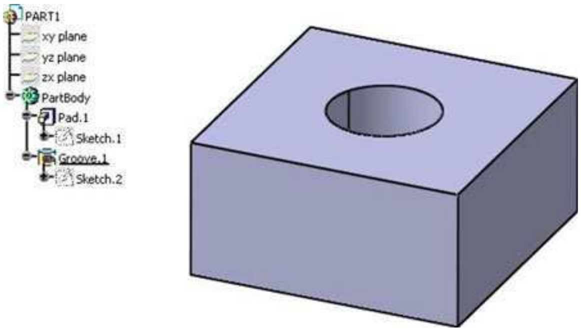


Fig. 10.6 Hole made as a “groove”

In Fig. 10.6 the hole is made by creating a long vertical rectangle which is then rotated round one of its long sides and subtracted from the block. The “history” says that the feature is a groove, but it is still a hole.

In Fig. 10.7 the hole is made by creating a rectangular block and then a hole through it with the hole operation. The description and the part shape correspond.

Fig. 10.7 Hole made as a hole

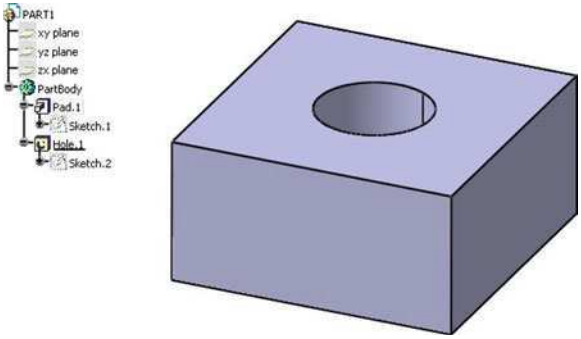


Fig. 10.8 Pocket which is a closed pocket

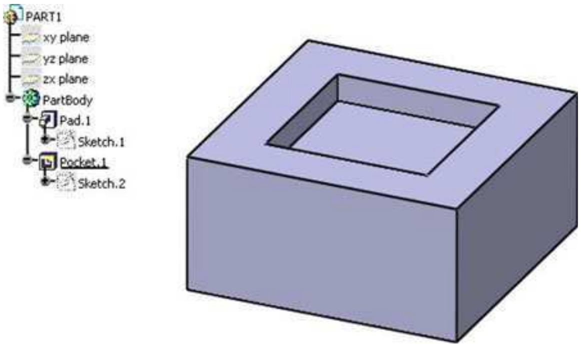
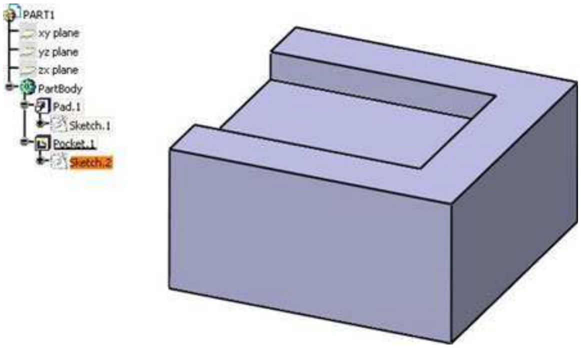


Fig. 10.9 Pocket which is an open pocket



What these examples are supposed to show is that you can create shapes in several different ways and that the history description does not necessarily correspond to what you would like to know about the shape. Now consider the next set of features.

In Fig. 10.8 the pocket really is a pocket. This is the classic shape, a cut out totally enclosed within a face. This is called a "closed pocket". If the pocket shape cuts through the boundary of the face the shape becomes an "open pocket", as shown in Fig. 10.9.

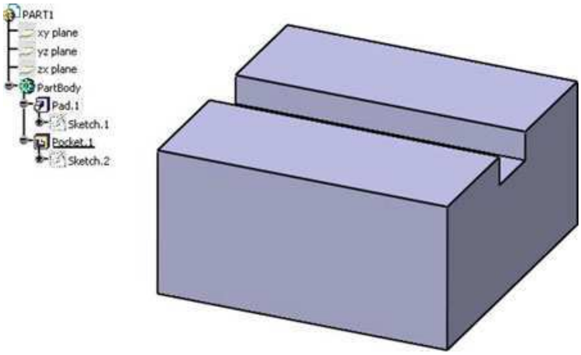


Fig. 10.10 Pocket which is a slot

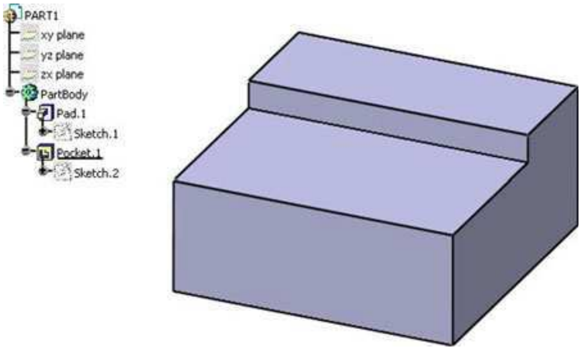


Fig. 10.11 Pocket which is a step

There is an important difference between the closed pocket and the open pocket for manufacturing. This is not recorded in the description, but the way that the tool enters the material is different.

In Fig. 10.10 the pocket is actually a slot. Again, a slot has different implications for manufacturing, in terms of how the tool enters the material as well as the size of tool used to make the slot.

A step, Fig. 10.11, is yet another feature which differs from a pocket in terms of manufacturing. It is also different from an open pocket because the feature is open on three sides.

In Fig. 10.12 the feature is even further from being a pocket. You might, with some systems, even be able to create a "pocket" which is totally enclosed in material, which ought to be described as a cavity.

What these examples are intended to show is that there is no control by the CAD system over the way that the operations are applied, nor the results. There are other ways to show that the shape elements created, even if they really do correspond to the feature description in the history tree, can be corrupted.

In Fig. 10.13 there is a pocket with an extrusion, which is as you might expect.

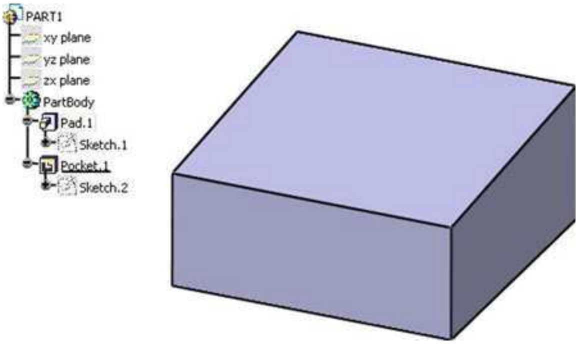


Fig. 10.12 Pocket which is a facing-off

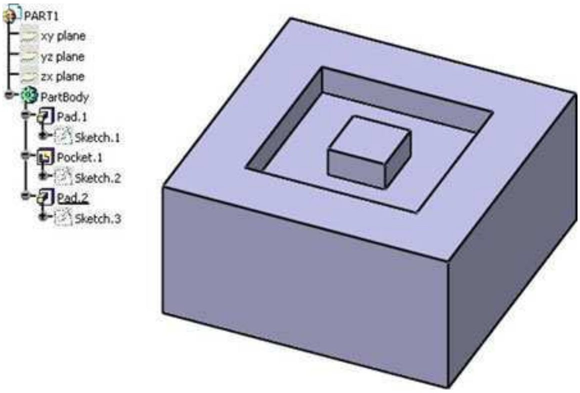


Fig. 10.13 Pocket and an extrusion—version 1

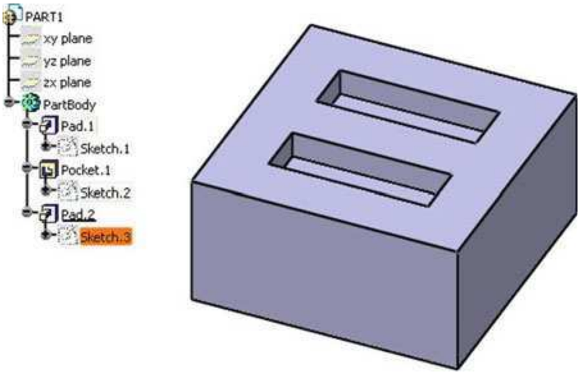


Fig. 10.14 Pocket and an extrusion—version 2

With a different extrusion, when the shape being extruded crosses the pocket boundary, the pocket turns into two pockets, Fig. 10.14. Ideally the CAD system should notify you that the pocket, supposedly created as a feature, has been corrupted.

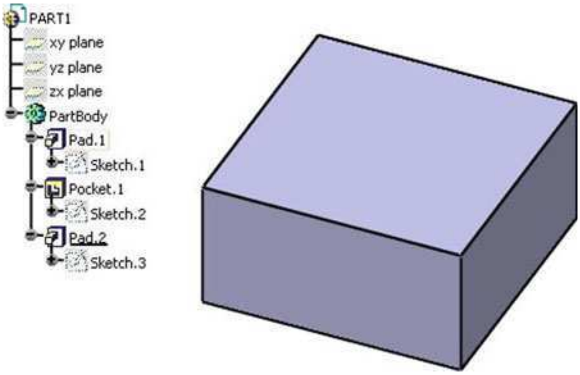


Fig. 10.15 Pocket and an extrusion—version 3

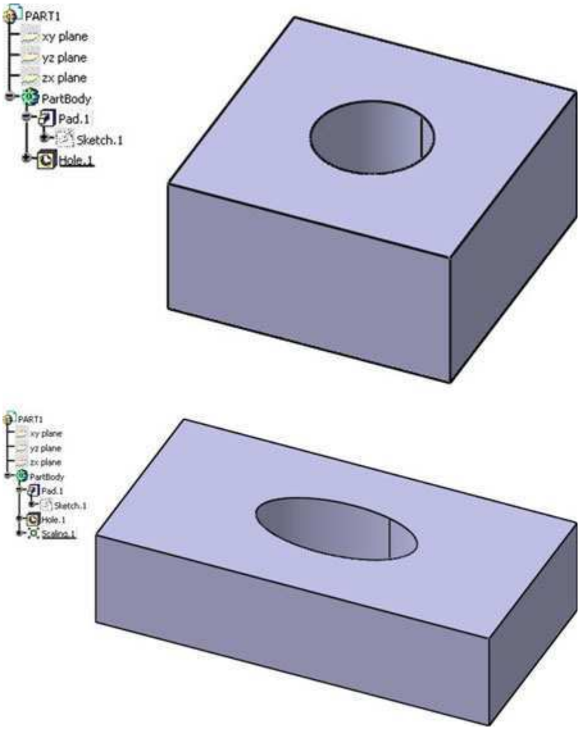


Fig. 10.16 Elliptical hole

If the shape being extruded surrounds the whole pocket disappears. Figure 10.15. People sometimes do this to eradicate a feature, but the net effect is a conflict of information between the finished shape and the description.

Figure 10.16 shows a different example of the way in which the feature origin is ignored. The top of the figure shows the original object, a block with a round, threaded hole. Underneath the object has been scaled unevenly, an unusual operation in CAD systems, and the threaded hole is now elliptical, presenting interesting manufacturing and functional problems. The scaling operation could

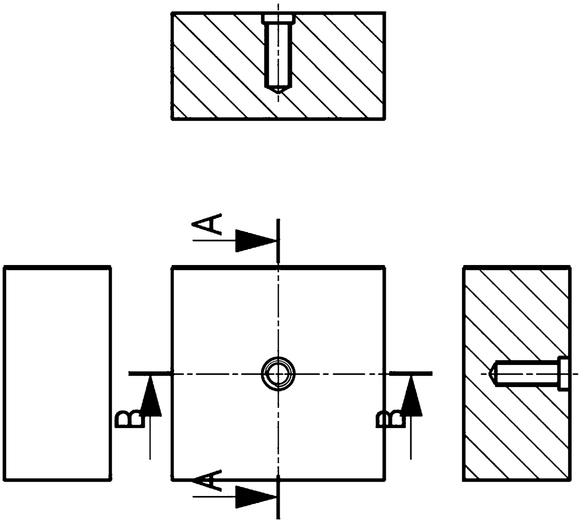


Fig. 10.17 Pointless hole?

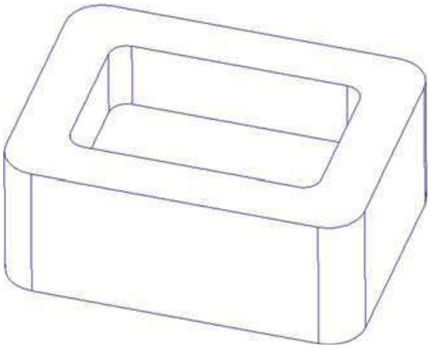


Fig. 10.18 Simple shape with rounded corners

have simply scaled the position of the hole centre line and the depth, not the final shape.

Finally, a different type of problem not related to erroneous shape. Figure 10.17 shows a slightly more subtle example of a problem hole created using the hole operation. The hole is countersunk to allow a bolt to be inserted into the object without protruding above the surface. The conical end option has been used to allow for tool shape during manufacturing. It is also marked as being threaded to allow a bolt to be screwed down directly. It might be considered to be a perfectly reasonable example of the use of the hole making operation, except that there seems no reason for the hole to be there. If the hole is countersunk then any bolt screwed into it will not hold anything externally. In this case the bolt would just disappear into the material with no functional effect, so why should it be there? This is a more difficult question, and there are many reasons for having holes, but many holes are there for connection reasons. It would be useful if such holes could be made in two objects at the same time, so that the connection would be apparent.

This is supposed to show that, while the use of feature operations can be useful for building up a part description, it is necessary to perform information maintenance to avoid conflicts. Simply giving operations feature names is not enough to make a feature-based modeller. For you, it is important to understand the limitations of the techniques used in CAD systems and so make sure that the information you create as accurate as possible. This is a problem for teaching CAD because very often the final shape has to be given, so students may take shortcuts to get to the final result rather than create the shape in stages using design logic. Take for example, the simple shape shown in Fig. 10.18.

It may be convenient to define a rounded shape, as shown in Fig. 10.19a, extrude it and then define another rounded shape, Fig. 10.19b, which is extruded downwards to create the pocket.

However, depending on what the rounded corners are for, it may be more logical to create a square shape, as shown in Fig. 10.20 and blend the edges. It is difficult to teach students to use CAD systems logically to provide meaningful information that is usable later if they are given the final shape to make.

What follows is about what I know of the state of the art of features and their use. This is intended for you to use to assess how advanced is your CAD system. Features are important for understanding shape at a high-level, so understanding what they are is an advantage in communicating between applications.

Fig. 10.19 Creating a pocketed shape

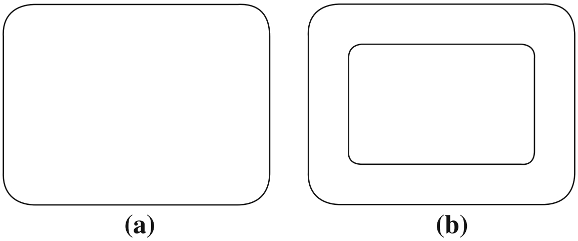
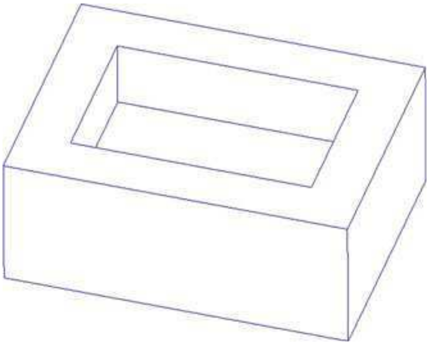


Fig. 10.20 Basic pocketed shape with square corners



10.1 Overview

The first thing to say is that the subject of features and feature work is enormous and has been the subject of books just about features [1]. What follows is a brief selection trying to give an idea of feature work rather than be a comprehensive survey.

The first published work on features in geometric modelling is perhaps that by Kyprianou [2], which is a classic piece of work on what is called “feature recognition”. However, Kyprianou was not alone in working on features and several variations appeared. Generally, the work fell into two categories:

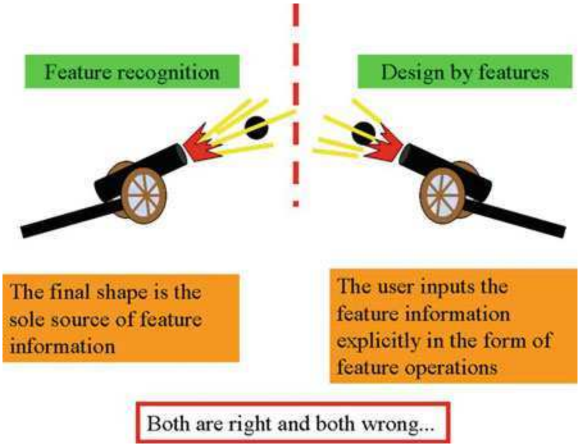
1. Feature recognition.
2. Design by features.

Feature recognition assumes that the final shape is the only source of information. Another technique, called “design-by-features” assumes that the feature information is added to the model explicitly by the designer. There was some disagreement over the use of these. This is summarised in Fig. 10.21. In fact, these techniques are not mutually exclusive and it is beneficial to use them in combination.

Feature recognition is an important technique, both for direct recognition and for verification of feature information acquired elsewhere. Feature recognition is also important for looking at shapes from the points of view of different applications. Features created for one reason, say ribs as strengtheners, may need to be manufactured as pocket sides.

Design by features certainly had, and has, an important role in promoting feature-based application development. This is a sort-of “what can I do if I have the feature information?” question. Feature information can be introduced realistically for specific purposes, especially in finishing operations. Although there are improvements that it would be desirable to make, the hole-making operation in CATIA is a good

Fig. 10.21 The feature war


Feature recognition
Design by features
The final shape is the sole source of feature information
The user inputs the feature information explicitly in the form of feature operations
Both are right and both wrong...

example of this. The operation seems really intended to add holes in a quick and easy fashion to an almost-finished design. Note, though, that not all features introduced explicitly are necessarily useful for another application. More will be said later, but CATIA's rib-operation is a good example of this. The rib is certainly a valid feature, but would not necessarily be useful as a rib for manufacturing.

A final general observation before describing these techniques in more detail is that, whereas most systems can more-or-less handle isolated features, interacting features cause problems. This is true of both feature recognition as well as design-by-features. Some methods and research that you see seem to work well but are based on isolated features.

10.2 Feature Recognition

As stated above, in pure feature recognition the final object is taken as the sole source of information.

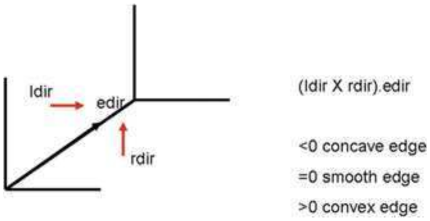
10.2.1 Kyprianou's Method

The first method published was that by Kyprianou [2], who worked on shape classification for databases. The aim was to generate a classification code for a shape so that it could be compared with shapes in a company parts database to check for similarity. The idea is that if a similar part to that just designed exists then it may be possible to adapt the existing part instead of generating a new part and hence reduce the parts needed by a company.

Kyprianou's method was based on classification of edges as concave or convex, thence face classification and finally feature recognition of linked facesets using shape grammars.

Edge classification as concave or convex can be done using a simple technique, illustrated in Fig. 10.22. Each edge in a boundary representation model is oriented and has a "right" face, a "left" face and a direction. The face directions are taken as the normal directions to the faces at the point on the edge being tested. At some point or points on the edge, the triple product ($ldir \times rdir$) · $edir$ is calculated, where "*ldir*" is the direction of the left face, "*rdir*" is the direction of the right

Fig. 10.22 Concave edge definition



face and "*edir*" is the edge direction. This gives you a number which is negative if the edge is concave or positive if the edge is convex at the point. The value can also be zero if the left and right face directions are parallel, for example, which happens with blended edges. Such edges are called "smooth" edges.

Note, though, that edges may not have the same classification along their whole length, as pointed out by Alan Smith in work on the Cambridge University BUILD system in which Kyprianou's work was embedded. For this reason the concavity test is usually applied at several points along the edge, the start and end as well as the middle, for example.

After the edges have been classified, all the faces in the object are classified as either "primary" or "secondary". A primary face is one which has one or more hole-loops and/or is bounded by a mixture of concave and convex edges. A secondary face is anything else, i.e. is a face with only one boundary which is made up only of convex or only of concave edges. The primary faces lie on the boundary of the feature and the arrangements of these faces give the features.

A simple example is given in Fig. 10.23. The four edges which make up the base of the boss are all concave. In addition, note that the face marked "f2" has a double boundary, hence contains a "hole loop".

The analysis for all the faces is given in Table 10.2.

The term "hl" denotes a hole-loop. The term "cx" denotes a convex edge and "cv" denotes a concave edge. Face 2 is classified as a primary face because it has a hole loop as well as four concave and four convex edges in its boundaries. Faces f8, f9, f10 and f11 all have a boundary with three convex edges and one concave edge, hence are all primary faces. All other faces are secondary faces.

Face f2 is considered to be the most important because it has a hole loop and more concave edges in its boundary. Because of the hole loop it is a separator face, separating the "root" (faces: f1, f3, f4, f5 and f6) from the "boss" (faces: f7, f8, f9, f10, f11). From this you might infer a simple definition of a boss might be a secondary face surrounded by convex edges with all adjacent faces as primary faces.

Fig. 10.23 Block with boss

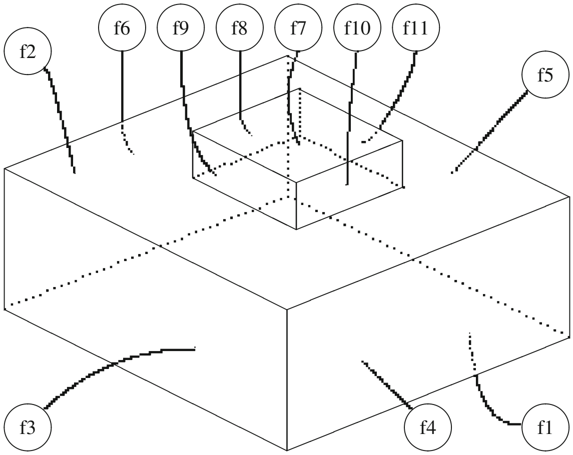


Table 10.2 Figure 10.23 face classification

Face	Primary or secondary	Holes/edges
f1	Secondary	4cx
f2	Primary	1hl, 4cx, 4cv
f3	Secondary	4cx
f4	Secondary	4cx
f5	Secondary	4cx
f6	Secondary	4cx
f7	Secondary	4cx
f8	Primary	3cx, 1cv
f9	Primary	3cx, 1cv
f10	Primary	3cx, 1cv
f11	Primary	3cx, 1cv

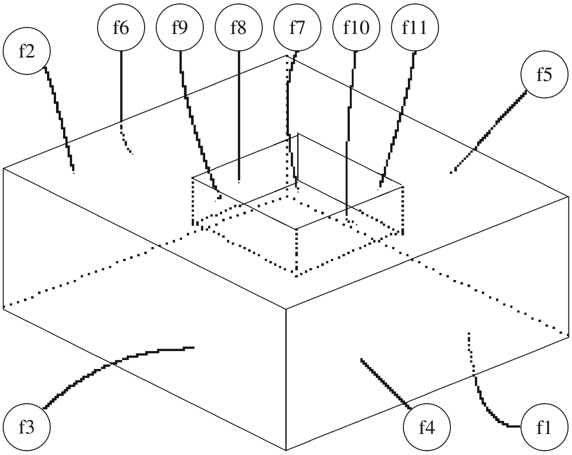


Fig. 10.24 Block with pocket

A second example is shown in Fig. 10.24 which shows a block with a closed pocket.

As with the block with the boss, the object faces can be classified as primary or secondary. The analysis for all the faces is given in Table 10.3.

Note that there is a strong similarity between these two feature analyses. The boss is an "extrusive" feature while the pocket is an "intrusive" one. A final simple definition for the pocket might be a secondary face surrounded by concave edges and all adjacent faces being primary faces.

10.2.2 Surface-Based Features

Although Kyprianou's work was ground-breaking, it had several limitations. Features do not always stem from convenient hole-loops and sometimes faces are

Table 10.3 Figure 10.24 face classification

Face	Primary or secondary	Holes/edges
f1	Secondary	4cx
f2	Primary	1hl, 8cx
f3	Secondary	4cx
f4	Secondary	4cx
f5	Secondary	4cx
f6	Secondary	4cx
f7	Secondary	4cv
f8	Primary	3cv, 1cx
f9	Primary	3cv, 1cx
f10	Primary	3cv, 1cx
f11	Primary	3cv, 1cx

shared between elements. Many improvements were made by after Kyprianou by Jared and his team, Smith, Anderson and Parkinson. Their interest was initially in producing manufacturing information automatically based on manufacturing features. This is a complex topic, though, and more will be said on this later.

In later work, with Tate and Swift [3], work was done on establishing methods to determine whether or not an object was symmetric. This is important for manufacturing, say, but it is not necessarily obvious from the construction method. It is not always natural for a designer to design half- or quarter of a part and then use a symmetry operation. The designer may prefer to design the complete part or may modify a design to create a symmetric part to make it easier to assemble. Symmetry is a special type of "Geometric Reasoning" and is an important consideration in assembly, for example.

A, somewhat, classic survey was done by Shah et al. [1]. There is a later survey by Parry-Barwick and Bowyer [4]. A lot of feature work exists and will not be mentioned here. The general notion of surface-based features is that the boundary representation of an object models its skin and this skin can be subdivided into regions, called features. What Kyprianou's method illustrates is a kind of shorthand notion, that the concave edges mark a limit for intrusion or the root of an extrusion. The next subsections are intended to give an idea of some of the other ways of looking at features and disentangling them from objects.

10.2.3 Feature Recognition in Dual Mode

An interesting alternative to Kyprianou's method was worked out by Falcidieno et al. In their method, the dual of the object was used as the basis for feature recognition. An example of an object and its dual is shown in Fig. 10.25.

The larger node at the centre of the dual corresponds to the face with a hole loop, and is, in fact, a multiple edge-set vertex. The dual figure is drawn as two octahedra, one on top of the other, but this is for convenience, the geometry is not

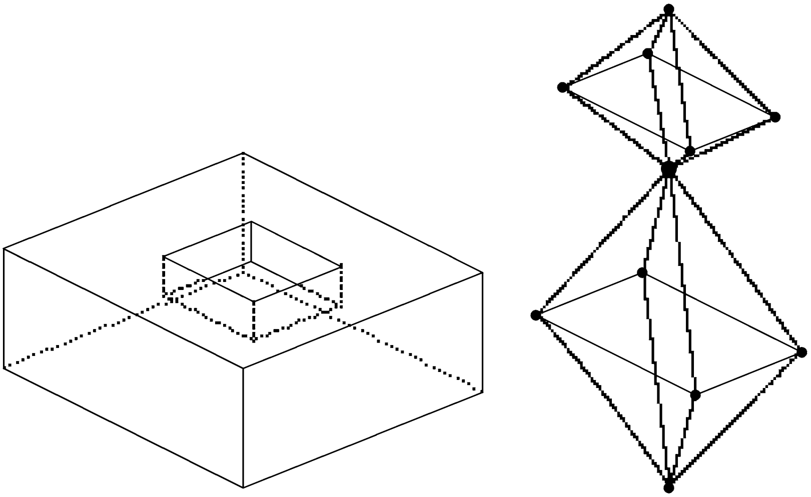


Fig. 10.25 Block with pocket and dual

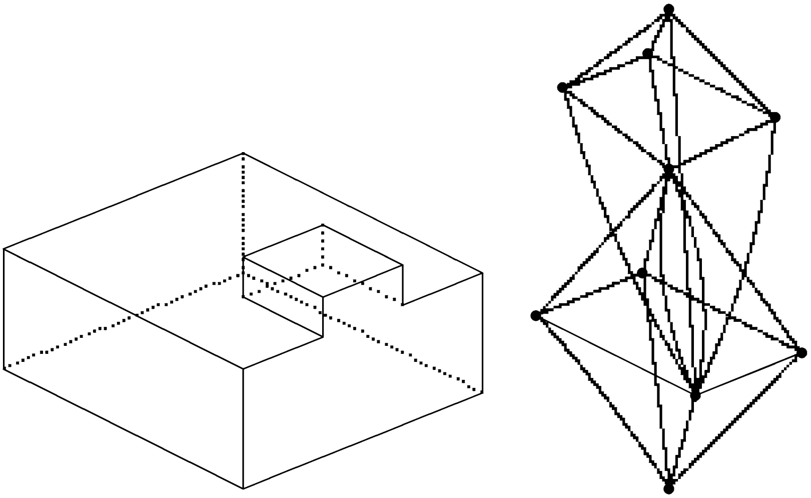


Fig. 10.26 Block with shared-face boss and dual

defined. In the feature representation method of Falcidieno et al. the dual parts are kept separate.

There is a complication when the features collide. In the object shown on the left of Fig. 10.26, one face is common to the base and to the boss. The dual of the object is shown on the right and it is clear that there is no clean separation point as with the object shown in Fig. 10.25.

Collisions between features, or collisions between features and the base have been a continual source of problems for feature recognition and representation. There are no easy answers to this problem.

10.2.4 Volumetric Feature Decomposition

A feature recognition variant, involving volumetric decomposition, was proposed by Waco and Kim. The idea of combining volumes to describe a shape is what lies behind CSG, or Constructive Solid Geometry, mentioned in Sect. 2.6. Some people found CSG representations a convenient support for feature recognition because they felt that the primitive objects in CSG corresponded to features. This is, of course, an over-simplification and recognising features from CSG also has its problems.

The alternating sums approach developed by Kim et al. is illustrated, for a simple two-dimensional case, in Fig. 10.27.

The original object is shown in Fig. 10.27a. The convex hull of this object is shown in Fig. 10.27b, c. The set difference between the convex hull and the original object is shown in Fig. 10.27d. This, if you like, is the amount by which the convex hull is too big. Taking the convex hull of this object, shown in Fig. 10.27e, f, this is then subtracted from the first convex hull. This creates an object which is slightly too small, by the two convex parts shown in Fig. 10.27g. The object would then be represented as Fig. 10.27c minus Fig. 10.27f plus Fig. 10.27g. The objects can be redescribed in terms of the primitive objects to give a CSG tree.

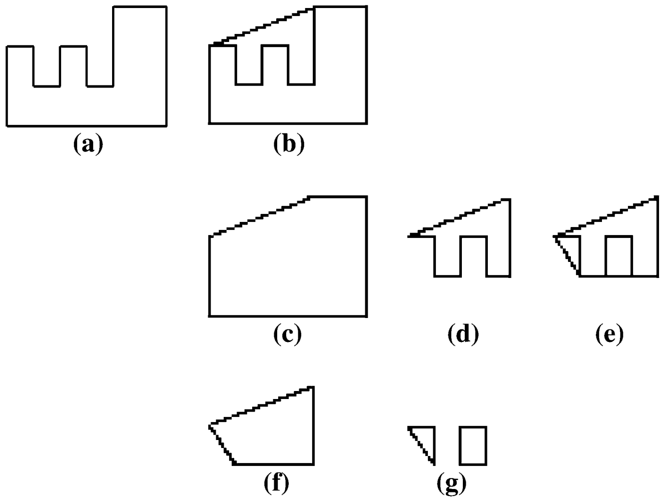


Fig. 10.27 Alternating volumes approach [5]

This is a very simplified illustration of the method to give an idea of the use of volumetric decomposition as a feature recognition method.

10.2.5 Features from the Medial Axis Transform

Yet another volumetric approach, from the medial axis transform, or MAT, was suggested by Renner and Stroud. The medial axis is a kind of skeleton of an object. It can be used to subdivide an object into convex pieces. The method, though, is a little weird because it works in so-called"dual space" like Falcidieno et al.'s work.

Figure 10.28 shows a simple object, shown in Fig. 10.28a, its medial axis, see Fig. 10.28b, the dual space nodes corresponding to the medial axis, shown in Fig. 10.28c and the grouping according to the convex element decomposition (Fig. 10.28d). This is a "positive" decomposition of the object itself, giving extrusive features. It is also possible to perform the same method on the negative object, giving, in effect, the medial axis of the space around the object. Decomposition of this would find the central hole.

One problem with the medial axis method is that it is complicated to calculate. The decomposition method developed by Renner and Stroud is interesting but needs more effort to make it stable. It has been demonstrated on simple examples, but complex objects cause problems, especially with negative objects. The medial axis provides information lacking in Boundary Representation, that is, the notion of spatial occupancy. For this reason it has a large potential in various applications. However, it is not yet a common feature of CAD systems.

10.2.6 Advantages, Disadvantages and Problems

A few of the advantages of feature recognition are:

1. Independent of object creation method.
2. Tailorable—recognition methods can be changed to get application-specific information.
3. Provides valid feature information when applied.

A few disadvantages are:

1. Difficult to do properly.
2. Interacting features.
3. Volumetric features such as thin walls are not detectable in a surface-based approach.

Although there have been several more-or-less successful pieces of software for recognising isolated features in objects, the problem of overlapping features rests as a stumbling block.

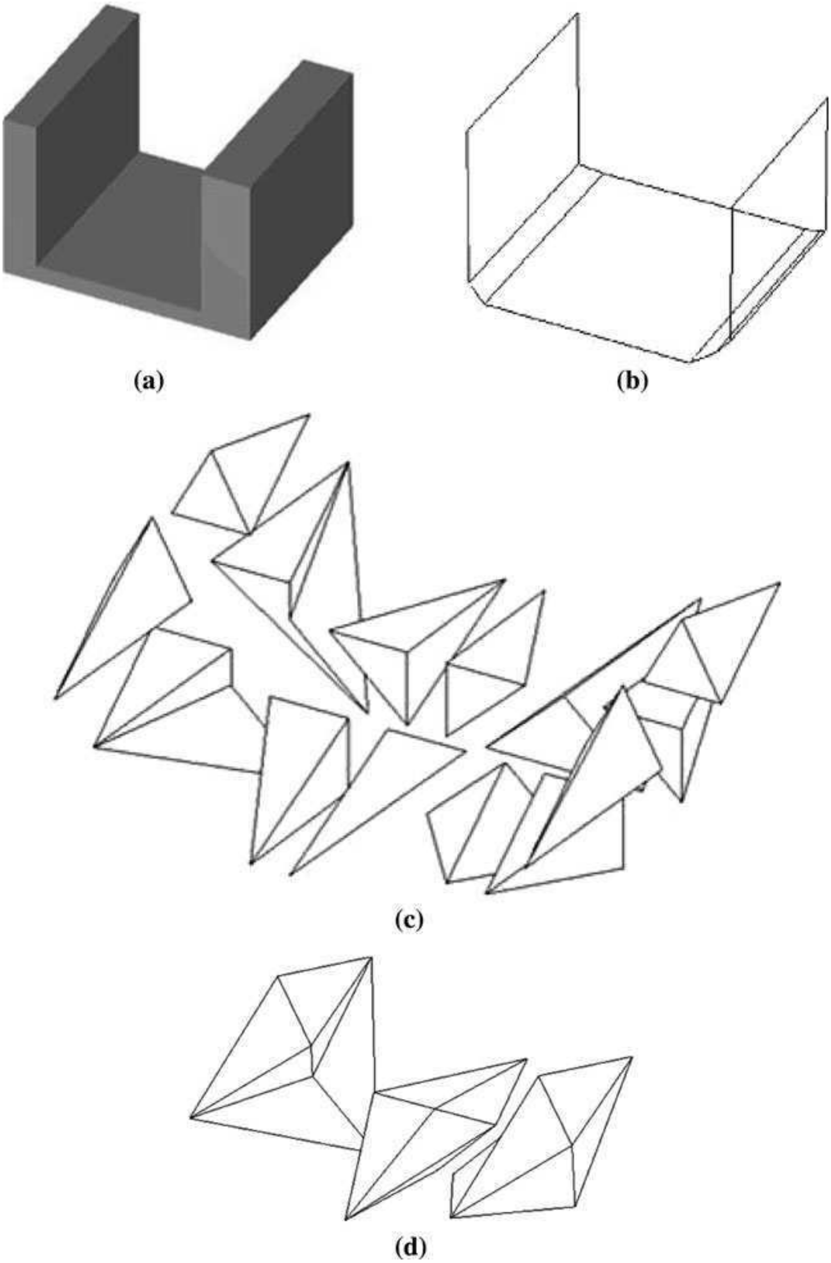


Fig. 10.28 Object, medial axis and nodes (from Stroud et al. [6])

There are some features which are not surface features, which can only be found from the volumetric notion of a part. Thin walls are an example of this.

10.3 Design by Features

"Design by features" means building in feature information directly into a model rather than having to rediscover it. There are two ways that I know about for doing this:

1. Feature operations—operations which insert shape elements into a model.
2. Editing in features—adding partial feature shapes into a model directly.

10.3.1 Feature Operations

Using feature operations means that the feature description of an object is contained in the "history tree". The history tree is a record of the operations performed to create a part, a recipe for making that object. This is useful as supplementary information but it is not usually the same as a feature description, unless you are very lucky, which is unlikely. See Chap. 12 for more details on history trees.

A feature operation should check how it is applied, add feature information to the model and maintain this information during modelling. It should also be capable of verifying that the shape produced is as described. To the best of my knowledge there are few, if any, implementations that do this. It is not easy to do this consistently because modelling code deals with geometric shape information, not symbolic information, so symbolic information tends to get left behind.

In order to demonstrate what could happen, consider the following example:

1. The user defines a basic shape, a rectangular block (Fig. 10.29a).
2. The user defines a pocket, Fig. 10.29b by defining a contour and extruding it downwards. The system controls that the defining contour lies entirely within the face. The result should be checked to see that the contour does not break through the bottom of the object. In this case, though, the bottom is well-defined. The bottom and sides are labelled as belonging to a closed pocket. The top face of the object is labelled as a "parent" to the pocket.
3. The user adds an open pocket, Fig. 10.29c. This is done by creating a contour which touches or crosses one of the edges of the top face and then extruding this downwards. The top face is labelled as the parent face, the base and the three side faces are labelled as before. The edge where the pocket opens out to the second face is labelled as an open boundary. I have not said that the second face is also a parent. This is an option but here I am using the logic that the face on which the contour was sketched is the parent face.
4. The user adds a boss, Fig. 10.29d. The boss contour is checked to see that it does not touch or overlap the boundaries of the top face, which it does not.

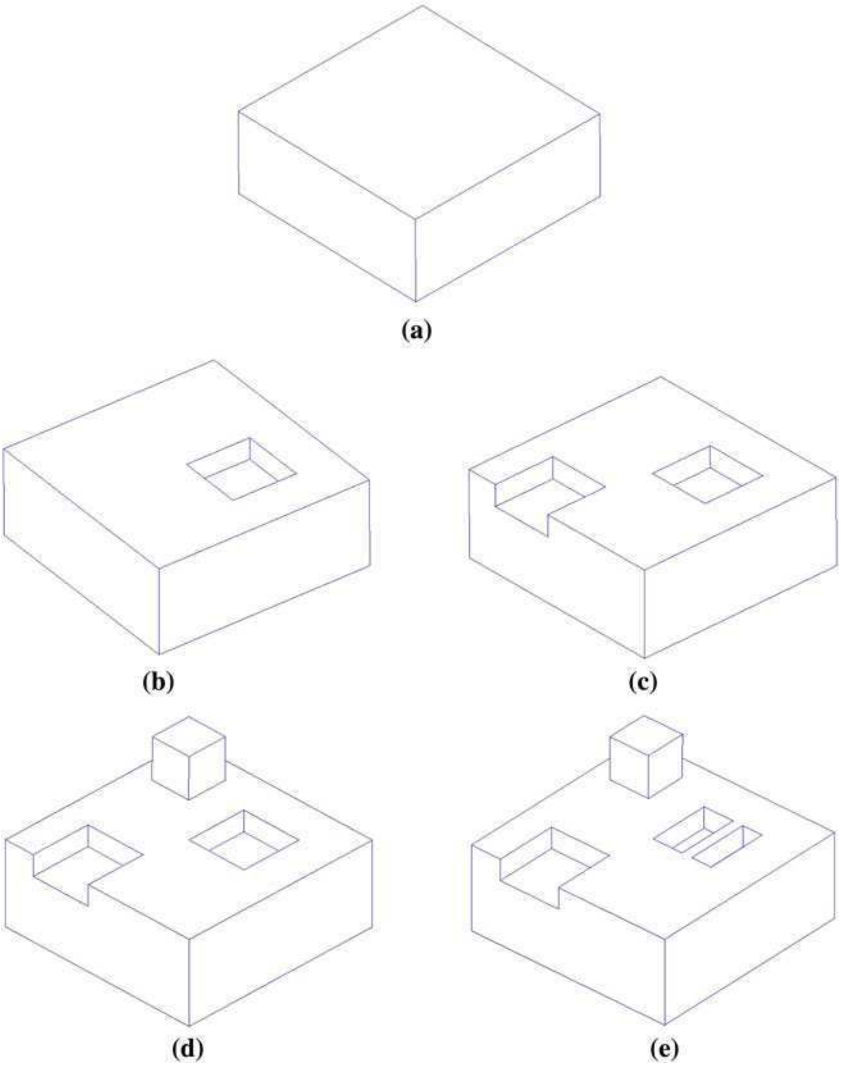


Fig. 10.29 Feature operations with pockets and bosses. **a** Original object. **b** Object with closed pocket. **c** Open pocket added. **d** Boss added. **e** Closed pocket modified

The top and sides of the boss are labelled as belonging to a boss and the top face is labelled as the parent of the boss.

5. The user modifies the original pocket by creating an overlapping, rectangular shape and extruding this downwards. The system notes that faces labelled as being pocket sides have been modified. It might note that the new intersection edge is concave, which indicates that the pocket size has been reduced.

There are a number of problems with trying to implement this in practice. Remember that CAD system developers cannot release experimental code, a system has to be reasonably stable in order to get it to users. The example above would need modifications of the basic code right across the system in order to be able to work with labelled shape entities. The modelling kernel ACIS has gone a little way towards this with their attribute handling mechanism, but this is not a complete solution in itself. As described in Chap. 4, many operations create an extra solid model as a partial result and then use Boolean operations to add this to or subtract it from the base model. It is to be expected that all the operations above would be done in this way. This gets round many problems and awkward cases, such as when a pocket is extruded through an object, or when the boss interferes with another part of the object. However, Boolean operations are general operations with no implicit knowledge of the shape they are producing. Specialised operations could be used, with a kind of special Boolean check to handle interactions. Using specialised operations would mean that you have information about what is expected in order to handle special cases. On the other hand, though, you would need to do a lot more implementation work in order to create them as they would be there only to create a particular shape. This is obviously less attractive to commercial system developers.

An alternative to having specialised operations is to have feature "verification" (see Sect. 10.4) to check the result. This is a sort of localised feature recognition task and would require having feature descriptions of the various feature and checking the description if any of the elements involved are modified. I don't know of any system that does this, though, it remains a technical possibility.

Note also that it is necessary to establish the rules for when a pocket becomes an open pocket. If there are at least three adjacent side faces then the pocket may be still an open pocket, but there are fewer then maybe the feature is a slot or step. However, if there are only two side walls, as for a slot, the feature may still be considered as a doubly open pocket if its size is too large to make by moving a tool through in one or two passes. It is a complex task to establish these rules, but this is necessary in order to be able to check the feature produced, or when modified, to maintain a relevant feature description.

Another type of problem comes when there is no convenient operation for producing a shape element. In this case general operations like extrusion or Boolean operations may be applied. The elements involved could be labelled as "To be recognised" and local recognition methods applied to just these.

A further type of problem is when "inadvertent" features are created, that is, features which appear as a bi-product of the operation applied. Consider Fig. 10.30. The original shape is shown in Fig. 10.30a. A boss is added in Fig. 10.30b, but since this touches three side faces a step has been created. Similarly, in Fig. 10.30c another boss has been added but, for manufacturing, a slot has been created.

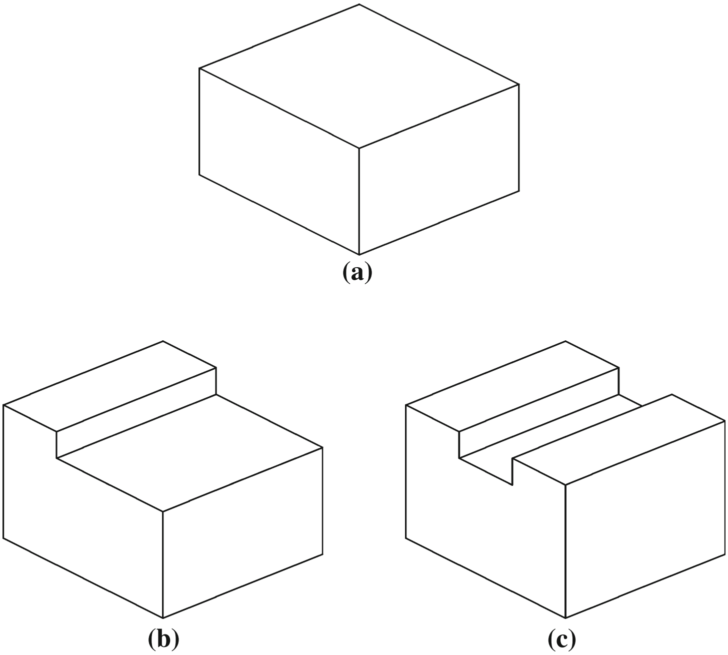


Fig. 10.30 Inadvertent feature creation. **a** Original object. **b** Boss added, step made. **c** Boss added, slot made

10.3.2 Editing in Features

An elegant alternative to feature operations, proposed by Ranta et al. [7], is to have a library of parametrised shape elements, features, which can be edited into a model to produce features. An example of an object, a partial slot feature and an edited object is shown in Fig. 10.31.

The original object is shown at the bottom left. The feature, a slot, as a partial object at the top, and the final object on the bottom right. One way that this can be done is to stretch the geometry of the partial slot object so that it is as large or larger than the object being altered, and then to perform a Boolean operation to add it in, as illustrated in Fig. 10.32.

Having a library of features makes it easy for the user to have an overview of the shapes that can be pasted into a model. Editing features into an object requires Boolean operation techniques, suitably adapted. It is possible to avoid the need for special operations because the feature identity is present in the library. This helps the developer by having more information for resolving special cases.

Technically, the Boolean operation for partial objects is not very different from the standard Boolean operation. A necessary condition for the application is that the Boolean interaction boundary does not cut the boundary of the partial feature.

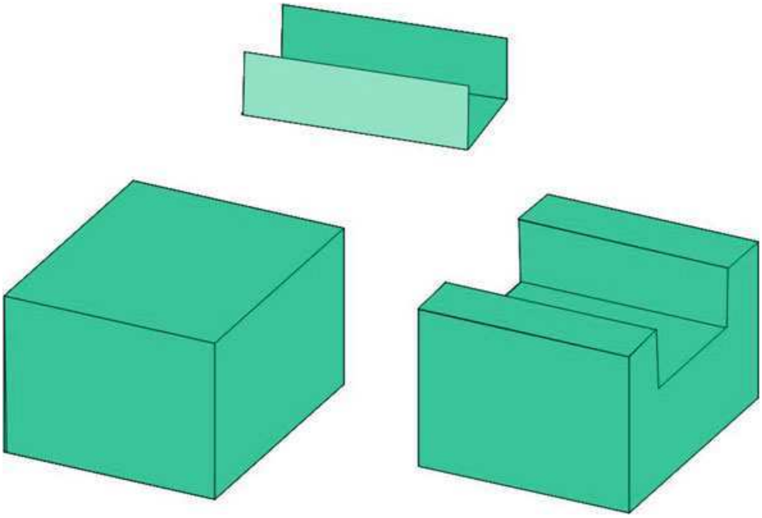


Fig. 10.31 Feature as partial model

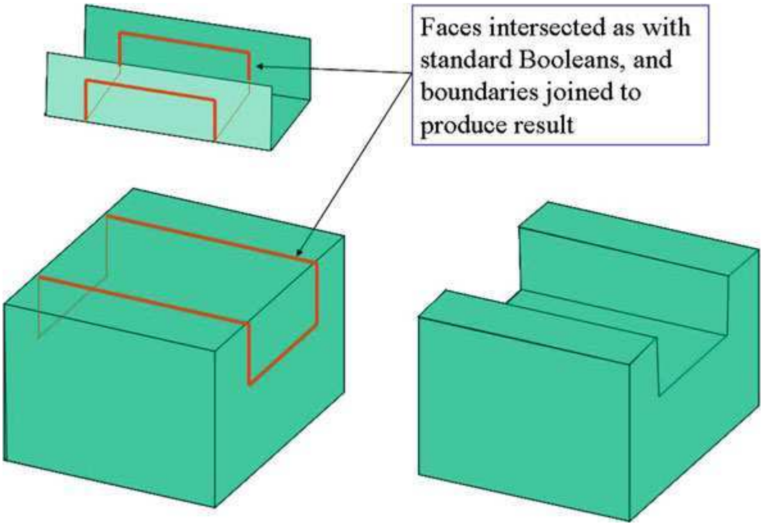

Faces intersected as with standard Booleans, and boundaries joined to produce result

Fig. 10.32 Boolean boundaries between partial object and object

The object and partial feature are separated along the Boolean interaction boundary, as in Fig. 4.3 and then the portion of the base object "outside" the interaction boundary is combined with the portion of the partial feature "inside" the interaction boundary (Fig. 10.32).

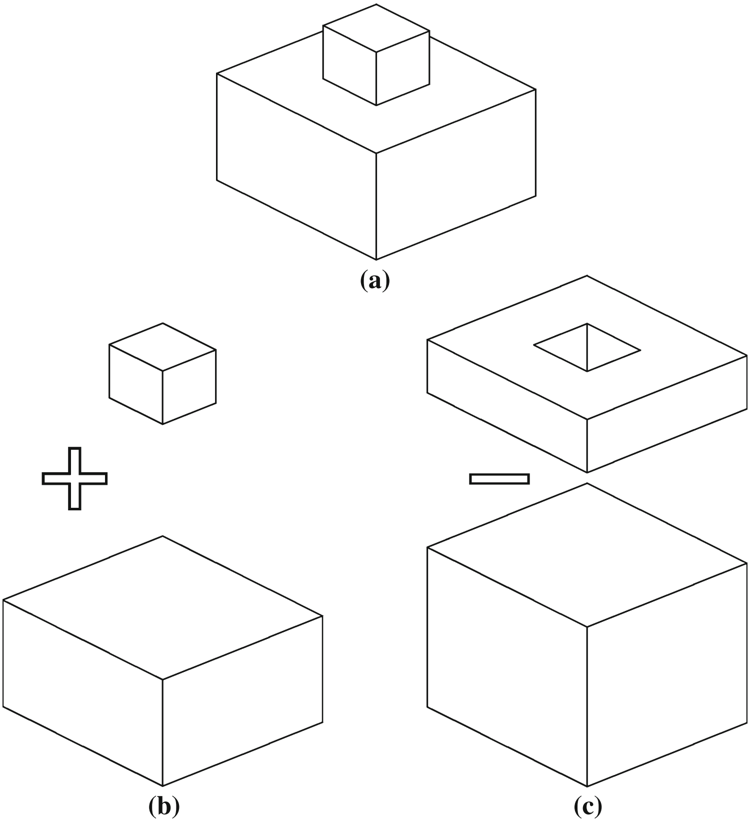


Fig. 10.33 Cube with boss. **a** Block with boss. **b** Block plus boss. **c** Block minus complex step

10.3.3 Feature Transitions

One early criticism of the design-by-features approach was that the feature description produced was static and not necessarily what was needed for an application. The notion of feature transitions came about to get round this.

Take, for example, a block with a boss on top, as shown in Fig. 10.33a. For the designer it is quite natural to create this as a block and then to extrude the boss on the upper face (Fig. 10.33b). However, if a manufacturing engineer wants to make this then she or he might “see” the object as a larger block from which material is to be removed, Fig. 10.33c.

The idea of feature transition is to reconfigure the boss feature information into the complex slot information set for manufacturing. Manufacturing is (usually) concerned with material removal, not with material addition. Normally you would not expect to make the object by creating a base block and then gluing or welding a smaller block on top, which is what is implied by the block plus boss variant.

Feature transitions for manufacturing, assuming metal removal processes, would take an extrusive feature and turn it into an intrusive one. Extrusive and intrusive features were described in Table 10.1.

10.3.4 Advantages, Disadvantages and Problems

A few of the advantages of design by features are:

1. Explicit feature information

A few disadvantages are:

1. Interacting features.
2. Inadvertent features are not recorded.

One researcher, François Sprumont, then of EPFL, suggested using only feature operations corresponding to manufacturing operations in order to build up a manufacturing related feature description. This is one approach to creating features in design for use downstream in manufacturing. However, the disadvantage is that the designer is forced to think in manufacturing terms, which may be a limitation on the fluidity of the design process. The idea remains interesting, though, because it is a way of forcing designers to think about a different application area. It would be interesting to see such an idea implemented and used in order to evaluate it.

10.4 Feature Verification

Feature verification involves checking a feature description introduced into an object to check whether it is still valid. This can be thought of as a localised feature recognition method where the feature elements are present but need to be checked to see if they still have the created properties.

In order to do this it is necessary to retain the feature information in terms of data structures recording the roles of parts of the feature and having a verification algorithm. For example, take the object with the pocket in Fig. 10.34a.

In the original object, the pocket base face, face A, is connected via concave edges to faces B, C, D and E. If the pocket is split, as in Fig. 10.34c, then base face A becomes two faces, A1 and A2, say. Face C is split into C1 an C2, say, and face E is split into face E1 and E2. Face A1 is then connected via concave edges to faces E1, B, C1 and a new face, F, say (not marked in the figure). Similarly, face A2 is connected via concave edges to faces C2, D, E2 and a new face G (also not labelled in the figure). So, the system should note that the original pocket cannot be considered as an entity in the changed figure and should be re-recognised as two new pockets. If the pocket had been

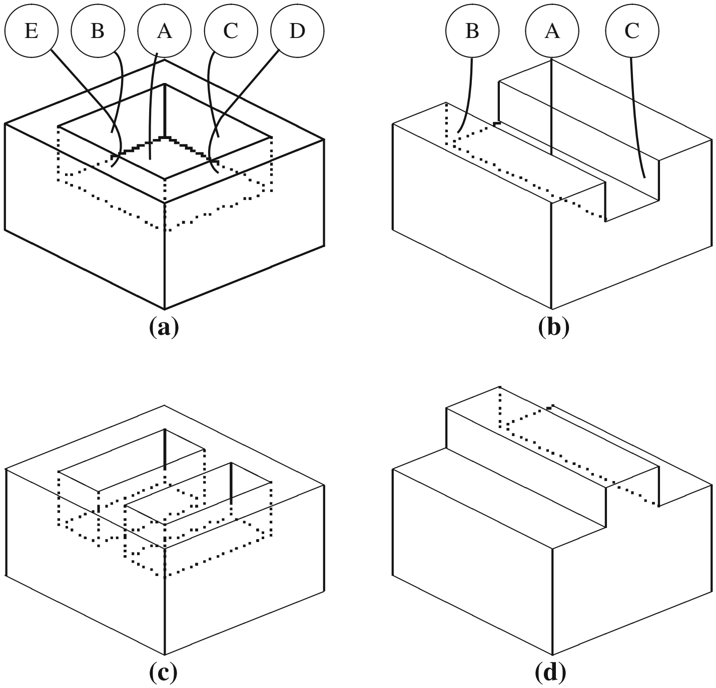


Fig. 10.34 Features and modifications. **a** Object with pocket. **b** Object with slot. **c** Object with modified pocket. **d** Object with modified slot

split by cutting out a slot, say, then the pocket might still be considered to be a pocket.

For the other shape, in Fig. 10.34b, the slot base face, face A, is connected to two side faces, B and C, via concave edges. If the geometry of the object is changed, as in Fig. 10.34d, then the original base face, A, is now connected via convex edges to faces B and C. The slot has been eradicated and the object needs to be re-recognised to find out what the result is.

Feature verification is related to feature transitions in that local re-recognition can be used both to check the features as well as to determine what features are there.

10.5 Feature Summary

Features are an important element of shape research because they transcend the level of the representation method and make shape elements directly referrable. However, feature research is difficult.

Many techniques have been developed and tried, but no simple solutions have yet been found. Basically, the two areas of research are: “feature recognition”, in which the final shape is traversed to discover feature information; and "design-by-features", in which the feature information is introduced explicitly. Neither of these is perfect on its own and probably the best solution would be provided by a mixture of the methods.

Feature recognition is good in the sense that the final shape is taken without knowledge of how it was created. The feature information extracted can be tailored to different application areas, in a similar way to that used by domain experts to view the same shape from different points of view. However, feature recognition is difficult.

Design-by-features is good in the sense that some, to what degree depends on different factors, feature information can be introduced explicitly. However, not all that feature information may be relevant to an application and there is a risk that the feature information is corrupted during object creation. The history tree of a CAD system is not the same as a design-by-features feature description. The history tree is simply a record of the operations used to create the shape, not all of which may correspond to features.

Feature transition and verification techniques can be a useful support tool to design-by-features, but these techniques do not seem to be in common use.

10.6 Using Features

This section describes briefly two aspects of feature use.

10.6.1 Design Features

Design features can be thought of as the principle shape elements which govern the design solution. For design there are requirements and often some known elements which need to be taken into account.

10.6.1.1 Structuring Features

It is not really reflected in current practice that there is a hierarchy of features in a model, Fig. 10.35. It is arguable about how many levels there are and what they are, this is a personal view.

There are elements of the design which are there for the fundamental requirements of the design. In addition, there are elements which are present for functional reasons, such as to fix one piece relative to another, with or without degrees of freedom. Then, on the third level in the figure, there are practical elements to guarantee physical properties such as rigidity, for example, or cuts to reduce weight. Finally, there are the elements which are introduced as compromises for manufacturing or assembly, for example.

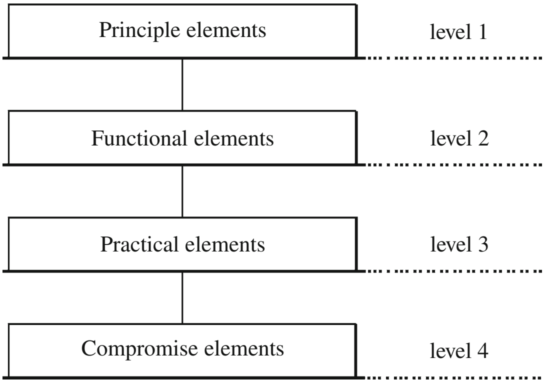


Fig. 10.35 Feature hierarchy

It may not be evident simply from the shape of a design which elements are there for what reason. Ideally the design "intent" would be recorded explicitly so that anyone in the production chain would have access to the knowledge. The purpose of thinking about feature structures is to improve the level of communication of the design.

10.6.1.2 Features as Independent Elements in Design

I once asked a student about the elements which he thought were important for a design to which he replied that a particular hole and the size of a supporting plate were important. He added that you couldn't have a hole, though, without an object. Actually, apart from current practice, there is no reason why a hole cannot exist without material around it. From a modelling point of view a hole could be represented as an open cylinder, such as those shown in Fig. 10.36.

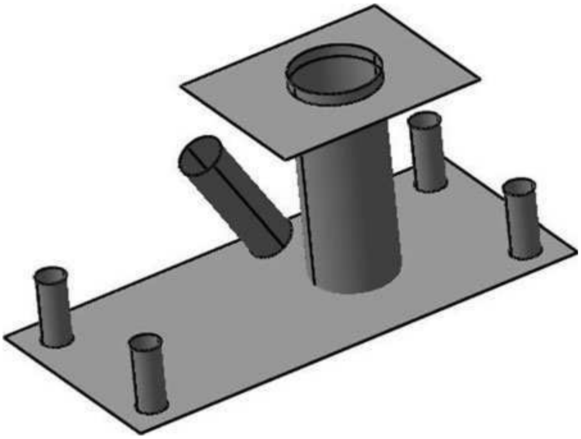


Fig. 10.36 Design features as partial objects

Pulling geometry off existing parts to share them with matching elements in assembly design was proposed by Kjellberg et al. in Sweden at the end of the 1970s and the beginning of the 1980s, so it is not a new topic. However, while this is an interesting subject, and technically feasible, it is not easy to create the closing elements of a model, so the tendency has been to remain with design an object first and adding matching feature elements later. This has weaknesses, though, because there is no link between the objects to help maintain consistency.

The way of doing this is to use non-manifold models to build up the object face-by-face, or with open facesets. CAD tools to do this can usually be found in connection with surface modelling, where individual surface elements are treated like faces, or rather pairs of back-to-back faces. When an object is complete there are two matching closed facesets, one on the exterior and one on the interior, so to create a solid the interior can simply be removed. See Sect. 4.12.

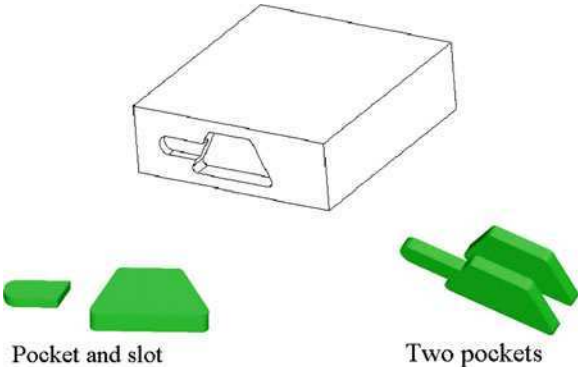
What is lacking, though, are tools to create the surface elements easily. Surface elements can be created, trimmed and joined, but the mechanics of doing this detract from the fluidity of design that is, or should be, the goal of a CAD system. Until the subject is investigated and new tools developed it is unlikely that this will be a convenient user method.

In the meantime, it is important to be clear about the key elements of the design, what are the design elements that influence the shape creation process. If these are, at least, documented as notes, or, better still, the information is recorded as informative notes attached to the design, then the design process becomes more logical. This is part of the process of structuring the feature information for other people in the production chain. If the key design elements match other elements in an assembly then obviously this will help generate tolerance information for use and assembly, and hence the manufacturing tolerances for design. Functional descriptions for communicating functionality and creation and annotation of design information in the model are the subject of research. One idea is to attach voice recordings to the model. This is also technically feasible, but would increase the storage requirements dramatically.

10.6.2 Manufacturing Features

A common misconception is that you can recognise features in a design automatically and then issue commands to mill these as a manufacturing plan. Although Jared and his team, in a project from 1982 to 1985, realised that this is not enough, people still propose this for automatic manufacturing. Jared and his team determined that, for manufacturing, it is necessary to make many process-planning decision which affect the manufacturing features. This affects the nature of the raw part, or stock, from which the part is to be made. If a part is to be made from a casting then some elements may not need to be machined, other elements, such as flat matching faces, may need to be finished, even though they might have no distinguishing shape elements in the original CAD model.

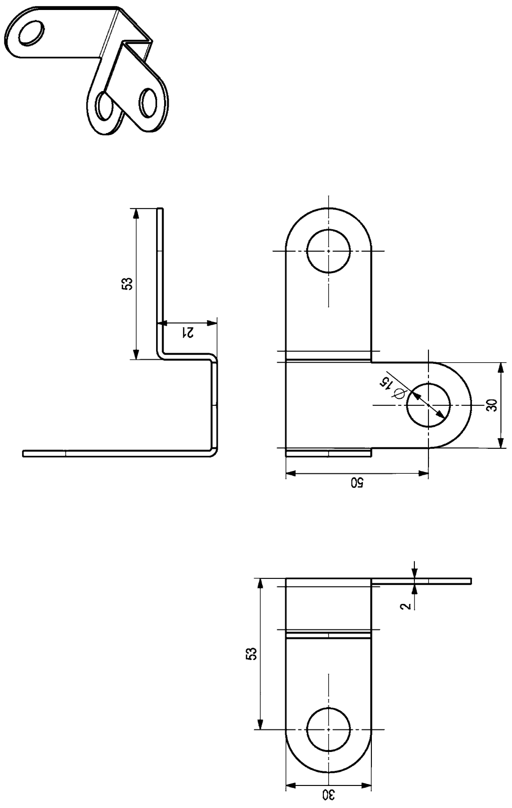
Fig. 10.37 Pocket and slot or two pockets?



Most recognise-and-make proponents assume implicitly that the part is to be manufactured from a rectangular block and that any concave element is to be machined in some way. Even with this approach, concave elements such as those shown at the top of Fig. 10.37, are ambiguous. The shapes at the bottom, where the solids show the material to be removed, illustrate two different decompositions. On the left the decomposition implies that the large region would be milled as a pocket and then a slot milled out from this. The decomposition on the right indicates the elements are to be milled as two pockets, one above the other. The decision about which to choose depends on many factors, such as the shape and size of the elements and the material, not just on the topological arrangement of the elements. For example, if the slot is wider than the widest milling tool then it would have to be milled as a pocket, even though the parallel slides seem to indicate that it is a slot. If the material is hard then it may be unwise or impossible to mill the whole pocket depth in one step. If the size of the elements is reasonably small, though, the most natural machining option might be to mill the pocket and then the slot. If you allow for milling from castings, then the possibilities increase again. It may be, say, that the pocket is simply for clearance while the tolerances on the slot are high. This would imply that the pocket does not need to be milled at all while the slot should be at least finished. Feature finding is an important aid, but automatic generation of machining instructions is still a long way off because human skills and decisions are needed, too.

Another counter example is shown in Fig. 10.38, from János Nyitrai's collection of mechanical engineering objects (Nyitrai, J.: Collected mechanical objects. Private communication, Nyitrai János, Budapest MúEgyetem, BME (1993)). The classical feature recognition methods would miss that this is a bent object, because this not what is usually checked for. It is clear that, even if there is no feature recognition system to recognise this as a bent object now, then there are sufficiently talented researchers in features to produce one. This is not the point. If you recognise this as an object made by bending then you assume that the material is flexible. If the material is brittle then it might be made by milling. You have to have extra information above and beyond the shape in order to plan the manufacturing.

Fig. 10.38 Object to be manufactured



Another kind of problem comes with the object shown in Fig. 10.39. Here the object is to be milled, from aluminium say. A classic feature recogniser should have no difficulty in finding the pocket and the step as manufacturing features. However, the pocket is not totally isolated from a functional point of view. It has thin walls, which require special treatment. This has practical effects on the order for machining and the cutting strategies and parameters. Thin walls, though, are volumetric features which are not normally found during feature recognition.

The idea of these small examples is not to try and convince you that feature recognition is impossible in connection with manufacturing, simply to point out that you cannot apply simplistic solutions and assume that the results will always be good. As Jared has pointed out, it is necessary to take several decisions during process planning which affect the manufacturing features needed and are certainly not trivial. This means that the geometric shape of a part is not the sole source of information for manufacturing planning.

A variant on the classic "recognise and mill" methodology was proposed by Malcolm Sabin in a keynote presentation at a CAM-I seminar in Cambridge in 1983. He proposed taking a final design and then building it back to the stock part. The list of features "undone" during the process of building back then becomes a

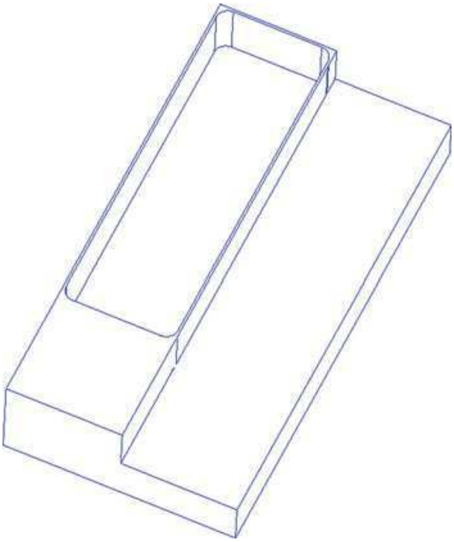


Fig. 10.39 Another object to be manufactured

simple process plan, to be ordered and optimised. This is an automated process rather than an automatic one and so the feature recognition process becomes a decision support tool for a human user rather than a replacement for a human planner.

Consider the object shown in Fig. 10.40. The original object is shown at the top. In the middle the drillable holes have been identified. These are round holes with a maximum limit set, hence the counter sink is not recognised because it would be milled rather than drilled. The holes are grouped according to axis direction and accessibility, hence on the left are the holes accessible from the *Z*-direction, while on the right there is a single hole for which the part would have to be re-oriented. Now here there would need to be a decision by the human. If the part is going to be made on a three-axis machine-tool, then it would be necessary to develop a new fixturing method. However, if the part is to be made on a five-axis machine then the two groups of holes could be merged. Finally, at the bottom is the part that is left after removing the drillable holes.

Another example is shown in Fig. 10.41. In this case the method indicates that the object should be machined from two directions. Even though the axis directions are parallel there are counterbores on one side and blind holes on the other side which mean that two directions should be considered.

Round holes are relatively easy to find and so these simple examples are intended to illustrate how this information can guide a user to a full process plan. The hole directions can be considered as indications of machining set-ups. The next step in this simple method is to look at each set-up, take the direction and look for profiles. The profiles indicate inside or outside profiles, pockets and even faces which might need to be faced off.

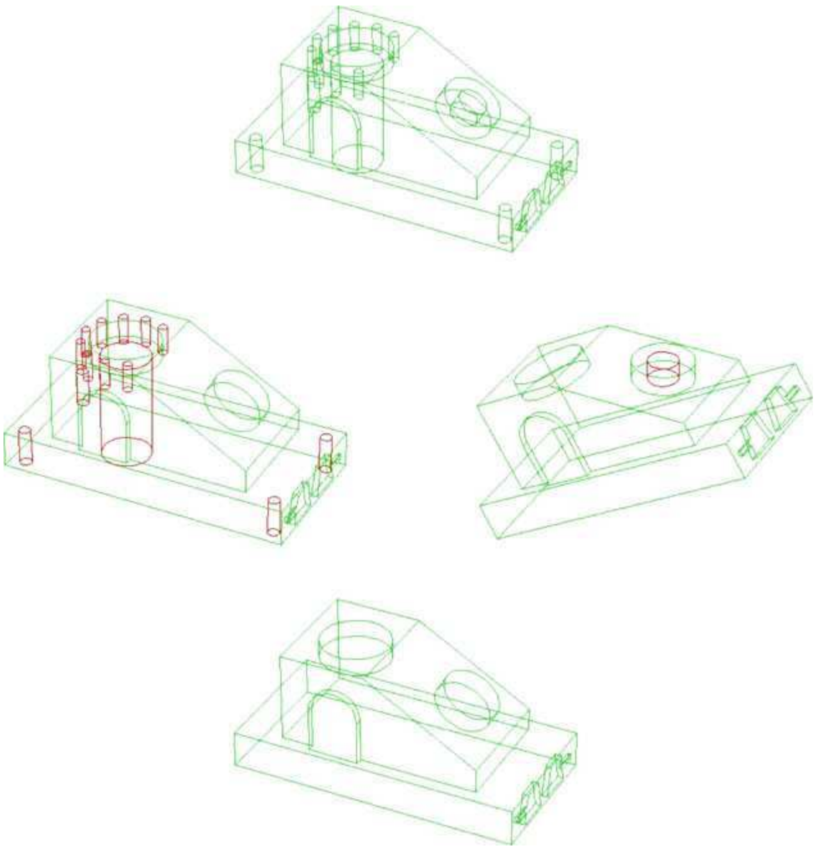


Fig. 10.40 Removing holes and orienting the ANC 101 object for manufacturing

A set of features derived from the new numerical control standard, STEP-NC or ISO 14649, are given in Appendix C as an illustration of an application feature set.

10.7 Chapter Summary

This chapter deals with the difficult and confusing subject of features. Features have been regarded as an important topic since early on in modelling development. Many methods have been developed and tried without any single solution becoming predominant. Unfortunately, there is also a tendency to misuse features or to present simplistic solutions which are not robust. There is some feature recognition in CAD systems, but this does not seem to be well developed. Features are sometimes confused with modelling operations in CAD systems, especially if the operations have feature names, but generally the feature information seems to

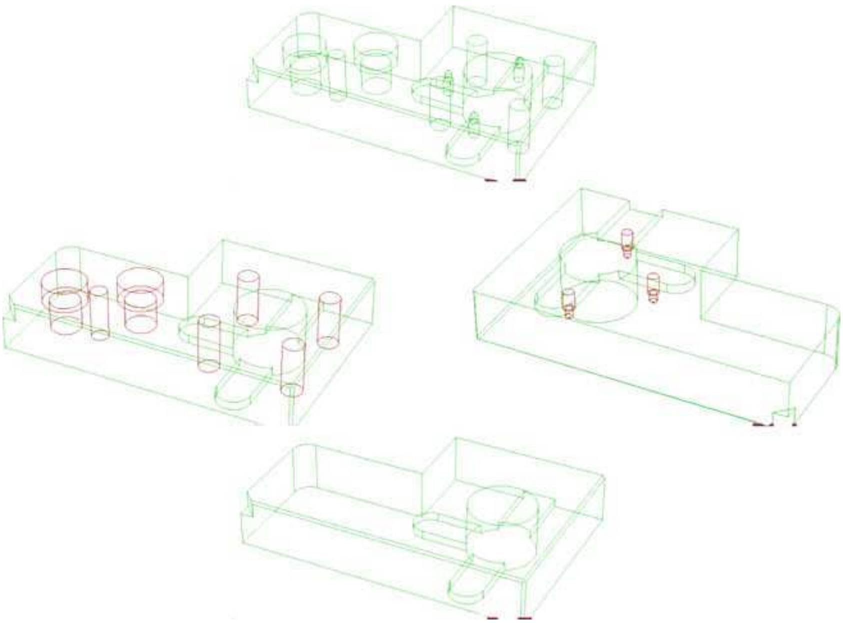


Fig. 10.41 Removing holes from the WZL STEP-NC test object

be neither preserved nor verified. This is a complex topic and probably needs much more time to become effectively integrated in CAD systems. However, features in manufacturing have already arrived at the controller level with the new standard ISO 14649. This, in turn, requires better feature development in both the design and planning stages to support it.

10.8 Exercises

10.8.1 Identifying Concave Edges

Identify the concave edges in the objects in Fig. 10.42.

10.8.2 Identify Primary and Secondary Faces

Classify each of the faces of the object in Fig. 10.43 with the slot as “primary” or “secondary”.

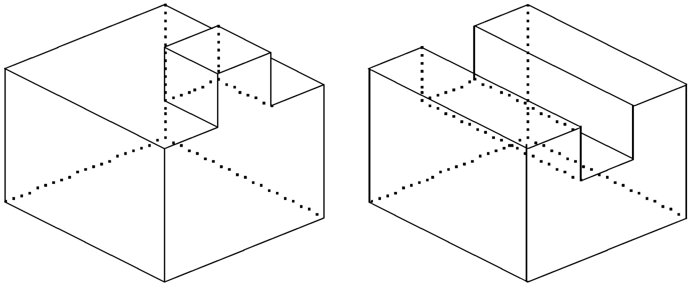
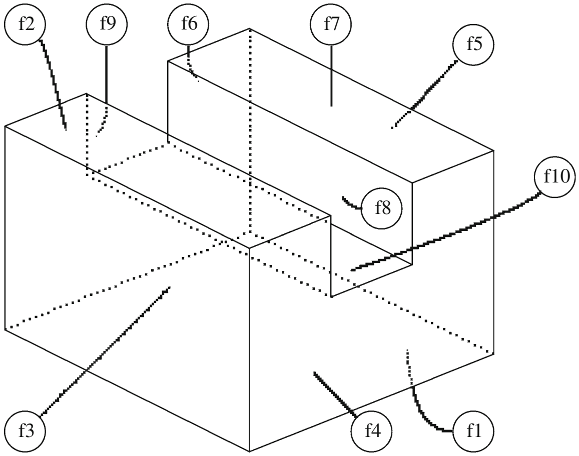


Fig. 10.42 Boss sharing common face and slot

Fig. 10.43 Slot with numbered faces



Face	Primary or secondary	Holes/edges
f1		
f2		
f3		
f4		
f5		
f6		
f7		
f8		
f9		
f10		

10.8.3 Face Adjacency Hypergraph

Draw the dual of the object with the slot and mark the edges corresponding to the concave edges of the original object. As a hint for drawing the dual, ignore the geometry and note that the front and back faces both have eight edges. In the dual, these will correspond to vertices with eight edges.

10.8.4 Design Feature Exercise

The aim of this exercise is to create a small assembly and to identifier the features in it. The assembly is shown in Fig. 10.44.

The axle has the shape shown in Fig. 10.45.

The basic shape shown on the right of the figure has roundings with radius 5, although you may decide for yourself their size.

The box is a rectangular block, 200 × 200 × 300, with wall thickness 5, and open at the back and front, as shown in Fig. 10.46. There are blends, radius 5, on the edges of the block and some holes, two for the axle and two to fix the plate, shown on the right of Fig. 10.47.

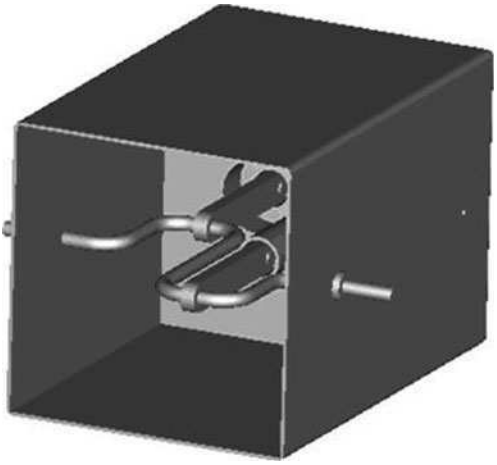


Fig. 10.44 Design features—final assembly

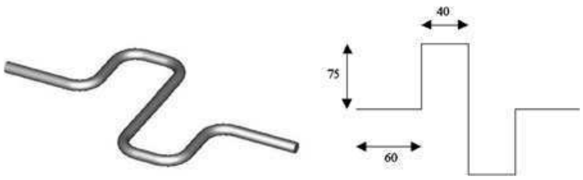


Fig. 10.45 Design features—axle

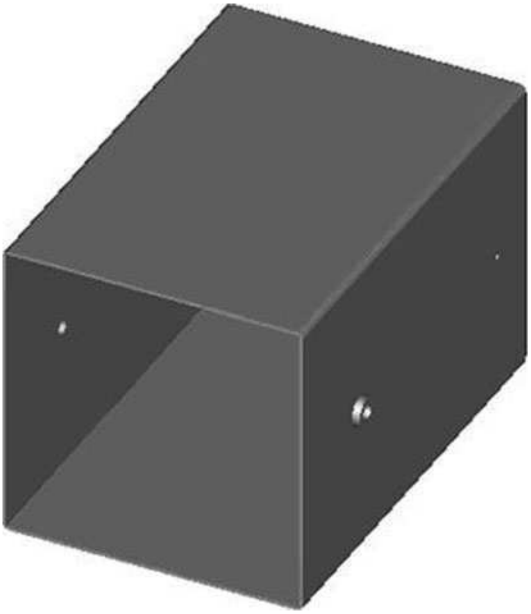


Fig. 10.46 Design features—product casing

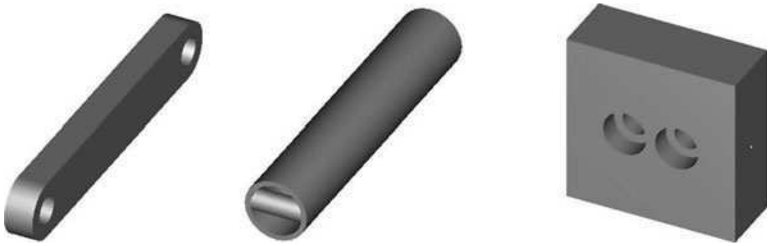


Fig. 10.47 Design features—piston elements and back wall

The "pistons" are in two parts, one being the piston head (Fig. 10.47—middle) and one which attaches the piston head to the axle (Fig. 10.47—left). Finally, there is the back wall, which is a rectangular block with two holes in the side for fixing it to the casing, and two guide holes for the pistons, shown on the right of Fig. 10.47.

For each element, identify the features and whether these are design features, manufacturing features or assembly features—each may be in more than one category. You may have noticed that it is impossible to assemble this product. What changes would you make to allow the product to be realisable, and which new features result?

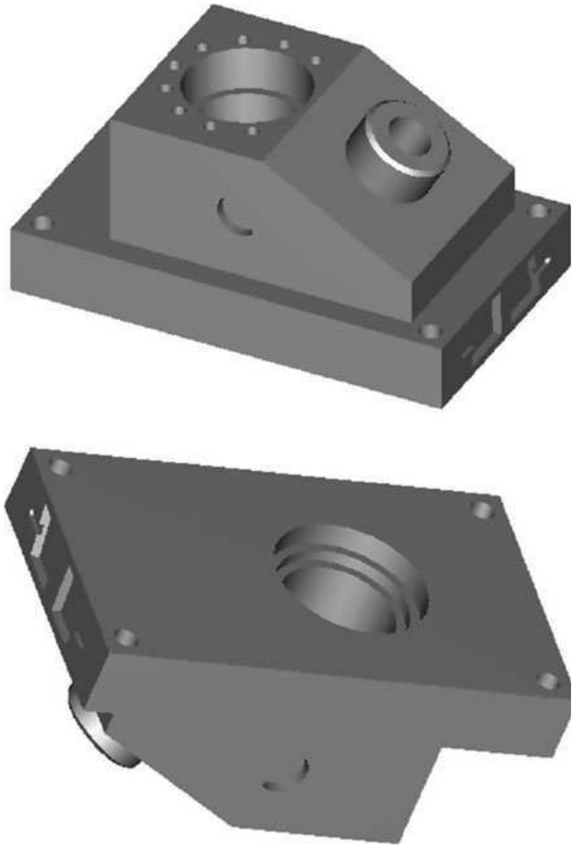


Fig. 10.48 Feature identification—ANC object

10.8.5 Feature Identification Exercise

Identify the feature is the object shown in Fig. 10.48 and say whether they are design, manufacturing, assembly or multiple-role features. Also, consider which of the features are compound features (features which depend on each other) and which are simple features.

References

1. Shah, J., Sreevalsan, P., Rogers, M., Billo, R., Mathew, A.: Current status of features technology. CAM-I Report R-88-GM-04.4 for Task 0 (1988)
2. Kyprianou, L.: Shape classification in computer-aided design. Ph.D. Dissertation, University of Cambridge, July (1980)

3. Tate, S.J., Jared, G.E.M., Swift, K.G.: Detection of symmetry and primary axes in support of proactive design for assembly. In: Bronsvoort, W.F., Anderson, D.C. (eds.) Proceedings of the Fifth Symposium on Solid Modeling and Applications, 9–11 June 1999
4. Parry-Barwick, S., Bowyer, A.: Is the features interface ready? In: Martin, R. (ed) Directions in Geometric Computing. Information Geometers Ltd., Winchester, pp. 129–160, ISBN 1-874728-02-X (1993)
5. Waco, D.L., Kim, Y.-S.: Geometric reasoning for machining features using convex decomposition. In: Proceedings of the Second Symposium on Solid Modeling and Applications, Montreal, pp. 323–332, May (1993)
6. Stroud, I., Renner, G., Xirouchakis, P.: A divide and conquer algorithm for medial surface computation. Computer-Aided Design **39**(9), 794–817 (2007)
7. Ranta, M., Inui, M., Kimura, F., Mäntylä, M.: Cut and paste based modeling with boundary features. In: Proceedings of the Second Symposium on Solid Modeling and Applications, Montreal, pp. 303–312, May (1993)

Chapter 11
Early-Phase Design

Design is a complex process about which much has been written and which covers a range of activities. It is not intended here to explain design as such, merely to explain how some of the information aspects should be transferred into CAD. This is not the same as having a PLM (Product Lifecycle Management) system, which is a different way of trying to coordinate product information. Product information integration means having different forms of computer support and making the information from these and from outside available within the CAD system. This chapter concerns what is termed here "early phase design", the initial steps to design a part.

Before describing the early phase of design it is, perhaps, useful to give a brief overview of the product lifecycle, as illustrated in Fig. 11.1, based on a classic reference for the design process by Pahl and Beitz. It is impossible to be definitive of all the elements in the lifecycle because of the huge diversity of products, hence only generalisations are discussed here. However, some common elements are shown.

In this simplified scenario, the product lifecycle starts with a market survey to establish that a product is needed. The product requirements phase sets out the role and expectations of the product based on where the product is seen to fit. This leads to a more formal definition of the requirements in terms of a product specification, which is taken as the start of the formal design phase here. The product specification leads to conceptual design, where a product solution is found and given initial geometric form in the embodiment phase. After the embodiment stage comes the detailed design stage, which is where traditional CAD is strong. The computer models from the detailed design stage may be analysed in various ways to check for strength. Once the models have been finalised the product enters the production stage and then the assembly of the components before being delivered to the user who uses it until it breaks, wears out or is no longer required, when it is disposed of.

Note that nowadays there is a tendency to at the whole cycle of a product, in that information and even physical parts come from the use and end-of-life phases back to the initial phases. However, the quantity and type of information is still being defined, hence the question marks. This will be dealt with later, in Chap. 14.

I. Stroud and H. Nagy, *Solid Modelling and CAD Systems*,
DOI: 10.1007/978-0-85729-259-9_11, © Springer-Verlag London Limited 2011

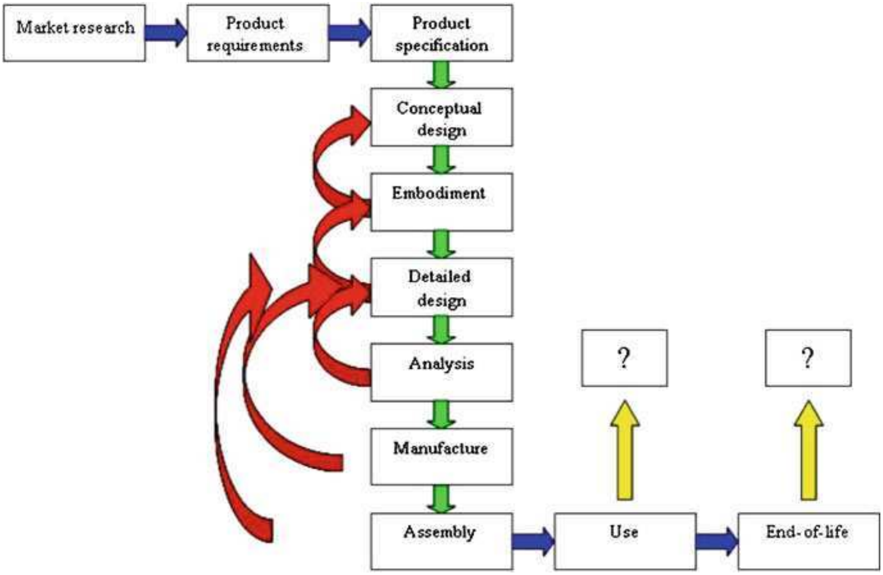


Fig. 11.1 Elements in the product lifecycle

That is the main flow in the product lifecycle. However, in reality, there are back flows, indicating that redesign is necessary. Each feedback costs time and money. The amount of money varies depending on where the problem is noted and how far back the change has to be made. For example, if a manufacturing problem is noted which can be fixed in the detailed design stage, for example adding edge roundings, then the cost may be small. However, if a problem is noted at the assembly stage which requires a change at the conceptual design stage, then the whole chain of embodiment, detailed design, analysis and manufacture has to be redone, maybe factory planning has to be redone, existing parts may need to be scrapped.

The general intention of early phase design research is to make design easier and hence help the designer to get the product right first time, so avoiding the feedbacks. Traditional design aids, like analysis, are classified as "reactive" because they take an existing shape and test it. In contrast, early phase tools need to be "proactive", helping the designer to find a solution. CAD systems currently tend towards the reactive methodology and deal with the detailed design phase and analysis, while many important decisions are taken prior to this in the early phase of design.

As stated above, traditional CAD concerns only the last part of the design process. Recently some tools have been added, in the form of PDM (Product Data Management) or PLM (Product Lifecycle Management) systems. While these may be useful for keeping track of and managing design documents, more advanced tools seem to exist only in research departments. Ideally, there should be design tools for creating, maintaining and integrating the early phase with the CAD system. This would mean that early phase decisions would be accessible by the designer in the detailed design phase. Such integration can really only be done by the commercial software developers, since it is necessary to modify the CAD system code. If there were to be development of an open-source CAD system to allow researchers to create their own CAD systems then this would set new goals for commercial systems. Until then, though, radical innovations seem likely to remain in the hands of the commercial developers. One suggestion, towards the end of this chapter, is the halfway step of creating design environments. Such environments could be imported manually or automatically into CAD systems to create information and geometry frameworks for detailed design.

The description here is not intended to be a survey of all methods, but rather illustrative of trends.

11.1 Tools and Techniques

The following are some of the tools and techniques used for design.

11.1.1 Expert Systems

Expert systems are a product of artificial intelligence research. The idea, put simply, is to recreate the workings of an expert by representing the expert's knowledge as a set of "if–then" rules in a database. The technique had some notable successes, the diagnosis of diseases in peanuts, for example, and diagnosis of human illnesses. An important rule-of-thumb for successful application is that the knowledge domain must be limited and reasonably small.

Expert systems can be applied in many domains in production, not only for design. This means that it may be possible to provide artificial experts as supports for the designer. For example, a manufacturing planning expert system may well be useful for a designer to check the "cost" of a design. The cost of a design may be measured in several ways, from simple manufacturing time to a complex environmental costing, such as done by Avram et al. [1], it depends on the complexity of the artificial assistant. This will be discussed further in Sect. 14.2.

An example of a manufacturing rule of the type in Jared's object oriented expert system is shown below:

```
if ($hole.radius \leq 10$ mm) then drill
else if ($hole.radius \leq 20$ mm) then { predrill; drill; }
else mill;
```

What this means is that if a hole radius is less than or equal to 10 mm then the hole can be drilled in one operation, if it is between 10 and 20 mm then it can be drilled with a smaller drill and then finished with a drill of the right size, otherwise it has to be milled.

The part is described for the expert system in terms of a set of manufacturing features, such as those described in Chap. 10, which were treated as "objects" in Jared's system. The features have a set of attributes, such as the radius (or diameter) of a hole, its depth, whether it is threaded or not, etc. Other features have other attributes.

11.1.2 Genetic Algorithms

A genetic algorithm is another problem solving technique that has been widely applied. The idea comes from biology, as the name suggests and, put very simply, the idea is that the solution evolves from an initial population. The technique has been used in many applications, not all of which are appropriate.

There is an initial population of possible solutions each of which is coded in such a way that the elements can be recombined or mutated to produce new solutions. There also needs to be some kind of evaluation function to allow new members of the population to be evaluated to see if they are better than their parents or fit to survive. In addition there has to be a culling strategy to weed out dead ends or less fit members of the population. The process stops after a certain number of generations have been generated.

Suppose that a preliminary solution to a problem is coded up as a set of strings, as on the top line of Fig. 11.2. For the next generation there is a set of transitions, such as "crossovers" or "mutations". Crossovers, as with people, is analogous to children inheriting properties from both their parents. Mutations can be thought of as random changes to strings to produce new elements. As the process proceeds some good solutions appear, which are kept, and bad solutions are removed ("die"). The process proceeds for a preset number of generations and the solutions

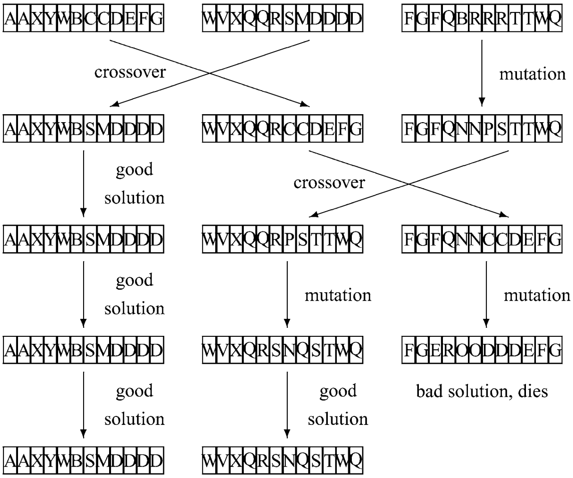


Fig. 11.2 Simple Genetic Algorithm (GA) illustration

remaining at the end evaluated to find the best one. Obviously this is a very simplified account of genetic algorithms, but this is not intended as a thorough treatment, only a brief overview.

So, then, what is the nature of the genetic algorithm method and when should it be used?

In my opinion, genetic algorithms are performing a sort-of "hill-climbing" procedure. The crossover technique aims to find solutions on a local hill, while the mutation technique looks to jump between hills to try to find a global optimum. The evaluation function is necessary to determine whether or not a solution is good or promising. This is also used to weed out solutions that are worse.

There is a large question about when to use a genetic algorithm, to which problems it can be applied. This is difficult to specify exactly. The technique is an interesting programming technique, but if a solution algorithm exists, or can be found, then it seems preferable to use the direct algorithm. Genetic algorithms were, at one time, fashionable. Fashion makes a graveyard of techniques, they become loosely applied and people are disappointed if they don't perform well. If the techniques don't perform well then it is the technique that is blamed, not its erroneous application. It is a pity if this happens because techniques can be better or worse for different problem areas. Genetic algorithms are interesting and can be applied profitably for certain problems. In the CAD domain they have been used for curve fitting, by Renner and Markus, for example. Curve fitting is a difficult problem with no good algorithmic solution, because the order of the curve is not defined, nor is the number of segments, positioning of knots or control points, etc. The validity of application in any particular domain has to be judged on a case by case basis.

11.1.3 Fuzziness

One aspect of early phase design is that there is uncertainty about a number of factors. This uncertainty is changed through decision making in different phases of the design process. In fuzzy mathematics, variables have a range rather than a single, or "crisp" value. This range also has distribution characteristics, as shown in Fig. 11.3. On the left (Fig. 11.3a) there is an even distribution, the fuzzy

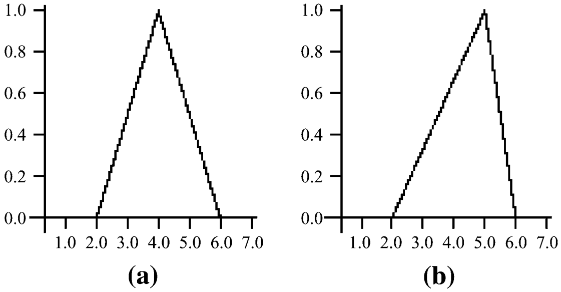


Fig. 11.3 Fuzzy distributions

variable could have any value between 2.0 and 6.0, with the value 4.0 as preferred, or most likely. The distribution need not be uniform, as shown on the right (Fig. 11.3b) where the preferred value is 5.0.

Mathematics with fuzzy numbers is, of course, more complicated than with crisp values. If you add the two variables in Fig. 11.3, what would the distribution of the result be? Don't worry about the question, a lot has been written on fuzzy techniques and fuzzy mathematics and this has been solved. The important point is to present the idea that values need not necessarily be certain.

For design, it is also important to be able to define approximate values, like "large", "medium" or "small" instead of numerical values, while software for model creation and visualisation needs numerical values. Fuzzy values define a sort of envelope for the geometry. This is difficult to arrange for traditional CAD systems with detailed geometry, but would be possible for the sort of layout system with simple geometry as described in Sect. 11.2.4.

11.1.4 Multi-Criteria Decision Making

In many applications there are many different, often competing, factors that have to be balanced. A simple example might be the weight of a part compared with its strength. If weight is not a factor then it is possible to "over-engineer" a part so that it is larger and thicker. However, this increases the weight, so when weight is also a factor there is a trade-off which must be considered.

Plotting two- and three-variable cases, as shown in Fig. 11.4, are possible but it is difficult to show in more dimensions. The lines may be curves, the scales may not be linear, but you should be familiar with this kind of representation.

A common way of coping with multi-criteria decision is to weight the different factors and assign them some score on this basis. The scores are added up and different alternatives compared by comparing the score. Most people know about

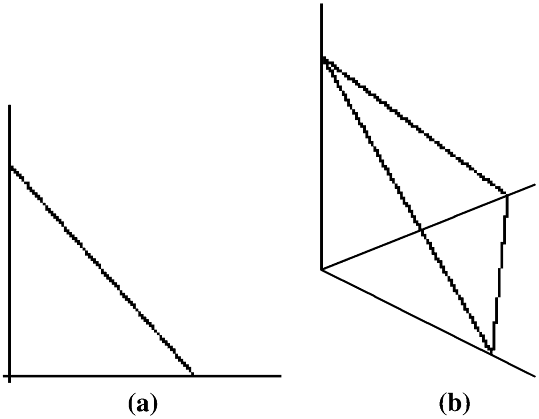


Fig. 11.4 Optimisation in two or three dimensions

this form of multi-criteria decision making. This is what happens when you buy some things like mobile telephones or cars. Different products have different options and the options add to the price. You might order a Swiss Army mobile phone with lots of tools or you might try and balance the different tool section to your needs.

11.2 Example Tools

There are various tools for design, some which have been implemented as software, others exist as techniques which could be implemented as software to provide a computerised support environment. Product Data Management (PDM) systems provide some help for keeping track of the textual information, but integration with software tools, especially CAD, CAPP and CAM, is limited.

The design procedure I will use as a basis for this section is that which is taught by Professor Horváth and Niels Moes from the department of Computer-Aided Design Engineering of the Technical University of Delft, and which I know from the E-GPR course. This may not be the only place where this methodology is taught and is certainly not the only methodology. However, this is my source of information and the method is clear and logical. Since this chapter is not really about design, but about tools for design I do not want to spend a lot of space on design techniques themselves. The reason for explaining the method here is so as to be able to put the tools and needs into a context.

The E-GPR course is an interesting learning experience being a course taught via video-conferencing techniques with six participating universities: Delft Technical University in the Netherlands, the Ecole Polytechnique Fédérale Lausanne in Switzerland, Ljubljana University in Slovenia, City University London in the United Kingdom, the University of Zagreb in Croatia and Budapest Technical University in Hungary. The students are divided into teams with members from different universities, so they are forced to adapt to cultural differences and learn to communicate with each other. The students are given a design problem by a partner company and are expected to produce a working prototype at the end of the course during a working week in which the students meet each other "in the flesh" for the first time.

The most successful courses, in my opinion, have been those in which the partner company outline the general problem and leave the students freedom to create something in that area. The project is helped by imaginative and helpful support from the responsible people in the partner company. The students are expected to go through a research phase, in which they learn about the problem and identify a particular sub-problem, a design phase in which they set up the functional requirements, a detailing phase during which they set the full details and a manufacturing phase during which they create a prototype.

The product idea is given by the company as a document which is made available to the students via a shared document area.

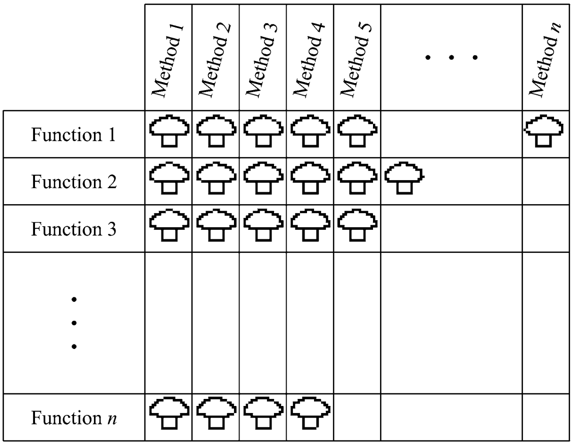
During the research phase the students use the internet as a means for research. The students use Skype or Adobe Connect for communication. There are shared whiteboard facilities for 2D sketching, but I have not yet seen 3D sketching facilities.

Once the initial research phase is over the students should have a set of functional requirements that guide further work. This set of functions is used for the creation of what is termed a “morphological chart”, a tool for focussing the students’ ideas. In the morphological chart, the product functions are written down vertically and the implementation methods for each function written horizontally, as shown in Fig. 11.5. There does not have to be the same number of methods for each function. The implementation methods, represented in the figure by little mushroom symbols, are sketched in the boxes. At this stage the students are expected to be creative and it is desirable if they include wild ideas in the methods. There is a filtering stage later during which the really hopeless ideas are weeded out, but if they do the filter too soon, during the morphological chart creation, they may miss a good idea. Science is catching up on science fiction and one of the team may know how to implement someone else’s wild dream. Niels Moes also stresses that the ideas generated are “team property”, that is, the students should not keep their ideas to themselves but put them into a common pool.

Once the morphological chart has been completed it is necessary to determine paths through this. Each path corresponds to a product variant. This is shown in Fig. 11.6 where a series of hypothetical paths have been sketched in. It is not obligatory to use different cells for every path, they can share cells, meaning that one of the functions can be fulfilled in the same way for two or more variants.

In the E-GPR course, once the products have been finalised, the students have to present their product variants and the partner company chooses one for each team to go further. With this product the students are then asked to produced what Niels Moes calls a “process tree”. This is a way of producing a checklists of

Fig. 11.5 Morphological chart



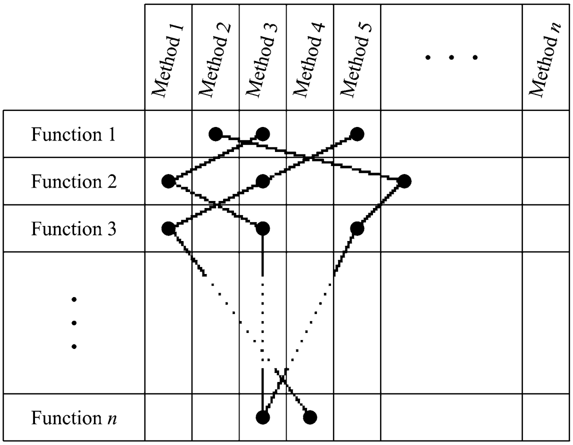


Fig. 11.6 Morphological chart with product paths

aspects for a product design in a series of phases. The word "process" means, in this context, a phase of the product life-cycle. Examples of these phases are the manufacturing phase, the assembly phase, the use phase and the end-of-life phase. The considerations for the phases should act as a way of checking the design to see how it fits in with the aims expressed in the process tree.

After this process the students perform the detailed design phase to create models of their designs.

It is clear that the following requirements need to be met for design tools:

1. Support "at a distance". E-GPR was set up as a course at a distance deliberately to expose students to the problems of working and communicating in geographically distributed places. However, supporting collaborative work is important even in the same company. There can be huge psychological, even if not geographical, distances between floors in the same office building. Good communication tools for synchronous activities are needed.
2. Document sharing facilities. By "document" I mean a variety of document types including text documents, pictures and CAD files. For companies this may involve security problems. A sort of library control system for taking out common documents to work on and putting them back and common viewing tools are needed.
3. Early phase design support. There are several possibilities, some explained below, for tools to help the critical early phase. Tools for sharing information and creating a common understanding within a design team are important to unify effort.
4. Visualisation sharing. 3D sketching would be nice. Although sketches are often 2D schematic diagrams, it is an unnecessary burden to sketch a 3D project when a solid is to be drawn. Having the possibility to create and place simple 3D shapes are needed as well as viewing tools to rotate the shapes, for example. Sketching in 2D exists, but needs to be made available for CAD as well.

11.2.1 Sketching Tools

There has been much work done on sketching tools for design. Back-of-envelope sketches and beer-mat sketches are important tools for designers, but they are limited, both in their audience and accessibility. Formal sketching tools to allow symbols to be sketched, lines and annotations in 2D are needed.

In 3D, simple shapes and lines can also help. This was the idea behind the development of generalised models by Professor Torsten Kjellberg and his team at KTH at the end of the 1970s and described in Chap. 6. Kjellberg et al. also pioneered the idea of a "product model", a model containing various types of information aimed at collecting the information from the initial steps in design.

Three-dimensional sketching is not as fluent as two-dimensional sketching, and is not the same as sketching a three dimensional object in two dimensions. Since the sketch medium, usually now paper, is two dimensional then sketching in two dimensions is natural. Even though three dimensional displays exist, and so three dimensional sketching is feasible, the process is not as natural as 2D sketching. This means that sketching in three dimensions requires an extra coordinate to be input, which breaks up the free flow of design. In one CAD system, called MEDUSA, at the end of the 1970s, three dimensional data was produced by picking in orthogonal view windows. This enabled the user to, say, define the *X* and *Y*-coordinates of a point in one window and then use a second window to define the *Z*-coordinate. The whole was arranged something like an engineering drawing.

Sketching in three dimensions, though, might involve something like the tools for layouts, described below, that is, definition of lines, curves and simple volumetric objects, placement tools and connection tools.

11.2.2 Functional Elements

The role of morphological charts and the process tree were described above. Both of these are easily amenable to distributed computer support. One of the important parts of the design process is to identify the functional decomposition of the problem. The functional elements are used for the morphological chart. The functional elements are determined by the design requirements.

Identification of the functional requirements is part of the expertise of the designer. It might possibly be helped by an expert system, or similar, if the product being designed is sufficiently specialised. However, this is unlikely and it is to be expected that the identification of functional elements is done by a human expert. The role of a computer-based tool is to allow the designer to record the decomposition and as many of the reasons behind the decomposition as possible. The information should be maintained in a clear format so that it can be accessed and used by other design support tools.

11.2.3 Partial Solution Databases

A powerful idea, pioneered by François Sprumont, to the best of my knowledge, is to build a database of partial solutions which can be consulted during design. This idea was implemented in two projects, MicroCE and ABCD. The MicroCE project is described below as an illustration of this technique. Further work was done by Duck-Young Kim on constraint satisfaction [2].

The partial solution database, or "design catalogue" starts with a functional element of the design and identifies possible solutions. The functional description, mentioned above, is a starting point. Certain of these functional elements may correspond to known solutions. The purpose of the tool is to find optimal known solutions for these known solutions.

11.2.3.1 MicroCE

The MicroCE project between several EPFL laboratories and the company Mecanex. Mecanex have a specialised product range of connectors for connecting rotating sensors, such as radar dishes, and fixed elements which use the data. Different applications need different sensors with different mechanisms. Customers came to Mecanex, filled in a data sheet with the characteristics of the sensor needed. The data sheet was then analysed by Mecanex's designers to produce a design and cost estimates.

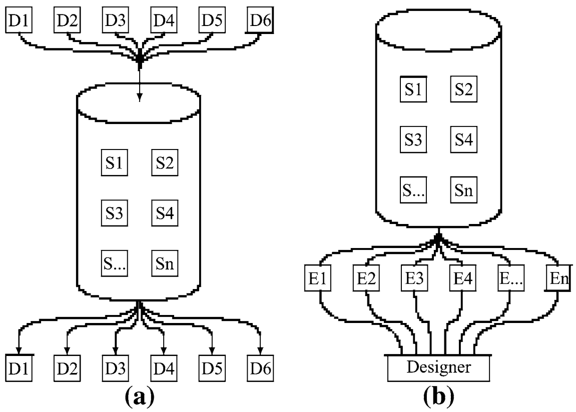
In the MicroCE project the aim was to produce a method, based on design catalogues, see below, to reduce the time for the design from three days, the normal time, to three hours.

11.2.3.2 Design Support Using Solution Catalogues

Design catalogues are a method of sharing company data between designers. They are collections of solutions to design sub-problems specific to the company. The design solutions from one designer are available for another designer, so that best solutions are shared. This is illustrated in Fig. 11.7a. All the designers put their partial solutions in the same common database. When a designer wants to select a partial solution, he or she evaluates the partial solutions and chooses the one which has the highest score, Fig. 11.7b.

There are several benefits about using design catalogues. One is that the company has a "knowledge resource", its principle solutions contained in a manner that is accessible to all. Another benefit is that designers can share their ideas and the design catalogue becomes a set of best-practices for a particular set of design problems. This also means that, if a designer is absent—on holiday, ill, or retired for example—the design solutions from that designer are still available. This is important for companies, especially SMEs, where the absence of

Fig. 11.7 Design catalogues



key personnel may be critical to success or failure. A design catalogue can also create company product standards. As a final example of a benefit there is that the identification of partial design solutions reduces the design work. This means that the designer is freed from routine work to concentrate on more critical problems.

There is a restriction to the use of design catalogues as there is to any support tool. Too much automation is not a good idea because, as pointed out by Nakazawa [3], this leads to standardised products and reduces a company's value. Nakazawa's comment was, put simply, that the design ability of a company is part of its value. If you share that throughout an industry by making it available via software then all companies can produce comparable products and so any particular company will lose its market edge. This is true of any design aid, not just design catalogues. It is important to plan properly in the application of all design tools and see them as support, not a replacement for human abilities.

In the Mecanex example, the design catalogue concerned rotational mechanisms for the sensors. François Sprumont also connected it to Csabai Attila's layout tool (see Sect. 11.2.4) and to special code to produce parametrised models of the sensors to make a more complete system. In a real system the designer would have completed the design with housing and other elements around the sensor.

In terms of the outline design process used for the E-GPR course, this tool is related to the morphological chart and would help in the selection of particular functional variants. The common database for the functional solutions allows sharing, even from geographically distributed sites.

11.2.3.3 Case Study

The company names are intended to be fictional. Any resemblance to an existing company is regretted.

Suppose that there is a company, called Okos Drive Motor Company, who make custom made motors for different companies. The have several different motors, electric, petrol, mechanical, steam, and so on. The size of these and the components depend on the requirements of the customer.

One of the customers of Okos Drive Motors is a company called Bolondok Incorporated who make a family of self-mobile shoes which can be customised according to user characteristics. The self-mobile shoe is a device like a shoe, worn on the feet, but which moves itself around without the user having to lift their feet. The product may be stupid, but this is to avoid conflict with real products. There are a number of solutions, such as:

Wheeled shoe	Good for large buildings, such as hospitals or shopping precincts (malls)
Lizard-legs shoe	A shoe which can be used outside on semi-rough terrain
Hover shoe	A shoe which can be used on rough terrain
Mud shoe	A shoe with a large surface area to be used over soft ground

A common component in these shoes is the motor, which is custom manufactured. The size and arrangement of the elements of the motor depend on the power required and the motor type. The client company, in this case Bolondok Incorporated, comes to Okos Motor Drives with the parameters of the required motor and Okos Motor Drives proposes a solution. In order to reach that solution, Okos Motor Drives evaluates its product range against the requirements, giving each a score.

Okos Motor Drives has the following solutions:

- Horizontal petrol engine. This is a petrol engine which is mounted horizontally with direct connection to the element being driven. The fuel tank is also part of the motor assembly.
- Indirect petrol engine. This is a petrol engine which is mounted at a distance from the driven element and connected via a flexible cable. As before, the fuel tank must also be considered as part of the motor solution.
- Horizontal electric motor. An electric motor which is mounted horizontally and connected directly to the element being driven. The battery pack placement has also to be considered as part of the assembly.
- Vertical electric motor. Similar to the horizontal electric motor, but there is an extra L-shaped drive element to connect the motor to the driven element. The battery pack is also a part of this solution.
- Indirect electric motor. The electric motor is mounted away from the driven elements and connected via a flexible cable. The battery pack is also part of this solution.
- Horizontal steam motor. In Okos Motor Drives catalogue, the motor must be mounted horizontally in connection with the driven element. The solution needs a water tank and fuel tank.

- Horizontal mechanical motor. This solution, too, is also only available as a horizontally mounted element, but has no need for separate power elements.

For this example, the evaluation is done in tabular form, as shown in Table 11.1. The final score is a weighted sum of the elements, with the weights being determined from the design requirements, here they are: 5 for power, −2 for weight and 2 for power source. The elements in the other columns are normalised according to how well they fit the task.

In this very artificial example, the vertical and horizontal mounted electric motors score the best with the horizontal petrol engine not far behind. Note that the weighting function is a way of method for *multi-criteria decision making* (see Sect. 11.1.4). The weights are used to combine a number of different elements into a single score for comparison. An alternative is to keep the different criteria separate and compare the criteria categories with each other. This is a complex topic and, here, the weighting function is used for simplicity.

The result of this simple example is that the Okos Motor Drive company can propose three solutions: the vertical and horizontal motor solutions and the petrol engine. If the correct dimensions of the elements can also be supplied then Bolondok incorporated can complete their design while Okos build and supply the motor. In the same way as done for the Mecanex example by Sprumont, the dimensions could be supplied in terms of gross dimensions and connection elements for a layout design, as described in Sect. 11.2.4.

This small artificial example is intended only to outline the basic principles of the use of design catalogues. A real application example and evaluation would be more complicated and also contain part of the company knowledge.

11.2.4 Layout Design

An interesting system for layout design was developed by Attila Csabai at the EPFL [4]. Csabai developed a system for producing a product structure with kinematic connections that could be used for problem decomposition and planning, simulation as well as providing an important support for design.

Table 11.1 Motor evaluation table

Motor	Power	Weight	Power source	Score
Horizontal petrol	1.0	0.7	0.5	4.6
Indirect petrol	0.95	0.8	0.5	4.15
Horizontal electric	1.0	0.8	0.65	4.7
Vertical electric	1.0	0.8	0.65	4.7
Indirect electric	0.8	0.9	0.6	3.4
Horizontal steam	0.2	0.2	0.1	0.8
Horizontal mechanical	0.1	0.3	0.8	1.5

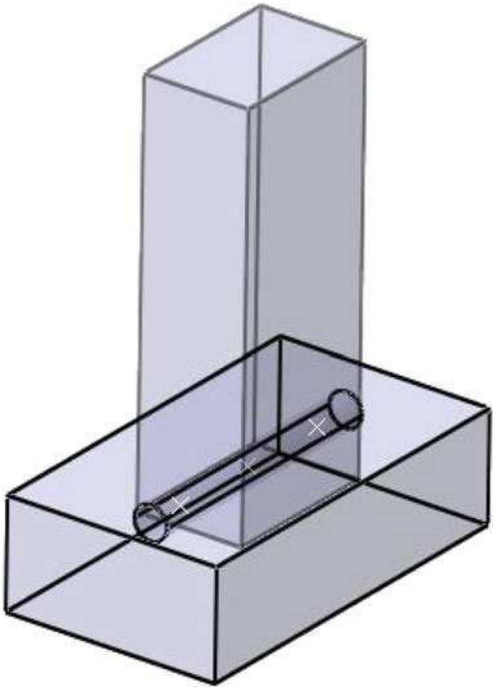
In Csabai's system, the functional elements of the product, identified by the designer, are represented as a connected set of *design spaces*. Each design space has its own coordinate system, with reference to which the elements are put, and a primitive shape. The design spaces were linked together using kinematic connections which allowed movement simulation. The primitive shapes in Csabai's system were rectangular blocks, cylinders and simple extruded shapes. However, the primitive shapes were not intended to be the same as the final object, merely geometric placeholders, so the simple extruded shape was not much used because of possible confusion with the final shape. Csabai also allowed the primitive design spaces to include the real objects, once these had been designed, so that the layout could also be used to visualise the assembled product.

To illustrate the concept, consider Fig. 11.8. The picture and others have been faked using CATIA as it is not currently possible to get pictures from Csabai's original system.

The layout consists of three elements, a base, an axle and a rotational part. The layout elements are not intended have the final shapes. This allows simple simulations with elements that can be changed easily. The kinematic simulation allows product demonstration both for a client as well as for members of a design team to create a common understanding of a product.

In Fig. 11.9 one element, the base element has been defined. In this example, the top two prongs extend just outside the primitive geometry of the base design space. If this happens in a real layout then the need for extra space is the subject of

Fig. 11.8 Simple layout



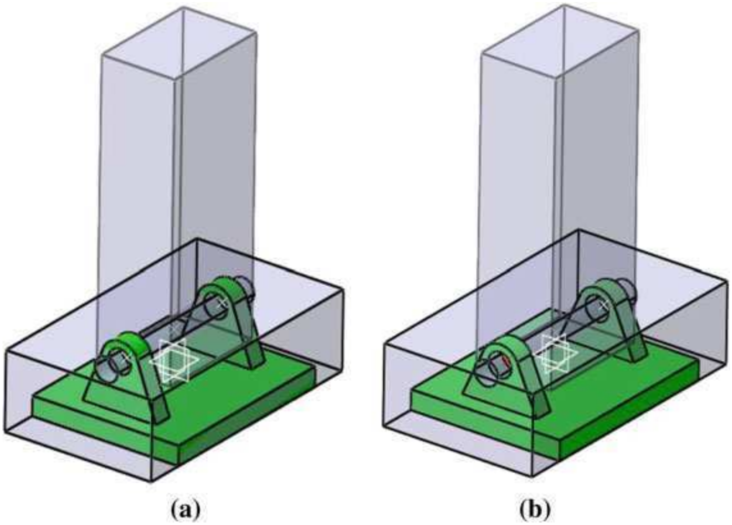


Fig. 11.9 Simple layouts with component

negotiation. Here, the extension does not interfere with any other design space so it might be allowed. Alternatively, as on the right of the figure, the connections may be modified so that the real part can be modified to lie within the design space. If there is a conflict, though, then the designers responsible for the two interfering design spaces need to negotiate as to how to resolve the common space. In Csabai's terms, collisions between primitive shapes did not indicate an error, merely a volume of space which needed to be negotiated. The primitive shapes were regarded as "soft solids" rather than "hard solids" where a collision implies an error.

The layout tool created by Csabai is also a support for problem solving. The first level of design spaces correspond to the functional elements identified by the chief designer. This is illustrated in Fig. 11.10. At the top is a single node corresponding to the whole product. On the second level are the functional elements, as shown in Fig. 11.10a. In Csabai's philosophy, each of the functional elements can be further subdivided, if necessary, into simpler sub-elements, as shown in Fig. 11.10b, which can be solved by a designer or broken down even further. This method allows successive reasoning steps about the product element for creation of a well structured design. This could be done by a single chief designer or allocated to a design team for elaboration. This method also leads to a coordinated design methodology which can be valuable for distributing work both locally and in a geographically distributed environment. In the end, the structure of the layout corresponds to the product structure and could, in theory, be exported directly to a CAD system.

Another aspect of Csabai's work was to create a collaborative design environment. In Fig. 11.11, also simulated, the base is shown solid in green with a connecting part shown transparently in red. A user is able to see both their own element in an assembly as well as a connecting part. A second designer might see the view in Fig. 11.12, with views of designer 1's part and designer 3's part as

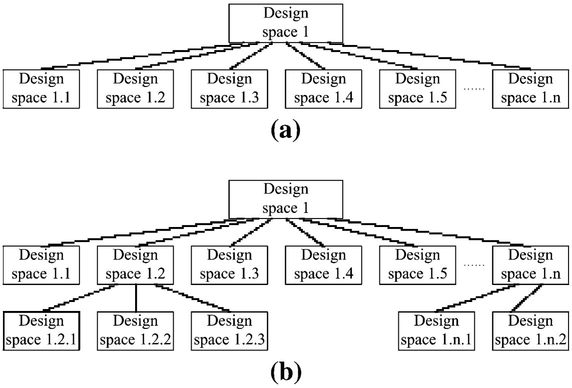


Fig. 11.10 Layout subdivision

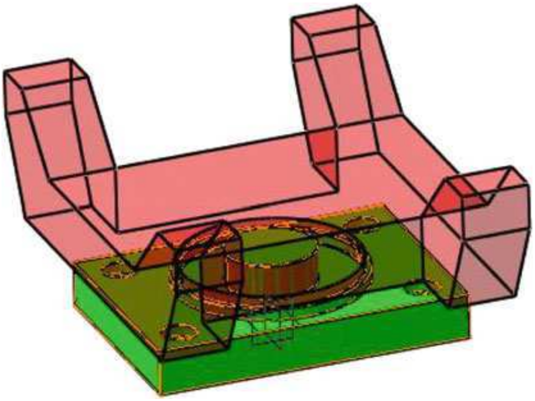


Fig. 11.11 Shared assembly

well. This makes it easier for the users to understand how their component fits together with other parts.

For the designer responsible for the rotating part there would be different connecting elements and the base and the other element would be shown as transparent, with the rotator solid.

Another aspect of Csabai's work was the notion of common features. This is illustrated, again with faked images using CATIA, in Figs. 11.13 and 11.14.

The base and rotator share a common rotational joint. By establishing common joints in the layout, Csabai was able to introduce them as feature pairs in the detailed design.

11.2.4.1 Layout Aspects

Some of the interesting aspects, in no particular order, of Csabai's work are:

1. Design spaces and placeholders for geometry.

Fig. 11.12 Shared assembly

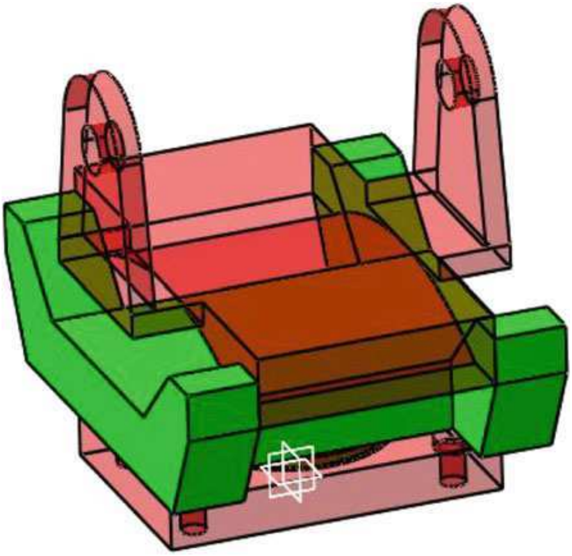
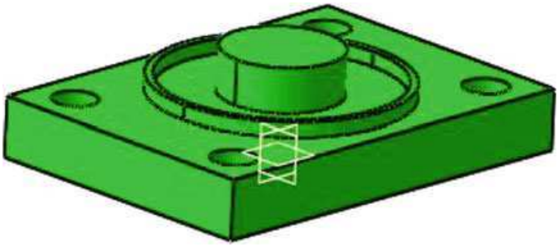


Fig. 11.13 Base with shared feature



2. Rough kinematic simulation.
3. Problem solving through decomposition.
4. Design distribution.
5. Collaborative design environment.
6. Soft solids.
7. Multi-body features.

Design spaces and placeholders for geometry. A design space is, basically, a local coordinate system. Assigned to each design space in Csabai's work was a simple geometric shape, rectangular block or cylinder, though he also considered simple swept shapes. The geometric shape is not intended to be the same as the final geometric element in the model, merely to be a simple placeholder for visualisation. At the same time, the simple shape gives boundary limits for the final design to act as a framework for the detailed design.

Rough kinematic simulation. In Csabai's system kinematic connections were defined between elements, allowing simple kinematic simulation of the final product. Such a simulation provides all participating designers with an

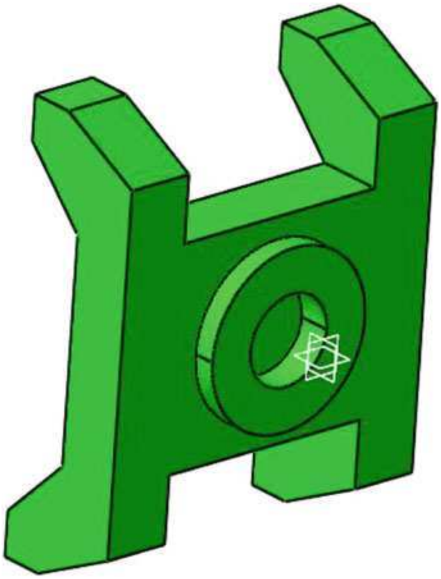


Fig. 11.14 Rotator with shared feature

understanding of how the product is intended to work and hence the functionality of the subparts or submodules.

Problem solving through decomposition. Csabai's work involved the same sort of problem solving style as found in Jackson's Structured Programming method, but with a geometric element. Many people do this almost automatically and the initial problem decomposition will probably exist as the functional decomposition of the product. However, Csabai's method allows the refinement of this first decomposition to identify sub-modules and product elements. The resulting structure also corresponds to a final product assembly structure.

Design distribution. A consequence of the decomposition method is that design spaces could be assigned to designers or design teams. This was the intention of Csabai and is a useful problem management tool. The connections between the modules acted as common points for communication to ensure consistency.

Collaborative design environment. Csabai conceived his system as a web-based system, allowing designers to collaborate in a distributed environment. The way that Csabai imagined the system to work is that the chief designer creates an initial layout with a set of design spaces, based on a functional decomposition and then assigns these to members of the design team. The system maintained a database, noting the person responsible, links to solid models, security and so on.

Soft solids. Because the geometric shapes in Csabai's system were placeholders for the final geometry, they were not considered to be rigid, final solids. This means that collisions between the geometric shapes during kinematic simulations are not errors, merely warnings that there is shared space. The collision volumes then become areas for negotiation between different designers to make sure that the final detailed geometry does not collide.

Multi-body features. Csabai allowed geometric features to be attached to connections between design spaces. This, in my view, is an important advance over the traditional perception of features, that is, that they are isolated elements in single objects. In Csabai's system, changing the feature meant that the change was propagated to both design spaces sharing the connection, maintaining consistency. Csabai also recognised that a particular kinematic mechanism could be implemented in different ways, which could be negotiated. For example, a rotational joint might be implemented as a pin-and-hole joint. However, in which part is the pin and in which the hole is not defined and open to negotiation between designers.

11.2.4.2 Example

An example of the use of Csabai's system, from Csabai et al. [5], concerns a hypothetical robot design sequence. Figure 11.15 shows a traditional, hypothetical, design sequence. Figure 11.16 shows another hypothetical design sequence using design spaces.

The designer, or design team starts by producing a detailed model of the geometry. The model is then animated to produce a kinematic simulation of the final product. Based on this, design modifications are made by changing the detailed geometry. The product is analysed to check its physical characteristics. The design is then modified, to reduce the weight of the arm, for example, while maintaining its stiffness. A new simulation reveals that the robot cannot reach certain areas so the upper arm is lengthened. A collision is also found and the arm

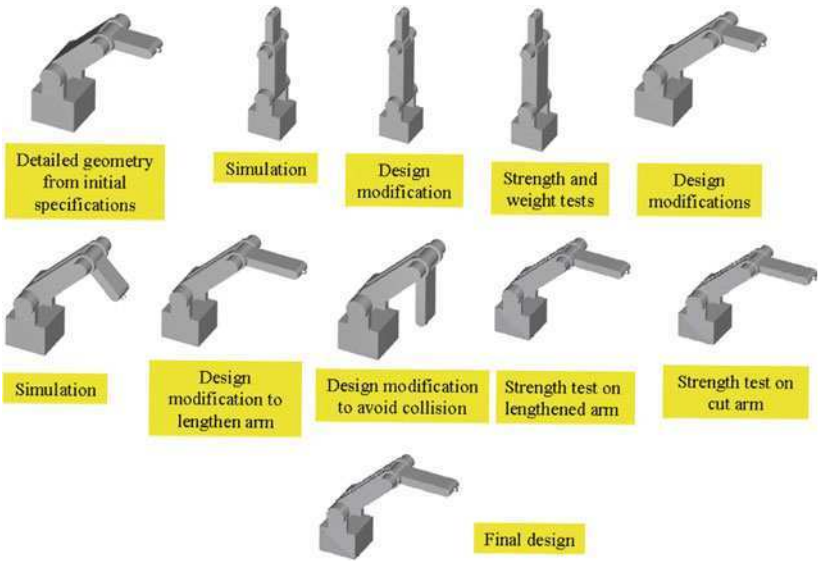

Detailed geometry from initial specifications
Simulation
Design modification
Strength and weight tests
Design modifications
Simulation
Design modification to lengthen arm
Design modification to avoid collision
Strength test on lengthened arm
Strength test on cut arm
Final design

Fig. 11.15 Traditional robot design with exact models (from [5])

shape modified to remove the collision. New strength and weight analyses are performed and eventually the final design is reached.

This is a hypothetical example of how a design might develop. With extended design teams and complicated products there is an increased risk of design disparity. Decisions about details of the design may be taken arbitrarily. A layout tool is not a magic solution but it does provide a simple visualisation means which can help to provide a common understanding of a product.

The designer begins by defining the main elements of the robot as rectangular blocks of the desired size and defines the kinematic connections between them. The initial primitives are animated and tested as to whether the robot can reach a particular height. The layout is modified by changing the lower arm design space primitive. This is a simple parametric change since the primitive shape is a rectangular block. A new test on reachability reveals that the robot cannot reach down low enough, so the upper arm primitive is changed, again a simple parameter modification. The collision volumetric parts between different design spaces are noted as area where care must be taken. When the layout modifications are complete the final design is produced. Simple analyses on the primitive elements can also lead to constraints such as for weight and stiffness which can then be communicated to a CAD system for detailed design.

As stated above, layout modelling is not a magical solution, it is a method for organising work and building knowledge. The possibility of animating rough shapes is a tool for augmenting comprehension about the intention of the whole product (Fig. 11.16).

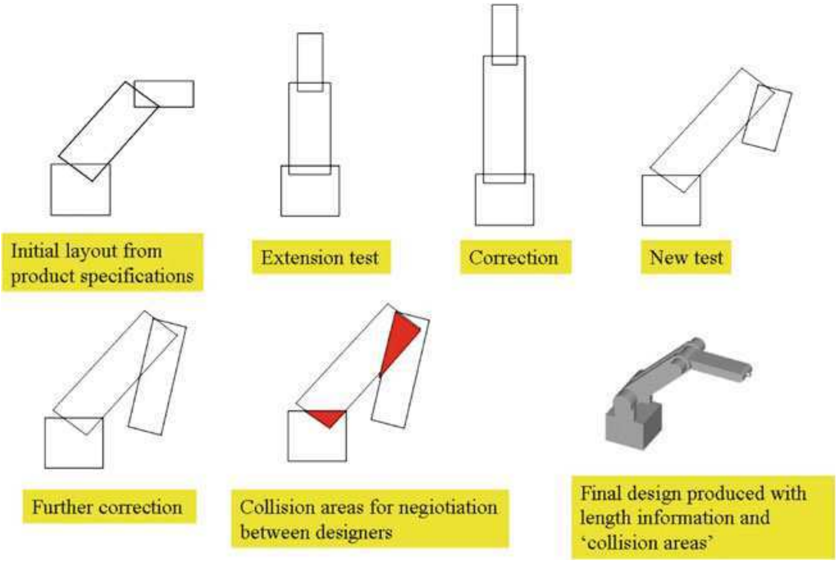

Initial layout from product specifications
Extension test
Correction
New test
Further correction
Collision areas for negiotiation between designers
Final design produced with length information and 'collision areas'

Fig. 11.16 Initial phase robot design with layout models (from [5])

11.3 Design Environments

The design environment is another idea *waiting in the wings* for an entrance into the CAD world. The design environment is a mechanism for communication of early-phase design to current CAD systems in the detailed design phase. The idea is to use the macro-language programmable facilities in a CAD system to import geometric elements from a file as help-, or construction-, geometry. This is a simple compromise which, while far from ideal, does extend the support by CAD systems.

An example of an environment file, for the layout in Fig. 11.8, might be:

```
webdes_rotator_group.htm <assembly> 3 6
<block> 1 webdes_rotator_base.htm 140 80 40 fix
<connection> 1 rotation 2 0 360 (−35,0,5) (1,0,0) (0,0,−1)
             3 rotation 4 0 360 (35,0,5) (1,0,0) (0,0,−1)</connection>
</block>
<cylinder> 2 webdes_rotator_axle.htm 6.25 100 free
<connection> 2 rotation 1 0 360 (0,0,−35) (0,0,1) (1,0,0)
             4 rotation 3 0 360 (0,0,35) (0,0,1) (1,0,0)
             5 static 6 0 0 (0,0,0) (−1,0,0) (0,0,1)</connection>
</cylinder>
<block> 3 webdes_rotator_top.htm 60 40 145 free
<connection> 6 static 5 (0,0,−70) (0,0,1) (1,0,0)</connection>
</block>
</assembly>
```

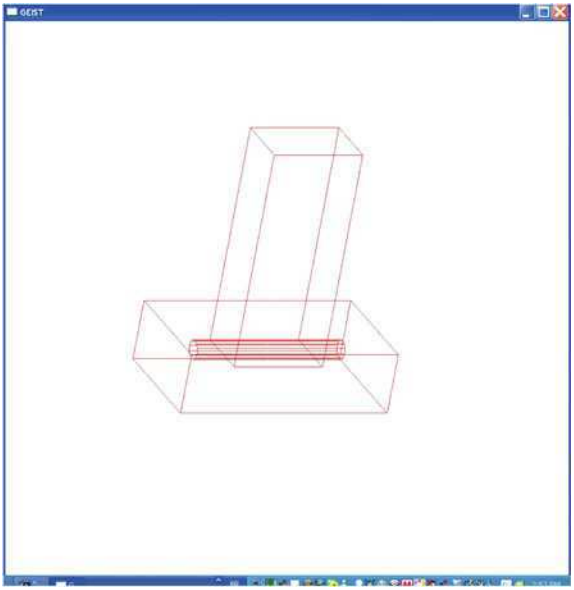
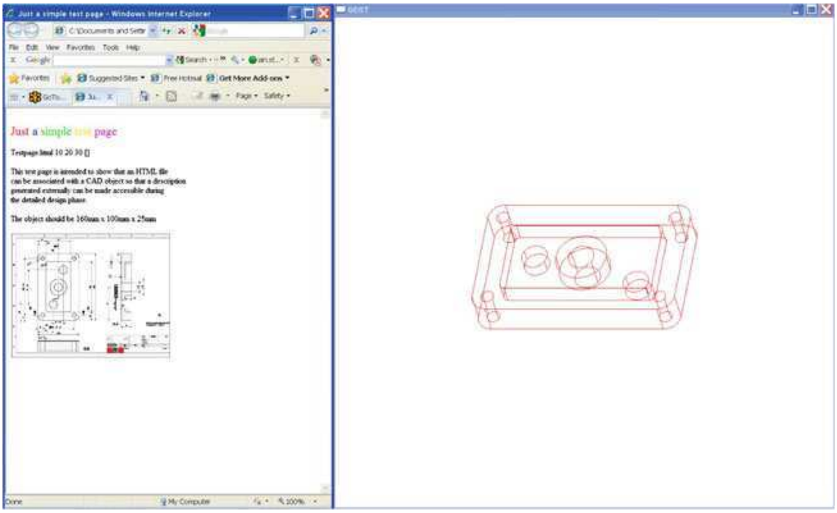


Fig. 11.17 Layout input from a design environment file


Just a simple test page - Windows Internet Explorer
File Edit View Favorites Tools Help
Favorites
Suggested Sites
Free Hotmail
Get More Add-ons
Page
Safety
Done
My Computer
100%

Just a simple test page

Testpage.html 10 20 30 []

This test page is intended to show that an HTML file can be associated with a CAD object so that a description generated externally can be made accessible during the detailed design phase

The object should be 160mm x 100mm x 25mm

Fig. 11.18 Associating an HTML document with a geometric model

When read in and the design spaces aligned, this gives a layout such as that shown in Fig. 11.17.

The purpose of the filename is to be able to link external documents, e.g. HTML or any of the familyML documents, with models. An example is shown in Fig. 11.18.

In the design environment, above, HTML filenames are given as part of the design space descriptions. The idea behind this is that external software could create an information environment, in a well-known and accessible format, which could then be accessed from inside a CAD system to provide more information for a designer. In addition, the designer would be able to traverse a structured information environment, possibly adding extra information or changing parameters. Although relatively simple and cheap to implement, the creation and use of design environments is lacking in modern CAD.

11.4 Proactive Tools

A lot of interesting work has been done on proactive methods by Jared, Swift and others at the universities of Cranfield and Hull. This is interesting not only because of the ideas but also because of the quality and high level of knowledge of the people behind them.

In the same way as the other early phase design techniques already mentioned, the aim is to get the design right first time and avoid expensive redesign steps. Some examples of papers on this topic are [6–11]. An important part of this work is the proactive nature to find a good solution in the initial phase of design and during the design process. The idea is not to complete the design and then analyse it, but to help the designer find good solutions.

The series of articles above are about an experimental design environment called the "Sandpit". The background to the work, described in [9], is that about three quarters of the product cost is determined by decisions during the design stage and that a large proportion of this results from assembly. The system analyses products at three levels: product group, product structure and component design. The system aims to help the designer right throughout the design process. With this methodology the user is both aware of assembly and guided while designing. This is the nature of proactive tools, to help and support the design process to arrive at a correct, or near correct solution rather than forcing expensive redesign when errors are found late.

11.5 Chapter Summary

This chapter deals with the important area of design in the early phase. The aim is to put the CAD tools into a context, to show obvious holes in current computer support for the design process. An example design process is described, based on that followed by Professor Horvath's group at the Technical University of Delft. This is used as background for the descriptions of tools for the early phase of design. The chapter ends with a short description of an interesting proactive system, the Sandpit by Jared, Swift and others, for supporting assembly.

11.6 Early Phase Design Exercises

11.6.1 Functional Requirements

What would be the functional requirements for a coffee machine? What are the functional elements of the coffee machine?

11.6.2 Coffee Machine Design

Design a coffee machine.

Suppose that the principle elements of a filter coffee machine with their approximate sizes are:

- Water reservoir (100 × 200 × 200).
- Stand (300 × 200 × 200).
- Hot plate (250 × 120 × 50).

- Jug (80 × 80 × 100).
- Filter holder (80 × 80 × 80).

Create a new assembly in your CAD system and represent the elements using rectangular blocks. Identify the inputs/outputs of these elements and their relative positions with respect to each other. Model the connections between the elements.

Now design the coffee machine using the rectangular blocks as guides and use the models of the connection elements to create the interfaces between the objects.

11.6.3 Morphological Chart for a Coffee Machine

The morphological chart is a tool for identifying interesting design variants. Produce a morphological chart for a coffee machine with the following elements:

Power: Mains supply; Battery; Human powered dynamo; Solar cell
Water source: Built-in reservoir; Bottled water; Mains supply
Coffee holder: Filter unit; Preloaded capsules;
Coffee receptor: Jug; Individual Cup; Multiple disposable cup unit
Sales gadget: Radio; Timer; Alarm signal when ready; MP3 player; Game player; None
How do you produce product variants from a morphological chart?
What would be your favourite combination?
What would be a successful product?
What would be a combination for an environmentally conscious household?

11.6.4 Functional Decomposition

Identify the functional units for a product such as a vacuum cleaner.

References

1. Avram, O., Stroud, I., Xirouchakis, P.: A multi-criteria decision method for sustainability assessment of the use phase of machine tool systems, Int. J. Adv. Manuf. Tech. 53(5–8), 811–828 (2011)
2. Kim, D.Y.: Development of a collaborative conceptual design support system. Ph.D. Dissertation, EPFL (2006)
3. Nakazawa, H.: Human-Oriented Manufacturing System (HOMS)—Does it suggest a new type of robot? IEEE Robotics and Automation Magazine, pp. 12–17, December (1998)
4. Csabai, A.: Layout modelling and evaluation methods and tools. Ph.D. Dissertation, EPFL, STI-IPR-LICP, Switzerland (2003)
5. Csabai, A., Stroud, I., Xirouchakis, P.: Container spaces and functional features for top-down 3D layout design. Comput. Aided Des. **34**, 1011–1035 (2002)

6. Barnes, C.J., Dalgleish, G.F., Jared, G.E.M., Swift, K.G., Tate, S.J.: Assembly sequence structures in design for assembly. In: Proceedings of the IEEE International Symposium on Assembly and Task Planning, ISATP 97, pp. 164–169 (1997)
7. Barnes, C.J., Dalgleish, G.F., Jared, G.E.M., Mei, H., Swift, K.G.: Assembly oriented design. In: Proceedings of the IEEE International Symposium on Assembly and Task Planning, ISATP 99, pp. 45–50 (1999)
8. Tate, S.J., Jared, G.E.M., Swift, K.G.: Detection of symmetry and primary axes in support of proactive design for assembly. In: Bronsvoort, W.F., Anderson, D.C. (eds.) Proceedings of the Fifth Symposium on Solid Modeling and Applications, 9–11 June 1999
9. Tate, S.J., Jared, G.E.M., Brown, N.J., Swift, K.G.: An introduction to the designer's sandpit. In: Proceedings of DFM 2000. Design for Manufacturing, Baltimore (2000)
10. Barnes, C.J., Jared, G.E.M., Swift, K.G.: A pragmatic approach to interactive assembly sequence evaluation. J. Eng. Manuf. **217**(4), 541–550 (2003)
11. Toro, C.R., Jared, G.E.M., Swift, K.G.: Product development complexity metrics: a framework for proactive DFA implementation. In: International Design Conference—Design 2004, Dubrovnik, pp. 483–490 (2004)

Chapter 12
History, Parametric Parts and Programming

The notion of maintaining knowledge about the object being built flourished at the end of the 1970s and the beginning of the 1980s. A Swedish idea, from Kjellberg, Eriksson et al. was what was termed "Uppkomstbeskrivning", a description of the definition of the origin of object parts. This has, to some extent been replaced by the history tree.

12.1 History Structures

It is common practice to maintain a list of operations used to make an object and allow this to be edited or rerun, called sometimes the "Construction History". This provides a way of remaking an object so that the user can edit input parameters and make new versions of the object. This idea may have evolved out of earlier practices of model creation, at any rate it is a useful facility in modern CAD systems. The idea has been extended to allow parametrisation of objects as well as direct manipulation of the history structure.

To put this into perspective, I give here examples of earlier methods. In early research it was common to write small command files to make objects and to run these through a command interpreter for object creation. That was before interactive graphics and other developments made it possible to create objects interactively. When interaction became easier another idea which became standard was to create a log file, a list of the operations used by a designer to create a shape.

I think that the notion of parametrisation grew out of these, but since I do not know the exact history about how parametrisation and history trees came about I cannot attribute the credit and will follow this viewpoint.

I. Stroud and H. Nagy, *Solid Modelling and CAD Systems*,
DOI: 10.1007/978-0-85729-259-9_12, © Springer-Verlag London Limited 2011

12.1.1 Command Files

An example of a command file, from 1979 for the BUILD research system [1], is given below. The original code is on the left, with comments in italics on the right.

```
comment Gehäuse (Rohteil) 23446   02 using construction lines
prompt
echo
2.5d                              Planar shape creation on the XY plane with
                                  global origin, package developed by
                                  Graham Jared
point 1 (0,6.4,0)                 Create point 1
point 2 (0,−6.4,0)                Create point 2
line 1 start p 1 dir vi           Create line 1 through point 1 in the
                                  direction (1,0,0)
line 2 start p 2 dir vi           Create line 2 through point 2 in the
                                  direction (0,1,0)
circle 1 centre (0,0,0) radius 6.4  Create a circle centre at the origin with
                                  radius 6.4
point 3 (28.9,6.4,0)              Create point 3
point 4 (22.5,3.4,0)              Create point 4
point 5 (19.5,2.4,0)              Create point 5
line 3 start (0,0,0) dir vi       Create line 3 through the origin in the
                                  direction (1,0,0)
line 4 start p 3 dir vj           Create line 4 through point 3 in the
                                  direction (0,1,0)
line 5 start p 4 dir vj           Create line 5 through point 4 in the
                                  direction (0,1,0)
line 6 start p 5 dir vj           Create line 6 through point 5 in the
                                  direction (0,1,0)
line 7 start p 4 dir vi           Create line 7 through point 4 in the
                                  direction (1,0,0)
line 8 start p 5 dir vi           Create line 8 through point 5 in the
                                  direction (1,0,0)
drawstyle dotted,label,-labf      Set drawing options for dotted, labelled
                                  construction geometry and
                                  no labelled faces
line 9 start (13.5,0,0) dir vj    Create line 9 through a given position in the
                                  direction (0,1,0)
line 10 start (0,−2.9,0) dir vi   Create line 10 through a given position in
                                  the direction (1,0,0)
line 11 start (7.5,0,0) dir vj    Create line 11 through a given position in
                                  the direction (0,1,0)
line 12 start (−5.7,0,0) dir vj   Create line 12 through a given position in
                                  the direction (0,1,0)
```

point 6 lxl 1 3 1 9 — *Create point 6 at the intersection of line 3 and line 9*

point 7 (10.6,−2.9,0) — *Create point 7 at a given position*

point 8 lxc − 1 12 c 1 — *Create point 8 at the intersection of line 12 and c1. The − sign is used to change the direction of line 12 for the intersection to give one of the points*

circle 2 centre (10.6,0) radius 2.9 — *Create a circle at a given centre with radius 2.9*

draw — *Produces the image in Fig. 12.1*

The 2D shape definition tool, created by Graham Jared, was an important advance in the BUILD system because it made it easy to define complex shapes. The first step in shape definition is to set up a set of “construction lines” which guide the shape creation. These construction lines can be points, lines or circles. The analogy is that these were like the pencil lines drawn by a designer before inking-in the final shape and erasing the pencil lines. The lines and circles have a direction which can be used to distinguish between intersection results. For example, in the line: “point 8 lxc − 1 12 c 1” there are two lxc (line intersect circle) results between line 12 and circle 1. Normally the one with the lower parameter value on the line would be chosen. The minus sign changes this so that the point with the higher parameter value is given as the result.

start p 8 — *Starts the “inking-in” procedure putting in edges and vertices. The start vertex is at the position defined by point 8, approximately at (−5.7,2.9)*

extend to lxc l12 c1 — *Create a new edge in a straight line to the intersection of line 12*

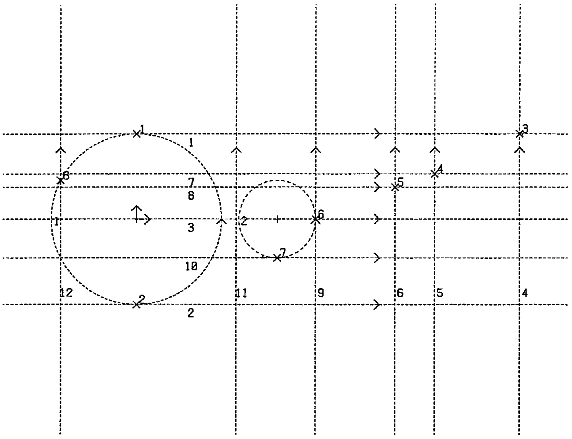


Fig. 12.1 Construction lines for MBB Gehäuse Rohteil

	and circle 1 (approximately at (−5.7,−2.9))
extend round c 1 to lxc l2 c1	*Create a new edge round circle 1, in counter-clockwise direction, to the intersection of line 2 and circle 1, to (0,−6.4)*
extend to lxl l11 l2	*Create a new edge in a straight line to the intersection of line 11 and line 2, at (7.5,−6.4)*
extend to lxl l10 l11	*Create a new straight edge to the intersection of line 10 and line 11, at (7.5,−2.9)*
extend to lxc l10 c2	*Create a new straight edge to the intersection of line 10 and circle 2, at (10.6,−2.9)*
extend round c2 to lxc l9 c2	*Create a new circular edge round c2 in counter-clockwise direction to the intersection between line 9 and circle 2, to (13.5,0)*
extend to lxl l8 l9	*Create a new straight edge to the intersection between line 8 and line 9, at (13.5,2.4)*
extend to p5	*Create a new straight edge to point 5, which could also have been defined dynamically as lxl l6 l8, at (19.5,2.4)*
extend to lxl l6 l7	*Create a new straight edge to the intersection between line 6 and line 7, at (19.5,3.4)*
extend to p4	*Create a new straight edge to point 4 at (22.5,3.4)*
extend to lxl l2 l5	*Create a new straight edge to the intersection between line 2 and line 5, at (22.5,−6.4)*
extend to lxl l2 l4	*Create a new straight edge to the intersection between line 2 and line 4, at (28.9,−6.4)*
extend to p3	*Create a new straight edge to point 3 at (28.9,6.4)*
extend to lxc l1 c1	*Create a new straight edge to the intersection between line 1 and circle 1, at (0,6.4)*

extend round c1 to p8	*Create a new circular edge around circle 1 in counter-clockwise direction to p8*
end	*Creates a closed shape by joining the first and last vertices*
drawstyle -circles, -lines, -points, -autoscale	*Set the drawing styles*
draw	*Produces the image in Fig. 12.2*
quit	*Leaves the sketching package*

The inking-in procedure creates the first geometric model which will be added to in the next steps. It is a planar model with the exterior profile of the shape to be made. It is possible, at this point, to extrude the shape and create a first solid model, but in BUILD it was also possible to define a planar model with all the interior faces defined and then to extrude these. This is closer to drafting practice in design.

set view vk	*Set the eye point to be above the object*
drawstyle man	*Set the drawing style to manual*
set scale 0.2	*Set the drawing scale*
setcentre	*Set the centre of drawing to the centre of the object*
draw	*Not shown as figure*
2.5d face 1 originv 14	*Define a shape on face 1 (the top face) using vertex 14 as origin. All positions are defined relative to this vertex, which is marked with axis directions in Fig. 12.2*
drawstyle dotted, label, -labf	*Set the drawing styles*
line 1 start (−6.4,0.0) dir vj	*Define line 1 with direction (0,1,0) through a point relative to vertex 14—in global coordinates at (22.5,6.4,0)*

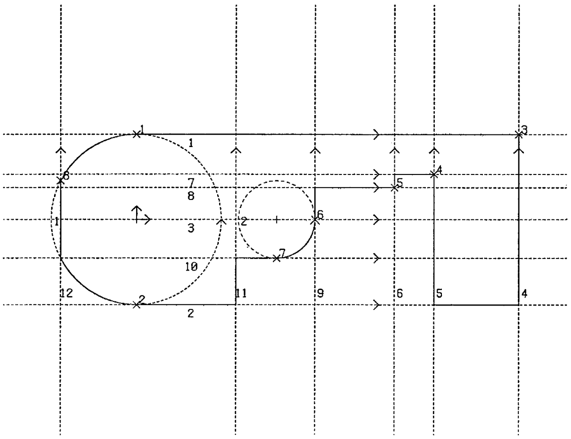
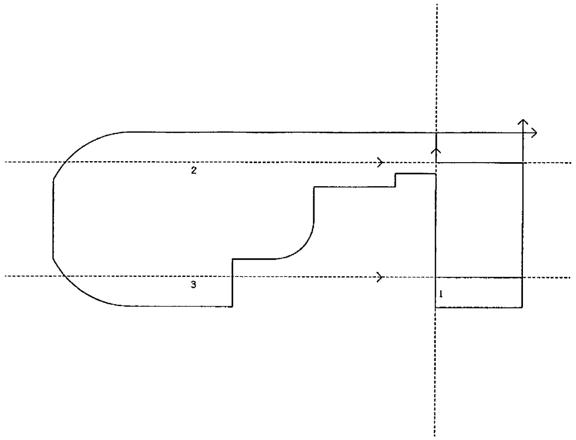


Fig. 12.2 Inked-in basic shape

line 2 start (0,−2.2) dir vi	*Define line 2 with direction (1,0,0) through a point with position relative to vertex 14—in global coordinates (28.9,4.2,0)*
line 3 start (0,−10.6) dir vi	*Define line 3 with direction (1,0,0) through a point with position relative to vertex 14—in global coordinates (28.9,−4.2,0)*
start lxl l2 e13	*Start the first prong shape at the intersection of line l2 and edge 13, breaking edge 13 at the same time, at (22.5,0,0)*
extend to lxl l1 l2	*Create a new straight edge to the intersection of line 1 and line 2, at (22.5,4.2,0)*
extend to lxl l1 e14	*Create a new straight edge to the intersection of line 1 and edge 14, at (28.9,4.2,0)*
end	*Close the shape by breaking edge 14 and closing the vertices*
start lxl l3 e11	*Start a new shape at the intersection of line 3 and edge 11, at position (22.5,−4.2,0)*
extend to lxl l3 e16	*Create a new straight edge to the intersection of line 3 and edge 16, at (28.9,−4.2,0)*
end	*Close the shape by breaking edge 16 and joining the vertices*
draw	*Produces the image in Fig. 12.3*
quit	*Leave the shape sketching package*

Note the use of simple numbers to identify faces, edges and vertices. This has a big disadvantage because the numbers can change during modelling. This can be seen in the above sequence when one shape has the command: “extend to lxl l1 e14” and three lines later there is a command: “extend to lxl l3 e16”, which is to a new edge created when edge 14 was broken by the first command. One of the

Fig. 12.3 Construction lines and initial shapes for face detailing



drawing styles used in BUILD was to draw objects with edge and vertex numbers to make it easier to identify elements. This solution, though, was imposed because there was no interactive graphics at that time and this sort of off-line programming with entity references was necessary.

setcentre draw	*Not shown in a figure*
2.5d face 1 originv 20	*Do the detailing of the front prongs of top face with the vertex marked in Fig. 12.4 as origin*
drawstyle dotted,label,-labf	*Set the drawing styles*
line 1 start (−15,0) dir vj	*Create a line in direction (0,1,0) through the point relative to the origin vertex 20—in global coordinates at (7.5,6.4,0)*
line 2 start (0,−2) dir vi	*Create a line in direction (1,0,0) through the given point—in global coordinates at (22.5,4.4,0)*
line 3 start (−19,0) dir vj	*Create a line in direction (0,1,0) through the given point—in global coordinates at (3.5,6.4,0)*
line 4 start (0,−10.8) dir vi	*Create a line in direction (1,0,0) through the given point—in global coordinates at (22.5,−4.4,0)*
start lxl l1 e14	*Start at the intersection between line 1 and edge 14, at (7.5,6.4,0)*
extend to lxl l1 l2	*Create a straight edge to the intersection between line 1 and line 2, at (7.5,4.4,0)*
extend to lxl l2 l3	*Create a straight edge to the intersection between line 2 and line 3, at (3.5,4.4,0)*
extend to lxl l3 e14	*Create a straight edge to the intersection between line 3 and edge 14, at (3.5,6.4,0)*
end	*Close the shape*
start lxl l3 e3	*Start a new shape at (3.5,−6.4,0)*
extend to lxl l4 l3	*Create a straight edge to the intersection between line 4 and line 3, at (3.5,−4.4,0)*
extend to lxl l4 e4	*Create a straight edge to the intersection between line 4 and edge 4, at (7.5,−4.4,0)*
end	*Close the shape*
line 5 start (−16,0) dir vj	*Create a new line in direction (0,1,0) through the point relative to the origin vertex—which is at global coordinates (−6.5,0,0)*
start lxl l5 e25	*Start a new edge at the intersection between line 5 and edge 25, at (6.5,4.4,0)*
extend to lxl l5 e30	*Create a straight edge to the intersection between line 5 and edge 30, at (6.5,−4.4,0)*
end	*Close the shape*
draw	*Produces the image in Fig. 12.4*
quit	*Leaves the sketching package*

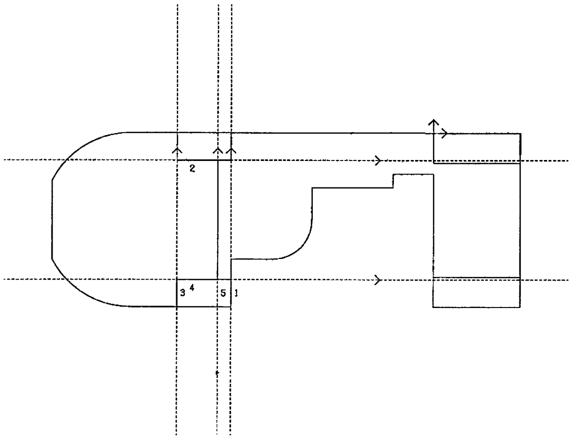


Fig. 12.4 Construction lines for MBB Gehäuse Rohteil front prongs

This step is similar to that of the previous part when detailing the back part of the object. Similar to that, it would have been possible to do the internal sketching and extrusion at the same time. However, since explicit edge and vertex numbers are used as references the 2D object is simpler and hence easier to identify the elements.

draw	*Not shown in the figures*
findface 10 18	*Find a face common to two edges. This is for information only and is used for verification*
2.5d face 1 originv 20	*Start detailing the back face between the rear prongs of the object*
drawstyle dotted,label,-labf	*Set the drawing styles*
line 1 start (−0.7,0) dir vj	*Create a line through the point relative to origin vertex, at (23.2,0,0)*
start lxl l1 e21	*Start a new edge at (23.2,−4.2,0)*
extend to lxl l1 e17	*Extend it to (23.2,4.2,0)*
end	*Close the shape*
quit	*Leave the 2D shape definition package*
findface 10 18	*Find the face common to two edges*
2.5d face 1 originv 5	*Start defining the oval face of the object*
drawstyle dotted,label,-labf	*Set the usual drawing styles*
circle 1 centre (3.1,2.9) radius 2.9	*Create circle 1 relative to the origin vertex—global coordinates of the centre (10.6,0,0)*
circle 2 centre (1.9,2.9) radius 2.9	*Create circle 2 relative to the origin vertex—global coordinates of the centre (9.4,0,0)*

line 1 start (0,5.8) dir vi	*Create line 1 with direction (1,0,0) through the point relative to the origin vertex—global coordinates (7.5,2.9,0)*
start v7	*Start a new edge at vertex 7, at (13.5,0,0)*
extend round c1 to lxc l1 c1	*Create a new circular edge, extending counter-clockwise round circle 1 to (10.6,2.9,0)*
extend to lxc l1 c2	*Create a new straight edge to (9.4,2.9,0)*
extend round c2 to lxc e33 c2	*Create a new circular edge, extending counter-clockwise round circle 2 to (7.5,0,0)*
end	*Close the new shape to form the base for the oval extrusion*
start lxc e33 c2	*Start a new part of the shape at (9.4,0,0)*
extend round c2 to lxc e5 c2	*Create a new circular edge counter-clockwise round circle 2 from (7.5,0,0) to (9.4,−2.9,0)*
end	*Close the shape*
quit	*Leave the 2D shape definition*
2.5d face 7 originv 3	*Create the shape for the hole at the front of the object*
drawstyle dotted,label,-labf	*Set the drawing styles*
circle 1 centre (0,6.4) radius 4.5	*Create a circle radius 4.5 with centre at the global coordinates (0,0,0)*
start (0,1.9)	*Start a new edge at (0,−4.5,0)*
extend round c1 to (0,10.9)	*Create a new semicircular edge in counter-clockwise direction to (0,4.5,0)*
extend round c1 to (0,1.9)	*Create a new semicircular edge round circle 1 in counter-clockwise direction to (0,−4.5,0)*
end	*Close the shape*
punch	*Punch the shape through to the underside of the 2D shape*
quit	*Leave the shape definition package*
draw	*Not shown in any figure*
2.5d face 1 originv 6	*Create the oval hole through the object*
drawstyle dotted,label,-labf	*Set the drawing styles*
circle 1 centre (−0.35,2.9) radius 2.25	*Create a circle radius 2.25 with centre at global position (10.25,0,0)*
circle 2 centre (−0.85,2.9) radius 2.25	*Create a circle radius 2.25 with centre at global position (9.75,0,0)*

line 1 start (0,5.15) dir vi	*Create a line, direction (1,0,0) through the global position (10.6,2.25,0), which is also tangent to the two circles*
line 2 start (0,0.65) dir vi	*Create a line through the global position (10.6,−2.25,0), direction (1,0,0), which is also tangent to the two circles*
start lxc l2 c1	*Start a new edge at (10.25,−2.25,0)*
extend round c1 to lxc l1 c1	*Create a new semicircular edge counter-clockwise round circle 1 to (10.25,2.25,0)*
extend to lxc l1 c2	*Create a new straight edge to (9.75,2.25,0)*
extend round c2 to lxc l2 c2	*Create a new semicircular edge counter-clockwise round circle 2 to (10.25,2.25,0)*
extend to lxc l2 c1	*Create a new straight edge to (10.25,2.25,0)*
end	*Close the shape*
punch	*Punch the new shape through to the underside of the object so that it becomes a hole through the object*
quit	*Leave the shape definition package*
draw	*Not shown in the figures*
2.5d face 8 originv 36	*Start a new shape to define the hole at the rear of the object*
drawstyle dotted,label,-labf	*Set the drawing styles*
circle 1 centre (3.25,4.2) radius 1.25	*Create a circle radius 1.25 centred at global coordinates (26.75,0,0)*
circle 2 centre (2.55,4.2) radius 1.25	*Create a circle radius 1.25 centred at global coordinates (26.05,0,0)*
line 1 start (0,2.95) dir vi	*Create a line through the global position (22.5,−1.25,0), direction (1,0,0), which is also tangent to the two circles*
line 2 start (0,5.45) dir vi	*Create a line, direction (1,0,0) through the global position (22.5,1.25,0), which is also tangent to the two circles*
start lxc l1 c1	*Start a new edge at (26.75,−1.25,0)*
extend round c1 to lxc l2 c1	*Create a new semicircular edge counter-clockwise round circle 1 to (26.75,1.25,0)*
extend to lxc l2 c2	*Create a new straight edge to (26.05,1.25,0)*

extend round c2 to lxc 11 c2	*Create a new semicircular edge counter-clockwise round circle 2 to (26.05, −1.25,0)*
extend to lxc 11 c1	*Create a new straight edge to (26.75, −1.25,0)*
end	*Close the shape*
punch	*Punch the shape through to the underside to create a hole*
quit	*Leave the shape definition package*
draw	*Produces the image in Fig. 12.5*

Unlike modern systems, BUILD created complex two dimensional shapes which could be extruded to create solids. Modern systems like singly connected profiles for extrusion, which makes for easier algorithms. The "punch" command was needed to distinguish between holes which traversed a planar shape and surface details.

set view (−2,10,10)	*Sets a static view position for graphics*
draw	*Produces the image in Fig. 12.6*

The graphics facilities were primitive at the time when this command file was defined and hence only static images could be generated. Setting the view position was the method for viewing an object from different directions.

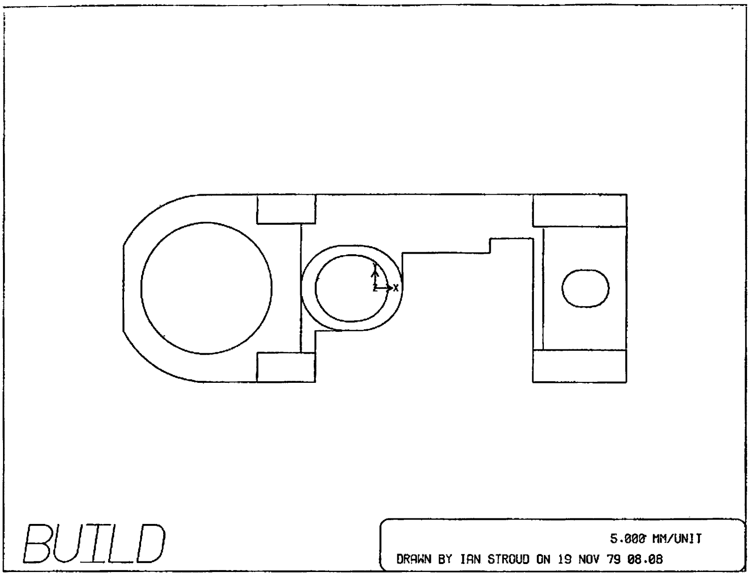


Fig. 12.5 Final 2D shape for MBB Gehäuse Rohteil creation, viewed from above

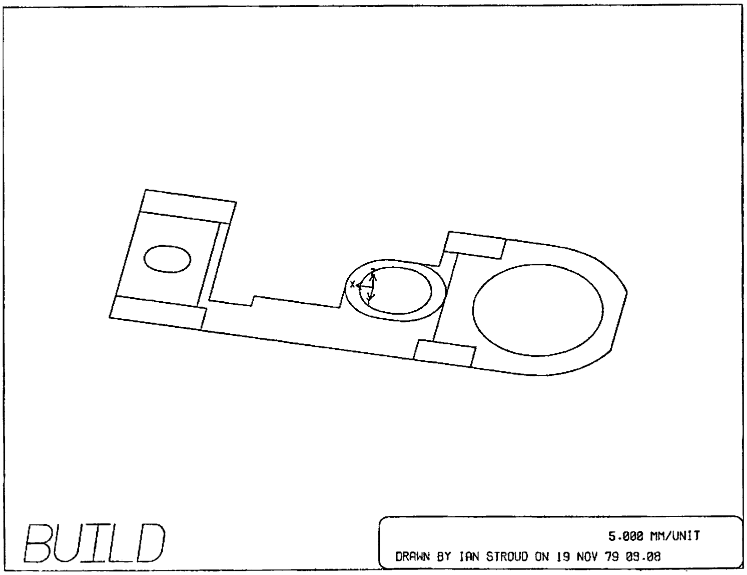


Fig. 12.6 Final 2D shape for MBB Gehäuse Rohteil creation

sweep body by (0,0,−5)	*Extrude the object downwards, which meant searching for the bottom face, which consisted of a simple outer contour with three inner circular contours*
drawstyle lhl, -label	*Set the drawing styles*
draw	*Produces the image in Fig. 12.7*

The drawing style “lhl”, which means “local hidden lines”, was a simple method for testing whether edges were definitely invisible and created slightly more realistic images. The resulting image also shows a number of extra edges in the extrusion direction between coplanar faces which result from the edge subdivisions due to the internal detailing on the top face.

sweep face 3 by (0,0,5)	*Sweep, or extrude, the first rear prong*
sweep face 4 by (0,0,5)	*Sweep the second rear prong*
draw	*Produces the image in Fig. 12.8*

Again, note the use of face numbers as simple identifiers. This identification method is fragile and needs constant modification if preceding steps are modified to create extra elements.

sweep face 5 by (0,0,5)	*Sweep the first forward prong*
sweep face 6 by (0,0,5)	*Sweep the second forward prong*
sweep face 1 by (0,0,4.7)	*Sweep the oval shape*
draw	*Not shown in any figure*

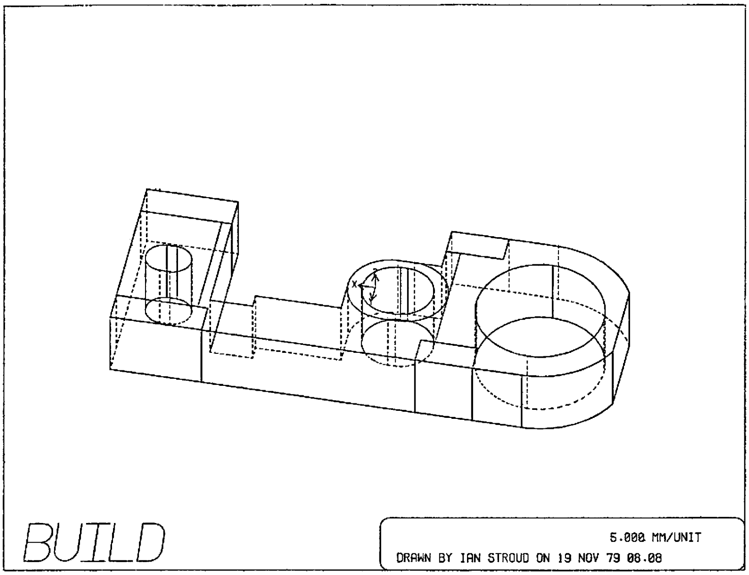


Fig. 12.7 MBB Gehäuse Rohteil with first extrusion

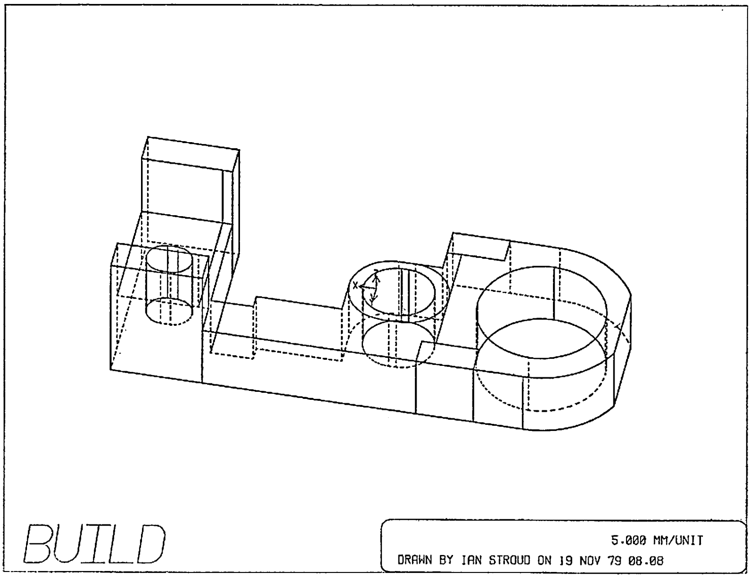


Fig. 12.8 First two prongs extruded

sweep face 8 by vk	*Sweep the rear step upwards by one unit*
sweep face 7 by (0,0,−1.4)	*Sweep the front of the object downwards by 1.4 units*
draw	*Produces the image in Fig. 12.9*
todisc camia	*Save the initial shape*
quit	*Leave the system to prepare for the second set of modifications*

Although it is possible to create the object in one session, the command file was broken into units. At the end of the first session, the object had all the basic shape elements defined and extruded, or swept. Some of the faces need to be tilted in the second step and extra elements added to create the final basic shape.

fromdisc camia	*Retrieve the basic shape saved at the end of the previous session*
drawstyle third,lhl	*Set the drawing styles*
setcentre	*Set the graphical centre to be the centre of the object to be drawn*
draw	*Produces the image in Fig. 12.10*

The object is drawn in "third angle view", which means that it has a similar layout to an engineering drawing. This kind of viewing is also useful for examining the relative positions of two objects prior to a Boolean operation, as will be seen in the next sequence.

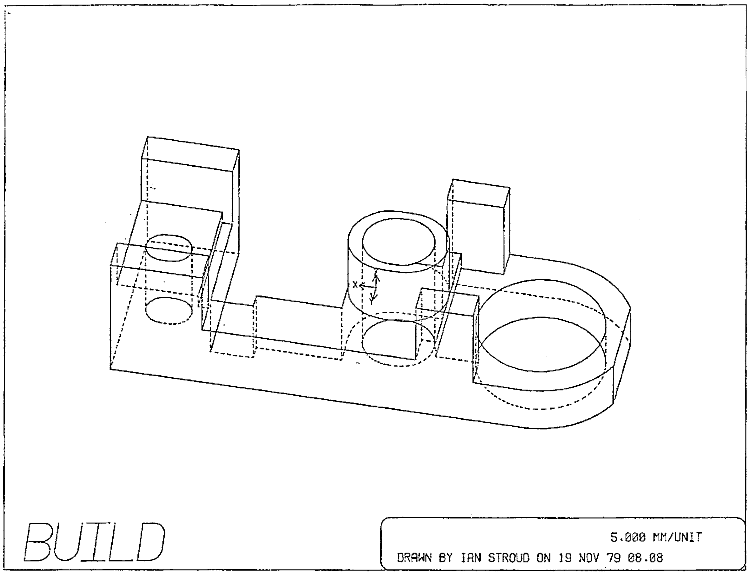


Fig. 12.9 MBB Gehäuse Rohteil with all faces extruded

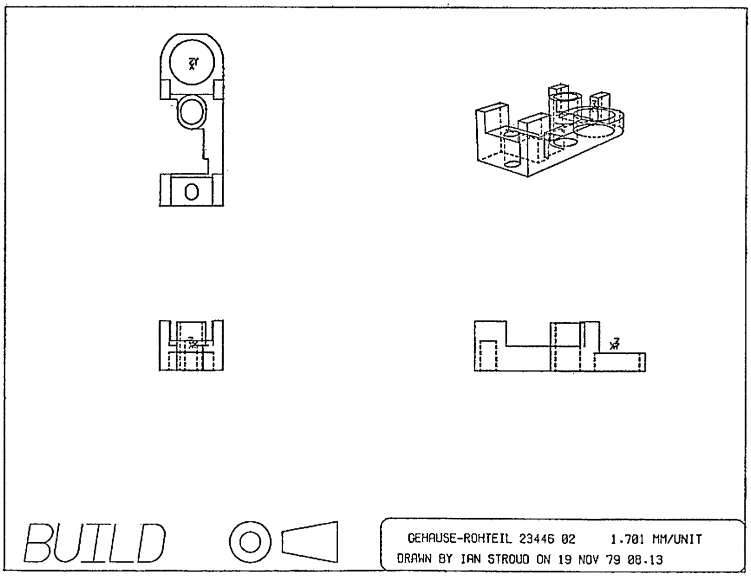

BUILD
GEHAUSE-ROHTEIL 23446 02 1.701 MM/UNIT
DRAWN BY IAN STROUD ON 19 NOV 79 08.13

Fig. 12.10 Third angle view of MBB Gehäuse Rohteil with all faces extruded

cube — *Create a standard cube, that is, a cube with one corner at global position (−1,−1,−1) and the other extreme corner at (1,1,1)*

modify by sx 3 sy 2.7 sz 3 mx 26.7 mz −4 — *Modify the shape and position of the cube, scaling it by 3 in the X-direction, 2.7 in the Y-direction and 3 in the Z-direction. The cube is then moved 26.7 in the X-direction and −4 in the Z-direction*

draw 2 — *Produces the image in Fig. 12.11*

Unlike many, but not all, modern systems, BUILD allowed uneven scaling. Instead of creating a block of the correct dimensions, a standard shape was created and then scaled to produce the correct shape. The cube is centred around the origin so the scaling is done first and then the translation, in the order that the transformation elements are written.

subtract — *Subtract the rescaled and repositioned block from the MBB basic object to create the cutout at the back*

drawstyle single, man — *Set the drawing styles*

set scale 0.2 — *Set the graphics scale to 0.2*

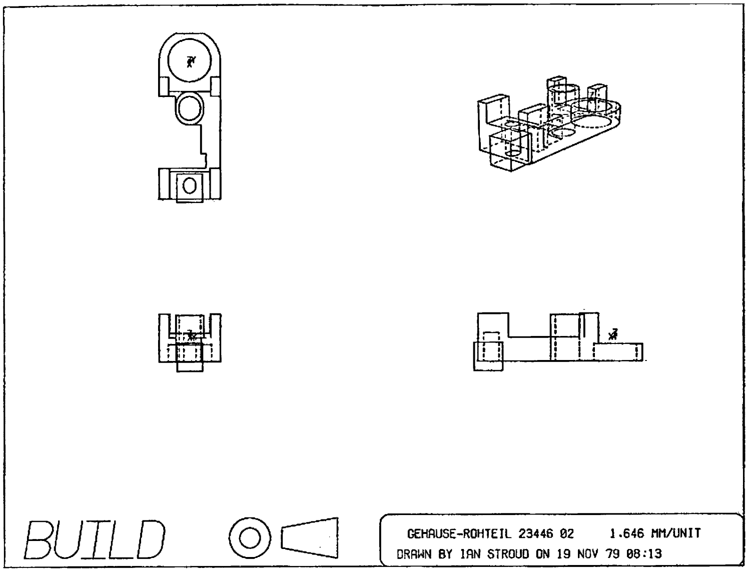

BUILD
GEHAUSE-ROHTEIL 23446 02 1.646 MM/UNIT
DRAWN BY IAN STROUD ON 19 NOV 79 08:13

Fig. 12.11 Third angle view of MBB Gehäuse Rohteil and cube to be subtracted

set view (−2,10,10)	*Set the viewing position*
setcentre	*Recalculate the centre of the object for drawing*
draw	*Produces the image in Fig. 12.12*

The drawing styles above reset graphics to a single view and set the style to "manual" to allow graphical rescaling.

modify by mx −3.5 mz −5	*Move the object in order to rotate the prong face*
detransform	*Fix the transformation*
tweak face 5 by ry 5	*Rotate the face and recalculate the geometry*
setcentre	*Set the graphics centre*
draw	*Produces the image in Fig. 12.13*

The face rotation, done by tweaking, modifies the face surface equations and recalculates the surrounding geometry to be consistent. The tweak definition specifies that the face is to be rotated about the *Y*-axis by 5°. This means that object has to be moved so that the correct object part is coincident with the *Y*-axis. The object is moved down 5 units and back 3.5 units, which means that the top edge of the front prongs is aligned with the *Y*-axis. The "detransform" command means that the translation is multiplied into the geometrical entities in the model so that the model is really positioned at the right place. Normally, a transformation was simply associated with the object and only used to change the object when the

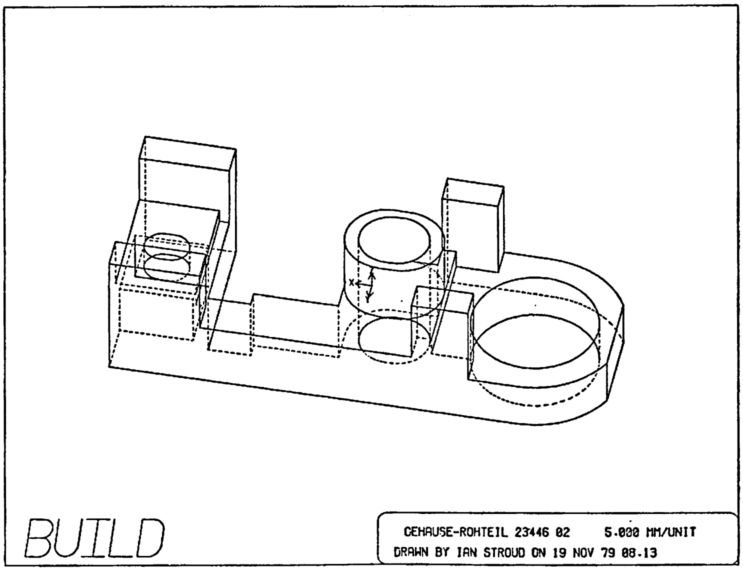

BUILD
GEHAUSE-ROHTEIL 23446 02 5.000 MM/UNIT
DRAWN BY IAN STROUD ON 19 NOV 79 08.13
X

Fig. 12.12 Single view of MBB Gehäuse Rohteil with extrusions and cutaway rear

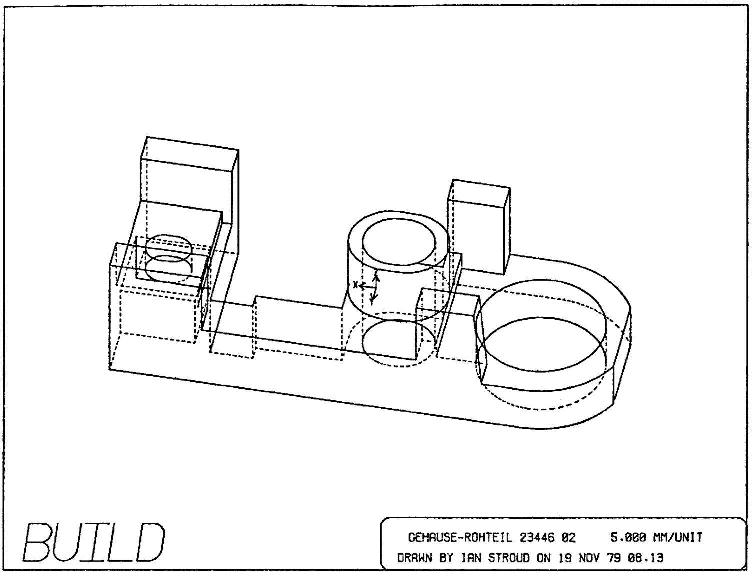

BUILD
GEHAUSE-ROHTEIL 23446 02 5.000 MM/UNIT
DRAWN BY IAN STROUD ON 19 NOV 79 08.13
X

Fig. 12.13 MBB gehäuse Rohteil with one tweaked face

absolute position was needed, for example for Boolean operations. This command enforces the multiplication under user control.

tweak face 8 by ry 5	*Tweak the other prong face*
modify by mx −19 mz 5	*Move the object so as to be able to tweak the rear prong face*
detransform	*Multiply the transformation into the object*
tweak face 30 by ry 15	*Tweak one of the prong faces by 15°, the other has to be changed using a Boolean operation*
setcentre	*Set the graphics centre*
draw	*Produces the image in Fig. 12.14*

This step performs the other face rotations which can be done by tweaking. The last face cannot be done in the same way because there is a topological change which has to be made.

cube	*Create a standard cube*
modify by mx −1 my −1 sy 5 sz 6 ry 15	*Move the block so that its top corner is at (0,0,0) and change the size of the object by 5 in the Y-direction and 6 in the Z-direction. Finally rotate the cube about the Y-axis by 15°*
draw 2	*Produces the image in Fig. 12.15*

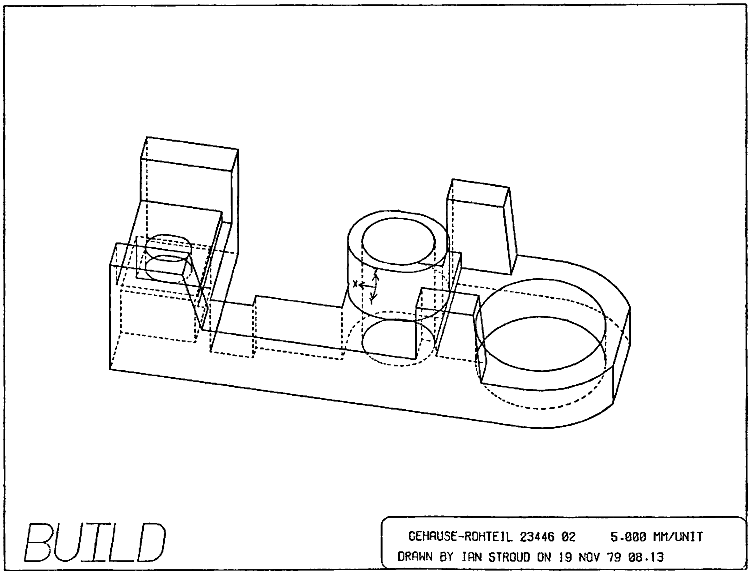

BUILD
GEHAUSE-ROHTEIL 23446 02 5.000 MM/UNIT
DRAWN BY IAN STROUD ON 19 NOV 79 08.13

Fig. 12.14 MBB Gehäuse Rohteil with three tweaked faces

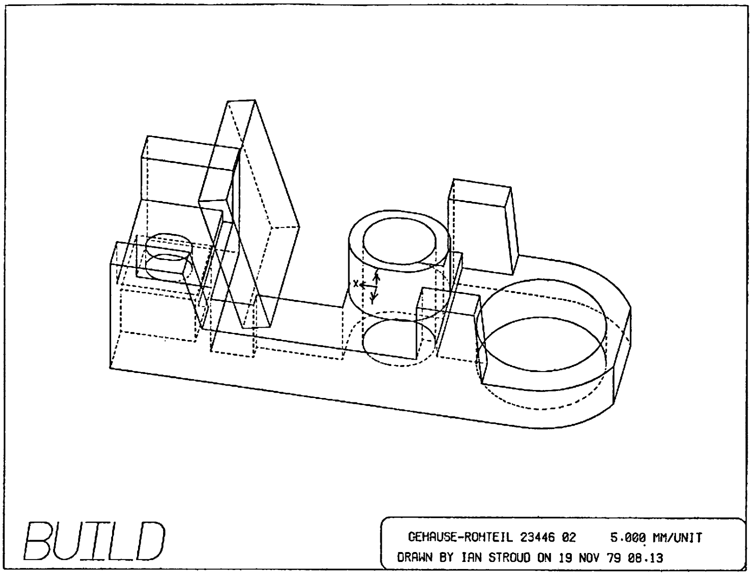

x
BUILD
GEHAUSE-ROHTEIL 23446 02 5.000 MM/UNIT
DRAWN BY IAN STROUD ON 19 NOV 79 08.13

Fig. 12.15 Using subtract to create a sloping face

This step is similar to that before to create a tool block with which to modify the basic object.

subtract	*Subtract the tool body from the base object to create the final tilted face*
setcentre	*Set the graphics centre*
draw	*Not shown in figures*
modify by mx −0.7	*Move the body back 0.7 units*
detransform	*Multiply the transformation into the body*
tweak face 11 by ry 15	*Tweak the small face of the step by 15°*
modify by mx −5.7 my −6.4 mz 5	*Move the body to its final position*
detransform	*Multiply the transformation into the body*
setcentre	*Set the graphics centre*
draw	*Produces the image in Fig. 12.16*
todisc camib	*Save the base body to disc ready for the next stage*

The commands shown above are part of the sequence for creating the MBB Gehäuse Rohteil used in the CAM-I test of 1979. The sequences ran the BUILD solid modelling system developed by the Cambridge University Computer Laboratory research group led by Ian Braid. The sequences use principally the work on shape definition and tweaking developed by Graham Jared, the Boolean operations developed by Alan Smith and Bernard Solomon and some local operations developed by Ian Stroud.

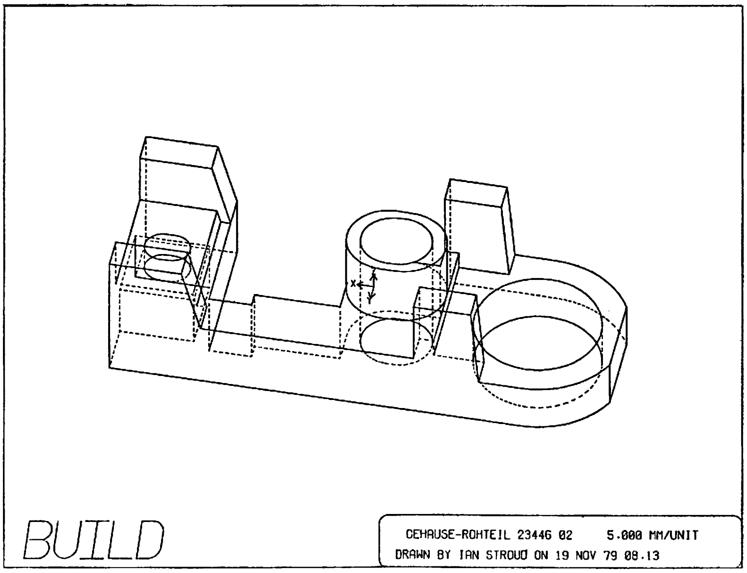

x
y
BUILD
CEHAUSE-ROHTEIL 23446 02 5.000 MM/UNIT
DRAWN BY IAN STROUD ON 19 NOV 79 08.13

Fig. 12.16 MBB Gehäuse Rohteil after first phase of creation

Although old, this command sequence illustrates several important points. The first is the use of construction geometry, the second is the references to the object parts that are used, the third is the stack-based operation, finally, there are the types of operations used. The use of construction geometry has become a standard part of CAD. Referring to object parts is important and is discussed below. The BUILD command interpreter also used a stack for object handling which allowed, for example, multiple objects to be used in a fluent manner. Several of the operations used in CAD systems today were already defined by the BUILD research group at the end of the 1970s. Although these have been improved in terms of speed and reliability, modern CAD developers have not introduced as many new operation types as could be expected from contacts with their clients.

The next sections deal with developments of the use of command files.

12.1.2 Log Files

Log files are, I believe, older than commercial solid modelling, but it doesn't really matter for what is said here.

A log file is intended to record what a system user does interactively in a file. In this respect the CAD system can be thought of as a graphical editor, creating a command file while building up a model interactively. This was important in early developments when systems weren't as reliable as they are now so that designers could redo their work and also pass problem files to developers for bug fixing.

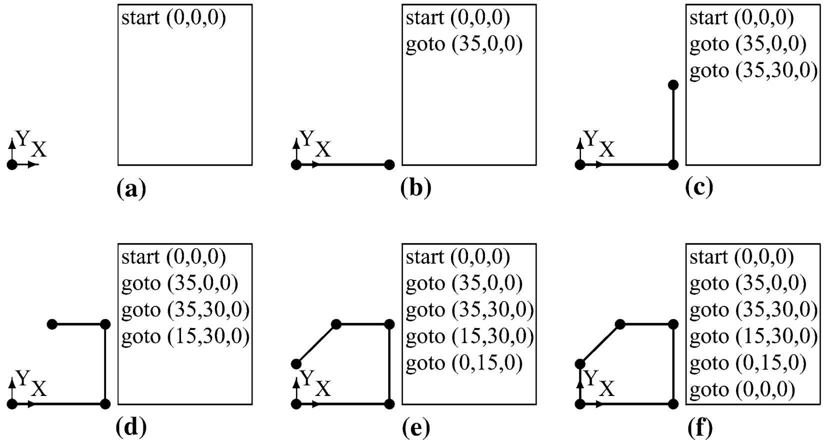
To illustrate how this works, consider Fig. 12.17. The user moves the cursor to the origin and clicks to indicate a start point, Fig. 12.17a. At the same time the system writes a line to the log file, such as “start (0,0,0)”. The user then moves the cursor to a new position, say (35,0,0) and clicks the mouse button again. The system draws a line from the start point to this position and writes a new line to the log file, “goto (35,0,0)”, Fig. 12.17b. The user again moves the cursor to (35,30,0) and clicks the mouse button, causing the system to create a new line from (35,0,0) to (35,30,0) and to write the line “goto (35,30,0)” to the log file, Fig. 12.17c. The procedure continues to (15,30,0) Fig. 12.17d, (0,15,0) Fig. 12.17e and finally to close up the figure, Fig. 12.7f.

For simplicity, construction geometry is not used in this example, although it is a common facility. As the model development proceeds so the logfile is extended. In Fig. 12.18 the first operation in the continuation might be a hidden operation, to set the shape into a surface, and the second an extrusion.

This would give a logfile such as:

```
start (0,0,0)
goto (35,0,0)
goto (35,30,0)
goto (15,30,0)
goto (0,15,0)
goto (0,0,0)
embed plane (0,0,0) (0,0,1)
extrude (0,0,25)
```

And so on. The idea should be fairly clear that, instead of building a command file using a text editor and imagining where everything is placed, a command file is built while the user interacts. In order to replay the files, though, it is still necessary to have a command interpreter.



```
start (0,0,0)
```

(a)

```
start (0,0,0)
goto (35,0,0)
```

(b)

```
start (0,0,0)
goto (35,0,0)
goto (35,30,0)
```

(c)

```
start (0,0,0)
goto (35,0,0)
goto (35,30,0)
goto (15,30,0)
```

(d)

```
start (0,0,0)
goto (35,0,0)
goto (35,30,0)
goto (15,30,0)
goto (0,15,0)
```

(e)

```
start (0,0,0)
goto (35,0,0)
goto (35,30,0)
goto (15,30,0)
goto (0,15,0)
goto (0,0,0)
```

(f)

Fig. 12.17 Creating a figure and log file

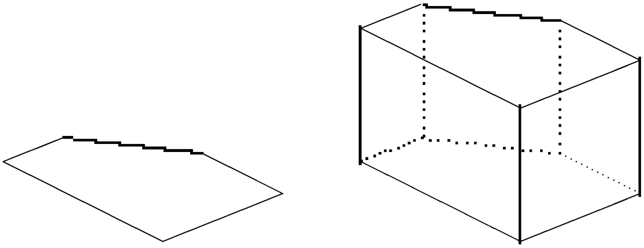
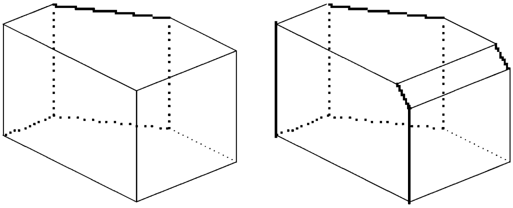


Fig. 12.18 Extrusion and log file

Fig. 12.19 Chamfering and log file



None of the operations above, though, need to refer to model parts. The same questions arise about handling model interactions as those raised above in the command file sequence. Suppose that the user chamfers an edge in the model, as shown in Fig. 12.19.

The simple solution to this is to record the command with the edge number, picked interactively, say, such as:

chamfer edge 9 by 8

This is certainly enough for replaying the sequence, but it has the same problems if the object is modified prior to the chamfer command. Edge 9 may not be the same as that originally intended. This becomes more of a problem for history files.

12.1.3 Internal History Records

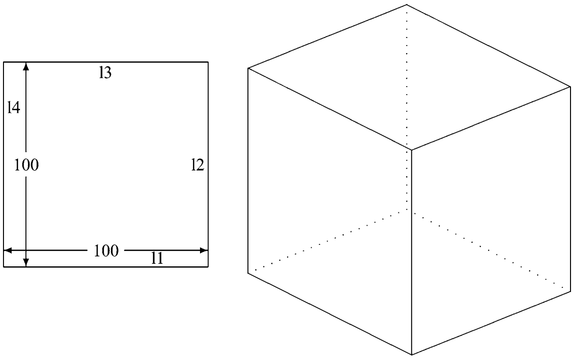
Finally, back to history files. If you can write a log file to a physical file, why not just store it in memory?

This is the basis for history records. Retaining the log file in memory allows the system to offer users the possibility of modifying command parameters and rerunning code. This may be the basis for parametric modellers, which were around in the late 1980s. One famous example of a parametric system is Pro-Engineer, although I do not know how it works. This system was innovative when it first appeared and these innovations were appreciated. Other systems followed by producing the kind of history records mentioned above, at least, I assume that they work in the way mentioned above. It is not the purpose to analyse the systems to reveal commercial secrets but merely to show how they might work and show the links to the initial command files.

One question about internal history records, though, is what you show to the user. It is not optimal if the user sees all the commands that the system uses to create the object. Sketch geometry is not necessarily shown in complete form. For example, to create a simple cube, as shown in Fig. 12.20, you might have the following operations:

2.5d	*Start a sketch*
line 1 start (0,0,0) dir vi	*Create a first line*
line 2 start (80,0,0) dir vj	*Create a second line*
line 3 start (80,70,0) dir -vi	*Create a third line*
line 4 start (0,70,0) dir -vj	*Create a fourth line*
perpendicular l1 l2	*Set a constraint. These are often done automatically by the system if two lines are almost perpendicular*
parallel l1 l3	*Another constraint which may be added automatically*
parallel l2 l4	*Another constraint*
distance l1 l3 100	*Set the distance between two lines to be 100, changing the geometry. This is equivalent to adding a dimension and changing the value to 100.* *A distance dimension also requires a parallelism constraint*
distance l2 l4 100	*Change the geometry so that the distance between the lines is 100*
start lxl l1 l4	*Start inking in at a particular point. The geometrical elements are infinite, even though the system may only draw the inked-in portions of that geometry*
extend to lxl l1 l2	*Make an edge to the second corner*

Fig. 12.20 Creating history files



extend to lxl l2 l3	*Make an edge to the third corner*
extend to lxl l3 l4	*Make an edge to the fourth corner*
joinup	*Make the last edge*
quit	*Exit from the sketch module*
embed plane vo vk	*Set the inked-in sketch, a wireframe model, into a surface to make it complete*
sweep body by (0,0,100)	*Extrude the body to create the solid*

Of this, the user might see:

Sketch	*Sketching information shown graphically*
sweep body by (0,0,100)	*Create the final solid. The embedding operation is a preprocessing step to the extrusion operation*

As a further illustration, consider again the sequence to create the basic MBB Gehäuse Rohteil.

Sketch1	*Sketching information shown graphically in Fig. 12.21. Only two dimensions are shown for simplicity, but normally the shape would need many more*
sweep sketch1 by (0,0,5)	*Create the final solid. The embedding operation is a preprocessing step to the extrusion operation. The sketch is added on a plane or planar face and the wireframe sketch is embedded into that plane*
Sketch2	*Sketching information shown graphically in Fig. 12.22. The sketch would be done on the top face of the object created in the first step*
sweep sketch2 by (0,0,5)	*Extrude the prongs, they can be done in one step because they are all the same height.*

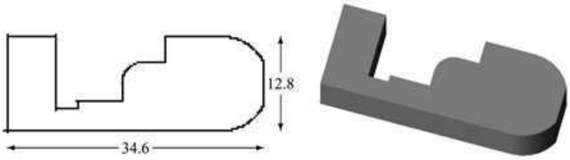


Fig. 12.21 MBB Gehäuse Rohteil—initial sketch

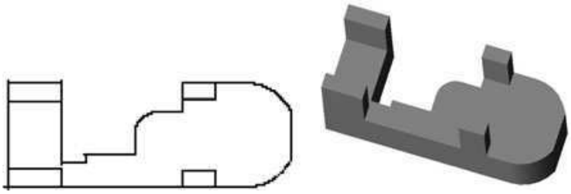


Fig. 12.22 MBB Gehäuse Rohteil—prongs extruded

	It doesn't matter that the sketch parts are not linked. In modern systems the swept, or extruded, shapes become separate objects which are then added to the original object using Boolean operations, as described in Chap. 4
Sketch3	*Sketching information shown graphically in Fig. 12.23. The sketch could be done using the object's own boundaries or as a larger object if a Boolean subtraction is used*
sweepcut sketch3 by (0,0,−1.4)	*Extrude the front part downwards. In modern systems this may be done as a linear sweep of (0,0,1.4) and a subtraction Boolean operation*
Sketch4	*Sketching information shown graphically in Fig. 12.24*
sweep sketch4 by (0,0,4.7)	*Extrude the oval shape. As before, this is now usually done by creating a separate object which is then added to the original*
Sketch5	*Sketching information shown graphically in Fig. 12.25*

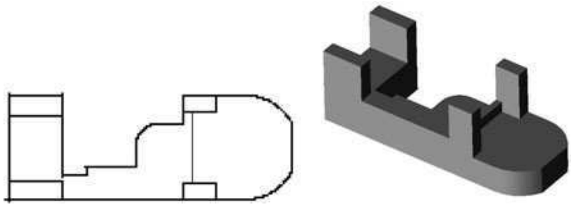


Fig. 12.23 MBB Gehäuse Rohteil—front swept downwards

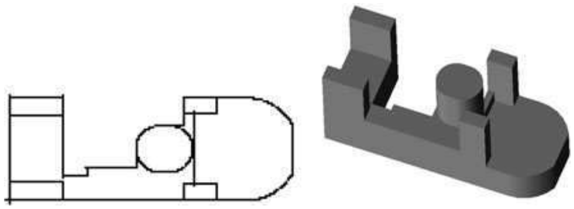


Fig. 12.24 MBB Gehäuse Rohteil—oval part extruded upwards

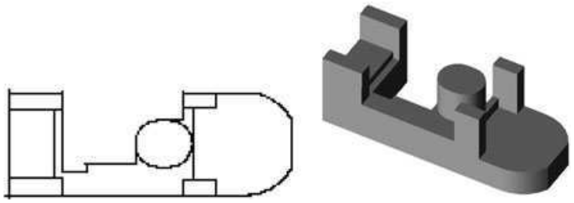


Fig. 12.25 MBB Gehäuse Rohteil—back step extruded upwards

sweep sketch5 by (0,0,1)	*Extrude the back step*
Sketch6	*Sketching information shown graphically in Fig. 12.26. The sketch can be done on the top surface of the prongs or on the surface of the oval part. The extent here is given as 10, but the extent could be to the bottom surface*
sweepcut sketch6 by (0,0,−10)	*Extrude the hole shapes downwards. Linear extrusion and subtraction operation again to give that the holes go right through the object. If Boolean operations were not used it would be necessary to sketch each hole separately on the correct face and extruded it through to the underside face to ensure that it breaks through*
tweak face x by −5 about edge xx	*Rotate the front prong faces by 5° about the bottom edge. The result is shown in Fig. 12.27*
tweak face y by −5 about edge xx	

Selecting the face and edge is a topic which will be dealt with later. In modern systems this is done by interactive picking, but the topic is not evident.

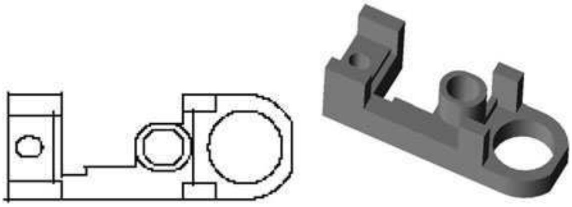


Fig. 12.26 MBB Gehäuse Rohteil—holes made

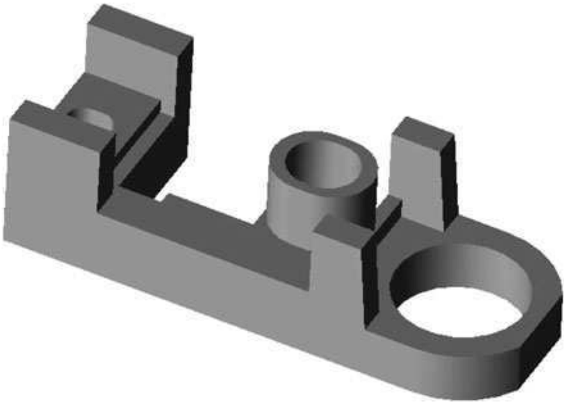


Fig. 12.27 MBB Gehäuse Rohteil—front prongs modified

Sketch7	*Sketching information shown graphically in Fig. 12.28*
sweepcut Sketch7 by (0,12.8,0)	*Extrude the wedge shape and subtract from the body. Fig. 12.29. This is an alternative to a tweak operation*
Tweak face z about edge zz by −15	*Rotate the small step face by 15°*
Sketch8	*Sketching information shown graphically in Fig. 12.30*
sweepcut sketch8 by (0,0,4)	*Extrude the square shape (linear extrusion and subtraction Boolean operation)*

The reason for showing this sequence is to demonstrate the way in which sketches are used to group shape definitions into visual groups. It is not necessary to maintain the individual curve information. The resulting history information shown to the user might be:

Fig. 12.28 MBB Gehäuse Rohteil—back prongs modified

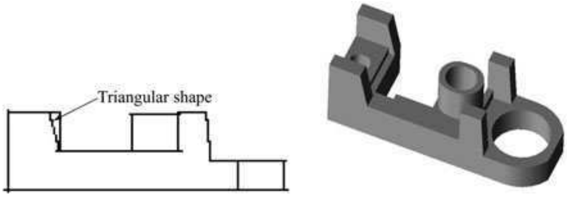


Fig. 12.29 MBB Gehäuse Rohteil—back step modified

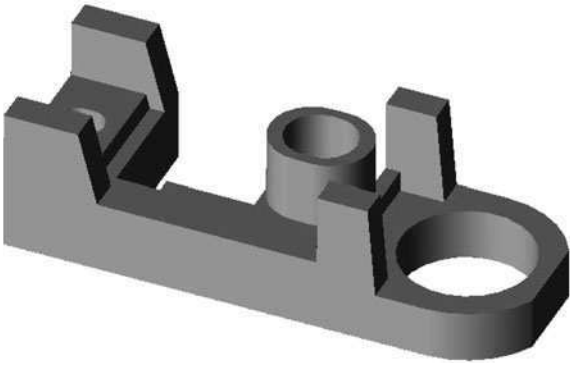
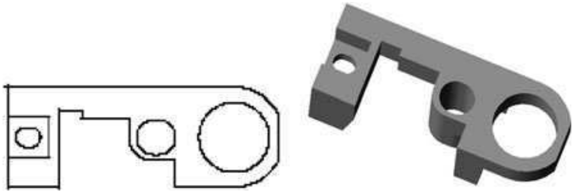


Fig. 12.30 MBB Gehäuse Rohteil—rear cutout made



Sketch1	*Figure 12.21*
sweep sketch1 by (0,0,5)	
Sketch2	*Figure 12.22*
sweep sketch2 by (0,0,5)	
Sketch3	*Figure 12.23*
sweepcut sketch3 by (0,0,−1.4)	
Sketch4	*Figure 12.24*
sweep sketch4 by (0,0,4.7)	
Sketch5	*Figure 12.25*
sweep sketch5 by (0,0,1)	
Sketch6	*Figure 12.26*
sweepcut sketch6 by (0,0,−10)	
tweak face x by −5 about edge xx	
tweak face y by −5 about edge xx	
Sketch7	*Figure 12.28*
sweepcut Sketch7 by (0,12.8,0)	
Tweak face z about edge zz by −15	
Sketch8	*Figure 12.30*
sweepcut sketch8 by (0,0,4)	

This is much more efficient than showing all the commands to create curves and the 2D profiles. Two things to note, though, are the way that you refer to object parts, to be discussed in Sect. 12.2 and the way that operation parameters are used to create parametric files, to be discussed in Sect. 12.3.

12.2 Referring to Model Parts

One of the key needs with command files and history trees is to be able to refer to parts of the model. This topic has already been mentioned in Sect. 7.4, but is mentioned here in more detail. A good reference for this topic is Várady et al. [2] which gives a good perspective for these techniques. Due to the different development and availability of graphics it was not always possible to use interactive techniques and hence several methods were tried. Although interactive graphics is widely available, it is not always appropriate and re-evaluation of a part is one of those areas where other techniques are needed.

12.2.1 Entity Numbers

Entity numbers are a basic element of modelling datastructures, though not in all systems. An entity number is useful for writing model data to a disk-file as well as for debugging. In the BUILD research solid modeller these were also used to identify elements for commands, as can be seen in the command sequence shown in Sect. 12.1.1.

What happened in BUILD was that, when an entity was created or revived after being deleted, it received automatically a unique number. This meant that the number of the entity depended on the order of creation of the entities of that type. Take the example of a cube, made here by creating a square on the $Z = 0$ plane with one corner at (0,0,0) and then extruding the result in the Z-direction (Fig. 12.31).

Suppose, now that I want to chamfer edge 9 by five units. The result object may be as shown in Fig. 12.32.

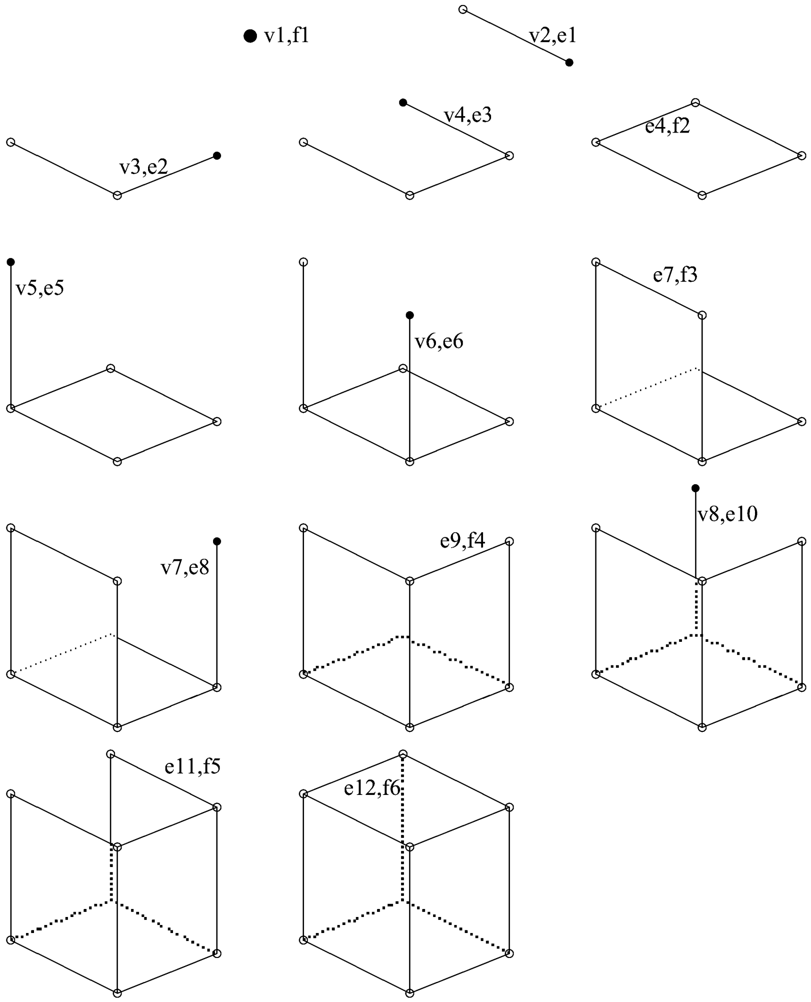


Fig. 12.31 Entity numbering when making a rectangular block

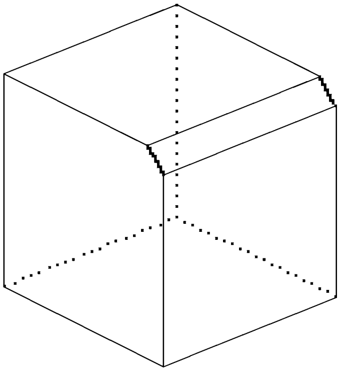


Fig. 12.32 Chamfered edge in block

The command file to create the object might be:

```
2.5D
start (0,0)
extend to (50,0)
extend to (50,50)
extend to (0,50)
extend to (0,0)
end
sweep face 1 by (0,0,50)
chamfer edge 9 by 8
```

If the shape creation part is changed to create a five-sided shape, with the following command sequence:

```
2.5D
start (0,0)
extend to (50,0)
extend to (50,20)
extend to (30,50)
extend to (0,50)
extend to (0,0)
end
sweep face 1 by (0,0,50)
chamfer edge 9 by 8
```

Then, the sequence for creating the five-sided block would be as shown in Fig. 12.33.

Chamfering edge 9 would then cause one of the vertical edges to be chamfered, as shown in Fig. 12.34.

As already mentioned, the use of entity numbers in command sequences is volatile and changing the commands in the sequence may cause odd results. This means that simple entity numbers are not really stable enough to support user interactions.

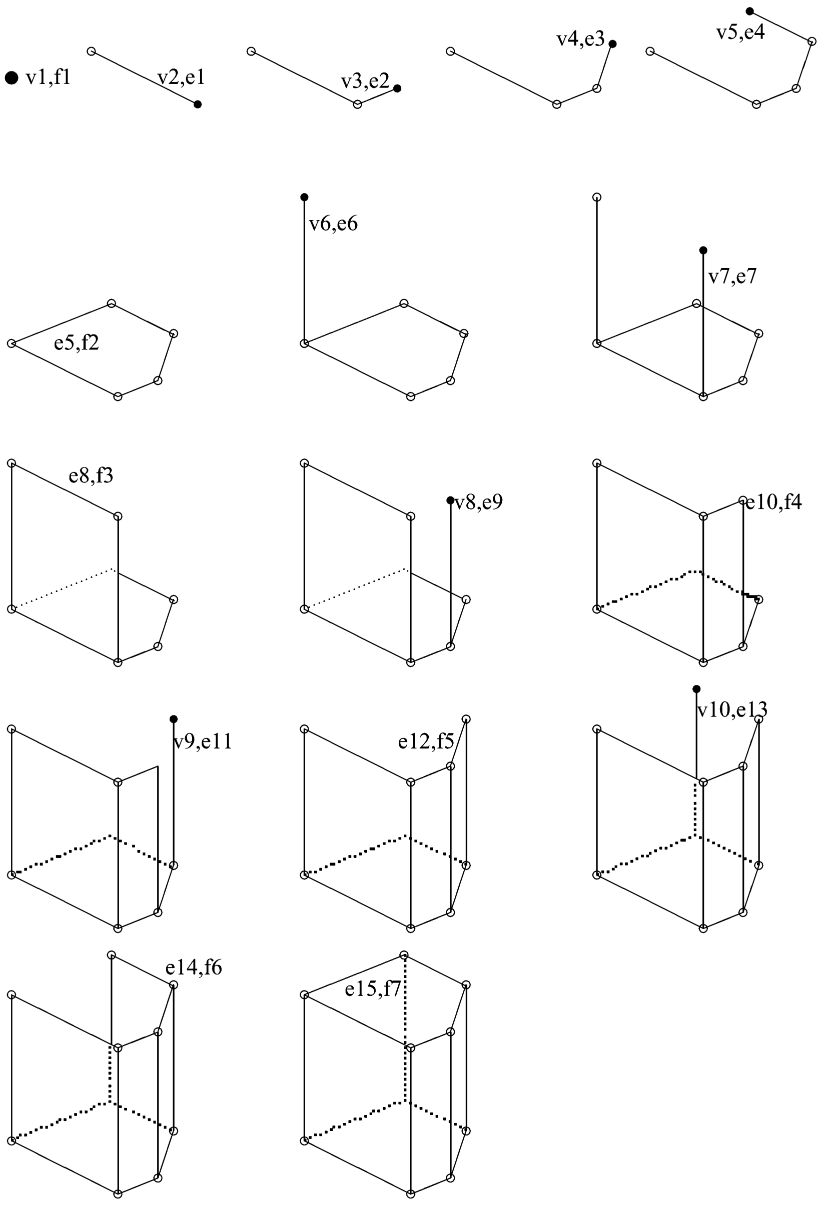


Fig. 12.33 Entity numbering when making a five-sided block

12.2.2 Topological Navigation

Topological navigation was a method invented by Chris Cary for the BUILD system to allow model entity identification as a series of qualified entities. What

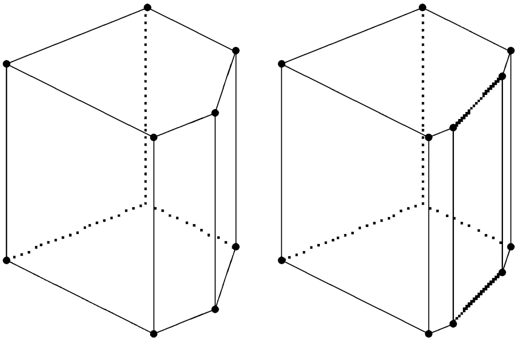


Fig. 12.34 Chamfering edge 9 in the five-sided block

Cary did was to use relative positions of elements to move around the object. Cary used six position qualifiers: TOP, BOTTOM, LEFT, RIGHT, FRONT, BACK with the three topological elements: FACE, EDGE, VERTEX. The method was a major advance over the use of entity numbers because entities were found dynamically based on position rather than on a predefined number which did not depend on shape. However, it was necessary to be able to visualise the target object and be able to identify unique sequences to find the element. Sometimes, too, it is difficult to find a valid sequence, but the method worked very well for the test objects used in the software development before interactive techniques were developed.

Take the case in Fig. 12.32 again. The command sequence for this, using Cary's navigation method, would be:

```
2.5D
start (0,0)
extend to (50,0)
extend to (50,50)
extend to (0,50)
extend to (0,0)
end
sweep face 1 by (0,0,50)
chamfer top edge of front face by 8
```

Note that edge 9 in Fig. 12.31 is not the top edge of the body, as edges 7, 9, 11 and 12 are all equally high. Nor is it the front edge of the body, since edges 2, 6, 8 and 9 are all equally forward. However, the front face is unique, that is face 4, and the highest edge on that is edge 9. Although edges 6 and 8 have one end vertex as high as edge 9, they extend lower than edge 9, so the definition of *top* is highest maximum and highest minimum. If several elements have equal extents then a warning should be given.

In order to chamfer the corresponding edge of the five sided block, shown on the top-left of Fig. 12.35, the sequence:

```
chamfer top edge of front face by 8
```

is also valid, producing the result shown at the top right of Fig. 12.35. In order to chamfer the adjacent top edge, as shown on the bottom left of Fig. 12.35, you might have the sequence:

chamfer top edge of right face of right edge of front face by 8
or *chamfer top edge of front face of front edge of right face by 8*
or *chamfer right edge of front edge of top face by 8*

Finally, to chamfer edge 9, bottom right of Fig. 12.35, you might have the sequence:

chamfer right edge of front face by 8

As an example of a more complex sequence, consider the selection of the edge around which to tweak the face of the MBB object, shown in Fig. 12.27. The sequence given was:

tweak face x by −5 about edge xx

Replacing "edge xx" by a navigation sequence might give:

tweak face x by −5 about right edge of back vertex of back edge of top vertex of front edge of left face

The use of this navigation comes from the era when command files were used rather than interactive techniques. The technique is based on human ability to

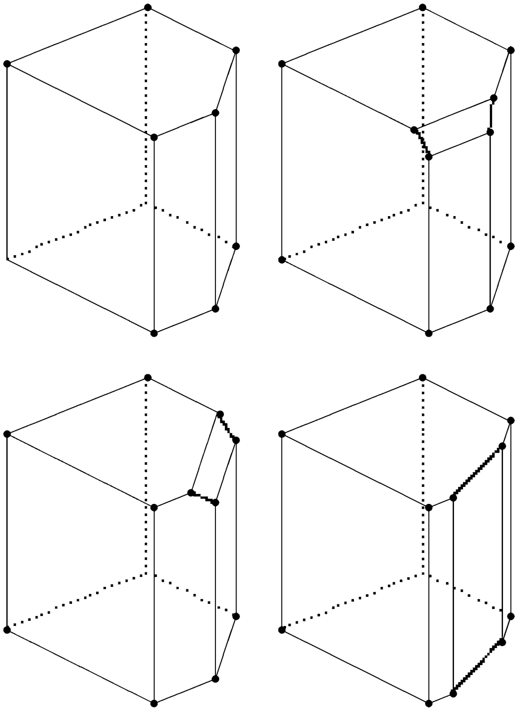


Fig. 12.35 Chamfering different edges in the five-sided block

visualise an object being made and it is hard to see how sequences could be generated automatically for history sequences.

12.2.3 Hit-Testing

Hit-testing is a graphical means of identifying elements by picking them in a graphics model on a screen and is now the dominant entity selection technique. This has already been described, in Sect. 7.4, but is here mentioned again in terms of command files and history.

Hit-testing is a method where a two dimensional point on the screen is converted into a model entity. There are two methods, one is the model-space method and the other the graphics-space method. In the model-space method, the place at which a user clicks on the image, is converted into a three dimensional point, and the view direction is used to create a line in space. This line is then intersected with the object in model space to recover a set of elements at proximity to which the line passes. In the graphics-space method a graphics entity is identified during drawing and then this entity linked back to the model in model-space. You don't really have to know which of these is used, merely that the act of picking returns a model element.

In command files the interactive pick can be replaced by a three dimensional point and a vector direction. This is done in command files in the ACIS modelling kernel, for example. In history files the interactive pick from the user can be recorded, together with the current visual transformation, and then the interactive pick can be reconstructed if the command sequence is replayed.

To understand the role of transformations in graphics, see Sects. 5.2.2 and 7.1. Take the example of Fig. 12.32 as an example. Suppose the user turns the cube so that it looks something like the figure on the left of Fig. 12.36. Suppose, also, that the user selects the middle of the edge for chamfering. The picking data for the hit test might be a line through the middle of the object, say (0,0,0) with a direction (−0.7071,0,−0.7071).

If the same sequence is applied with the new object, then the picking misses an edge, as shown on the right hand side of Fig. 12.36. Even though record the picking seems a good idea there are limitations. For the second, the CAD system should

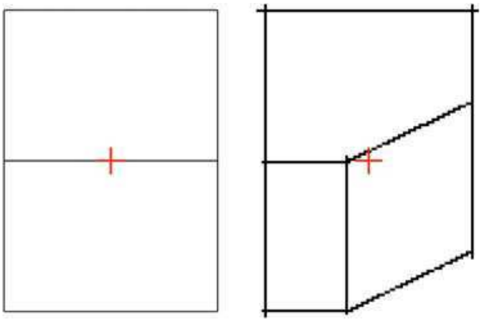


Fig. 12.36 Hitting the edge to be chamfered in the cube

probably flag an error and disallow the command. Recording picking data is certainly a method that has been used at one time for history recording. The other main method used currently is the "persistent naming" method, described next.

12.2.4 Persistent Naming

"Persistent naming" is a technique in which the identity of model parts operated upon retain their identity. In command files, it was difficult to refer to model parts, as explained above. Persistent naming is a little like the use of entity numbers in the first command files. However, unlike entity numbers, persistent names are transmitted and transformed to give a historical record. The following is how persistent naming might work because there are several possibilities. One method is described by Mun and Han [3]. The following is a simplified method intended to illustrate the principle without being close to any existing commercial method.

Take again the example from before with the extruded square shape. Figure 12.37 shows the basic shape with some simple names.

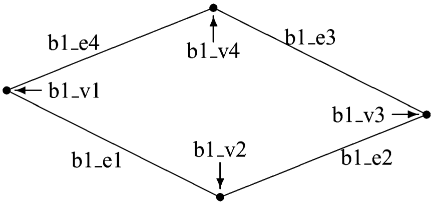


Fig. 12.37 Square with persistent names

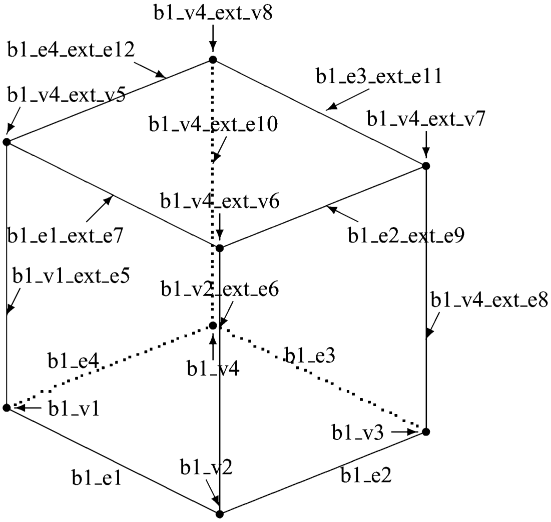
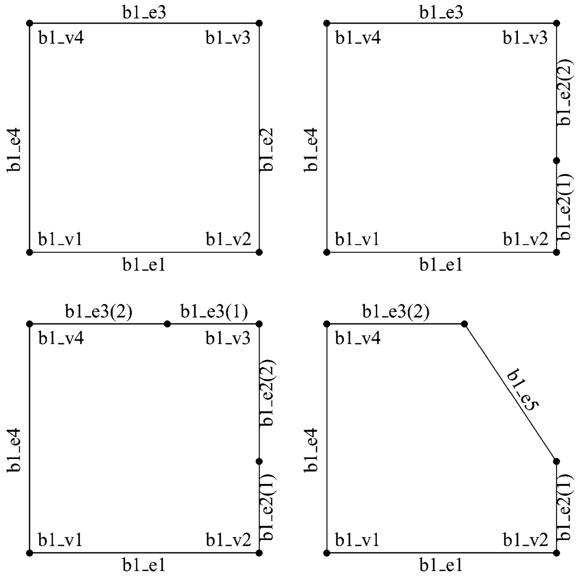


Fig. 12.38 Cube with persistent names

Fig. 12.39 Modified square with persistent names



When the object is extruded the new elements receive names based on the old names, as shown in Fig. 12.38.

Pick the same edge, that is, the one marked “b1_e2_ext_e9” in Fig. 12.38, and chamfer it, as in the earlier examples. Now, modify the original sketch, shown in Fig. 12.37, by cutting off a corner, as shown in Fig. 12.39.

When this new figure is extruded, the chamfered edge now has a slightly different name: “b1_e2(1)_ext_e9”. The “(1)” denotes a modification so it is relatively easy to match the old name part “b1_e2_ext” with “b1_e2(1)_ext” simply by ignoring the name modifier. The last part, the “e9” is not used for matching. Note, though, that if edge “b1_e2” had been totally replaced then the method would not have worked.

Not being an expert on persistent naming, I have tried to explain what I understand to be the mechanism behind persistent naming. In summary, it is a method for keeping a trail through the development of the model in order to minimise reworking. However, it is not an infallible method.

12.2.5 Do What I Mean

Do not expect magic solutions from any system. There is no DWIM CAD function, the illusive “Do What I Mean” function, and it is unrealistic to expect CAD systems to know always what you want. Really, it’s up to the user to avoid complicated sequences and extensive editing of the history tree. It may even be worthwhile redoing the design at the end if you want to have a logical command description of an object.

The methods described around part identification try and help the user to create a consistent model creation description which can be used as a part history and can be modified. These methods may help but they may also hinder by creating unexpected side-effects of changes. This may be helped by improving the computer-aided design support chain but in any case remember that it is necessary to check the model produced. Only the user can do this properly.

12.3 Parametrisation

True 3D dimensioning is rare. Hillyard [4] wrote his dissertation on the subject of dimensioning and tolerancing in 2D and 3D. The methods proposed by Hillyard for three dimensional dimensioning are complex to use because they used a matrix for modifying vertex positions and the surface and other geometry had to be modified indirectly. There are also three dimensional constraints to be take into consideration, which Hillyard describes. However, it is a pity that this method is not used as it is a direct way of manipulating objects.

What is currently used in CAD systems is a mixture of 2D dimensioning and operation parameter manipulation. In this sense, parametrisation means naming the operation parameters as well as 2D dimensions and allowing the user to change these, directly or indirectly. An interesting possibility which is found in many systems is to allow dimensions to have vales computed from other dimensions. This is important for creating product families. An early example, from Braid and Hillyard involved the analysis of simple spanners. These are not simply scaled versions of each other. The shape of each spanner depends on the size of the nut which it is to fit. As with other families, there is a principle, or driving, dimension and other dimensions defined in terms of these.

Taking a simplistic example, of the spanner shown in Fig. 12.40, you might have the following relationships between the driving dimension, *w*, and the other dimensions shown, *R hw* and *l*:

if $w < 4$ then $R = w + 2$ else if $w < 10$ then $R = w * 1.4$ else $R = w * 1.2$
if $w < 4$ then $hw = w$ else if $w < 18$ then $hw = w * 0.667$ else $hw = 12$
$l = R * 4.4$

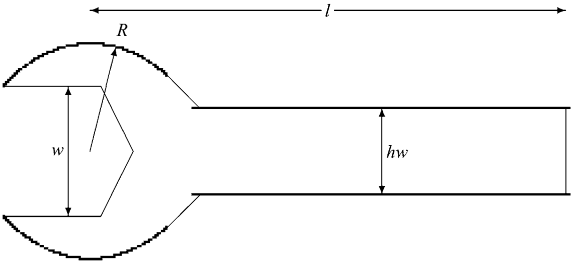


Fig. 12.40 Simple spanner model

This is not intended to be an "industrial strength" example, merely an illustration of how dimensions can be related. If the operation parameters are also available for functional descriptions then it may also be possible to add:

if $w < 4$ then $extrusion\_distance = 2$ else $extrusion\_distance = w * 0.125 + 2$

This gives a way of defining part families, although a real example may be quite complicated.

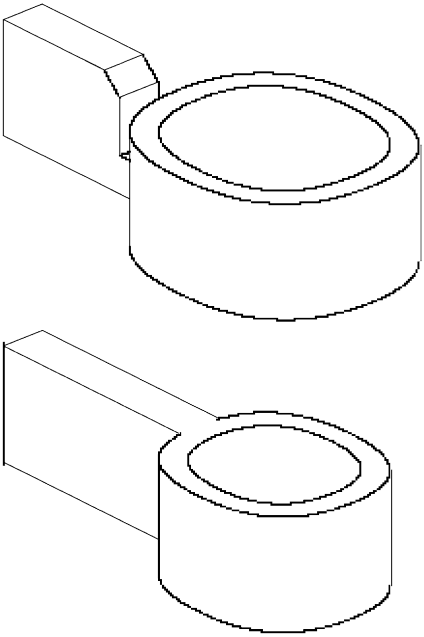
A complication to this is when the model topology changes for certain values of the driving parameter or parameters. This may mean that certain operations fail because the elements to which they refer disappear.

At the top of Fig. 12.41 is the product when the circular part is relatively large while that on the bottom is when the circular parts are smaller. When there is the gap between the vertical part and the circular part the top front edge of that is chamfered. Suppose that this is made in two steps from a basic shapes, such as those shown in Fig. 12.42. The first shape is defined and extruded, say 60 units. In the next step the long horizontal part is added and extruded 20 units. In the final step the final thick part is added and extruded 40 units. A chamfer is added to the front edge of the thick part.

If the following rules are applied:

if $R_1 < 60$ then $R_2 = R_1 + 10$ else $R_2 = R_1 * 1.2$
$l_1 = 100, l_2 = (R_1 + R_2) * 0.5$
if $R_1 < 60$ then $l_3 = l_1$ else $l_3 = l_1 - (R_2 - R_1) * 1.5$

Fig. 12.41 Simple product with variable topology



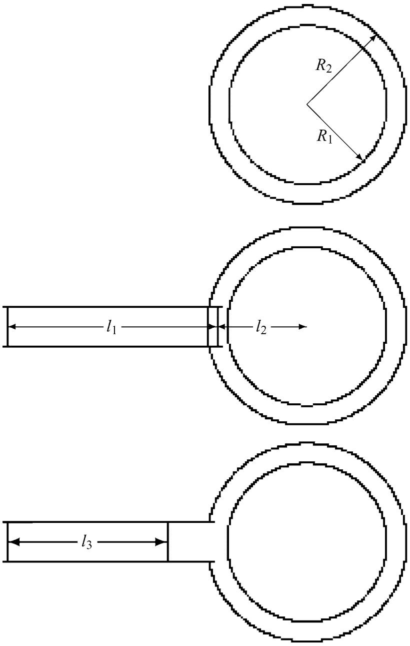


Fig. 12.42 Variable topology product basic shapes

Then it is clear that when R_1 is less than 60 the shape at the bottom of Fig. 12.41 will result and there will be no edge to chamfer. One solution to this is to produce two sequences. Another solution may be allow operations to be conditional, either checking parameter values or checking for the existence of topological elements. I have not seen the second solution in practice, though.

12.4 Undoing and Redoing

The method for handling changes described previously concerns having a list of operations invoked by the user and rerunning them. There is another method of undoing and redoing operations which can be used to undo work, especially if an operation fails. This is the so-called "blackboard" method.

In the blackboard method, the system maintains a list of changes to the state of computer memory. Each change may be the creation of an element, its modification or its deletion. In the case of modification, the old values of the element are recorded so that they can be reinserted. In the case of deletion, the old record is simply preserved.

If an undo is forced then the system simply runs back down the list until the previous check point, deleting new entities, changing back the values of modified elements and recreating deleted entities. This undo, though, is also recorded so that the user can "redo" a change which has previously been undone.

Note that this method also has an overhead in terms of memory use. For this reason, if a change is undone and then the model modified, it may no longer be possible to redo that change because the redo records have been deleted to save space.

A variant on this method, employed in I-DEAS for a while, was to copy complete models as part of the construction history. This also meant that the I-DEAS disc files could get quite large for complicated models.

12.5 Macro Languages and CAD System Programming

Macro languages and programming interfaces provide a way of extending a CAD system by adding user defined functionality. They both require more knowledge about how modelling operations work in order to use.

Macro languages and system programming use what is called the "Application Programming Interface", also known as an API, to access the main functionality of the CAD system. The main difference is that with system programming you use the computer language of the CAD system and have access to data structures and more functionality, while with macro-languages you use an interpreted interface.

Macro languages come in two forms: CAD system private form or interpreted computer language forms. The CAD system private form is like the command interpreter formats described at the beginning of this chapter in Sect. 12.1.1. The computer language forms are interpreted forms which are linked to the CAD system code. An advantage with computer language forms is that the syntax and several utilities already exist. Examples of the use of computer languages with CAD and modelling are the use of LISP by Kjellberg et al. of KTH during the early 1980s (also used later as an interface to the ACIS modelling kernel) and the use of Visual Basic in CATIA.

An Application Programming Interface might have the following components:

- Main modelling functions
- Element traversal and collection
- Euler operations
- Geometric utilities

The main modelling functions may include:

- Boolean operations
- Extrusion
- Wire model extrusion
- Reflection/symmetry
- Chamfering
- Blending
- Shelling
- Thickening
- Draft angle

An example of an API function, in some arbitrary form, might be:

API_BOOLEAN_ADD(RESULT **result_code, BODY *b1, BODY *b2)

where “b1” and “b2” are pointers to two bodies. If there are multiple body results, because b1 and b2 do not overlap, then these are chained together. The result code is to communicate warnings or errors as a list of results.

There are two levels to work at. On one level you can use high level functions. As an example of this, take a function to create a strange insert, shown sectioned in Fig. 12.43. The shape of the cut out is a truncated pyramid topped by a double cone as a sort of push fit connector. This is not intended to be like any existing connector and I apologise if it resembles any commercial product.

Suppose that there is a function called “MAKE_INSERT”. It is necessary (advisable) to define the function and its parameters in a formal way, something like this:

```
FUNCTION MAKE_INSERT(RESULT **result_code, point connect_pos,
                    vector connect_dir, real connect_rad, BODY *target);
INPUT PARAMETERS
    result_code     The error code
    connect_pos     The connection position
    connect_dir     The connection direction
    connect_rad     The connection radius
    target  The target object
OUTPUT PARAMETERS
    none
ERRORS
    Given radius is zero or negative
    Given direction vector is zero length
    Target object is NULL
COMMENTS
```

The function creates a connector insert shape at the given position and in the given direction, with a given radius. The connector shape is created in the given body.

The code might be something like the following:

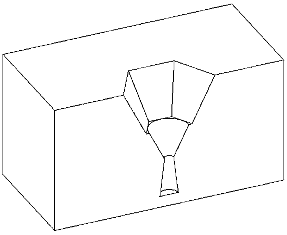


Fig. 12.43 Strange insert (section)

```
 1 FUNCTION MAKE INSERT(RESULT **result code, point connect pos,
 2                             vector connect dir, real connect rad, BODY *target);
 3 BEGIN
 4     if ( connect rad < toleps )
 5        result code = error("invalid connect rad");
 6     else if ( target == NULL )
 7        result code = error("NULL body given");
 8     else if ( ABS(zdir) < toleps )
 9        result code = error("Zero connect direction");
10     else BEGIN
11           BODY *b1, *b2;
12           FACE *f, *xf;
13           VERTEX *sv, *v;
14           point bpos, vpos;
15           vector fdir, xdir, ydir;
16           real area;
17
18           connect dir.normalise();
19           xdir = vector(-connect dir.y, connect dir.x, 0);
20           if ( ABS(xdir) < toleps ) xdir = vector(1,0,0);
21           ydir = connect dir*xdir; xdir = ydir*connect dir;
22           xdir.normalise(); ydir.normalise();
23           make pyramid(result code, connect pos, connect dir,
24                  connect rad, connect rad*0.5, connect rad, 6, &b1);
25           bpos = connect pos + connect rad*connect dir;
26           make cone(result code, bpos, connect dir,
27                  0.433*connect rad, 0, 0.866*connect rad, &b2);
28           add(result, b2, b1);
29           bpos = bpos + 1.5*connect rad*connect dir;
30           make cone(result code, bpos, -connect dir,
31                  0.2*connect rad, 0.2, 0.634*connect rad, &b2);
32           add(result, b2, b1);
33           subtract(result, b1, b);
34        END
35 END
```

Lines 4–9 are some simple checks on the parameters. Lines 18–19 set up a local coordinate system based on the connect_dir parameter. The connector shape base is created as a primitive on lines 23, 24. The two cone shapes are created as primitives and added to the base using Boolean operations (lines 25–32) before subtract the shape from the original body (line 33).

As a lower level programming example, take a function to create a pyramid or truncated pyramid. This can be created fairly easily using Euler operators, as it is a variant of the extrusion or sweep command.

The formal definition would be something like this:

```
FUNCTION make_pyramid(RESULT **result_code, point base_centre,
                      vector zdir, real base_radius, real top_radius,
                      double height, int side_num, BODY **pyrb)
INPUT PARAMETERS
      result_code                 The error code
      base_centre                 The centre of the pyramid base
      zdir                        The direction of the pyramid axis
      base_radius                 The radius of the pyramid base
      top_radius                  The radius of the pyramid top
      height                      The height of the pyramid
      side_num                    The number of sides of the pyramid
OUTPUT PARAMETERS
      pyrb                        A pointer to the pyramid made
ERRORS
      Given radii are both zero
      One or both radii are negative
      Given zdir vector is zero length
      Given height less than or equal to zero
      Number of sides less than three
COMMENTS
```

The function builds a pyramid with base centre at the given point and axis in the given direction. If the top radius is larger than the bottom radius then the pyramid will be inverted. If the two radii are equal then the result will be a prism, but one or both radii must be greater than zero.

The actual code for such a function is relatively easy to write using tools such as the Euler operators. For example, in a sort-of pseudo code you might have:

This is more complicated than the first example because it uses lower level operations to build up the object. As before, the function starts by testing the parameters in lines 5–11. Then, the necessary variables are declared in lines 13–20 and a local coordinate system set up in lines 22–25. The function first creates a base with *side_num* sides, in lines 26–58. The *polyareavec* function (line 60) computes the normal to a face based on the vertex positions so that the base surfaces can be computed (lines 61–66). The function then finishes the pyramid by stepping round the base creating the side faces (lines 86–125).

These two examples are intended to show you the style of different programming levels, the first using high-level functions and the second lower level manipulation functions. Which is appropriate depends on the nature of the problem, but obviously the second style requires a much more thorough knowledge of the CAD system functions.

```
FUNCTION make pyramid(RESULT **result code, point base centre,
                      vector zdir, real base radius, real top radius,
                      real height, int side num, BODY **pyrb)
BEGIN
   if ( base radius < 0 ) result code = error("base radius negative");
   else if ( top radius < 0 ) result code = error("top radius negative");
   else if ( (bottom radius < toleps) AND (top radius < toleps) )
      result code = error("bottom radius and top radius zero");
   else if ( ABS(zdir) < toleps ) result code = error("Zero Z-direction");
   else if ( height < toleps ) result code = error("Invalid height");
   else if ( side num < 3 ) result code = error("Invalid number of sides");
   else BEGIN
         FACE *f, *xf;
         LOOP *l, *ol, *xl;
         EDGE *e, *nexte, *xe;
         VERTEX *v, *ov, *sv, *xv;
         vector fdir, xdir, ydir;
         point cent, vpos, fp0, fp1, fp2;
         real ang, ang step, area, rad;
         int i;

         xdir = vector(-zdir.y, zdir.x, 0);
         if ( ABS(xdir) < toleps ) xdir = vector(1,0,0);
         ydir = zdir*xdir; xdir = ydir*zdir;
         xdir.normalise(); ydir.normalise(); zdir.normalise();
         mbfv(base centre + base radius*xdir, &v, &f, pyrb);

         ang = 0; ang step = (PI+PI)/side num;
         if ( base radius < top radius )
            BEGIN
               rad = top radius;
               cent = base centre + height*zdir;
            END
         else BEGIN
               rad = base radius; cent = base centre;
            END
         for ( i=0; i<side num; i++ )
            BEGIN
               ang = ang + ang step;
               if ( i == side num-1 )
                  BEGIN
                     xf = mfe(v, sv, v->get edge(),
                              sv->get edge());
                     xl = xf->get loop(); xe = xl->get edge();
```

```
                    xe->set curve(new CURVE(
                                new STRAIGHT(v->get pos(),
                                    sv->get pos())));
                END
            else BEGIN
                    ov = v;
                    vpos = cent + rad*cos(ang)*xdir +
                            rad*sin(ang)*ydir;
                    v = mev(ov, vpos, ov->get edge());
                    xe = v->get edge();
                    xe->set curve(new CURVE(
                                new STRAIGHT(ov->get pos(),
                                    vpos)));
                END
        END
    polyareavec(xf, &fdir, &area); fdir.normalise();
    xl = xf->get loop(); e = xl->get edge();
    ol = lopel(e, xl); f = ol->get face();
    xf->set surf(new SURFACE(
                new PLANE(cent, fdir)));
    f->set surf(new SURFACE(
                new PLANE(cent, -fdir)));
    if ( ((top radius < base radius) AND
                        (fdir DOT zdir > 0)) OR
    ((base radius < top radius) AND (fdir DOT zdir < 0)) )
        l = xl;
    else l = ol;
    ang = 0;
    if ( base radius < top radius )
        BEGIN
            rad = base radius;
            cent = base centre;
        END
    else BEGIN
            rad = top radius; cent = base centre + height*zdir;
        END
    e = ecclv(l, sv); ov = mev(sv, cent+rad*xdir, e);
    xe = ov->get edge();
    xe->set curve(new CURVE(
            new STRAIGHT(sv->get pos(), ov->get pos())));
    sv = ov;
    for ( i=0; i<side num; i++ )
        BEGIN
            nexte = eccel(e, l); xv = vcwel(e, l);
```

```
          fp0 = xv->get pos(); xv = vopev(e, xv);
          fp1 = xv->get pos();
          ang = ang + ang step;
          vpos = cent + rad*cos(ang)*xdir +
              rad*sin(ang)*ydir;
          if ( rad < toleps )
            BEGIN
              xf = mfe(xv, sv, ecclv(l, xv), ecclv(l, sv));
              fp2 = sv->get pos();
              fdir = (fp1-fp0)*(fp2-fp0); fdir.normalise();
              xf->set surf(new SURFACE(
                  new PLANE(fp0, fdir)));
              xl = xf->get loop(); xe = xl->get edge();
              ol = xe->get lloop(); l = xe->get rloop();
              xe->set curve(new CURVE(
                new STRAIGHT(xv->get pos(), vpos)));
            END
          else BEGIN
              if ( i == side_num-1 ) v = sv;
              else BEGIN
                  v = mev(xv, vpos, nexte);
                  xe = v->get_edge();
                  xe->set_curve(new CURVE(
                    new STRAIGHT(xv->get_pos(), vpos)));
                END
              xf = mfe(ov, v, ecclv(l, ov), v->get_edge());
              fp2 = ov->get_pos();
              fdir = (fp1-fp0)*(fp2-fp0); fdir.normalise();
              xf->set_surf(new SURFACE(
                    new PLANE(fp0, fdir)));
              xl = xf->get_loop(); xe = xl->get_edge();
              l = xe->get_lloop(); ol = xe->get_rloop();
              xe->set_curve(new CURVE(
                    new STRAIGHT(ov->get_pos(), vpos)));
            END
          e = nexte; ov = v;
        END
      xf = l->get_face();
      xf->set_surf(new SURFACE(
            new PLANE(sv->get_pos(), zdir)));
    END
END
```

As a final comment, an interesting possibility which, as far as I know, has not yet been exploited is the use of the DJINN format, described in Sect. 9.3.1, as an interface. The advantage of DJINN is that, if it is provided as an interface, new functionality can be implemented directly (or even indirectly using DJINN itself) and provided in macro form for other CAD systems. This would be an important step in generalising construction history standardisation.

12.6 Chapter Summary

This chapter deals with the use of command files in modelling and CAD, with part identification, with parametric parts and with programming. The aim of the chapter is to explain how command files were once used as input and then seem to have become a way of creating a construction history. The chapter also deals with the problem of model element identification. Part parametrisation is described as well as brief summaries of undoing and redoing and programming interfaces.

12.7 History, Parametric Parts and Programming Exercises

12.7.1 Topological Navigation

In the object in Fig. 12.44, determine the topological sequences for different elements.

Try to find expressions for the marked elements, faces 1, 2, 3, edge 1 and vertex 1 in the figure. Assume that FRONT is in the $+X$ direction, BACK in the $-X$ direction, RIGHT in the $+Y$ direction, LEFT in the $-Y$ direction, TOP in the $+Z$ direction and BOTTOM in the $-Z$ direction.

12.7.2 Permanent Naming Exercise with Chamfer

Make a square 100 × 100 and extrude it upwards 100 units. Chamfer the top edge, by 10 for example, as shown in Fig. 12.45.

Now edit the sketch so it looks like that in Fig. 12.46 and allow the CAD system to redo the sequence.

Two variants of what might happen are shown in Fig. 12.47. Either one of the top front edges is chamfered or both are. If only one is chamfered then the CAD system does not apply persistent naming at the 2D level.

A variant on this is to create the 100 × 100 × 100 cube with a chamfered edge as before and then to go back and, after the cube has been made but before the chamfer, insert a cutout from the base face upwards through the chamfer.

12.7.3 Writing a Command Sequence

Write a command sequence to create the object in Fig. 12.48.

Create the object by making a square, 100 × 100 and extruding this 40 units. On the top face, a rectangular shape 50 × 25 should be created and extruded 60 units. On the front face a 50 × 20 rectangle is created and extruded 45 units. Finally, the edge marked with an error on the bottom right of the figure should be chamfered. The exercise is about writing the command sequence, not about making the object in a CAD system.

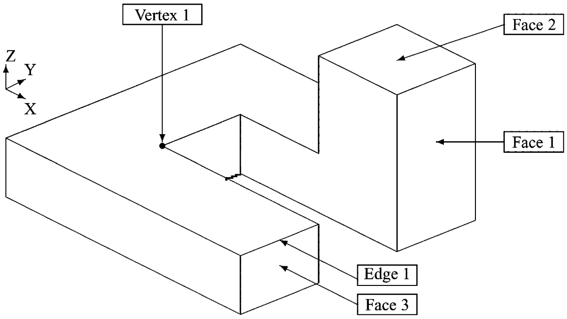


Fig. 12.44 Basic figure for topological navigation

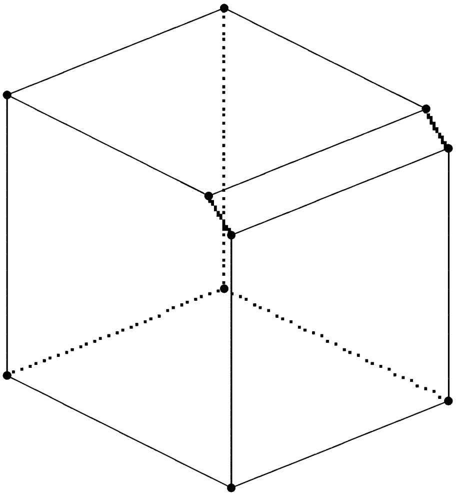


Fig. 12.45 Cube with chamfered edge

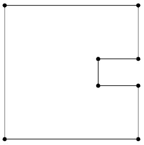


Fig. 12.46 Modified basic shape for chamfer test

12.7.4 Parametric Object Exercise

Use your CAD system to produce a parametric model. The shape to be parametrised is shown in Fig. 12.49.

A is the driving dimension.

$B = A * 1.5$

if $A < 50$ then $C = 5$ else $C = A * 0.2$

if $A < 50$ then $D = A * 0.65$ else if $A < 100$ then $D = A * 0.55$ else $D = A * 0.5$

$E = D$

$F = D * 0.25$

if $A < 50$ then $G = A/6$ else $G = A/7$

$H = A * 0.25$

if $A < 60$ then $I = A/2$ else $I = A * 0.6$

$J = H * 0.5$

Can you set a condition on A? Try setting the condition:

if $A < 50$ then $A = 50$

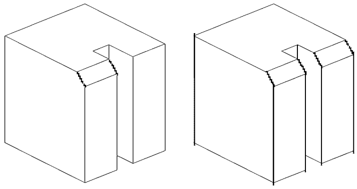


Fig. 12.47 Modified shape with chamfered edge (variants)

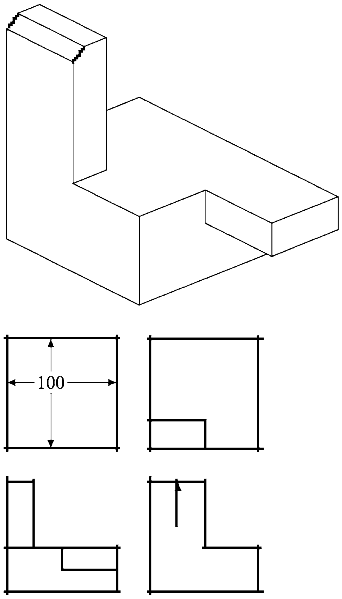
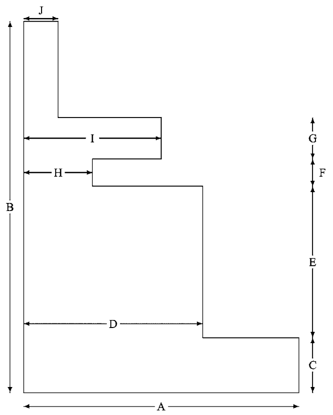


Fig. 12.48 Object to be created with a command sequence

Fig. 12.49 Object to be parametrised



References

1. Stroud, I.A.: BUILD picture book. CAD Group Document 104, Cambridge University Computer Laboratory, November (1979)
2. Várady, T., Gàal, B., Jared, G.E.M.: Identifying features in solid modelling. Comput. Ind. **14**(1–3), 43–50 (1989)
3. Mun, D.-W., Han, S.-H.: Identification of Topological Entities and Naming Mapping Based on IGM for Parametric CAD Model Exchanges, Int. J. CAD/CM. 5(1), 69–81 (2005)
4. Hillyard, R.C.: Dimensioning and tolerancing in shape design. Ph.D. Dissertation, University of Cambridge Computer Laboratory, Technical Report No. 8, June (1978)

Chapter 13
Assemblies

Most products consist of a set of linked objects organised in a structure called, at least sometimes, an “assembly”. The classical assembly structure defined in the BUILD system, perhaps even earlier, has a structure consisting of an assembly, containing a list of instances each referring to a single object or an assembly. The use of instances and transformations also appears in computer graphics, see Newman and Sproull [1], for example.

The structure mentioned in Sect. 2.10 is shown in Fig. 13.1. The modern structure also includes constraints, which define relations between elements of the assembly.

With the exception of the CONSTRAINT entity this is the classic form of assembly description. This is not only a practical and realistic way of representing assemblies but has the important advantage of reducing the memory and disc usage for complex assemblies. It is practical because every part is represented fully only once, which makes it easy to identify the basic components needed in a product. Another practical result is that it is possible to animate the assembly, simulating motion for example, just by changing a transformation matrix and not the whole geometry associated with an object.

Constraints set up relationships between assembly elements that need to be respected when changing assembly element positions. Examples of constraints are planar face contacts or cylinder in cylindrical hole. The constraints link instances not objects because it is the instances which are unique in the assembly.

Some examples of assembly structures (from Stroud [2]) are shown in Figs. 13.2, 13.4, 13.6 and 13.8.

Figure 13.2 shows a simply assembly where all parts in the assembly are unique. There are no repeated sub-structures in the assembly and, because there are only a few elements, it may be conveniently represented as a flat assembly. The structure is shown in Fig. 13.3. (To be pedantic, the instances are actually chained together in a list, but they are shown here as though the Group of Objects references each one separately because it seems clearer.) In the figure, circles are used to represent objects or object groups, while triangles are used to represent instances.

I. Stroud and H. Nagy, *Solid Modelling and CAD Systems*,
DOI: 10.1007/978-0-85729-259-9_13, © Springer-Verlag London Limited 2011

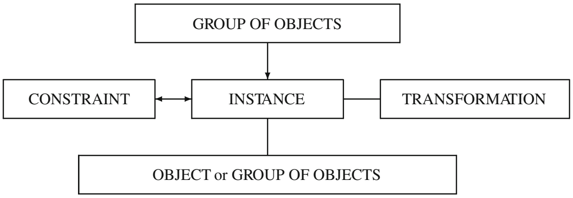


Fig. 13.1 General assembly datastructure

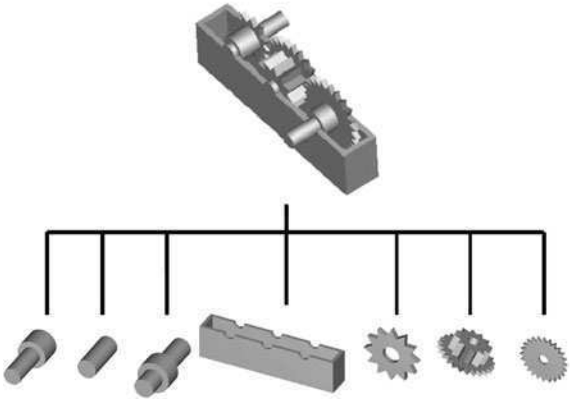


Fig. 13.2 Simply assembly

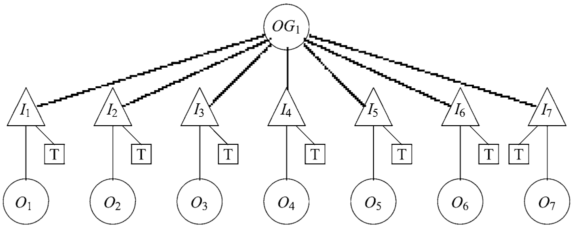


Fig. 13.3 Assembly datastructure for the simple assembly

At the top level there is one group of objects, OG1. This has a list of seven instances, $I_1 \ldots I_7$. Each instance has a transformation and refers to one single object, in this assembly. The constraints are not shown, but instance I_4, which references the frame object, would probably be fixed in position so that the other instances move to it. This is a bit like physical assembly where the frame would be put in the assembly area and the other subparts moved to it. Constraints will be dealt with later, in Sect. 13.4, so will not be dealt with further here. The constraints serve to determine the transformation matrices which change the apparent position of objects in the assembly.

The next assembly example, Fig. 13.4, shows the sort of structure that might result with common standard elements like nuts and bolts. Here, there should be only one full model for each standard element and multiple references. This is important both for saving space and for the standard Bill of Materials (BOM) determination (Sect. 13.7). In the assembly there are only four "real" objects while there are ten instances. Each instance appears to be a separate object. In the original assembly

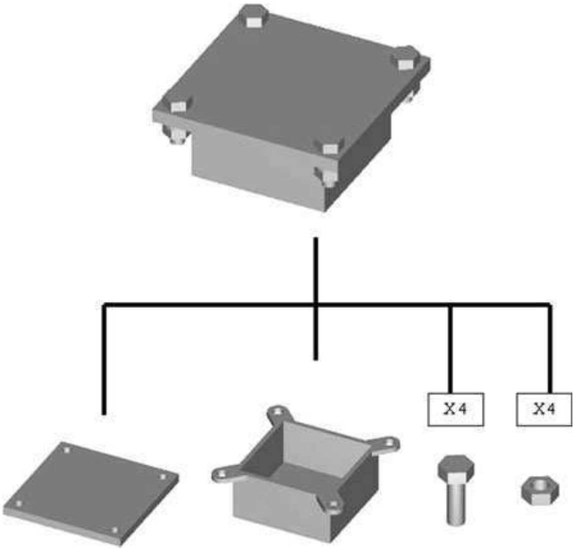


Fig. 13.4 Assembly with repeated standard elements

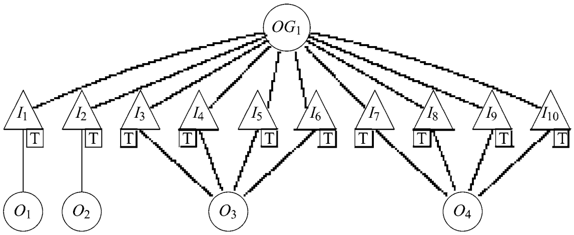


Fig. 13.5 Assembly datastructure for the standard elements assembly

datastructures and possibly also now, each real object model maintained a count of the number of times that it was used, meaning that the bolt model would have a count of four, as would the nut model. For creating the Bill of Materials it is simply necessary to print out the model name together with this count to give the required information. This is one reason why it is important to use the instancing mechanism instead of just copying models. It is harder, and time consuming, for a CAD system to compare the geometry and topology of two models to see if they are the same or different than to compare two model references to see that they refer to the same object.

The structure resulting from such an assembly is shown in Fig. 13.5. For the repeated elements, the transformations associated with each instance are used to place the graphics image of the part in different places, giving the effect of multiple objects. Graphics systems use transformation matrices for several purposes and it is easy to pass over the instance transformation matrix to the graphics system to view the different instances.

Another type of structure is shown in Fig. 13.6. Here, there is a repeated subassembly as an illustration of the way in which assembly structures can be hierarchic. Logical structuring of the assembly is important, both for efficiency and also

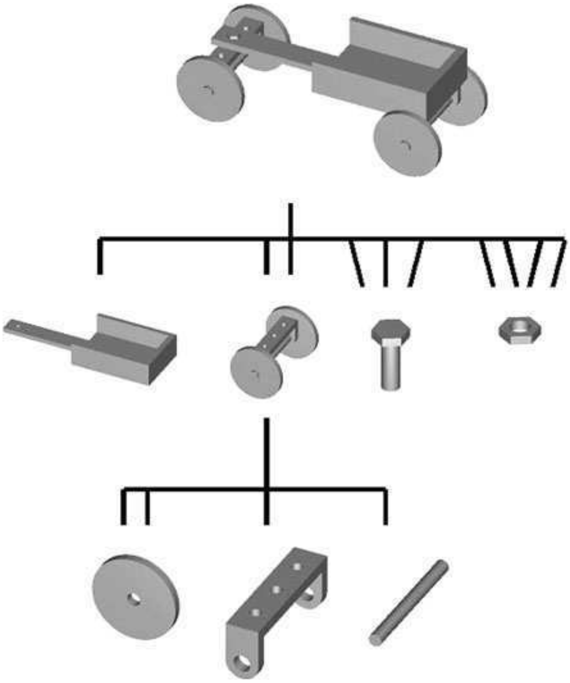


Fig. 13.6 Assembly with sub-assemblies

for communicating design intention. With very complex assemblies the space saving aspects of using instances of subassemblies is still important. One aspect of this is that the subassemblies should be treated as rigid. This is because the whole assembly is repeated and it is unreasonable to expect the system to have an assembly with one set of transformations in one place and another set of transformations in another place. So, if a subassembly is changed, then all instances of that subassembly are expected to change. In order for a CAD system to have different positions for subassemblies then it would be necessary to maintain a hierarchy of transformation matrices or, easier, to copy the subassembly structure.

Note that there are only six unique models in this structure, although it appears that there are sixteen pieces, as illustrated in Fig. 13.7. Although this is a simple assembly, there is a significant saving of memory. As products become more complex so the saving increases. However, it is not always possible to create multi-used subassemblies for space saving purposes, but structuring an assembly may be important anyway. The reason, mentioned above, of communicating design intention is important. For production, a subassembly may represent a complex component which can be physically assembled in parallel with the main assembly, hence saving time.

The final example of an assembly structure is that developed by Lars Wingård of the Royal Institute of Technology (KTH) in Stockholm. The structure is shown in Fig. 13.8 and is a model of an ASEA (as it was then) industrial robot by Wingård and Palm. The model was used both as a demonstration of animation and, later,

Fig. 13.7 Assembly datastructure for the repeated sub-assembly structure

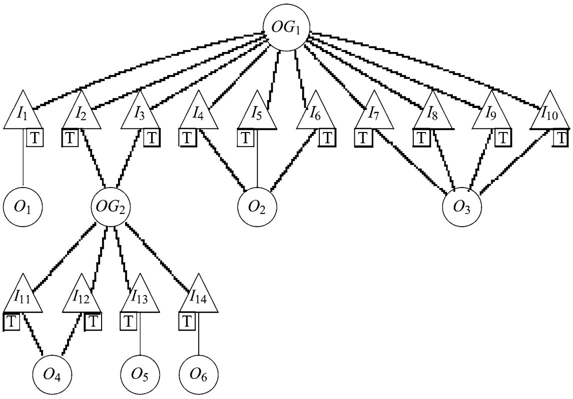
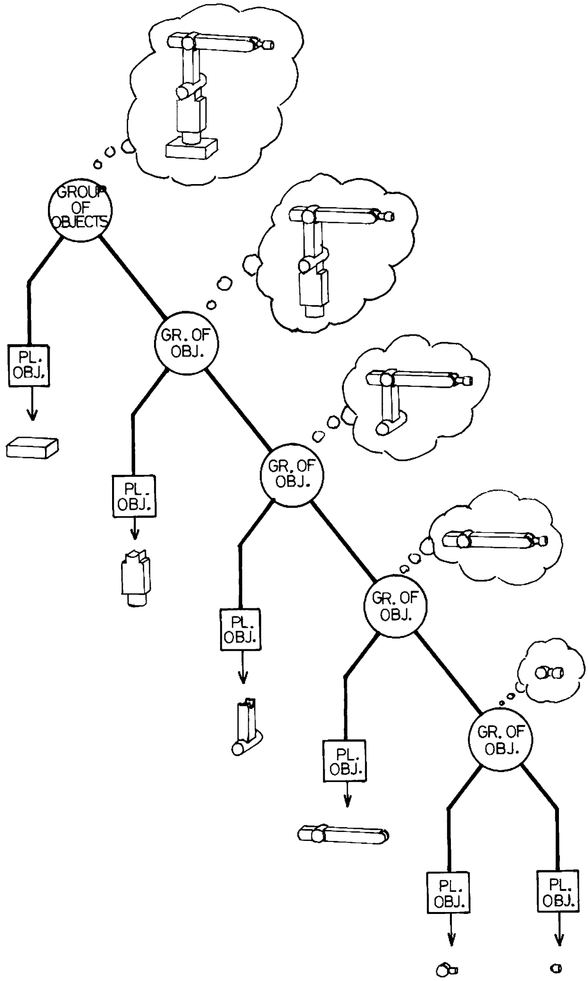


Fig. 13.8 Complex kinematic assembly (by Wingård)



for simulation of welding. Wingård arranged the structure so that a transformation corresponded to each driven joint of the robot. Thus, by modifying the transformation matrices, the robot model was made to move. Each subgroup of objects was treated as a rigid subassembly which could be moved as a whole according to the programmed joint movements. The corresponding datastructure is shown in Fig. 13.9.

This is, perhaps, an extreme structure and has, to a certain extent, been replaced by the use of constraints. However, this is an efficient method, since it does not need a constraint solving process to calculate the transformation matrices. Note that there are only six unique objects in the assembly, each used once, while there are ten instances.

These examples are intended to show that there are many ways to determine an assembly structure and, as usual, it is up to the user to be aware of what he or she is doing to use assemblies effectively.

One thing to remember is that it is important to use the instancing mechanism rather than simply copying models. Although copying models may seem to produce the same result this represents a waste of space. Also important is to reflect on the structure of the design and, if possible, to group elements into logical groups. As with creation of single model designs, though, the user's reasoning for an assembly structure may not be clear to other people accessing the data. The lack of easy mechanisms for annotating models is still a problem in CAD and this information may have to be put in separate documents, even on paper.

13.1 Instances

An instance is a reference to an object or sub-assembly together with a transformation to define how that object or subassembly is positioned and oriented with respect to the global coordinate system. You should understand that an instance entity is very simple and so is much smaller, in terms of memory use, than the object or object group which it references. Another thing to note is that an instance is not allowed to reference any assembly or group-of-objects which contains it directly or indirectly. An example of a self-referencing assembly is shown in Fig. 13.10.

Obviously, if such an assembly structure were allowed then it would be traversed endlessly. In order to avoid this, when you ask to add an assembly to an assembly, then the assembly being added is traversed to see that it does not contain the assembly to which it is being added. This is done automatically by the system, but you can try creating a self-referencing assembly to see what the system does.

13.2 Transformations

Transformations have been described in Sect. 5.2.2. Transformations work assuming the homogenous coordinate systems described in Sect. 5.2. As described in Sect. 5.2.2, a transformation has the following form:

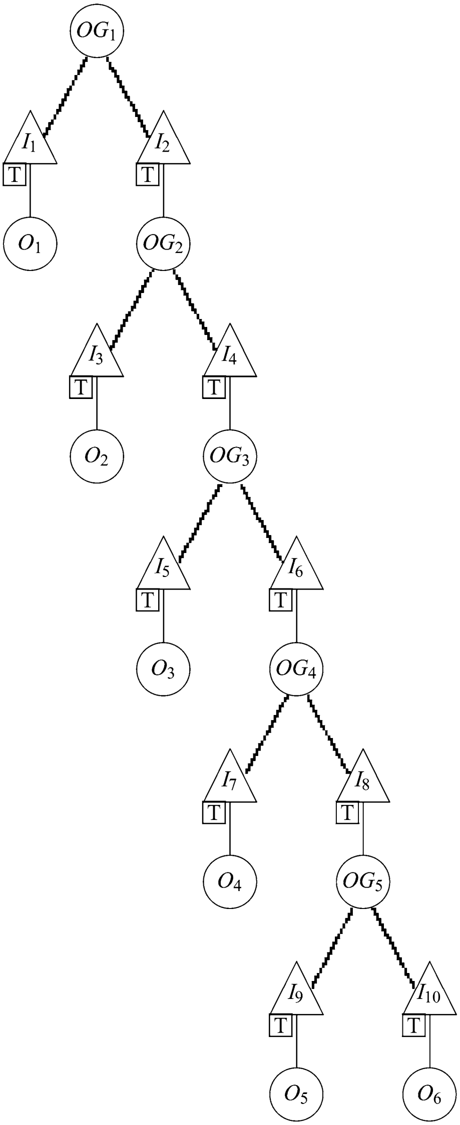


Fig. 13.9 Wingård's assembly datastructure for the kinematic assembly

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix}$$

The part:

$$\begin{bmatrix} a & b & c \\ e & f & g \\ i & j & k \end{bmatrix}$$

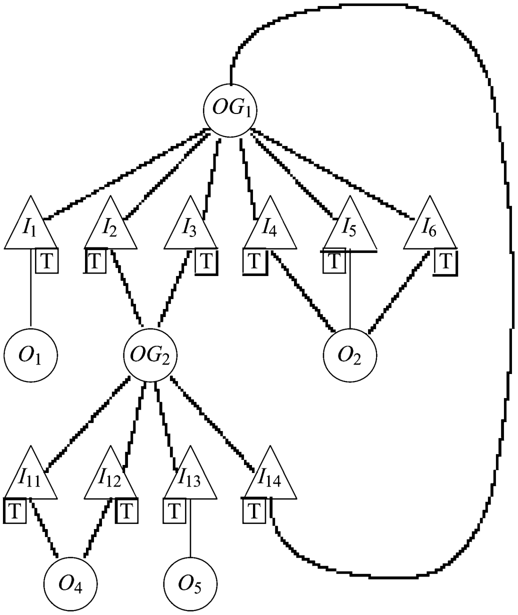


Fig. 13.10 Assembly with self-reference

describes scaling and rotation. The part:

$$\begin{bmatrix} d \\ h \\ l \end{bmatrix}$$

defines the translation. The elements m, n, o can all be zero, though not necessarily, but this makes the matrix easier to define. The element p should, nowadays, usually be 1. Nowadays, the transformation information is created through the use of constraints rather than explicitly, but more of this later.

Transformations need not be represented as 4 × 4 matrices, but this is a convenient common form. Transformations could, for example, be held with rotation, translation and scaling information held separately. Holding scaling separately is, perhaps, used to avoid having uneven geometric scaling. However, scaling should not be used as part of a transformation matrix in an assembly.

Figure 13.11 illustrates the use of transforms to change positions and orientations. The group of objects consists of four instances, each referring to a 20 × 20 square, centred at the origin. Instance I_1 has a transformation matrix which moves the square (−30, −20, 0). Instance I_2 has a transformation matrix which is the identity transformation and so makes no change. Instance I_3 rotates the square and moves it 30 units in the X direction while instance I_4 is another simple translation, by the vector (60, 10, 0).

In practice, not all transformations which can be represented are useful. Section 5.2.2.2 mentions some of the "blacklisted" transformations which are

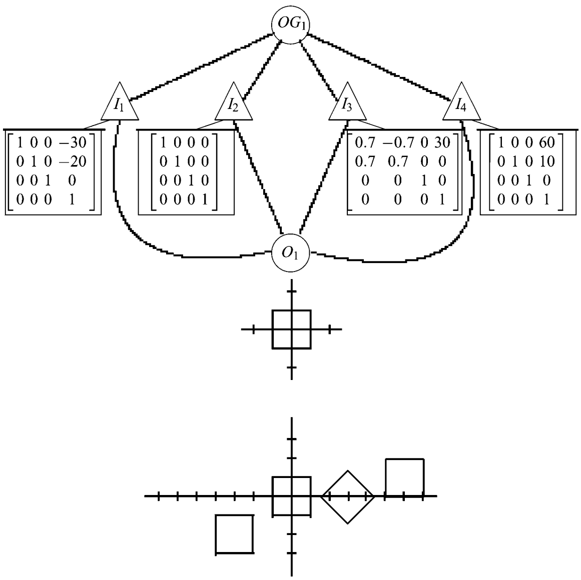


Fig. 13.11 Transforming an object

usually disallowed because they cause problems. For assemblies it is possible to be more rigorous and there is an informal proposition, here, that only rotations and translations should be allowed in assemblies. The reason for this is that, when physically assembling products, it is only possible to translate and rotate parts, not to transform them physically, except in the restricted case of parts with elastic material properties which are ignored here. CATIA allows symmetry transformations in assemblies. The reason for this may be the use of CATIA in the aircraft industry. Aircraft CAD assemblies are so big that they stretch the limits of the technology. Allowing, for example, symmetry transformations for wing assemblies would, of course, reduce the size of the assembly. However, it can lead to unnecessary duplication and other problems.

There is an exercise on symmetry assemblies which is intended to indicate that the presence of an easy way of doing something may not be desirable because it makes it easy to do the wrong thing. In general, my opinion is that symmetric parts should be considered and created, if needed, at the part level and that assemblies should have only rotations and translation transformations.

13.3 Exploding Assemblies

This topic does not really belong here, but it is difficult to see where to put it. It is sort-of “anti-constraint”, so has been put here. It is useful as a precursor for constraining assemblies.

An “explosion”, here, means that the elements in an assembly are moved apart temporarily, disregarding any constraints. The reason for doing this is so as to be able to “see” the subcomponents and, hence, to be able to select elements to create constraints.

There are two types of explosion, manual and automatic. With the manual method the user moves the objects, for example by dragging them out of the way. With the automatic method the system calculates new positions for the objects and moves them. Since the manual method is up to the user, only the automatic method will be described here.

There are different algorithms for automatic explosion, two are described here. Both of these use the centre of the assembly as a start point and move the instances outward from this. The centre of the assembly can be calculated conveniently using the boxes surrounding the objects in the assembly. It is usual in modelling to have a box, or some other simple shape, surrounding elements in a model and the model itself. Each instance takes this box, copies it and transforms the copy to produce the instance box. The instance boxes are then “summed” to find the box surrounding the objects in their current positions. The centre of this summed box is the centre of the assembly.

In the first algorithm, the CAD system uses the positions of the instance box centre relative to the assembly centre to define a vector direction along which to move the instance. The size of the vector offset may be given by the user, arbitrary or calculated. In the second algorithm the positions of the instances are put on a circle with some radius on an arbitrary plane through the centre of the assembly.

The radius of the assembly can be calculated from the instance boxes, for example. The sum of the diagonals of all boxes can be thought of as the length of the perimeter of the assembly circle. The radius is easily calculated from the sum by dividing by 2π.

An important thing to note with assembly explosions is what happens to the fixed instances in the assembly and existing constraints. There is an exercise about this at the end of the chapter. What is interesting, first of all, is whether or not an instance which has been marked as fixed is moved. If it is moved, then the question is whether it will be moved back to its original position if the constraints are applied. It is worth checking with a simple test assembly before you have to use the tool for serious work.

13.4 Constraints

Constraints are relations between simple geometric elements, that is, planes, lines and points, which determine positions and orientations of the bodies to which those elements belong. The constraints form a connection graph over and above the normal assembly structure described above. As objects are modified, the constraint graph is verified and new solutions calculated. Aligning two entities involves calculating a transformation matrix, which is then applied to an instance to change the apparent position of the object or sub-assembly to which it refers.

The constraints fix various degrees of freedom for the objects. An object has six degrees of freedom, three translational and three rotational. Removing degrees of freedom means that relations are established between different objects and the relations, or constraints are used to define object positions. In order to find the positions of the objects based on the constraint set you have defined, the CAD system uses what is called a "constraint solver". This works through the constraint set finding possible positions for each object, if it can. If the system of constraints has been correctly established then all the objects will be in the expected places. If too few constraints have been established then one or more objects may have strange positions. Note, though, that sometimes it is necessary to leave one or more degrees of freedom because the assembly is not rigid, but has mechanisms. A simple mechanism is a cylindrical axle in a cylindrical hole, which allows the axle to turn around the hole axis. More about mechanisms later.

Constraints are set between instances, not directly between model elements. To understand this, consider Fig. 13.12. Suppose that there is a constraint of coincidence between the top face of one block and the bottom face of the other block, as shown on the left of the figure. If the constraint were between the faces of the object referenced then this would mean that the block would be flat. The constraints should refer to the instances and, through these, to the faces to be constrained. What this means in practice is that either the point and normal of the

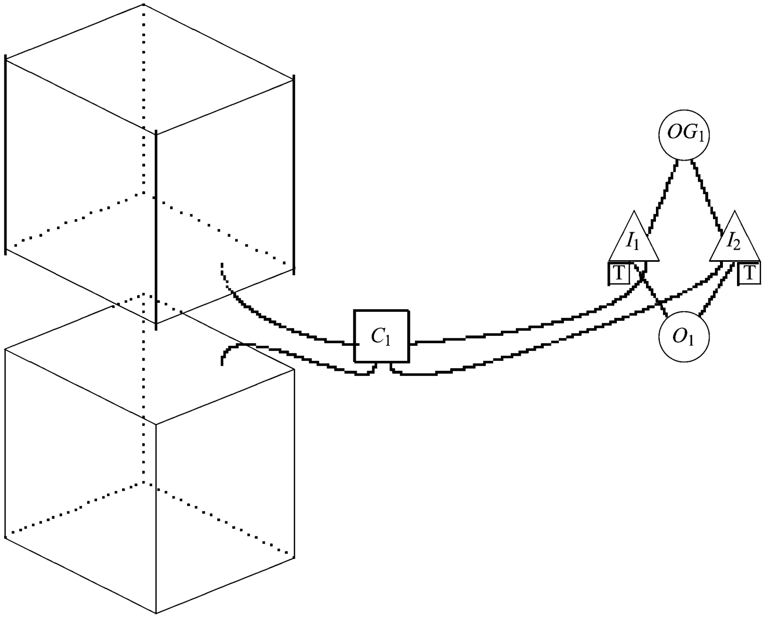


Fig. 13.12 Constraint references

surface are copied and transformed or that the constraint contains a face reference and the point and normal information are copied whenever the constraint is evaluated.

Constraints appear in various ways. The common ones are summarised in Table 13.1.

It is currently usual that the faces are planar faces, except for the special cylinder case, and that the edges are straight. This could be relaxed slightly and some extra special cases allowed, but the constraints generally work with points, lines and planes so it is more logical to restrict the constraints to planar elements.

The topological constraints define transformations which are multiplied into the transformation of the constrained instance. The geometry of each constrained topological element is used to define a simple element, point, line or plane, which is then, in turn, used to calculate a transformation with the geometric constraints, defined below.

Table 13.1 Constraints

Constraint	Form	Comments
Fixed		This is to fix an object in its current position
Parallel	Dir–dir	Parallelism between two elements
Perpendicular	Dir–dir	Perpendicularity between two elements
Concentric	Point–point	Concentricity between two elements
Vertex–vertex	Point–point	Simple move from one vertex to another
Vertex–edge	Point–point	The edge alignment point is calculated to be the closest point on the edge
Vertex–face	Point–point	The point to which to move is calculated from the face surface
Edge–edge aligned	Point–point dir–dir	One point on the movable line is moved onto the fixed line and then the line rotated so that the directions are parallel and in the same direction
Edge–edge opposite	Point–point dir–dir	Same procedure except that the movable line is rotated to be in the opposite direction
Edge–face	Point–point dir–norm	A point on the line is moved to the face and the edge direction rotated to be perpendicular to the plane normal
Face–face aligned	Point–point norm–norm	This is where the faces are in the same direction. The point of the movable face is moved onto the other face. The movable face is rotated so that the normal is in the same direction as the fixed face
Face–face opposite	Point–point norm–norm (anti)	This is where the planes are in the opposite direction. The same procedure as before, but the movable face is rotated so that the normal is in the opposite direction to the fixed direction
Cylindrical face–edge	Point–point dir–dir	The cylinder axis is taken as a line and the edge and line are aligned
Cylindrical face–cyl. face	Point–point dir–dir	As before, the cylinder axes are used for a line–line constraint

It is also possible that, instead of making the elements exactly coincident, the CAD system allows you to define an offset value. If the CAD system does not allow this it may be possible to create supplementary geometric elements, planes, lines and points, and to align these to get the desired effect. Note, though, that if you define two points to be at a certain distance then you are implicitly defining that one point lies on a sphere with the other point at the centre. You are not defining an exact position for the point. Similarly, a point at a distance from a line defines that the point lies on a cylindrical surface with the line as an axis. Similarly, the line will be tangent to a sphere with radius as the given distance. A point at a distance from a plane means that the point lies on a plane offset from the original plane.

13.4.1 Fixing an Instance

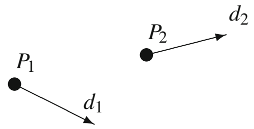
This is mentioned here, but is not quite the same as the other constraints. This is important because one of the instances in the assembly should be "grounded" or fixed and the others moved to it. For the user this is important because sometimes the user expects one part to move whereas, when a constraint is defined, the other part moves, depending on some internal decision. Fixing one instance, then moving its constrained partners, then the partners of the constrained partners, and so on, is a clearer process. Fixing an instance removes all its degrees of freedom.

13.4.2 Parallelism

Parallelism can be used to establish a relationship between edges or surfaces. This is, again, easiest to do with straight edges and planar faces. A constraint of parallelism removes two of the rotation degrees of freedom, but rotation about the edge or surface normal is still possible and three translation degrees of freedom remain.

Figure 13.13 illustrates the geometric elements used for calculation. A constraint for parallelism involves a rotation. The rotation axis is defined from the two line directions or the surface normal directions. If the directions are already parallel and aligned then there is nothing to do. If the directions are parallel but not aligned (same direction) then there is an option to rotate the lines, this can be done about any axis perpendicular to the line. If the two directions are not aligned then the cross product of the line directions defines the rotation axis, the line direction defines the X-axis, say, and hence the rotation angle can be determined.

Fig. 13.13 Making two lines parallel



13.4.3 Perpendicularity

Perpendicularity is similar to the parallelism constraint, it removes one rotation degree of freedom. If the two directions—line directions or surface normals—are parallel then the rotation axis can be any axis perpendicular to this direction.

13.4.4 Concentricity

Concentricity is a constraint between a point and a circular curve or between circular curves. It removes two rotational degrees of freedom and two translational degrees of freedom. If two circles are involved the normals to the circles are aligned to be parallel and then the centre of one circle moved to be on a line through the centre of the other circle with direction of the circle normal. If a point and a circle are involved then the point is moved to be on a line through the centre of the circle with direction of the circle normal. Alternatively, the circle centre is moved so that it is on a line through the point with direction of the circle normal.

13.4.5 Aligning Two Points

Figure 13.14 shows what the alignment means.

To align two points with the coordinates: $p_1 = (x_1, y_1, z_1)$ and $p_2 = (x_2, y_2, z_2)$ you get the transformation matrix:

$$\begin{bmatrix} 1 & 0 & 0 & x_1 - x_2 \\ 0 & 1 & 0 & y_1 - y_2 \\ 0 & 0 & 1 & z_1 - z_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

13.4.6 Aligning a Point and a Line

The basic procedure is illustrated in Fig. 13.15. The point on the line closest to the given point is calculated and then the given point is aligned with this calculated point.

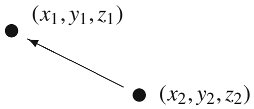


Fig. 13.14 Aligning two points

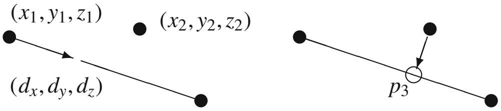


Fig. 13.15 Aligning a point and a line

The given data are the line start point, $p_1 = (x_1, y_1, z_1)$, the line direction, a normalised vector, $d = (d_x, d_y, d_z)$ and the point to be aligned, $p_2 = (x_2, y_2, z_2)$. The formula for calculating the closest point is:

$$p_3 = p_1 + ((p_2 - p_1) \cdot d) * d.$$

The transformation matrix is then:

$$\begin{bmatrix} 1 & 0 & 0 & x_3 - x_2 \\ 0 & 1 & 0 & y_3 - y_2 \\ 0 & 0 & 1 & z_3 - z_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where (x_3, y_3, z_3) are the coordinates of p_3.

13.4.7 Aligning a Point and a Plane

The alignment is to move the given point to a point on the plane closest to the given point, as illustrated in Fig. 13.16.

The given data are a point on the plane, $p_1 = (x_1, y_1, z_1)$, the plane normal, a normalised vector, $n = (n_x, n_y, n_z)$ and the point to be aligned, $p_2 = (x_2, y_2, z_2)$. The formula for calculating the closest point is:

$$p_3 = p_2 - ((p_2 - p_1) \cdot n) * n.$$

The transformation matrix is then:

$$\begin{bmatrix} 1 & 0 & 0 & x_3 - x_2 \\ 0 & 1 & 0 & y_3 - y_2 \\ 0 & 0 & 1 & z_3 - z_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

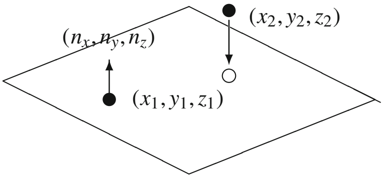


Fig. 13.16 Aligning a point and a plane

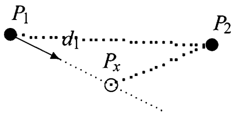


Fig. 13.17 Aligning two lines—point to line alignment

13.4.8 Aligning Two Lines

This is more complicated than the point transformations because the line has to be rotated as well as translated. There is also a slight complication because the lines have a direction as well, and the alignment could be so that they have the same direction or have opposite directions.

Assume that the lines are defined by a point and a direction. Anyway, if they are defined by two end points then it is easy to calculate the direction, provided the end points are not coincident, in which case there is an error condition. The first step is to move the point of one line onto the other line (Fig. 13.17). The closest point on a line to a given point is easily calculated as:

$$P_x = P_1 + ((P_2 - P_1) \cdot d_1)d_1$$

where P_1 and P_2 are the line points and d_1 is the normalised direction of the first line.

The first step in aligning the lines is, therefore, a translation of $P_x - P_2$.

Assuming that the point P_x has the coordinates: $P_x = (Px_x, Px_y, Px_z)$ and $P_2 = (x_2, y_2, z_2)$ you get the transformation matrix:

$$\begin{bmatrix} 1 & 0 & 0 & Px_x - x_2 \\ 0 & 1 & 0 & Px_y - y_2 \\ 0 & 0 & 1 & Px_z - z_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The cross product of the two line directions gives the sine of the angle between them, the scale product defines the cosine, from these two values it is possible to determine the angle, θ, between the two lines. The point P_x and the cross product of the two line directions defines the rotation axis. If the magnitude of this is zero then the lines are aligned, in the same or opposite directions.

The line direction of the fixed line, the normal to the two lines and the mutual orthogonal direction define a local system of coordinates for the transformation, Fig. 13.18. The Z'-axis is the rotation axis. The transformation matrix for the rotation can be calculated by rewriting the line vector in terms of the local coordinate system, rotating it by the required angle around the local coordinate system Z' axis and then retranslating the result back into the global system. The required angle is defined using the x- and y-coordinates in the local coordinate system. This transformation matrix is then combined with the translation matrix, calculated earlier, to give the final matrix.

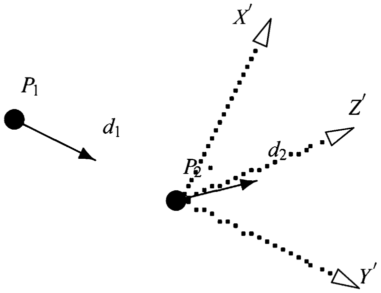


Fig. 13.18 Aligning two lines—direction alignment

13.4.9 Aligning a Line and a Plane

This is illustrated in Fig. 13.19. The first step is to move the point of the line to the plane and the second to rotate the line into the surface so that its direction is perpendicular to the plane normal. The transformation of the line point onto the plane is the same as for the case of aligning a point and a plane.

The rotation axis is defined by the cross product of the line direction and the plane normal. If these two are parallel then any vector normal to the plane normal can be used as a rotation axis. As with aligning two lines, the plane normal, the rotation axis and the mutually orthogonal vector form a local coordinate system. The line is rotated around the rotation axis by the required amount to give the rotation transformation matrix.

If the plane is to be moved to the line, then the point of the plane is moved to the line using the point–line constraint calculation, and then the surface normal rotated so as to be perpendicular to the line direction.

13.4.10 Aligning Two Planes

This is basically a point–plane constraint together with a rotation of one normal to be parallel and aligned, or parallel and opposed. A transformation is calculated to move the point of the plane to be moved into the fixed plane. Then, a rotation transformation is calculated to rotate the movable plane so that its normal is parallel to the fixed plane normal (aligned or opposite direction). Figure 13.20

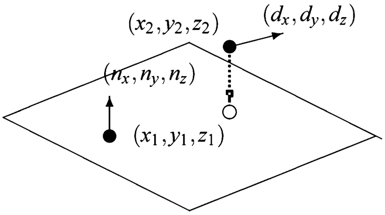


Fig. 13.19 Aligning a point and a plane

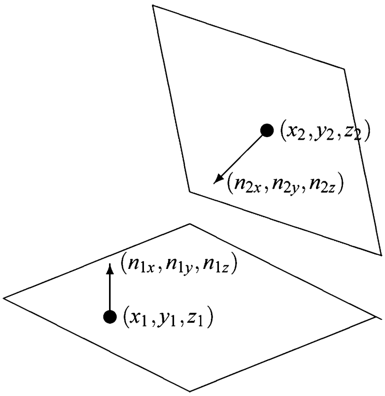


Fig. 13.20 Aligning two planes

shows the geometric elements for the calculation. The rotation axis is defined by the cross product of the two plane normals.

All the details of this should now be familiar so will not be repeated.

13.5 Kinematic Mechanisms

In the sense used here, kinematic mechanisms differ from simple constraints because they are groups of constraints with a logical purpose. Building up an assembly with simple constraints may produce the same constraint set as that created by a kinematic mechanism package but this is not certain. A kinematic mechanism package provides a set of mechanisms, each with a framework of constraints with empty slots which are filled by the user. So, for example, a rotational mechanism might require the user to identify the axis, the stop face and limits. The user is, therefore, guided into producing a constraint set which is logical. In addition, kinematic packages should contain extra facilities not found in the assembly package. Kinematic packages exist in different forms, from the dedicated software to simpler ones included in CAD systems. This section is intended to give a brief overview of a few types rather than an exhaustive survey.

Kinematic mechanisms are intended to animate assemblies so that a designer can get an impression of how things function. As well as watching the outside of the mechanism it is possible to set some parts as transparent so as to see the interior of the mechanism. Note, though, that in general it is too costly to calculate positions from the models themselves. For example, a gear mechanism would not calculate the position of one gear from the model of the other, but the mechanism would involve setting a movement ratio between the two gears, such as 2:1. This would mean that turning one gear instance through an angle α would produce an automatic rotation of $\alpha/2$ of the other gear instance. This is calculated directly from the rotation ratio.

Generally, kinematic simulations are produced by posing the assembly in a series of static positions which are replayed sufficiently quickly so that the mechanism appears to be in constant movement. In a previous version of I-DEAS this was done by recording a set of transformation definitions which were plugged back into the assembly.

For each pose, some kinematic packages perform analyses, with Boolean operations, to see if components collide during operations. As Csabai [3] pointed out, since the analysis is done only on static poses then there is a risk that some collisions may be missed if the movement is simulated with large steps between poses. Alternatively, if the steps are too small then the simulation takes a lot of time. Nor is this problem solved by using a swept volume technique, which generates the volume of space through which an object moves. A simple Boolean operation would give the common volume, but the time element, when each object is there, is not given. However, the visual aspects of simulation can be an important aid for the user.

Kinematic connections between elements provides a similar framework to that imposed by constraints, but is different because the connections are functional. Some of these were mentioned briefly in Sect. 8.4. Examples are:

- Rigid connections
- Rotational connections
- Sliding connections
- Universal connections
- Screw threads
- Gear wheels
- Springs
- Cams and followers

Generally, the mechanisms are set up and then one, perhaps more, entity is used as a driving entity to animate the others. For example, in the assembly shown in Fig. 1.49, there is a handle. In a simulation this might be defined as rotating at constant speed while the positions of other instances in the assembly are calculated from this. A few of the possible kinematic elements are shown in Fig. 13.21.

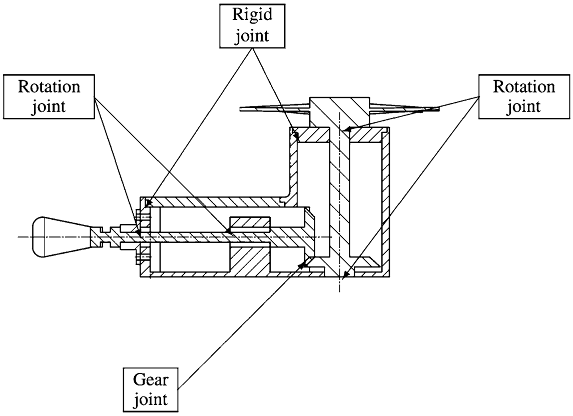


Fig. 13.21 Kinematic connections for simple winder assembly

13.5.1 Rigid Connections

Rigid connections mean that one instance is moved with the other. The lid and the box in the assembly in Fig. 13.4 would have a rigid connection. Rigid connections can also be created between elements which move together. For example, if a shaft is made of two parts that are bolted together then one shaft part would be connected rigidly to the other (Fig. 13.22).

13.5.2 Rotational Connections

Rotational connections might contain an axis definition and a positioning contact surface on each of the two objects being animated, for example. An example is shown in Fig. 13.23.

The user would identify the components, that is, the axis of one rotation element and the base face to provide one coordinate system. The matching elements of the other components provide a second coordinate system for calculation. The two *X*-axes allow calculation of angular limits, for example when the rotation is blocked as in Fig. 13.24.

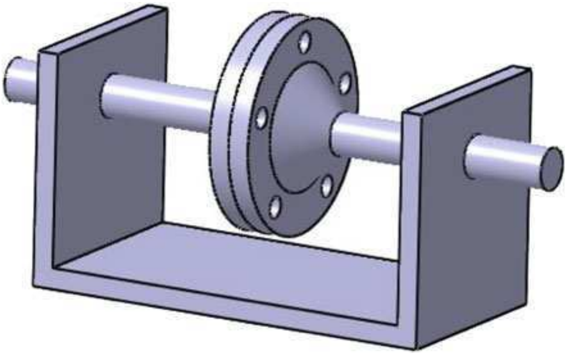


Fig. 13.22 Axle with two rigidly connected components

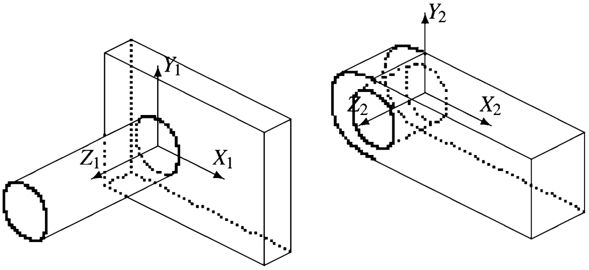


Fig. 13.23 Rotation kinematic connection elements

13.5.3 Sliding Connections

Sliding connections need one or two contact surfaces. There are two cases shown in Fig. 13.25, on the left where the two objects have one contact surface and on the right where there are two contact surfaces.

Again, note that the idea of simple kinematic simulation is not necessarily to check for collisions but to calculate positions quickly enough to make a reasonable visual simulation. Dynamic collision detection algorithms do exist, though, so check the CAD system to see if it offers this possibility.

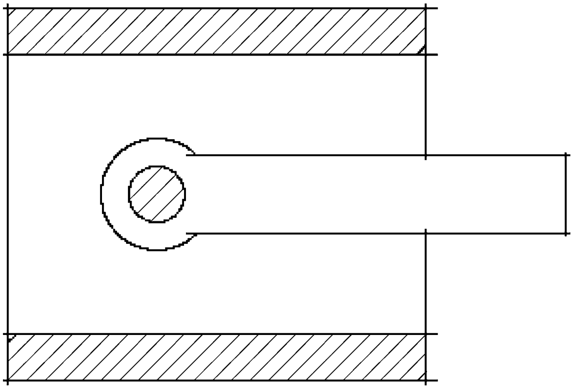


Fig. 13.24 Rotation connection with limits

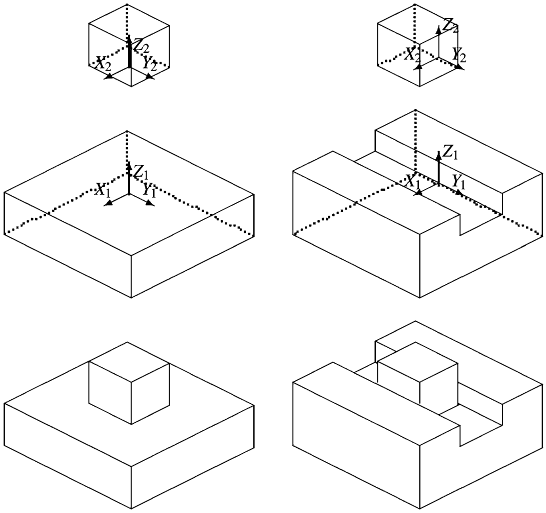


Fig. 13.25 Sliding kinematic connection elements

13.5.4 Universal Connections

Universal connections are spherical connections with a common point aligning the centres of positive and negative spherical surfaces.

As with the rotational joint, the aligned coordinate systems allow setting of limits for such a joint. Figure 13.26 shows a simple example of such a joint.

13.5.5 Screw Threads

A screw thread has a functionality as well as a shape. The shape modifier describing a screw thread is useful for manufacture and identification of a particular feature. The functional characteristics are useful for simulation. As one threaded element turns against another there is a lateral movement of the objects relative to each other. Calculating this movement is much easier if the thread sizes are given explicitly than if the information has to be recalculated from the geometry.

As shown in Fig. 13.27 the basic elements needed are the same as for a rotational connection, but the angle measured by movement around the *Z*-axes needs to be accumulated to calculate the translation along those axes.

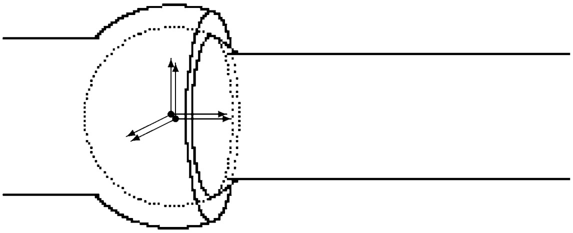


Fig. 13.26 Universal kinematic connection elements

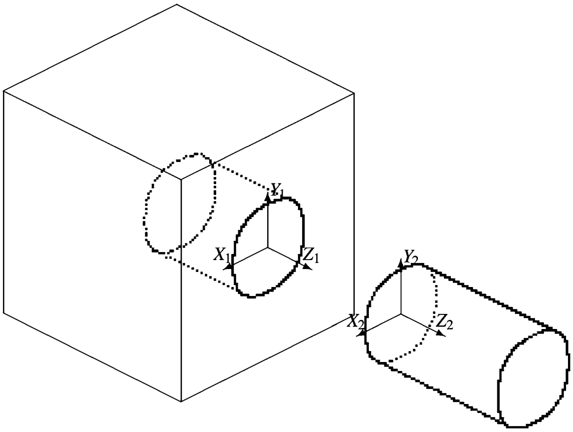


Fig. 13.27 Screw kinematic connection elements

13.5.6 Gear Wheels

As for threaded objects, a pair of gear wheels move with a certain specific relationship to each other. In a simple case, if one gear wheel has 24 teeth and another has 48 teeth, then one turns twice as fast as the other. One of the gear wheels can be considered as an input, or driving wheel, the other the output. If the smaller wheel is the driving wheel then the output is half the rotational speed of the input. If the larger wheel is the driving wheel then the output is twice the input. The user has to establish the relationship that there are two interacting gear wheels and has to specify the ratio, the simulation then calculates output movement from the input movement regardless of the real geometry. This is quick and easy to provide a realistic simulation. If the simulation were done by analysing contact surfaces and forces then it would take too long to be useful for a designer.

Figure 13.28 shows some examples of different gears, but without the full geometry. On the top left there are two gears in the same plane. On the top right there are gears which are perpendicular to each other. At the bottom of the figure is an example of a worm gear. All these have a functional connection so that rotating one instance causes the connected instance to be rotated. For the examples on the top line of the figure, the output rotation is in the opposite direction to the input direction. At the bottom, only rotating the worm gear will cause movement, trying to rotate the output gear should result in a blocked motion.

Note that using a functional transmission in this way means that the gears do not have to be full models because the motion is not derived directly from the model but from input parameters. This is easier and more practical for the implementer but does not really detract from the functionality for the user. It also means that the user can choose whether to represent a gear by an exact model or by a simplified model with a note attached to say that the model represents a gear. This was mentioned in Chap. 8. If a gear is a standard part that is to be ordered then it may make little sense to model them exactly unless the user wants an advanced simulation.

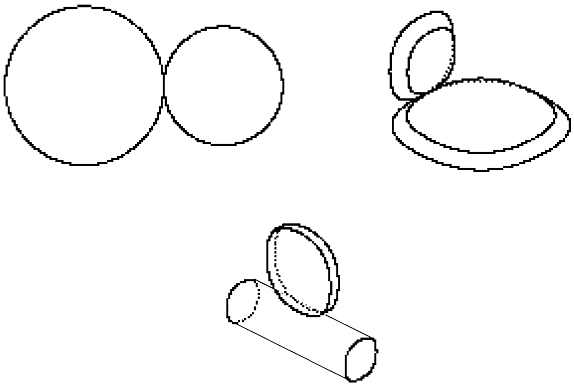


Fig. 13.28 Examples of gear kinematic connection elements

13.5.7 Springs

A spring is a complicated geometric element which is more important for its function than for its precise shape. As stated before, one of the assumptions for CAD modelling to work is that the model is rigid. A spring is not a rigid shape, it is a functional, elastic one. Therefore, springs do not fit in well with standard modelling techniques. It may be satisfying to model the spring as an extrusion along a spiral, or a loft between suitable profiles, but the model will stay in one position. It is technically possible to have a dynamic model using, say, facetted models and a functional warping so that, graphically, such a model appears to move. However, it is not certain that the result is worth the effort required to do this. A spring is another example of a functional connection between two instances in an assembly.

13.5.8 Cams and Followers

A cam provides a non-uniform profile which can be used to change the angle of a "follower". Figure 13.29 shows a very simple example of a cam and a simple bar follower. The top shows the initial state, where the radius of the cam is small. As the cam rotates, middle figure, so the larger distance starts to raise the bar. Eventually, the cam reaches a point where the distance between the centre of

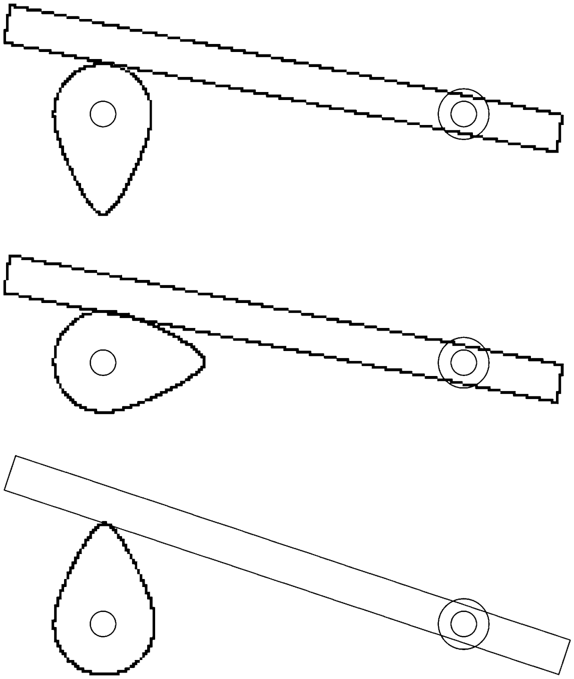


Fig. 13.29 Examples of cam kinematic connection elements

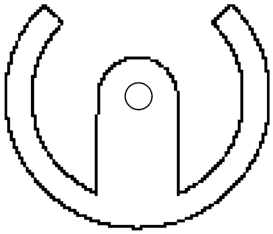


Fig. 13.30 Example of a bad 2D cam

rotation and the cam profile is a maximum and the bar achieves its maximum angle to the horizontal. Attaching further elements to the follower means that various levers can be pulled or pushed in varying amounts. This is a very simple example but there are many other examples of cams with varying shape.

A cam and a follower can be simulated using two parametric profile curves. Think of the profile as a single parametric curve in t where $t = 0$ represents a rotation of 0° and $t = 1$ represents a rotation of 360°. Evaluating the curve at any intermediate point and rotating the point through the appropriate number of degrees gives the contact point between the cam and the follower.

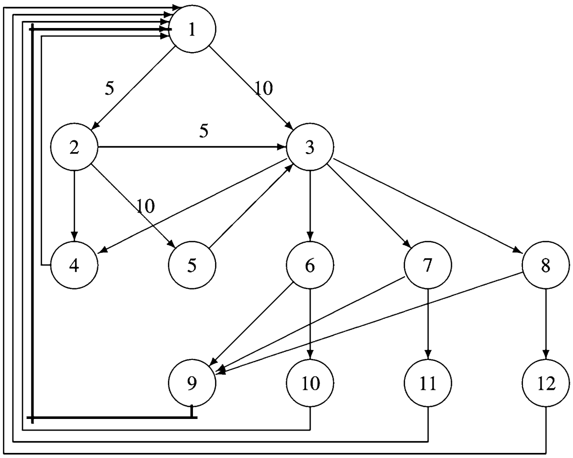
This is another example of a simple technique that can give incorrect results. If you consider the cam example in Fig. 13.30 then you should not expect the method to work correctly. This may, or may not be checked for by the system. For quick visual evaluation of a mechanism, though, the method described above is reasonable.

13.6 Mechatronics Simulation

Mechatronics was mentioned in Sect. 8.5. Simulation of mechatronics is technically feasible but is unlikely to exist in the CAD system. Research is going on in this area so it may come during the next few years. There are various elements that could be included in the simulation, such as display panels, which could be included fairly easily from a technical point of view. It should be possible to simulate the display in the same way as materials colourings are imposed on an image. However, to make a complete simulation work, it would be necessary to find a standardised way of communicating functional behaviour and connecting this with external elements. This is an area of interest for research but it is a little early to come to any definite conclusions. Activation of different elements in the simulation model, simulating switch activation, needs to be allowed.

One method which may be useful for simulation is the finite state machine method developed by Turing. Apparently, Turing thought of it as a mechanism to explain the functioning of DNA. What it means, briefly, is that there are a number of inputs to a "machine" and, with each input, the machine changes state. In any state the machine can perform an action and change state. Figure 13.31 shows a simple example of a state transition diagram for a simple vending machine.

Fig. 13.31 Example of a state transition for a simple vending machine



The vending machine can take tokens of five or ten units and is to deliver three products, each of which costs ten units and which can be selected by push buttons. The states are as follows:

1. Initial state: In the initial state, pushing any of the product selection buttons or the cancel button has no effect. A display should be illuminated to show that the machine is ready to accept coins. If a five unit token is inserted then the state changes to state 2. If a ten unit token is inserted then the state changes to state 3.
2. Part paid state: The display should show that five units have been entered. Pushing any of the product select buttons will have no effect. Pushing the cancel button changes the machine to state 4. Inserting five units will change the machine to state 3. Inserting ten units will change the machine to state 5.
3. Full paid state: The coin input is blocked. The display should inform the customer that product selection can take place. Pushing the cancel button changes the machine to state 4. Pushing the product 1 button changes the machine to state 6. Pushing the product 2 button changes the machine to state 7. Pushing the product 3 button changes the machine to state 8.
4. Cancel state: The machine should release all tokens accumulated so far and change back to state 1.
5. Overpaid state: The only combination that can cause this is if a five unit token and then a ten unit token have been inserted. The machine should release the five unit token and change the machine to state 3.
6. Product 1 state: The machine is ready to deliver product 1. If there are no products left then change to state 9, otherwise change to state 10.
7. Product 2 state: The machine is ready to deliver product 2. If there are no products left then change to state 9, otherwise change to state 11.
8. Product 3 state: The machine is ready to deliver product 3. If there are no products left then change to state 9, otherwise change to state 12.

9. Product exhausted state: Display a message apologising for the inconvenience and release the tokens paid before changing to state 1.
10. Product 1 delivery state: Release one unit of product 1. Release the tokens paid to the token collection unit. Change the machine to state 1.
11. Product 2 delivery state: Release one unit of product 2. Release the tokens paid to the token collection unit. Change the machine to state 1.
12. Product 3 delivery state: Release one unit of product 3. Release the tokens paid to the token collection unit. Change the machine to state 1.

A simulation system would need to have a library of inputs, such as coins, tokens, buttons, switches, keyboards, etc. The user would have to identify the states and what happens. The system should try and identify missing states, such as when possible inputs have been neglected. The user would have to identify display messages and outputs. For simple cases this might work.

Another method would be to use macro-programming techniques to allow the user to program behaviour. The macro programs would have to be associated with components in the product as non-geometric information. This sort of technique would require programming knowledge by the designer. Similar comments for the inputs and outputs also apply. An example of a macro-code program for the same problem might be:

```
WHILE (NOT global stop)
{
    total = 0; display “ready”;
    WHILE (total < 10)
     {
         if ( inserted(5) ) { display “5”; total = total + 5; }
         else if ( inserted(10) ) total = total + 10;
         if ( total > 10 ) { release 5; total = total - 5; }
         else if ( total = 10 ) { block token slot; display “select product”; }
         if ( cancel ) { display “cancel”; release all; total = 0; display “ready”; }
     }
    WHILE (total = 10)
     {
         if ( cancel ) { display “cancel”; release all; total = 0; display “ready”; }
         else if ( product1 button )
             { if ( product1 OK )
                 { collect tokens; release product1; total = 0; display “ready”; }
             else { release tokens; display “sorry”; }
         else if ( product2 button )
             { if ( product2 OK )
                 { collect tokens; release product2; total = 0; display “ready”; }
             else { release tokens; display “sorry”; }
         else if ( product3 button )
             { if ( product3 OK )
                 { collect tokens; release product3; total = 0; display “ready”; }
             else { release tokens; display “sorry”; }
     }
}
```

Although it is a little early to make definite statements it would be useful to monitor this possibility for the future.

13.7 Bills of Materials

A Bill of Materials (BOM) is a list of the entities in a product together with the number of times that they occur. This can be done by traversing the assembly structure and printing the object name together with the number of times the object is instanced in the assembly. If the assembly structure has been constructed correctly then this is enough. If, however, the assembly structure has copied elements then the BOM will contain multiple entries for parts which may have the same name.

The assembly structure at the top of Fig. 13.32 would give rise to a Bill Of Materials such as:

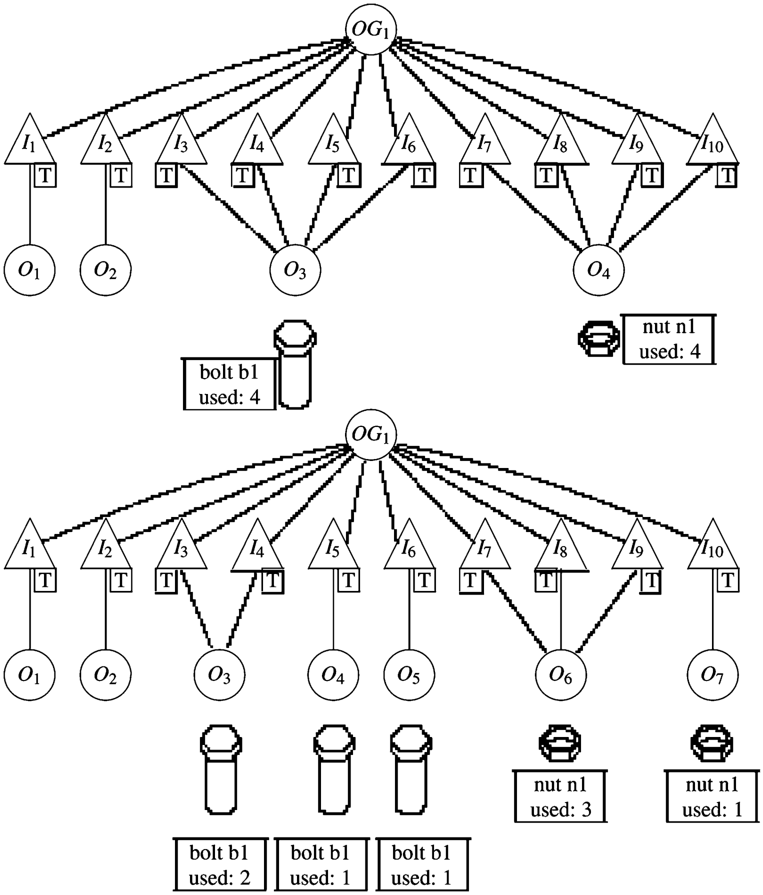


Fig. 13.32 Assembly datastructure for the standard elements assembly

Part	Number
Box	1
Lid	1
Bolt b1	4
Nut n1	4

The structure at the bottom of Fig. 13.32 would give rise to a Bill Of Materials with repeated elements such as:

Part	Number
Box	1
Lid	1
Bolt b1	2
Bolt b1	1
Bolt b1	1
Nut n1	3
Nut n1	1

It should be stressed, again, that the mechanism for creating a correct bill of materials relies on having a correct assembly structure. Try producing a bill of materials for an assembly and check for multiple occurrences of the same item. If you do have these, then it is likely that either your assembly structure is wrong or that your naming is wrong. If the system has an internal numbering then it may be possible for it to check that an item has been used several times. This is not a stable possibility as not all systems do this. Also, imported assemblies will not have this, with present importation techniques, so you cannot rely on the CAD system to cover up your mistakes.

13.8 Chapter Summary

This chapter deals with models with multiple parts, so-called assemblies. Assemblies are relatively simple, from a datastructuring point of view, but getting the arrangement of entities right is more complex. The chapter describes how static constraints work. The chapter then goes on to describe kinematic constraints. A brief mention is made of the possibility of simulating mechatronics components. Finally, the chapter describes how a Bill Of Materials (BOM) is created.

13.9 Assembly Exercises

13.9.1 Align a Cylindrical Peg in a Cylindrical Hole

The peg has an axis which passes through the point (5, 0, 10) and is in the direction (0.577, 0.577, 0.577). The hole has an axis through the origin (0, 0, 0) in the direction (0, 0, 1). Calculate the transformation matrix to align the peg with the hole.

Say in what way the transformation matrix differs in the three following cases:

1. Peg diameter 10, hole diameter 10.
2. Peg diameter 5, hole diameter 10.
3. Peg diameter 10, hole diameter 5.

13.9.2 Align a Square Peg in a Cylindrical Hole

For those unfamiliar with the expression, the term “a square peg in a round hole” is usually taken to mean a misfit, or someone or something which is unsuited to a particular purpose. Apparently, though, in old wooden buildings, square pegs in round holes were an old-fashioned type of push-fit where the square peg deformed the hole, or had some material sheared off, to provide a tight, rigid joint. Here, this type of fit is used to explain how supplementary geometry can be used for assembly.

Create two objects, one with a round hole and one with a square extrusion, as shown in Fig. 13.33. Fix (or ground) the block with the hole in it. Now create a free-standing line at the centre of the square extrusion. Align this with the centre line of the cylindrical hole and align the planes, top face of the block and bottom face of the peg to complete the constraint set.

Two things to notice. First, there is still one degree of freedom left for the peg. It can rotate around the hole centreline even though, in reality, it wouldn’t. The other thing to notice is that the peg is larger than the hole, but, as in the previous exercise, the CAD system won’t normally tell you. For all push-fit parts in an assembly, the real piece will deform and remain in place in the real assembly, but the model is treated as rigid by the CAD system and so may overlap the model of the target piece. if you want to have a model of the deformed piece then this will have to be created separately.

13.9.3 Assembly Symmetry

As an illustration, it is possible to perform the following experiment if your CAD system does allow symmetry transformations. The experiment involves creating a simple assembly of an L-block and bolts as in Fig. 13.34.

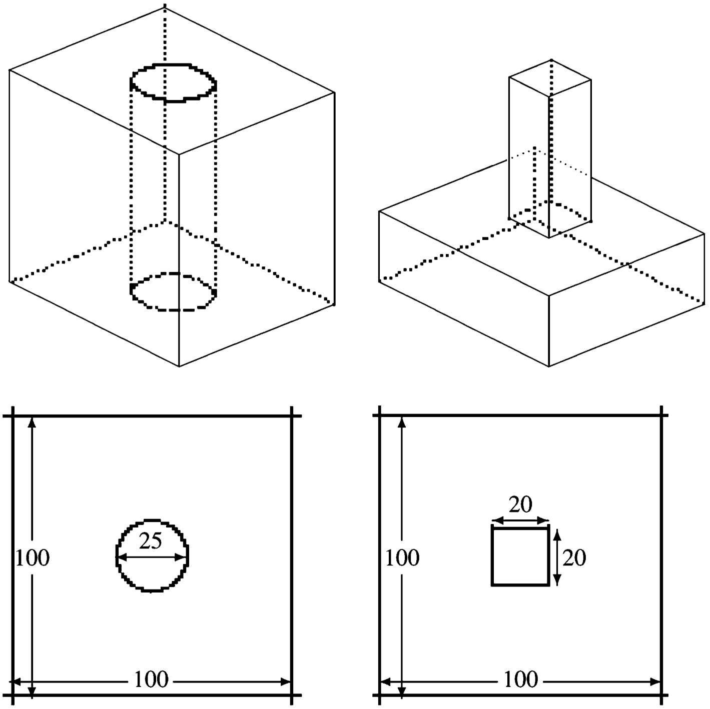
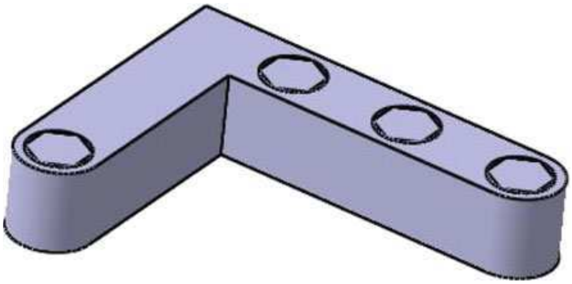


Fig. 13.33 Parts for "square peg in round hole" test

Fig. 13.34 L-block assembly for reflection



Make an L-shape with rounded ends (just to make it pretty), as shown in Fig. 13.35. Extrude the shape 20 units. Make a pattern of three holes (radius 5), with counterbore, radius 7.5, depth 5 in the vertical arm, one in the horizontal arm. Thread the holes.

Make a hexagonal shape, radius 7.5. Extrude this five units. Create a circular profile, radius 5, on the top face and extrude this a distance of 15 units. Thread the circular extrusion.

This should give you two objects like those shown in Fig. 13.36.

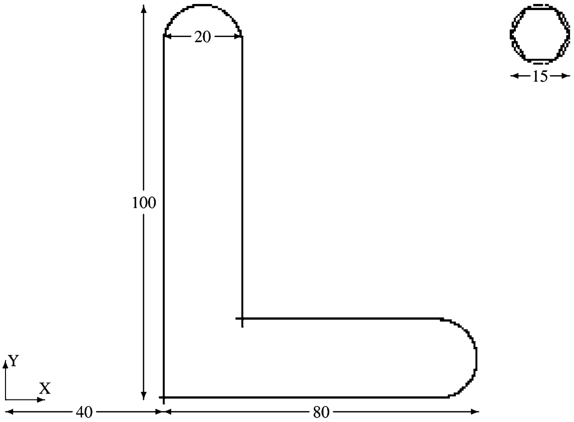


Fig. 13.35 Objects for L-block experiment

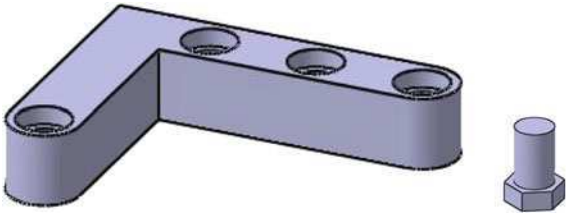


Fig. 13.36 L-block and bolt for assembly

Make an assembly with four instances of the bolt and the L-block to give the result shown in Fig. 13.34. Now create an assembly and insert the first assembly. Reflect the instance of the first assembly to give the result shown in Fig. 13.37.

Examine the result and determine which of the two structures shown in Figs. 13.38 and 13.39 is the result. In Fig. 13.38 the transformation of is a reflection transformation. In Fig. 13.39 the whole group of objects is copied and transformed.

If the CAD system just creates a new reference then the object shapes are not created explicitly and so there is no model for other applications. However, this is the most efficient for modelling large assemblies.

If the CAD system copies the structures, as in Fig. 13.39, then new models are created. One question, though, is how many models there should be. In reality, you can see that the bolt model should not be copied but translated. The bolt model stays the same shape after the symmetry operation. Also, what about the threaded holes? Have they changed in sense so that a right-handed thread has become a left-handed thread?

Yet another question concerns whether part shapes can be modified to avoid a symmetry operation. For the L-shaped piece in the example, it may be possible to give the L-shape a counterbore on both sides of the object so it can be turned over and used. True, it would mean an extra manufacturing operation, which may not be desirable, but would reduce the number of separate parts in a company.

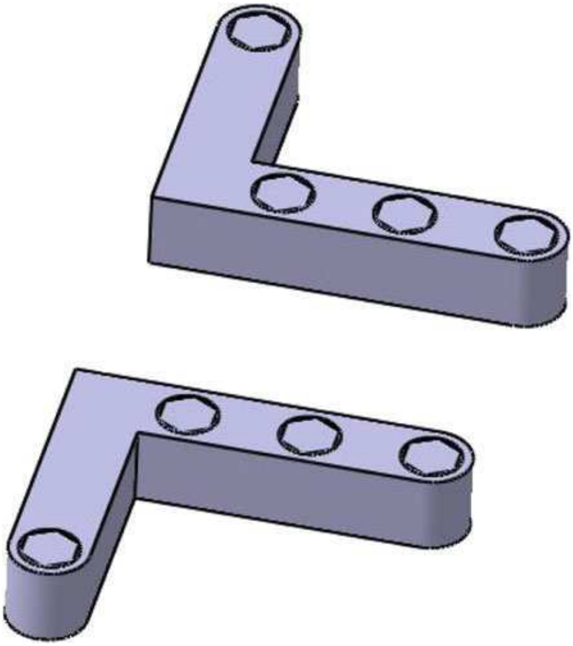


Fig. 13.37 Symmetric assembly

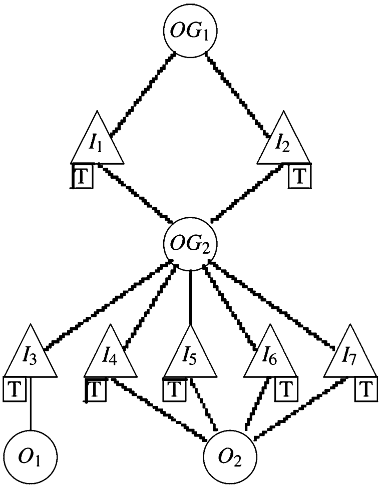


Fig. 13.38 Data structure 1 for symmetrical assembly

13.9.4 Explosion Test

Create an assembly with one cube, 100 × 100 × 100, and four cylinders, radius 20, height 80. Fix the cube and then explode the assembly. What does the CAD system tell you about fixed part? Does it leave the cube in place and move the other parts? Does it move the cube but move it back if you apply the constraints?

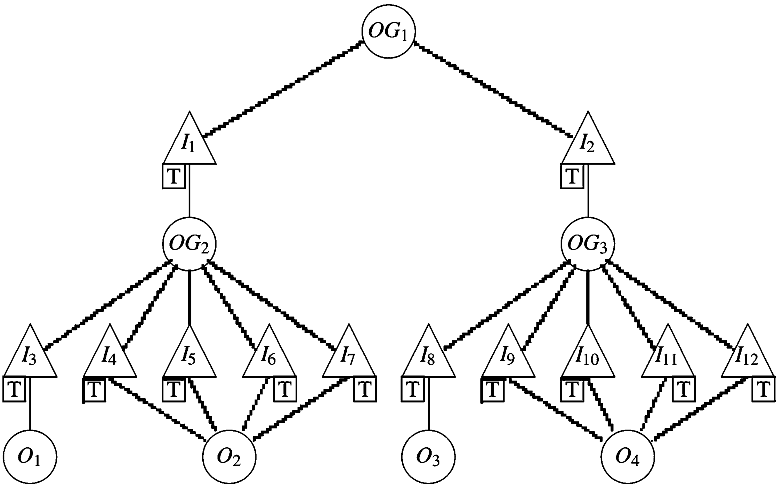
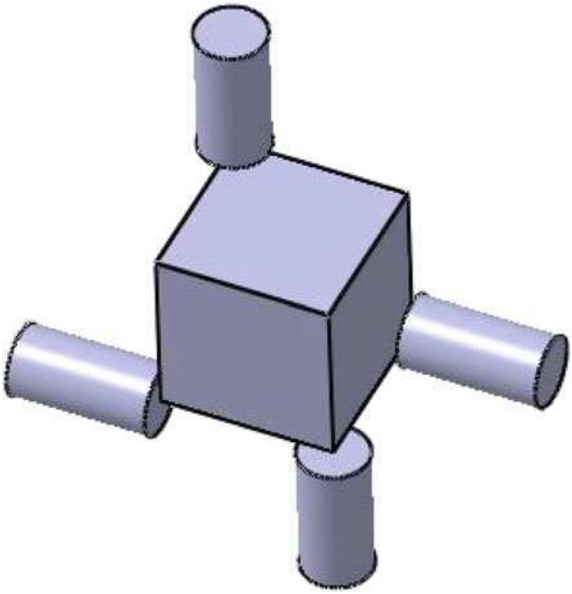


Fig. 13.39 Data structure 2 for symmetrical assembly

Fig. 13.40 Cube with constrained cylinders



In CATIA there is an “update” icon to reestablish the constraints. Other CAD systems have similar icons, find out which one your CAD system has.

Now establish a set of constraints between the cube and the cylinders. Align one end of the cylinder with a face of the cube. Then, align the cylinder axis with an edge of the cube, such as in Fig. 13.40. Re-explode the assembly and check that the system can reestablish the constraints after the explosion.

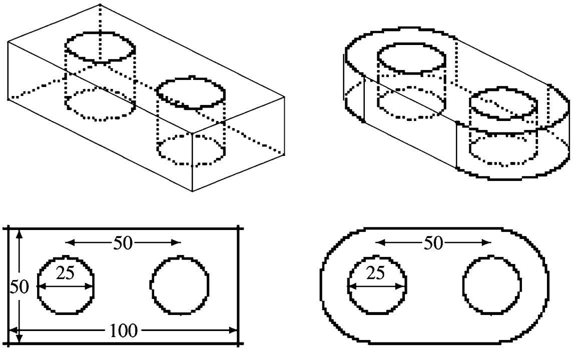


Fig. 13.41 Parts for over-constrained test

13.9.5 Over-Constraint Exercise

Create two pieces with holes at a known distance, such as those shown in Fig. 13.41. The exact dimensions are not so important so long as the hole centres line up. The holes can actually have different diameters just so long as the centre lines match. In the example below, the rectangular block is 100 × 50, extruded 25 units. The two holes have diameter 25 and are 50 units apart. In the other part, the basic part is a 50 × 50 square with two rounded ends and holes the same diameter and distance apart.

Put the two pieces into an assembly and set constraints so that the top plane of the rectangular block and the bottom plane of the rounded piece are coincident. Now align one pair of holes, one in the rectangular block and one in the rounded part. Then, try and align the other two holes. Does your CAD system let you do this?

I was quite pleased to see that I could align both pairs of holes in CATIA v5. This is a natural constraint set to use, but the problem is that the constraint set is over constrained. Another system, I-DEAS, used to refuse this type of constraint set because over the over-constrained nature of the set. This is technically correct, but was difficult for users. In the I-DEAS implementation the constraints could be set up as an alignment of the first two holes and then a constraint of parallelism between the vectors between the hole centres. A personal opinion is that the CATIA implementation is more natural for the user, but what happens if you change the distance between the hole centres? Go back and set the distance between the hole centres in the rectangular block to be 30 units. What does your system do? If it allowed the initial constraints to be set up then there is now a problem because both constraints cannot be fulfilled. This is where the technical correctness of the old I-DEAS implementation is better, because the objects are rigid and the change in distance between the holes implies that one object should be stretched. It is another example of where technical correctness and user-friendliness diverge.

Incidentally, this is another strength of Csabai's idea about features spanning two objects, described in Chap. 11. If features are to match, as in this case, it is far better to set them up so that the features are controlled by one set of parameters, distance, radius or whatever, and produced in both objects in one step.

References

1. Newman, W.M., Sproull, R.F.: Principles of Interactive Computer Graphics. McGraw-Hill, New York (1979)
2. Stroud, I.A.: Boundary Representation Modelling Techniques. Springer, Heidelberg (2006)
3. Csabai, A.: Layout modelling and evaluation methods and tools. Ph.D. Dissertation, EPFL, STI-IPR-LICP, Switzerland (2003)

Chapter 14
CAD in a Community

It is to be expected that design and manufacture take place in a community of applications and experts. This is because of the complexity of the different areas of production. It is difficult to be an expert in all of them and hence much design is done as part of a team. Some aspects of this have been mentioned in Chap. 11 for the design phase, but other application areas also need to be considered. In this respect it should be noted that CAD is not just about creating a correct model but also about communication. The communication aspects are very important. New areas of information and new tools are changing the methods for communication and it is important to understand these for efficient CAD use.

14.1 The Product Lifecycle

Traditionally, companies have manufactured products and then the buyer takes over responsibility with little or no information back to the company. This has changed with the modern view of recycling and is changing even more, with even the possibility to monitor and store component data with the product itself.

The product lifecycle is generally divided up into sections: Beginning-Of-Life (BOL), Middle-Of-Life (MOL) and End-Of-Life (EOL). The traditional notion of product development has concerned a simple CAD-CAPP-CAM-CNC chain at the beginning of life. This needs to be updated, though, to fit new theory and practice. The Middle-Of-Life, or "Use phase" is where data about component behaviour can be recorded and stored. At the End-Of-Life this data can be collected to see if a component can be reused. Data about the product at the end of life is also important to help designers make improvements.

14.1.1 The Old CAD-CAPP-CAM-CNC Chain

There is, at the time of writing, a sort of tradition of how information is passed from design to manufacturing which seems to be the traditional, paper-based

I. Stroud and H. Nagy, *Solid Modelling and CAD Systems*,
DOI: 10.1007/978-0-85729-259-9_14, © Springer-Verlag London Limited 2011

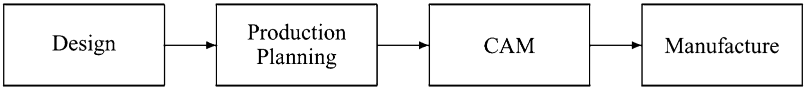


Fig. 14.1 Traditional notion of the production sequence

method, using an electronic medium. The CAD-CAPP-CAM-CNC chain has evolved over the years without having been planned, and is shown in Fig. 14.1. Although new tools are available the chain still bears the hallmarks of the old paper-based technological sequence.

Each section on the chain received information decided on in a previous section with limited feedback. For example, the production planner received 2D drawings of a part with annotations about tolerances and so on. From this, the production planner had to decide on the manufacturing method and details from which to prepare the toolpaths to send to the shopfloor. The designer had to produce drawings containing information in an indirect form using tolerances, for example. The idea was that the designer knew how the part fitted in to a product and so decided the tolerances necessary so that the part would fit in. Contact surfaces needed a finer finish than surfaces which did not touch any other part, and so on. Also, the main information communicated was shape based for manufacturing and assembly.

CAD systems produce three-dimensional objects, from which two-dimensional projections are computer, and these three-dimensional objects could be communicated and examined. This presents a problem in how to communicate the tolerance and other information. Certainly it is possible to annotate the 3D models, but this has to be done so that the information is not conflicting or just plain wrong, as explained in Chap. 8.

Another aspect is that modern data exchange methods, such as STEP, are exposing weaknesses in the traditional information flow which need to be handled by researchers and software developers. STEP has pulled together the traditional “islands” of CAD, CAPP, CAM and CNC, creating advanced possibilities but also showing the holes in the information set. New information types and sources have come into being, but there is other information not yet covered. As explained above, parts and products can be monitored during their functional lives, creating new types of information. There is also an increased interest in recovering components after the product has been discarded and reusing these. This reduces the environmental impact of production as well as reducing the waste going to landfill. Reuse and recycling, though, need their own types of information.

It is clear that there is more information available than can be communicated simply by drawings and that the amount of information makes it inefficient to use paper as a means of communication. In order to understand what information is needed, though, it is necessary to understand how different areas should fit together, not just how they have been fitted together.

14.1.2 Modern Life-Cycle View

A more modern notion is to look at the whole lifecycle and Fig. 14.2 shows a more complete cycle. Um et al. [1] give a fuller view of the cycle with information needed to be fed back at the end of life.

The cycle is not a pure cycle in that there are several "short circuits" where information may be fed back to the first parts of the lifecycle from any of the later stages without having to complete the cycle. What this means for design, though, is that you both create and receive information. The information you create is partly the shape and partly a reasoning process. The information you receive is partly a set of requirements and partly information about the performance of past products.

Data exchange standards have only gone so far in addressing the complete chain. This is, of course, natural, because the whole production environment is complex. The tendency has been to address the known parts and the information which is known to be necessary for communication. However, there is also much else that should be recorded and communicated. The lack of a full range of tools hinders development. The lack of a complete research environment covering all aspects of production hinders the development of tools. A method of creating federated research groupings to put together many universities is a promising avenue of development to overcome this. Without a method for researching the complete production and use environment we will be stuck with engineering inefficiency. Without this method, research will lead to local improvements, which may be good in themselves, but we risk missing globally effective solutions. This section is about some of the aspects of this that are evident now.

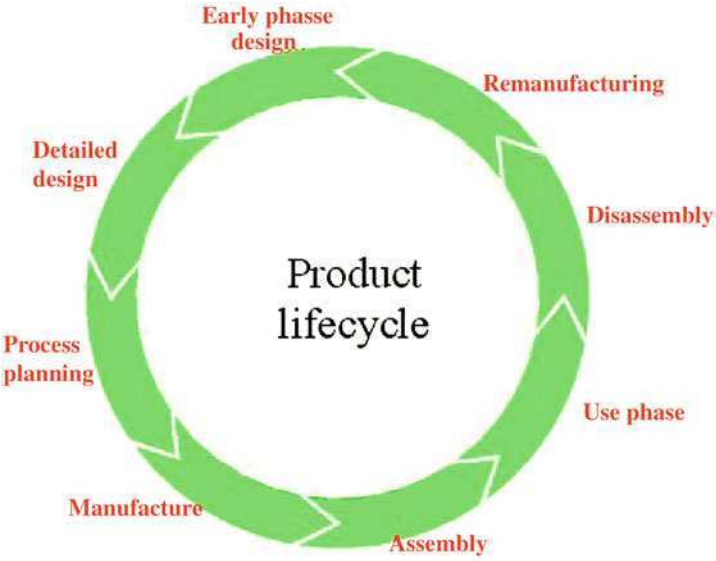


Fig. 14.2 More complete view of products and production

It is possible to give a simplified analysis of some of the elements as an illustration of how CAD may change during the next few years.

14.1.3 Early Phase of Design

In the early phase of design, when generating concepts, the principle information is the product idea (Fig. 14.3). However, extra information can be supplied about known solutions and their performances. This was done by Sprumont et al. with the MicroCE project, as described in Chap. 11. Sprumont's design catalogues with their evaluation mechanism provide a convenient way of introducing such information. The elements of the chosen concept need to be communicated for the layout design. The product structure and product information are also generated.

Concept research involves finding product solutions. The concept solutions involve functions and maybe partial solutions for functions.

14.1.4 Layout Design

In layout design, the product elements from the concept research are supplied to identify the elements in the layout. Additional constraints may be identified as well as known components which can be imported directly. The output is expected to

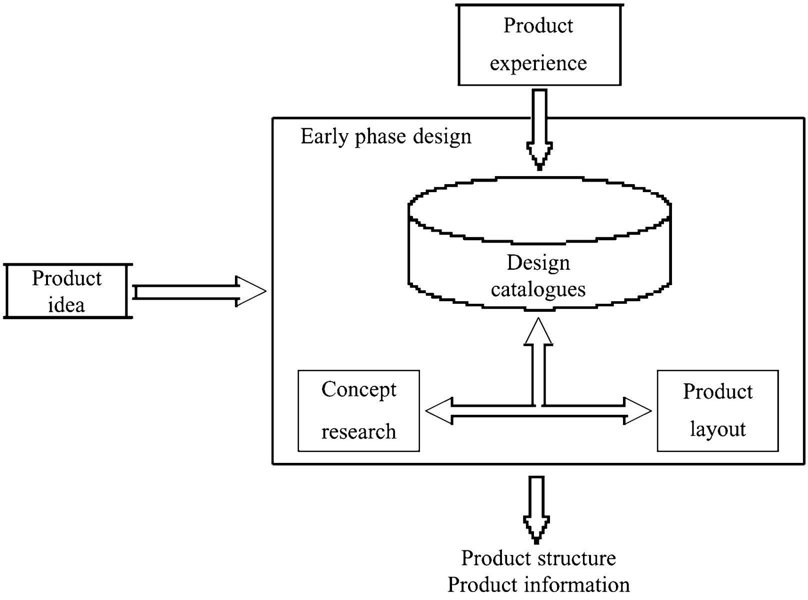


Fig. 14.3 Information for the early phase of design

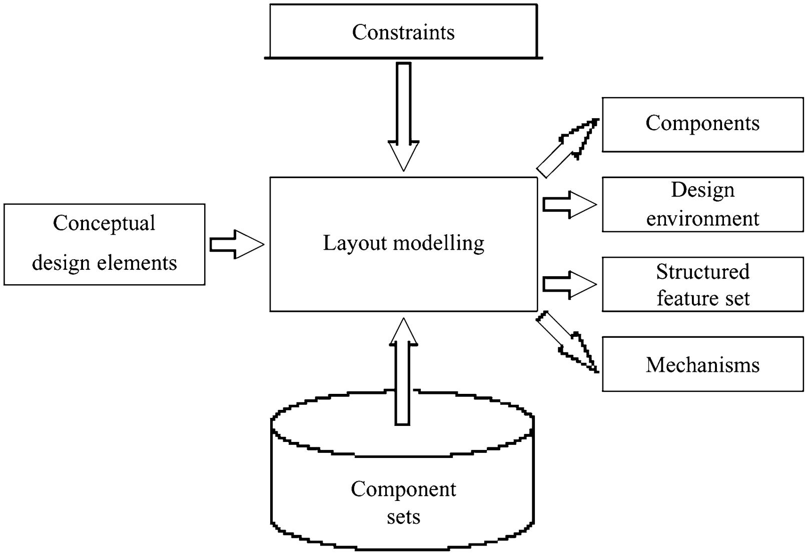


Fig. 14.4 Information for the layout phase of design

be a set of components, a set of design environments (described in Sect. 11.3), connection features in a structured feature set and mechanisms Fig. 14.4.

Layout modelling gives the first geometric elements to a project. This provides an important framework for the detailed design process. The simple geometric elements provide design space limits within which detailed design can take place. According to Csabai's philosophy [2] some conflict areas may also have been identified. The connections provide concrete geometric features with a reason for their presence. This is one element in the establishment of a structured feature set for manufacturing. It is also important for setting manufacturing tolerances. The connection features provide a bridge between objects for establishing consistency during the detailing. The kinematic simulation provides understanding about the functionality of parts as well as early identification of mechanisms in the product. The structure of the layout can be imported into the CAD system or left separately as a product structure.

14.1.5 Detailed Design Phase

The detailed design phase is what is currently called CAD. Note that, here, this includes the analysis phase for strength and weight calculations, for example. The designer using the CAD system should have information support from the earlier phases, not just in documentation form but as information in the CAD system itself. The product structure can come from the layout module as well as design environments from the different parts. The result of the design and analysis process

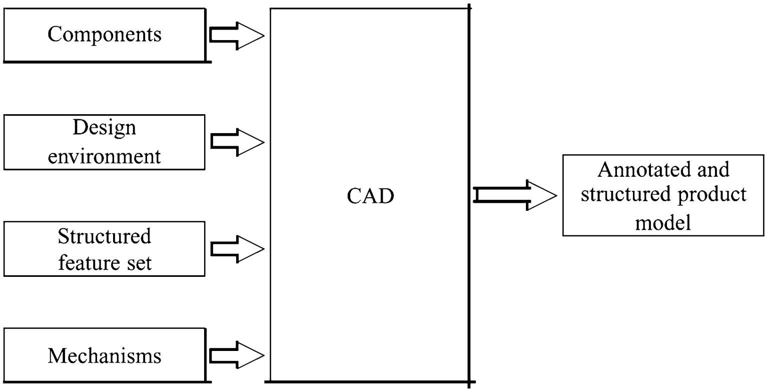


Fig. 14.5 Information in the detailed design phase

should be an annotated and structured product model. Normally, a product model is a flat structure of faces, edges and vertices. A *structured product model* is a model in which the different elements are categorised in order to convey meaning for manufacture and other purposes Fig. 14.5.

Current CAD is essentially a one person task with little support to find solutions. Current CAD is also *reactive* not *proactive*. "Reactive" means you design something then analyse it or examine it from different viewpoints whereas "Proactive" means the system helps you to find good solutions. At the moment the onus is on the user to get it right but the extra information is intended to help the user find good solutions and hence to move CAD towards being proactive. A proactive tool for design for assembly was described in Sect. 11.4.

An aspect mentioned earlier, in Sect. 1.7.6, is collaboration and multi-user systems. Collaboration in CAD is sometimes called "concurrent engineering" and will be dealt with in Sect. 14.2. Typically, with current CAD systems, this is somewhat asynchronous in nature because it is performed through databases and data exchange. However, collaborative CAD should mean synchronous working where you see other people's work at the same time as they do it.

14.1.6 Manufacturing Phase

Manufacturing is a complex topic involving knowledge about many domains. There are many different ways to make parts and the cost of manufacture can vary greatly with these different methods. Improved information helps to rationalise decision making in manufacturing (Fig. 14.6). One aspect is improved knowledge about the shape to be made through structuring and annotation. With current methods there is no apparent difference between the shape elements and the process planner is forced to use his or her knowledge and experience to guess about elements. However much information is present, though, the planner needs

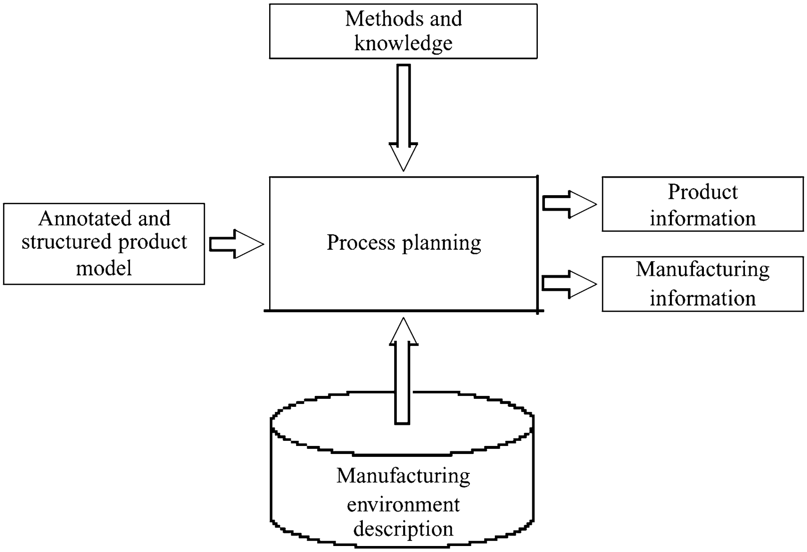

Methods and knowledge
Annotated and structured product model
Process planning
Product information
Manufacturing information
Manufacturing environment description

Fig. 14.6 Manufacturing phase

good knowledge about manufacturing methods and the manufacturing environment available.

It is possible to distinguish between elements in the model, for example with the decomposition shown in Fig. 10.35 as:

- elements that are fundamental to the design,
- elements that are functional,
- elements for practical purposes, and
- elements which are compromises.

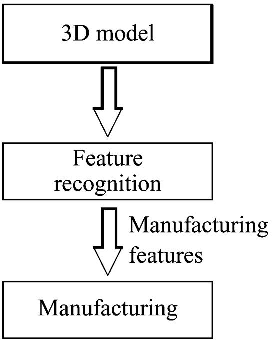
The fundamental elements are those parts of the shape which are there to fulfil the design requirements.

The functional elements are those parts, like bolt holes for fixing to other parts in an assembly, which are there to ensure the functional requirements of the part. Note that these features typically span two objects, an idea which comes from Csabai [2], as described in Sect. 11.2.4.

The practical elements are those elements which have been decided based on physical properties, like chamfers to remove sharp edges or stiffeners to ensure rigidity.

Finally, the compromise elements are things like draft angles and blends, added to make it possible to make a part by moulding, or blends added to allow pocket milling, for example.

This is not meant to be definitive or exclusive. Different ideas exist, the aim is just to illustrate the idea that there is a structure of design information that needs to be created by CAD tools, which can then be standardised and finally communicated. Improved knowledge about why different parts of the model are there helps the planner to focus on key elements.



this is wrong...

Fig. 14.7 3D process planning—current philosophy

Figure 14.7 shows a scenario presented often but which is too simplified to be widely applicable. This is, in essence, the method originally conceived by Jared and Smith for a project based on the BUILD system and the feature recogniser of Kyprianou in the early 1980s. During the project, reported by Parkinson [3], the team quickly realised that feature recognition on its own is inadequate. Process planning decisions have to be made in order to determine relevant features and orientations as well as manufacturing methods. Unfortunately, more than 20 years later, this lesson still hasn't been learned and this simplified scenario still appears.

Another method, based on Sabin's view of building back, is illustrated in Fig. 14.8. Here, there are several phases of feature recognition as planning progresses. If the design model is properly structured then some of these elements, such as the connection features, might be available directly from the model without having to recognise them, hence the term "acquisition". The connection entities require functional tolerances to make them work. These functional tolerances may be supplied directly by the designer, or inferred from a functional model, or both. In order to achieve the functional tolerances the planner has to determine the corresponding manufacturing tolerances and determine the manufacturing processes available to achieve these. Having removed the finishing features, a further feature acquisition step is needed on the intermediate part. It may not be necessary to manufacture all of the features found, hence the need for planning decisions in association with the shape recognition. Once this has been done the raw part has to be decided on and, if necessary, extra steps taken if, for example, the raw part is to be produced by casting.

This is a simplified model, but it is intended only to show why the simple "recognise-and-mill" philosophy is insufficient and why more information is needed from the designer. Manufacturing and manufacturing planning is a complex topic so this short description is not intended to do more than illustrate some of the aspects that need to be rethought in a revised production chain.

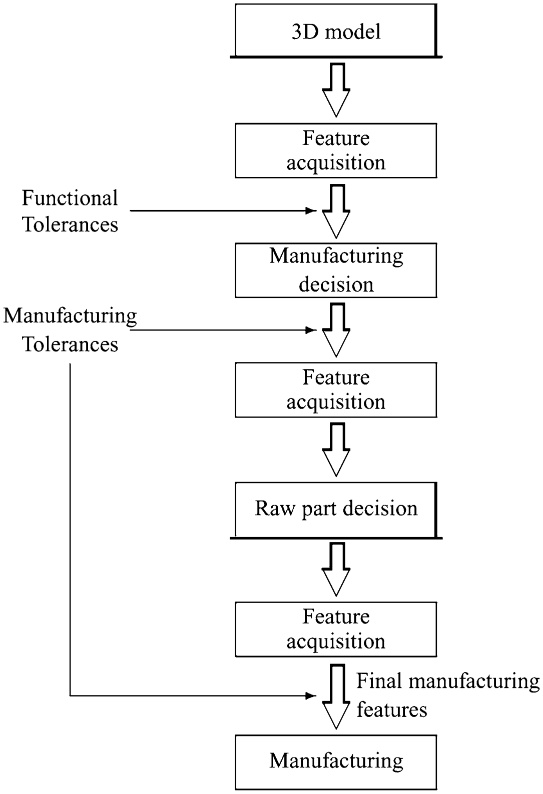


Fig. 14.8 3D process planning—different philosophy

14.1.7 Use Phase: Middle of Life

The use phase can be thought of as where the product is field tested. The relevance for design is in knowing about the performance of different elements. The advances in computer technology, the increased use of mechatronics and chips means that information can be stored with components for analysis. For example, the car industry has already embraced advanced electronics in cars. Data about components can be accessed when the car is serviced and the results stored in a company database. Figure 14.9 illustrates the information elements for the use phase.

The existence of product performance data can be exploited in design by feeding it back to the designer. This would allow designers to rationalise design in new versions of products. The history data should be a valuable company resource provided that it can be used like this.

14.1.8 End of Life Phase

The end-of-life phase is also an important source of information for design. At the end of its life, a product has a greater or smaller value depending on how much of

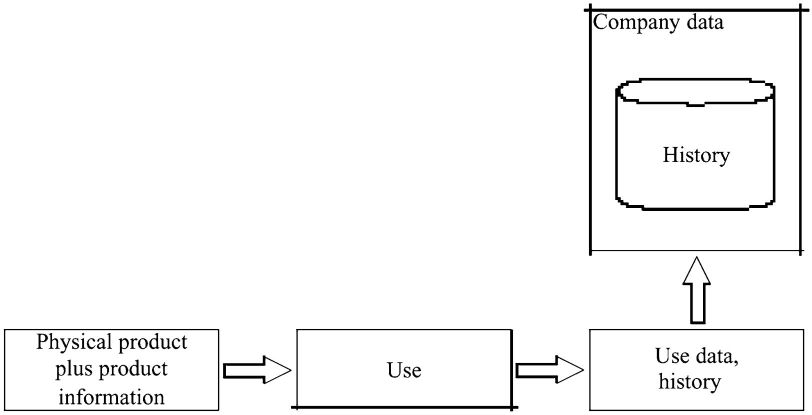


Fig. 14.9 The use phase

it can be reused or recycled. Elements which have to be disposed of in landfill have a negative value Fig. 14.10. Again, this is a complex topic. It is not intended to cover this here in any detail, merely to explain some of the aspects as an illustration of the effect on CAD.

A critical issue in recycle is material separation, the division of materials into different types. Some plastics, for example, cannot be reused if mixed with other types of plastic. Material separation affects how you connect entities together. There is a lot of information about the choice of connections in products for disassembly. If, for example, you glue parts together then it is much more difficult to separate them than if you bolt them together. Connections which need tools are harder to disassembly than connections which can be disconnected without tools. Some connections are designed to snap together and the connections broken for disassembly. There are a lot of variants. This is another illustration of the benefit of maintaining explicit connection information from the layout phase of design.

The information about the state of products at the end of their lives indicates possible design changes. The choice of materials that can be recuperated may change design decisions at the start of the project, hence the ability to reuse or recycle product parts at the end-of-life. Similarly, the presence or absence of recycled parts can affect design by, for example, increasing protection for critical parts. Finally, parts disposed of in landfill should be minimised. Even incineration of elements is a way of recovering energy and hence affects material choice.

The end-of-life phase also has an effect on production planning. The possibility of reusing components means that fewer new components of that type need be manufactured. This changes production priorities as well as introducing new pseudo-manufacturing operations like cleaning. Remanufacturing of parts is related to manufacturing from castings, which affects process planning.

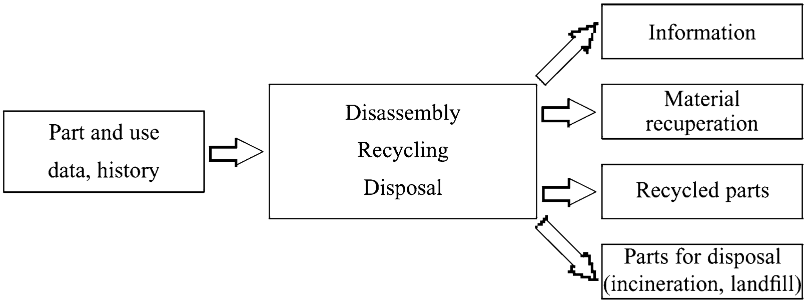


Fig. 14.10 Information for the End-Of-Life phase

14.2 Concurrent Engineering

In the chain, described above, a critical part is communication between the different areas. This assumes that the activities occur "asynchronously", one after the other. Another approach is to perform activities, at least in part, at the same time, allowing communication between the experts in different areas. This is generally termed "concurrent engineering".

The following explanation is divided up to illustrate different aspects of concurrent engineering. Firstly, it is necessary to consider who might participate in the process. Then, it is also necessary to set up an infrastructure to allow this work to proceed. Finally, it is necessary to consider the strategy needed to achieve the goals of concurrent engineering.

Two other aspects are considered in this section. The first is how synchronous CAD systems might be achieved, to allow real concurrent working. The second aspect is the replacement of human experts by expert systems for certain well-defined areas.

14.2.1 Participants

Here the meaning of concurrent engineering is when a team of experts in different fields work together to produce a design (Fig. 14.11).

What concurrent engineering aims to do is to make expertise from different viewpoints available to a designer during the design process. Note, this description concerns the detailed design phase, but similar considerations apply for the early phase of design. It is just that, with the lack of unified tools throughout the design process, this is done differently at the moment.

In Fig. 14.11 the designer is supported by expertise from five directions:

- The chief designer, who might want to check the progress and the connections with the other parts.
- The manufacturer, who might make comments about the manufacturability of a part, adding blends and so forth.

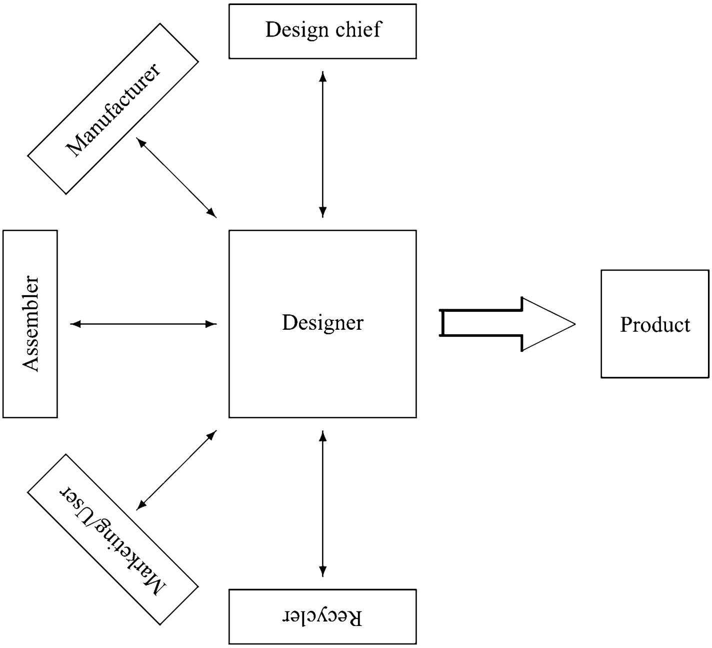


Fig. 14.11 Concurrent engineering

- The assembler, who might want changes to make assembly easier.
- The user/sales personnel, who might want to check the expected use of a product.
- The recycler, who might make comments about the reuse value of the part, the material, the attachment methods and so on.

What concurrent engineering seeks to do is to put the different stakeholders together in a way that the aspects are considered at the same time, rather than sequentially, hence the name. Another way of thinking about this is to consider a time-line. Figure 14.12 shows a hypothetical example where all tasks are carried out sequentially.

Ideally, concurrent engineering would be carried out in full synchronous mode, like a teleconference, with a collaborative CAD system. At the moment, though, concurrent engineering is often carried out by the designer saving the part to a database so that it is available for uploading by experts or expert systems. These then examine, change and return models. A time line for this might look like that shown in Fig. 14.13.

This is a simple example to give an idea of what might happen. By sharing data early it is possible to work in parallel and shorten the development time.

14.2.2 The Infrastructure

Computer practice has changed since the early days of single central computers to a network of powerful workstations. This means that it is necessary to plan

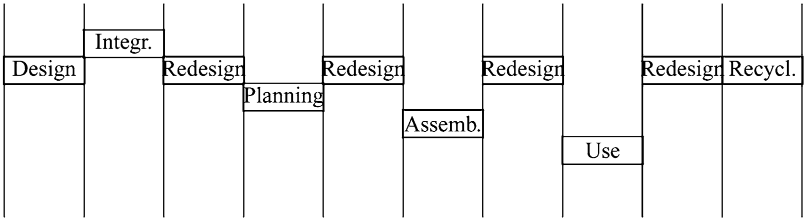
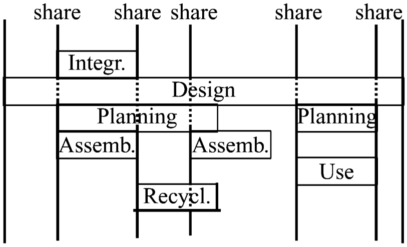


Fig. 14.12 Hypothetical non-concurrent engineering time-line

Fig. 14.13 Hypothetical concurrent engineering time-line



concurrent engineering round a distributed computing environment. With the internet, this distributed computing environment could be within one company or it could be at geographically remote sites. Some researchers report that even different floors in the same building represent a barrier to efficient communication so it is likely that some infrastructure is needed even within comparatively close locations. With large companies, planning production in several locations, there may be several experts in the same competence area, possibly with different requirements.

In order to share data it is necessary to have a common work space. This common area will, presumably, be an area of disk space where models and other data files can be placed and from which they can be retrieved. The type of model data exchange is illustrated in Fig. 14.14.

Another aspect of the concurrent engineering infrastructure is direct communication between participants. This could be done in a number of traditional ways using external means, such as memos, letters or telephone calls. There are more direct, computer-based means, though, which allow communication. Email, chat programs, whiteboards and so on have all become part of current computer practice. There are also more complete video conferencing systems, either separate or running on a normal PC. One advantage of these is that sessions can be recorded and replayed at a later date.

There is, though, competition between computer-based communication with video and sound and other software, which may disturb smooth communication. Computationally expensive activities, such as Finite Element Analysis, FEM, may cause disruption. An alternative is to have two machines, one for communication and one for software applications.

Security is a major concern for this type of working. Some companies do not allow chat programs because of the danger of communicating sensitive data immediately. This is, of course, potentially less of a problem if the communication is done in one site on an intranet without access to the internet. Common data areas, too, risk attack from outside the company. These problems, though, are not specific to CAD use and concurrent engineering. However, it is necessary to be aware of them and plan for them.

14.2.3 Strategy and Communication

A critical aspect is how to share the data. Working via a database gives you the effect shown in Fig. 14.14. This requires adequate version control to ensure that the model is not changed differently in two places at the same time. One method is to allow only the designer to change the model while the experts explain the changes they would like.

The question about how and when to share models, how to arrange changes and so on, lies outside the realm of the system. The system can provide a platform, but there are no hard-and-fast rules about how to use this. Note, though, that structuring the CAD work, especially if it can be structured using early phase design software, is beneficial. For example, the kinematic connections between layout elements are of interest for assembly experts and recycling experts. Kinematic connections can often be realised in different ways in the final design. For example, a static connection may be realised by gluing, welding, bolting together (with three or more bolts) or clipped together, for example. There are many more options. The method could be commented on before the start of the detailed design phase and hence the designer would have more information about the connections when the detailed design is being developed.

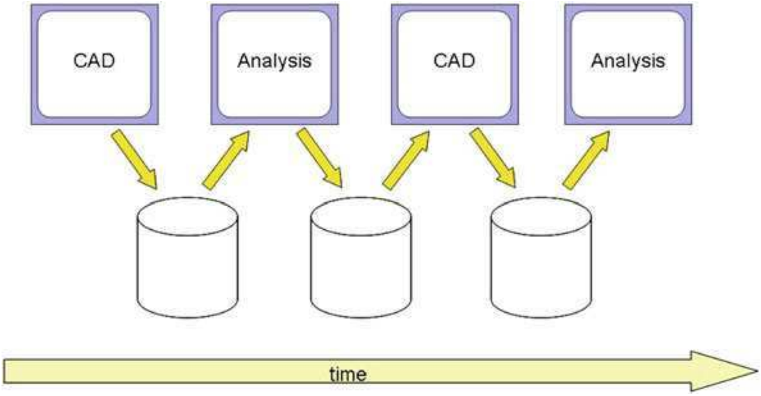


Fig. 14.14 Data sharing for concurrent engineering

Even if there are no computer-based early-phase design tools it is advisable to structure the CAD development as far as possible. The model elements were categorised in Sect. 14.1.6 as: Elements that are fundamental to the design, elements that are functional, elements for practical purposes, and elements which are compromises. The fundamental elements can be shared as early as possible, for understanding of the design, even if they are in draft form. The functional elements indicate interactions with other application areas, hence should lead to feedback. The feedback will give rise to practical purpose elements and also to compromise elements.

One possibility for model modification is to let different experts make changes to the model to improve it from their aspect of expertise. In this case it is necessary to organise a library-like system so that two or more people cannot access the model at the same time.

Another possibility, mentioned above, is to allow only the designer to change the model based on comments from collaborating experts. If a single user makes the changes, though, the graphical picking methods commonly used for defining model elements may not be available for other experts. See below for more comments. This means that it is necessary to work out other methods for communication. These are related to the methods described in Sect. 12.2.

- Picking information—one possibility is to pass picking information, see Sect. 7.4, in the form of a graphic transformation and screen coordinates. I-DEAS used to show this information in its macro files. The information is available somewhere in the CAD system, but may not be available to share as data. Essentially, 2D graphics position is interpreted as a line into the screen and the transformation defines the part orientation and placement. From these you can get a 3D line definition which is intersected back with the model to find the picked element(s).
- Naming—another possibility is to name model elements and communicate these. This is not like the permanent names mentioned earlier, but user defined names which have some logical sense to the CAD user and experts. Names are volatile, though, because the element associated with the name may disappear or be modified during modelling.
- Topological navigation—the method devised by Chris Cary. This, too, is orientation dependent and so needs to communicate a graphics transformation. The selection would be passed as a graphics transformation followed by a text string identifying the element. Probably, though, this would be harder to use than picking information.
- Feature references—feature references means using feature identifications as sort-of shape-based names. One problem with feature names is if the same elements are named in different ways. An example is shown in Fig. 14.15. The different feature names come about from applying different feature recognition methods. If the features are identified centrally and the model labelled with just one set of feature names then this would be a way to avoid the problem.

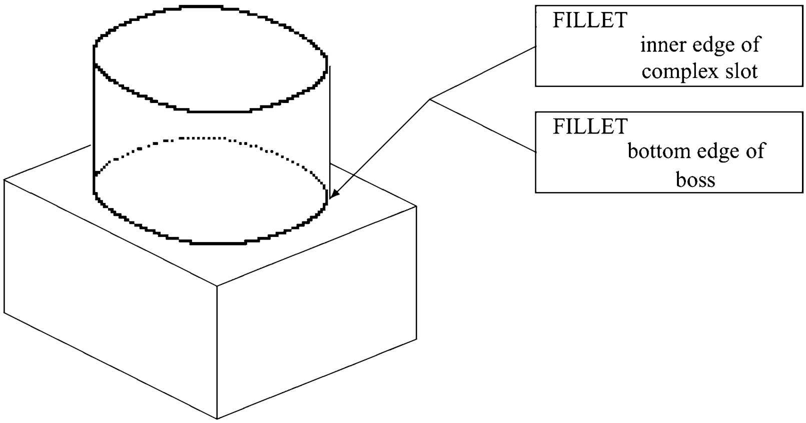


Fig. 14.15 Naming model elements

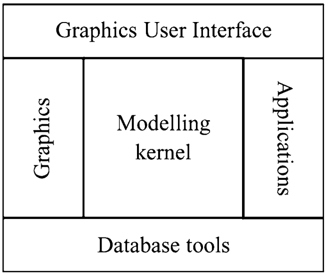
14.2.4 Synchronous Collaborative Systems

Yet another possibility for handling modification is to have only one model but allow multiple access via a collaborative system. Synchronous collaborative systems are technically possible and have been for many years. At least one system developer has announced a system with collaborative engineering, so it appears that this is a trend. Since I do not know how these work I give here a personal solution to show that it is possible. My method for synchronous collaboration is shown in Fig. 14.16. Essentially, the functional elements of the CAD system, shown at the top of the figure, are decoupled and organised as communicating processes. Commands are passed as command strings, which were described in Chap. 12. Having a standardised command shell, based on DJINN say, would make it easier to have plug-and-play applications.

What this means is that commands are created locally and put on a stack in the shared disk area. The CAD system picks the commands from the common area, performs the commanded task and delivers graphical results and possibly exchanges models. Commands need identifiers so that they can be undone or modified.

14.2.5 Experts and Expert Systems

The concurrent engineering systems mentioned above have an implication that the experts in the system are human. This is not necessarily so. It is possible to have expert systems for limited access, or apply software modules to analyse models. Model checking, for verification that transmitted and received models are correct, is described in Sect. 14.3. Application-based analysis software is described in Sect. 14.4.



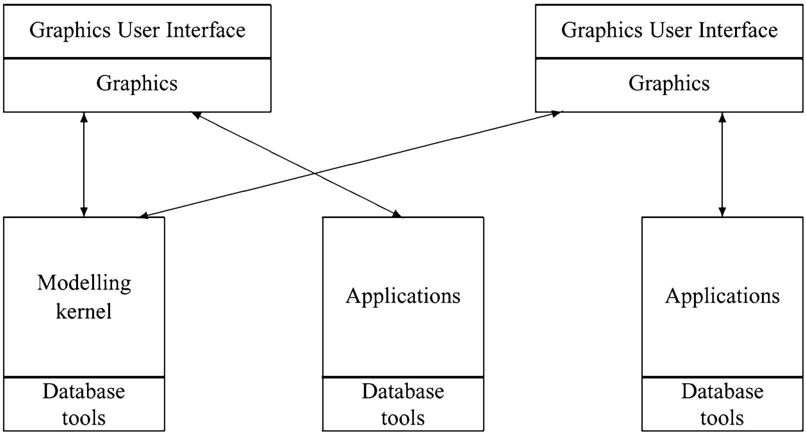


Fig. 14.16 Collaborative engineering by functional decoupling

Human experts are generally superior to software for checking because of their broad knowledge and ability to extrapolate. Software can compete well for boring tasks and for certain well-defined application areas. There is also the advantage that expert systems can be interrogated to find out why they reached a decision.

Checking software is useful but cannot provide the proactive support that a human expert can. Some examples of these are described in the next section.

14.3 Model Checking and Healing

Model checking and healing techniques are there to indicate problems in the model, and possibly to fix them. What they do is to go over a model and check that various conditions are fulfilled. Quite what is done varies between systems.

Model checking involves quite a lot of checks, but not all of which may be interesting for the casual user. Checking can be very useful for systems developers as well, who need to check the correctness of operations before they are released to a CAD system. The checking can involve extensive topological checking as well as geometric checking. Probably it is the geometric operations that are more interesting for the CAD user. One exception, though, is checking for non-manifold topology.

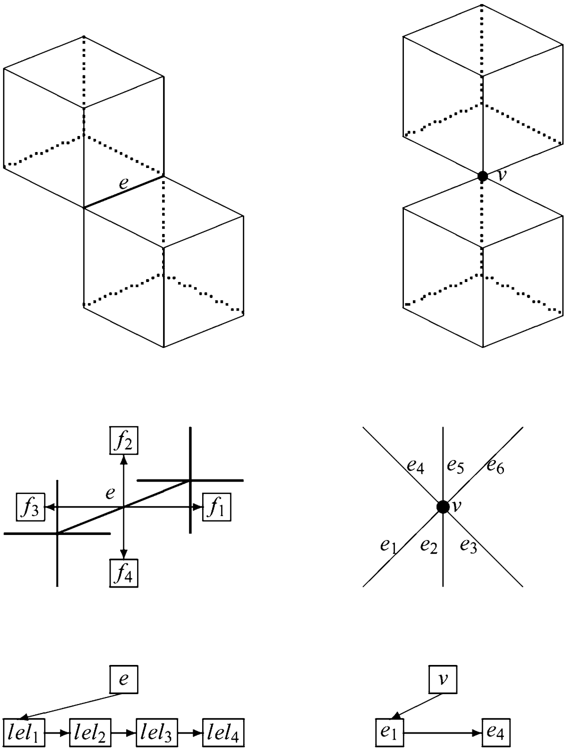
Healing is the term used for correcting the problems found by the checker. Model healing can involve making the geometry consistent or performing more complex operations like Boolean operations. Some possibilities for healing are outlined below.

14.3.1 Non-Manifold Topology

Non-manifold topology, which was described in Chap. 6, is not an error, but indicates an object which cannot be made in reality. Checking for these is easy, because it is only to scan through the model looking at every edge and every vertex to see if more than two faces are adjacent to an edge and more than one edge set refers to a vertex. Figure 14.17 shows simple examples of a non-manifold edge (left) and a non-manifold vertex (right).

In Fig. 14.17, the non-manifold edge, marked e in the figure at the top left, is adjacent to four faces, called here f_1, f_2, f_3 and f_4, as shown in the figure,

Fig. 14.17 Non-manifold edges and vertices



left-middle. This is now usually arranged using four loop-edge links, marked lel_1, lel_2, lel_3 and lel_4, as shown on the bottom left of the figure. For the non-manifold vertex shown in the right-hand column of Fig. 14.17, the non-manifold vertex has two sets of edges. In the middle of the right-hand column these are marked as e_1, e_2 and e_3 in one set and e_4, e_5 and e_6 in the second set. The vertex maintains a pointer to one edge in each separate edge-set, as shown at the bottom of the right-hand column of Fig. 14.17.

In order to heal the object it is necessary to have user interaction to determine whether the objects just touch or just miss. If they just touch then there is material which needs to be thickened, if they just miss then a little material needs to be removed. This is easier for edges than for vertices.

For edges, the chain of links has to be broken into pairs, each with an edge running between the same two vertices as the original edge. How you choose the pairs determines whether the objects touch or join. In the example in Fig. 14.17, if the pairs are (lel_1, lel_2) and (lel_3, lel_4) then the objects touch. If the pairs are (lel_2, lel_3) and (lel_4, lel_1) then the objects miss each other. If the objects miss then the vertices have to be split and the edges could be filleted off with a small, user-determined radius. If the objects just touch then they should also be rounded off, or treated somehow to add a small thickness.

For vertices, if the vertices miss then the healing is easy, it just involves creating a new vertex at the same position as the original vertex and moving one edge reference from the original vertex to the new vertex. The vertices can then be rounded off slightly, if needed. For a vertex where the objects just touch it is necessary to replace the vertex with a ring of edges which can then be rounded off slightly to produce a narrow neck around the object.

14.3.2 Geometry Calculation Errors

One use for model checking and healing is when you have imported a model from a different system and want to know if there are any problems before working on it. It is common that different systems use different tolerance values for geometric calculations. For intersection curves calculated between surfaces it is common that points are calculated iteratively, moving them closer to the surfaces until they lie, within some tolerance, on both surfaces. This is illustrated in Fig. 14.18.

The central solid line in the figure represents the real curve while the dots represent points calculated on the two surfaces. The intersection curve either passes through the points or is fitted to pass close to them. The outer dotted line pair represents the tolerances of the geometry creation system, while the inner pair represent the tolerances of the receiving system. If the system which generated the points is system 1, with a looser tolerance than system 2, the receiving system, then there is a risk that some of the calculated points may lie outside the tolerance limits of the receiving system. Such points are shown as open circles in Fig. 14.18.

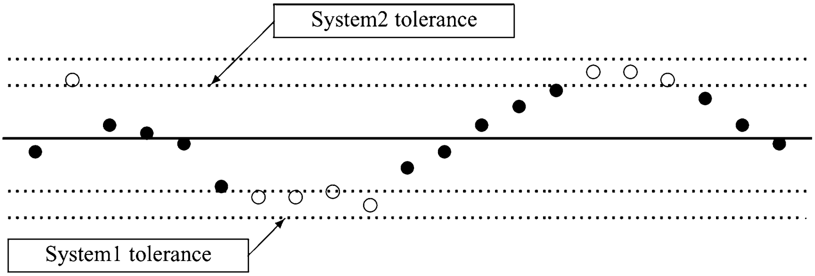


Fig. 14.18 Point positioning and tolerances

Tolerances are necessary for the functioning of CAD systems because of the way numbers are represented. Pfeifer [4] made a thorough analysis of different types of tolerances, which have already been mentioned in Chap. 5, such as:

- Tolerance based on the precision of the computer system.
- Distance tolerance value.
- Angular tolerance value.
- Geometric tolerance.
- Relative tolerance.
- Polynomial tolerance value.

One or more of these may be available for the user to change in CAD systems, hence the need to check tolerance-based geometry. The problem may even occur when both the sending and receiving system are the same software, if the tolerance is not stored as part of the model file, though this is less likely.

The healing process is helped for intersection curves if the representation for the intersection curve includes the two surfaces that were originally used to create the curve. If these are not available directly from the curve representation then it is necessary to use the body structure to find the surfaces of the faces adjacent to the edge referencing the intersection curve. When the surfaces have been retrieved then they can be intersected back to calculate a new curve at the tighter tolerance.

14.3.3 Joining Errors

Another type of geometric error that can occur when trying to match free-standing geometry. This can be necessary if, for example, models are exchanged using IGES. IGES does not communicate the model topology directly, so this has to be reconstructed. One solution is to create a set of surface patches as faces and then to sew these faces together along coincident edges in order to create a connected model. This is helped if the IGES file uses bounded geometry, for example, where the elements correspond roughly to topology. Failing that, it is necessary to do some geometric matching which, by virtue of the necessity for tolerances, mentioned above, it is harder to match items.

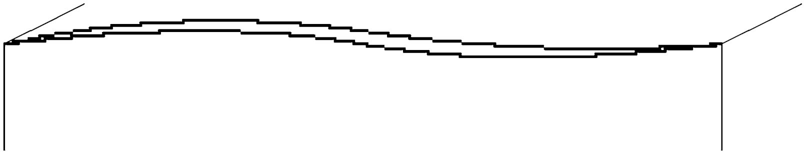
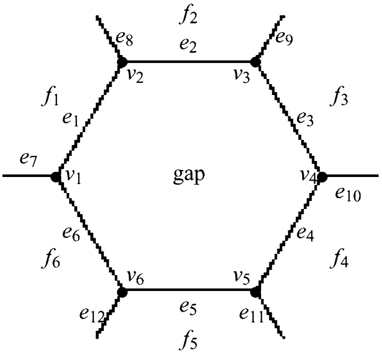


Fig. 14.19 Curve mismatching

Fig. 14.20 Missing face example



One potential problem is illustrated in Fig. 14.19. Two curves are not matched because of tolerance errors and hence not joined.

Joining errors are characterised by edges with only one adjacent face or by edges which are "sharp", i.e. which have the same surface on both sides but with opposite normal directions. Similar edges can also appear if, for example, a whole face is missing because the solid is being recreated from an incomplete surface model.

Mending a model with this kind of fault might be done by recalculating the geometry, moving the geometry closer. It might also be done by recalculating an intersection curve in the same way as described in Sect. 14.3.2. Yet another possibility is that the surfaces are trimmed back, as is done for blending, and new surfaces added which do meet, although this is a poorer solution because it increases the model complexity.

Mending missing faces is more complicated unless the edges around are continuous. In Fig. 14.20 there is a missing face, marked "gap" surrounded by six edges, e_1–e_6. These edges can be found by stepping round the hole, from face to face, finding neighbouring edges with only one face or neighbouring sharp edges. If edge e_1 has been identified as a one-sided or sharp edge of face f_1, you step clockwise round v_2 and find edge e_2. Then find the opposite vertex to v_2 along e_2, which is vertex v_3. This is repeated, the edges round v_3 are traversed clockwise until the next partially defined edge, edge e_3 is found, and so on. Eventually the start vertex is found and the gap boundary defined. This process can be problematical if several holes, or gaps, touch at vertices, but this is also manageable with a little care.

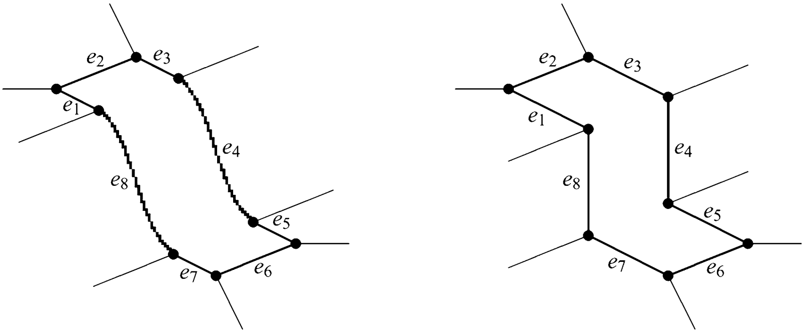


Fig. 14.21 Missing face example with continuous and non-continuous geometry

The next problem is to set in the new geometry. Figure 14.21 shows examples of gaps which are continuous, on the left, and discontinuous, on the right. In each case, the gap is surrounded by edges e_1 to e_8. The gap on the left, though, can be filled reasonably with a single surface while that on the right seems to be three gaps run together into one, and is less appropriate as a single surface piece. This is not say that it is impossible to put in a single piece of surface. With NURBS, for example, having coincident knots would allow you to have a single surface with discontinuities, but there is a difference between what you can do and what you should do. The "correct" solution would be to divide the gap into separate pieces, bridge it with extra topological elements to create multiple faces and put each into a separate, simple surface. However, this is not necessarily easy.

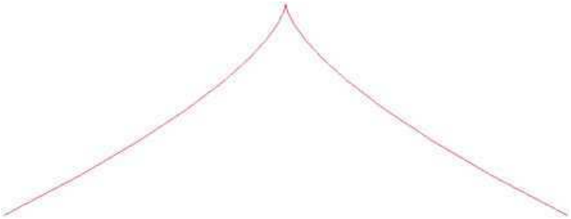
Remember, as with all these healing suggestions, it is much easier for you to see what to do than it is for the computer. You have a global view and a global understanding of the problem while the computer has only a very localised view. A human equivalent might be going for a walk on your hands and knees looking closely at the ground you are crawling over and trying to understand the countryside from that. This is inherent, so it is sometimes easier to have algorithms which demand user interaction rather than trying to make the healing totally automatic.

14.3.4 Geometric Singularities

Geometric singularities are related to the problem of self-intersecting geometry, described next. Geometric singularities are not necessarily a problem unless they lie within an edge or face. Take, for example, a simple cubic 2D Bézier defined by the control points: $(-10,0)$, $(10,10)$, $(-10,10)$, $(10,0)$ mentioned in Sect. 5.5.6. This curve is shown in Fig. 14.22. If you analyse the curve, there is a well defined position at $t = 0.5$, but the tangent has zero length.

Using a part of this curve is possible, it is just the inclusion of the singular point which can cause problems. Detection of this would involve checking the geometry

Fig. 14.22 Geometry with degenerate point



to find the singular point or points and checking whether these lie on topological elements in the model. If they are then the user should, at least, be warned about these when checking the model.

It is difficult to do automatic healing for this kind of problem because the choice of the geometry rests with the user.

14.3.5 Self-Intersections

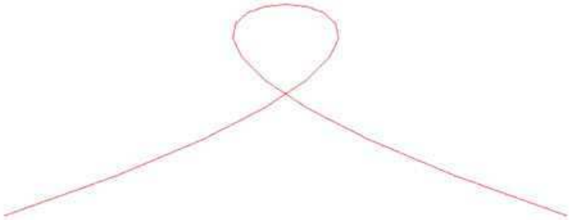
There are two types of self-intersections which might occur: self-intersecting geometry and self-intersecting models.

Geometric self-intersections are similar to geometry with singularities in that part of the geometry may be useful even though there can be problems with using it. Take, for example, another simple cubic 2D Bézier, which was also mentioned in Sect. 5.5.6, and is defined by the control points: $(-10,0), (20,10), (-20,10), (10,0)$ (Fig. 14.23). If you evaluate this at $t = 0.17267316$ and at $t = 0.82732684$ (the roots of the equation: $7t^2 - 7t + 1$) you find the same intersection point: $(0, 4.285714, 0)$

Similarly with surfaces, although the self-intersection may be a curve. At points along the self-intersection curve there is an ambiguity, so that a point may lie at different parametric positions with different normal vectors. However, unlike degenerate geometry, self-intersecting geometry can be valid and useful. A good example of this is the self-intersecting torus. Four types of torus can be readily identified: 1. Normal, 2. Touching, 3. Apple torus, 4. Lemon torus. These are illustrated in Fig. 14.24.

Although this may seem a little academic, these can appear naturally. For example, if you create the shape in Fig. 14.25 you have all four types of toroidal surface. The radius 12.5 circular arc gives you an apple toroidal surface, the radius

Fig. 14.23 Self-intersecting geometry



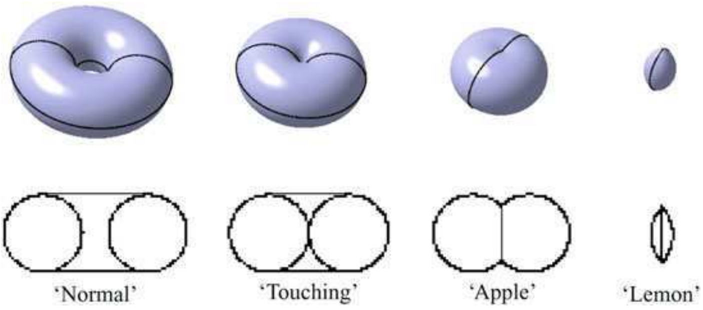


Fig. 14.24 Torus examples

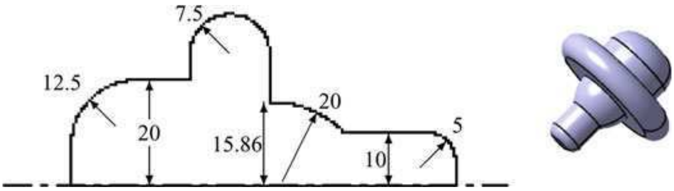


Fig. 14.25 Torus examples in a simple model

7.5 arc gives you a normal toroidal surface, the radius 20 arc gives a lemon toroidal surface and the radius 5 arc gives a touching toroidal surface.

It is clear that the model which results from rotating the profile about the axis is a reasonable model. None of the critical points appears in the final model and hence the model should not cause any problems. This is an illustration of the earlier statement that the presence of self-intersecting geometrical elements does not necessarily mean that the model is invalid.

Another type of self-intersection is where model parts cut into each other. Sections 4.2.10.7 and 4.7 describe how self-intersecting objects may be made. Another method is to sweep self-intersecting geometry or overlapping geometry and thicken this, as described in the exercises at the end of this chapter.

Healing self-intersecting objects may be done with a variant of the Boolean operations, although I have not seen this available in commercial systems.

14.3.6 Sharp Faces

Sharp faces are faces which taper off. Figure 14.26 shows an example of such a face. This is not an error, but can cause confusion when calculating tool paths, for example.

The horn-shaped face shown on the right may cause problems because, at the extremes, the distance between intersection points across the face is less than

Fig. 14.26 Example of sharp face

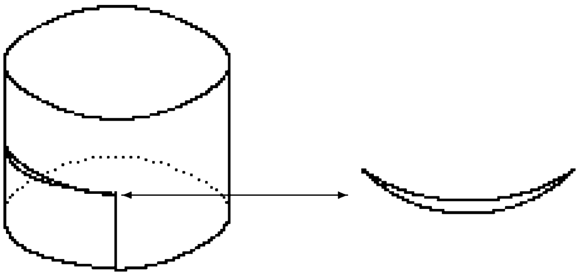
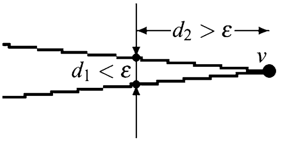


Fig. 14.27 Distances and tolerance in a sharp face



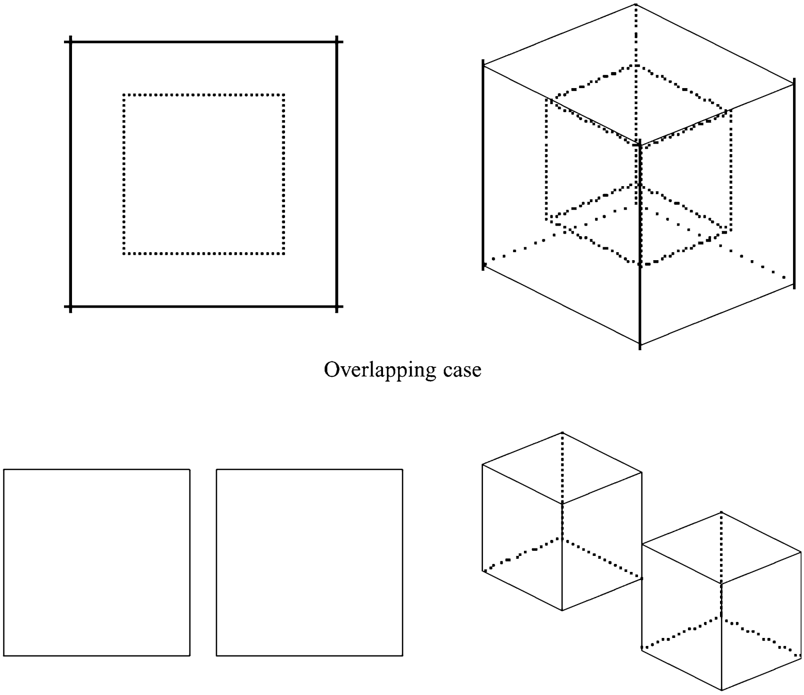
tolerance whereas the distance from the vertex is greater than tolerance. Figure 14.27 illustrates this. ε is the tolerance for the smallest distance between two points. d_1 is the distance between two intersection points cutting across the face, d_2 is the distance from the vertex v of the face. In effect, the sharp end portions of the face act like a fuzzy line due to the problem of geometric tolerances. This is not an error, but it is worth being aware of if your CAD system can detect it.

14.3.7 Multi-Piece Solids

The current tendency is that a model file contains only a single model. While this is comprehensible it means that objects with multiple parts may appear to be a single object and the differences hidden inside the datastructures. Multi-piece solids can be created inadvertently during modelling when model parts do not touch. These may be made by extrusion, for example, or by chamfering. They might also appear as the result of changing parameters in a parametrised model to inappropriate values.

There are two ways that a multi-piece object might be represented: using multiple shells; using multiple volumetric elements. For shells, see Fig. 2.17. A shell is a set of connected faces. The ACIS kernel, for example, allows multiple "chunks" of material in its datastructure. You do not really need to know how multiple pieces are represented, just that they can be. What is more interesting is whether the pieces overlap or not. Partial overlaps should have been detected when checking for self-intersections, so the pieces either completely overlap or do not overlap at all. Figure 14.28 shows two simple examples of overlapping and non-overlapping structures.

The case where the structures do not overlap is probably wrong. It would certainly cause problems for the standard manufacturing techniques and, if



Overlapping case

Non-overlapping case

Fig. 14.28 Simple, multi-piece models

separate pieces are really required, then they should be in separate parts of an assembly. The overlapping case is correct, from a modelling point of view, since the material is continuous, but can cause problems for manufacturing, say. It might be possible to manufacture some objects with some of the rapid prototyping techniques. Otherwise the model would have to be split, manufactured and the pieces joined. It is worth giving a warning for objects with cavities.

14.4 Design for ... Something

There are various analysis tools for analysing a design from particular points of view. A well-known set of software modules for this were developed commercially by Boothroyd and Dewhurst. The purpose here is not to describe these modules but to put them into the context of CAD presented in this chapter. Note, though, that there are conflicts between the results of these analyses and, as usual, there may be a trade-off between optima rather than having one clear solution. Design for assembly and design for manufacturing, for example, can fight each other. For manufacturing it is better to have simpler parts which are easier to manufacture. For assembly, it is better to merge parts, where possible, to have fewer parts to assemble.

The reason for this section is not to give an exhaustive description of the techniques and their workings. The intention is simply to say that they exist, a little of what they do and why they are used. It is the last that is important here. The techniques themselves are described better by the people who sell them or developed them.

14.4.1 Design for Manufacturing

This technique attempts to improve products from the point of view of manufacturing them. The method developed by Boothroyd and Dewhurst, pioneers in this technique, is partly based on features. There are various other factors, such as corner radii, which affect the size of the tool that can be used. The method analyses a process plan for a part and indicates how it can be improved. The process plan can be built up alongside the design of the part and tested several times to optimise the design.

One question is about *why* optimise manufacturing. One factor is cost. Another factor is social, if parts are to be manufactured in different places in the world. Traditionally, parts were designed, manufactured and assembled by a company, but there is a possibility, now, that some products are designed in one country and manufactured and assembled somewhere else. This means that the company cannot always control the manufacturing conditions and may want to consider simplifying manufacture or simplifying assembly, depending on local workforce skills. If the product is intended for a country with a lower level of manufacturing facilities, or fewer natural resources, then it can also be important to rethink manufacturing.

14.4.2 Design for Assembly

This technique attempts to improve products from the point of view of assembling them. This is another technique pioneered by Boothroyd and Dewhurst. There are also many interesting papers on proactive design for assembly by Jared, Swift and others, already mentioned in Sect. 11.4. Jared and Swift's work fits closer to the idea of concurrent engineering than does the reactive analysis of Boothroyd and Dewhurst.

Again, there are a variety of techniques for improving the ease of assembly of products. One of these techniques is to merge parts. Merging parts, though, risks complicating or changing the manufacturing techniques for the part. Figure 14.29 shows a version of one of the standard examples, although I don't know to whom to attribute it. At the top left of the figure is an assembly with ten pieces: one box,

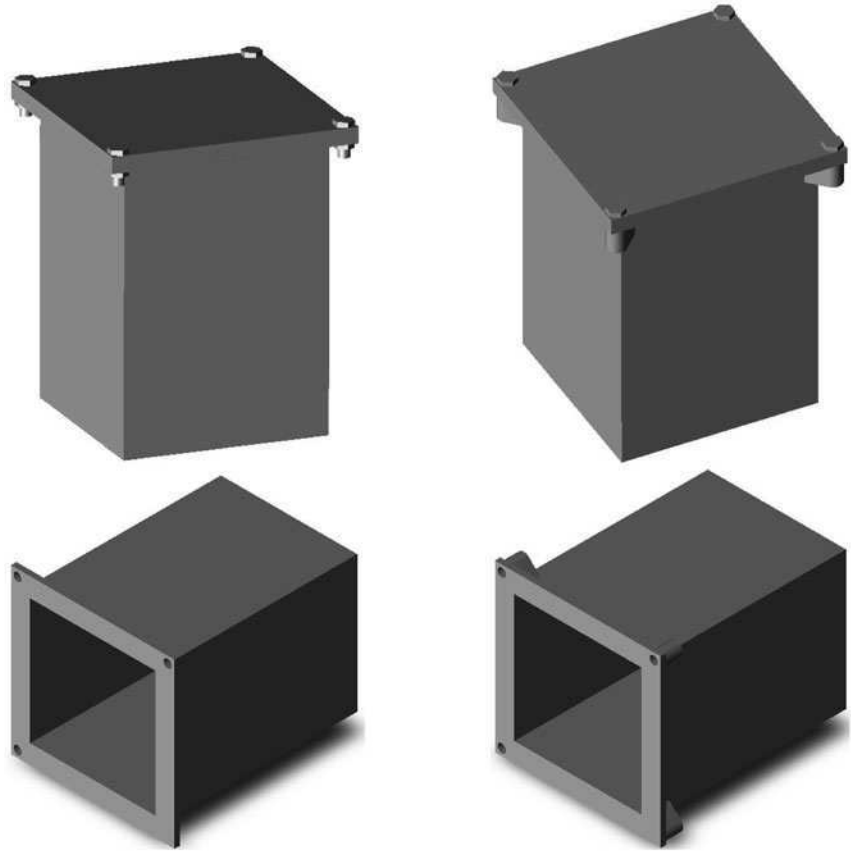


Fig. 14.29 Reducing elements in an assembly

one lid, four bolts and four nuts. At the bottom left you can see the box one its own. It has a flat rim with four holes through which the bolts are inserted. On the right you have an alternative. At the top is an assembly with six pieces, one box, one lid and four bolts. The reduction is made by integrating the bolts with the box, adding the screw threads into extrusions from the box.

There are other techniques that help in improving the assemblability, such as reducing the weight of components. This might be done by creating pockets or holes in the design. This, too, complicates the manufacture, but note that the manufacturing features introduced into the design have a different purpose to those for connection. This means that, probably, they need different tolerances to those for connections. The features for reducing weight also have a different purpose and it is useful to be able to communicate this information with the model.

Symmetry is another important factor for assembly. This can be another important consideration for the early phase of the design process so as to save work by using the symmetry operation.

Csabai [2] also worked out assembly and disassembly paths for the layout phase. His method was based on the space occupancy and is important because it could be used to check the simple layout primitives before detailed design starts. Usually, the earlier that methods can be applied the better, since it costs less to make modifications.

Again, there can be social reasons for facilitating assembly. One of these reasons might be if the product is to be assembled by a less skilled or less motivated workforce. Automatic assembly also needs easier assembly methods.

14.4.3 Design for Disassembly

Design for disassembly is important for the end of life of manufacturing. Disassembly is not necessarily the same as reversing assembly, because sometimes parts can be broken because their material will be reused, not the parts themselves. For material recycling it is important to have material separation and hence certain connection methods are not good. Connections may be replaced by breakable connections. Connections which need tools to disconnect them can be reduced.

Disassembly for repair is slightly different than disassembly at end-of-life as product damage should be minimised. Easy access to parts which are likely to need maintenance or replacement is a consideration at the layout stage.

Note, again, though the importance of having explicit connection information from the beginning. If this can be introduced in the early phase of design then it provides useful information for the detailed design.

14.4.4 Eco-Evaluation

There is a method, the "Eco-Indicators method", which seeks to cost the static parts of production. The method is demonstrated for part of a machine tool in [5]. The work, carried out by CeSI (Centro Studi Industriali), the Centre for industrial studies in Italy, was done for the analysis of part of a machine tool for the NEXT project, a project on the next generation of machine tools.

In his presentation in [5], Luca Mozzanica describes how eco-evaluation works, defining the relevant time span and geographical territory for the analysis. He points out the difficulties involved in setting the analysis up and defining the results. As an alternative he describes the Eco-Indicators method, or EI99, which calculates the ecological loading using a common unit called the millipoint. This does not allow the different contributions to be seen but does allow alternatives to be compared to choose the best option. Mozzanica illustrates the method very well, using the example of how a cast-iron ram and an aluminium ram can be compared to choose the material. Mozzanica shows how the different elements, such as transport, material, manufacturing and recycling are used to build up the analysis.

For any given part you might have the following elements:

1. Material. The millipoint score is a combination of elements concerning the acquisition (mining, etc.) and the processing per unit of weight. The score is then multiplied by the weight of the part to be made to give a total millipoint score.
2. Transport. The millipoint score is calculated as a cost per kilometre that a part is transported. The score is a weighting of fuel use, pollution and other factors.
3. Manufacturing. The millipoint score is calculated as a cost per unit weight of material removed. The energy costs, lubrication costs, tool and machine wear and so on are included in this.

The Eco-Indicators method is useful for understanding the contributions but does not always give a clear picture of how the scores should be broken down into different elements. The different contributions from the various factors are hidden in the millipoint score and it can be hard to see how the millipoint value was calculated. The technique can be helpful in choosing between variants as well as in making clear a number of factors to be considered.

14.4.5 Evaluation for Ecological Manufacturing

As part of the work for the NEXT project one task was on evaluating the use phase of manufacturing. The work, led by Professor Paul Xirouchakis and carried out by Vincent Capponi and Oliver Avram, was aimed at assessing how parts are made.

The eco-indicators method is not really accurate enough to take into account the factors during the use phase. Capponi, in his presentation in [5], uses the example of driving between Geneva and Annecy. The eco-indicators method would calculate the cost in terms of the distance whereas the style of driving and the traffic conditions will affect the actual ecological effects.

Figure 14.30 shows the organisation of the GREEM software. The shape description is given as a set of manufacturing features. The machining options and the evaluation criteria are given by the user. The evaluation of the part with the specified options and evaluation criteria is performed with the help of a database of experimental data contributed by project partners (ASCAMM, CeSI, CRF, Danobat, EPFL, Fatronik, FIDIA, IFW, MAG, ONA, Tekniker). The result of the analysis is a report indicating which machining options contribute to the total score. In a second level of development, Avram analyses the moving masses of the machine based on a machine tool model with data about some of the machines used for the experiments.

This kind of method can be used for evaluating process plans from different points of view. If used during the design phase then it may be possible to modify some of the shape features in a part to optimise the shape. Otherwise, it is probably more useful for the process planner than for the designer. If the part is to be made in large numbers then it is important to minimise the resources used. Also, the moving masses of machine tools can be analysed to set the part orientation for machining so as to reduce energy use, if possible.

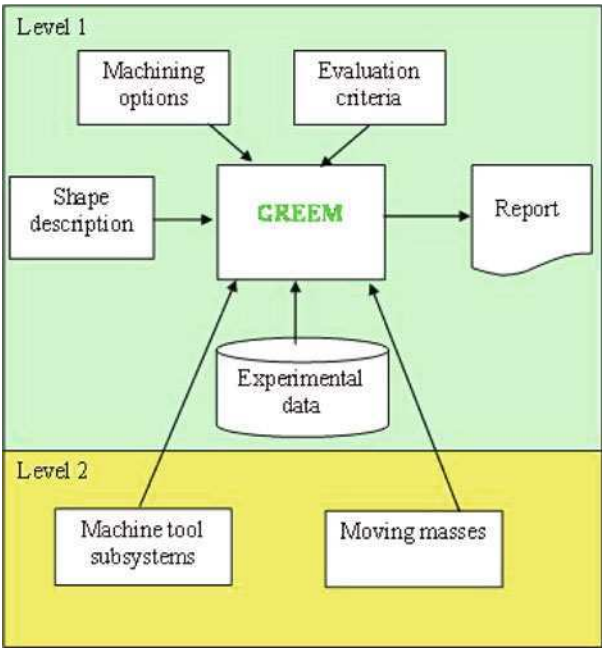


Fig. 14.30 GREEM organisation

14.5 Engineering Case Studies

Case studies sometimes involve dramatic failures which catch the public attention, at other times they involve smaller complications. I first heard about study of engineering failures from researchers at the late Professor Kosuke Ishii's laboratory at Stanford. Study of this topic is not to criticise people's failures but is intended to lead to improvements in both design and processes. There are a number of these and it is important to understand why things have failed in order to reduce risks. Studying both success and failure is a useful occupation to increase understanding and breadth of vision.

What is given here is not intended to be a complete analysis of any of the topics, merely a very small introduction and an indication of some lessons to be learned. A proper treatment of the subject would take much more space than is available here.

14.5.1 The Vasa

During the seventeenth century Sweden was a dominant nation in the Baltic and the then King, Gustav Adolf, wanted to have an impressive flagship as part of his fleet. The new flagship was designed and built. It set sail but capsized and sank in Stockholm's outer harbour. Analysis of the ship's structure revealed that the ship was top heavy and had not been loaded with enough ballast. However, had there been enough ballast, reports say that the lower gunports would have been underwater. A commission investigated the loss of the Vasa but reached no definite conclusion. Another, later, report suggested that the King himself had ordered an extra deck to be built, overruling the engineers and it was this extra addition that was the cause of the problem.

If this latter report is true it is an example where a client has overruled the engineers with the result that the product failed. It may seem easy to dismiss this as an example over too much power in the hands of one person, but there are modern examples of politicians overruling engineers. This is more than an academic example because of the legal aspects associated with design failures. If a design has been compromised by the wishes of a client then it is important to note this. There are more complications, too, with the move towards letting individuals design and manufacture their own products. If the product fails, then who is responsible? Is it the client or the design software? This topic falls outside the scope of this book, but the notion of engineering integrity needs to be maintained, even against powerful opposition.

14.5.2 The Titanic

The fate of the Titanic is a well-known example of engineering failures. The Titanic was billed as being unsinkable, but hit an iceberg during its maiden voyage in 1912 and sank with the loss of over 1,500 lives among the 2,200 passengers and crew. The titanic was supposed to be unsinkable because it was constructed with a number of compartmental segments. The theory was that if the hull was breached in one of these segments the damaged segment could be isolated and the ship would remain afloat. In fact, the gash through the ship traversed several sections. Another problem was that the list the ship developed while sinking hampered launching of lifeboats. Both of these problems have been well analysed to increase the safety of modern ships. Although the Titanic has been used as a romantic image of disaster, it is important to regard it as an example for engineering analysis.

It is impossible to think of everything, so it would have been difficult to have thought of the combined problems of the Titanic in advance. An alternative design, that of the Great Eastern by Isambard Kingdom Brunel, used a double hull which proved more durable than the Titanic's compartments. The Great Eastern hit an underwater rock which tore a huge gap in the hull. However, only the outer hull was breached and the ship was able to survive the accident. Advanced simulation might have helped. Aircraft simulators enable both pilots and designers to check critical cases of failure with risking the pilot.

14.5.3 De Havilland Comet

After the end of the Second World War the De Havilland company had the vision of using the newly developed jet engine technology in a civil airliner, the Comet. The project came to fruition and the first aircraft entered into service. Unfortunately the initial success was marred by accidents and the aircraft was grounded during the analysis. The analysis revealed a design fault which was then corrected and the aircraft continued in service.

The use of finite analysis software to check the strength of designs is now commonplace. Also, analysis of fatigue and material failure has advanced a long way, making understanding of component behaviour more complete. Software techniques for physical simulation exist. This means that physical testing can be preceded by virtual testing.

14.5.4 Symmetrical Components

This example comes from Stanford and concerns a product which had two symmetric variants. I do not want to identify the actual product here. In the variants the basic product was complete but needed some extra elements depending on its position. Extra tubes had to be fitted down the left side in the left-hand products and the right-hand side in right-hand products. The problem was that the left and right tubes were similar in shape, so sometimes a left-tube was fitted to a right product, or vice versa, requiring it to be bent slightly in the process. The bending process weakened the tube, which could have caused product failure, but quality control identified the problem.

The point is that the left-hand and right-hand parts were too similar, and hence the assembly personnel were not easily able to identify the correct part. In this case, it would have been better to make parts which were obviously different, either by shape or by colouring them in some way so that they could be identified quickly.

14.5.5 The Mercedes A-Class

This example has been analysed and presented by Professor Petra Badke-Schaub of the Technical University of Delft. The product was, fortunately, not a disaster but needed drastic redesign after the prototype stage. According to Professor Badke-Schaub's analysis the design team was young, highly motivated, well supported by senior management, but the initial product was unstable. According to the analysis the problem was that there was a lack of self-criticism within the design team, everyone believed in the result.

This example illustrates the need for a sort-of "Devil's Advocate" in design, someone who can examine the product critically and criticise it. It is common in software development that users encounter difficulties with software because they are using it in a manner other than that perceived by the software developer. Having an independent evaluator is valuable because that person will look at the product from a different point of view and hence provide information about public perception. Having several evaluators is also better. There is a related topic, called “Augmented Reality”, whereby a simple physical model is enhanced by virtual reality methods to give a perception of a finished product. This provides yet another method of testing before a real prototype need be produced.

14.5.6 The Challenger Space Shuttle

The space shuttle Challenger lifted off and broke up soon after take off in 1986. The blame for the explosion was put on a connection in one of the booster rockets called an “O-ring”. There were other circumstances surrounding the events, though, such as the low temperature at the time of launch, which were said to have contributed. Apparently the design flaw had been noted but not remedied. Again, it has been noted that engineers warned of the potential problem but had been overruled, similarly to the case of the Vasa. There was a report that the boosters could have been made differently to avoid having an O-ring, but there was a political decision to let a different manufacturer make the booster.

14.5.7 NASA’s Mars Probe

I rely on a very interesting article by Oberg [6] for information about this example. NASA engineering could be termed “Extreme Engineering” in popular terms because they have to deal with extreme conditions as well as a lot of special cases. The case in question is another special set of circumstances, but Oberg’s article suggests some interesting elements.

In 1999 NASA’s Mars probe crashed into Mars rather than landing softly. At the time there were reports that one design team had used inches while another had used millimetres and that this had contributed to the problem. Oberg suggests in his article that there were more complex reasons for the loss of the probe. First, he says, the original design was for a probe with two solar panels but one of these was removed to save money. The result was asymmetrical and the pressure of solar wind kept turning the spacecraft so several corrective manoeuvres had to be carried out. Unfortunately, the motors to turn the space craft were not centred about the centre of gravity of the spacecraft so that each manoeuvre moved the probe slightly off course. Finally, the distance of the probe from the Earth meant that the triangulation method used to calculate the position of the probe was less precise than for closer targets.

There are several interesting points in Oberg’s article, but one important one is the problem of redesign. After the removal of the solar panel all the subsequent design steps should have been redone to produce a new design. This highlights a weakness in current CAD, that there is no modelling of the design reasoning process and the decisions made. This is a complex topic which needs more research. It emphasises the need to work logically and to record as much information as possible to make the reasoning process clear for other collaborators.

14.6 Chapter Summary

This chapter deals with the environment in which CAD is used. First of all, the chapter explains that CAD is not an isolated activity but forms part of a cycle. The chapter describes the major stages and the sort of information that is available. The chapter describes briefly the notion of concurrent engineering and different aspects of this. Next, there is a description of different analysis techniques to support analysis of designs. Finally, the chapter presents a brief description of some engineering examples that can provide useful information.

14.7 Assorted Exercises

Note, these checking exercises are not concerned about whether or not the objects can be made, they are about what can be detected by the CAD system checker, if one exists.

14.7.1 Non-Manifold Checking Exercise

Make the object shown in Fig. 14.31. You can do this by creating a 100 × 100 × 100 block and then cutting out square portions, 60 × 60 in three orthogonal directions.

On the face marked with an arrow, create a 60 × 60 square which just touches the interior hole in the face. Extrude this upwards 60 units so that all edges of the new part just touch an edge of the original figure. This means that the result object should have 12 non-manifold edges. Check whether the analysis tool finds these non-manifold edges.

14.7.2 Degenerate and Self-Intersection Exercise

Make a simple cubic 2D Bézier defined by the control points: $(-100, 0)$, $(100, 100)$, $(-100, 100)$, $(100, 0)$. This gives you a curve similar to that shown in Fig. 14.22. Add extra straight edges round this to create a closed figure, extruding it upwards to create an object like that in Fig. 14.32. Check if the extrusion operation warns you about the object while making it and whether it is detected by any checking software. What happens if you cut through the degenerate part?

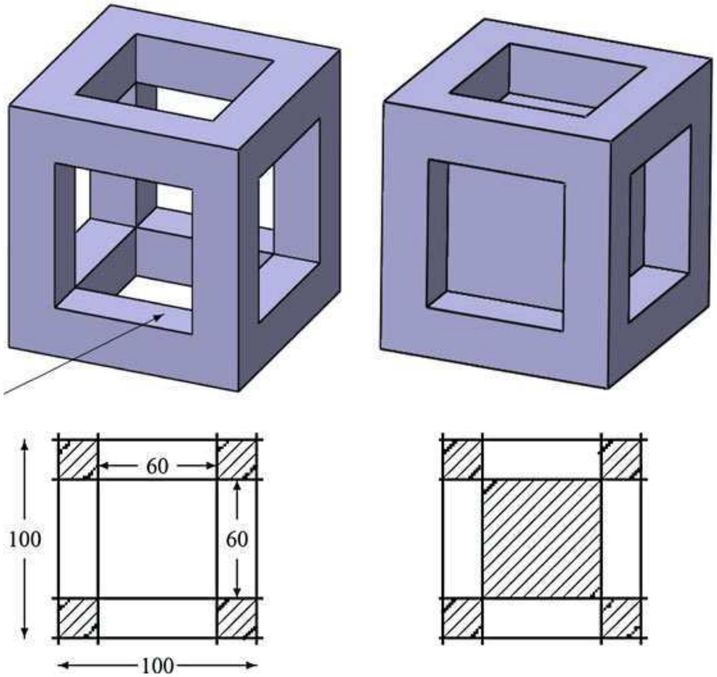
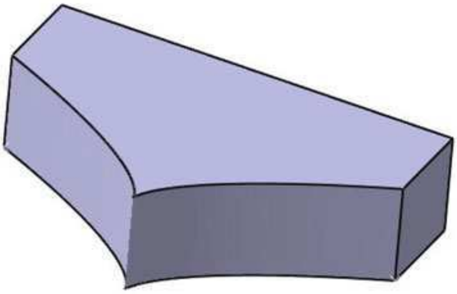


Fig. 14.31 Non-manifold example

Fig. 14.32 Object with singularity



Make a second cubic 2D Bézier defined by the control points: $(-100, 0), (200, 100), (-200, 100), (100, 0)$. This gives you a curve similar to that shown in Fig. 14.23. Try extruding the shape to create a self-intersecting shell (or sheet) object and then give this thickness to create the object in Fig. 14.33. Does the system warn you at the extrusion stage that the curve is self-intersecting?

If the system does complain about the self-intersecting curve then you can still create a self-intersecting object by modifying the control points. Create a curve with the control points: $(-100, 0, 0), (200, 100, 10), (-200, 100, 20), (100, 0, 30)$. This gives you a curve something like that shown from the side in Fig. 14.34.

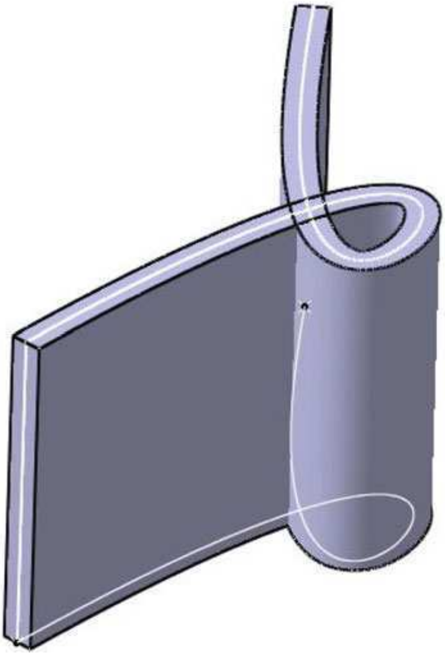


Fig. 14.33 Object with self-intersecting geometry

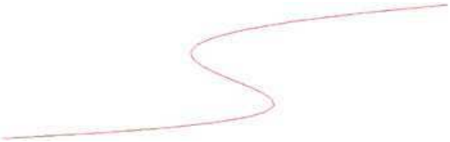


Fig. 14.34 Object with self-intersecting geometry

The curve is no longer self-intersecting, but if you extrude it 100 units, say, you would get an object like that shown in Fig. 14.35.

Check whether this object is detected during the extrusion stage. Check, also, whether it can be detected by analysis tools.

14.7.3 Sharp Face Test

Make an object like that shown in Fig. 14.36. The angles of the "teeth" are: $90°, 80°, 70°, 60°, 50°, 40°, 30°, 20°, 10°, 5°, 2°, 1°$. The height of the teeth could be 40 units, for example. Extrude the shape 20 units and check to see which teeth are flagged as sharp.

Subtract the object from a larger block with double the height so as to make the sharp faces as pockets and check whether the same teeth are flagged as sharp.

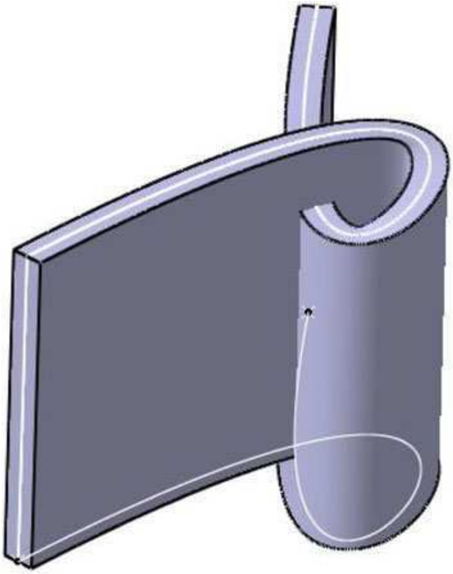


Fig. 14.35 Another object with self-intersecting geometry

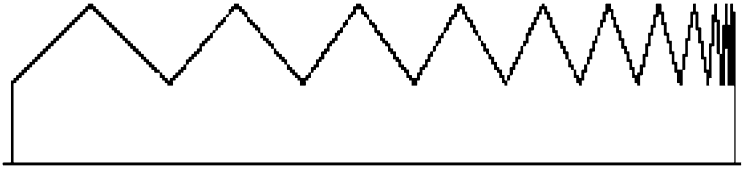


Fig. 14.36 Object for checking sharpness

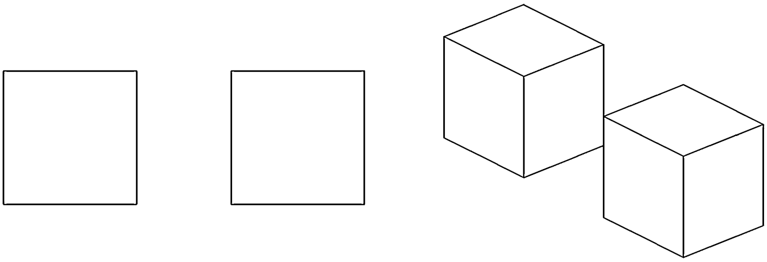


Fig. 14.37 Blocks for multi-piece checking

14.7.4 Multi-Piece Objects

Sketch two 100 × 100 squares and extrude them 100 units. Figure 14.37 It is not an error to make objects like this as an intermediate step, but the CAD system

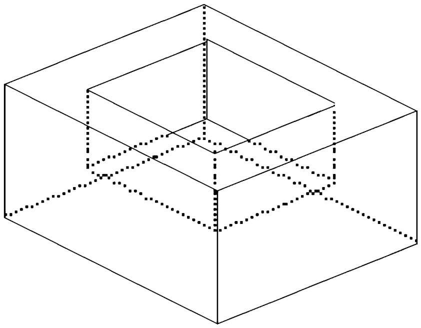


Fig. 14.38 Half-shape for cavity creation

should really signal that these are separate pieces. A model check should also identify that the model is a multipiece body.

Create a square block 100 × 100 × 100 and subtract a 60 × 60 × 60 from the middle. Alternatively you can create a half shape, like that shown in Fig. 14.38 and reflect it about the top face.

Although the object is correct and a single model, a CAD model checker should indicate that the model has multiple shells indicating a cavity.

References

1. Um, J.-Y., Yoon, J.-S., Suh, S.-H.: An architecture design with data model for product recovery management systems. Resour. Conserv. Recy. 52(10), 1175–1184 (2008)
2. Csabai, A.: Layout modelling and evaluation methods and Tools. Ph.D. Dissertation, EPFL, STI-IPR-LICP, Switzerland (2003)
3. Parkinson, A.: The use of solid models in BUILD as a database for N.C. machining. In: Proceedings PROLAMAT 1985, Paris, France, 293–299 June 1985
4. Pfeifer, H.-U.: Methods used for intersection geometrical entities in the GPM module for volume geometry. CAD J. **17**(7), 311–318 (1985)
5. Avram, O., Bueno, R., Capponi, V., Copani, G., Gallino, A., Mozzanica, L., Stroud, I.: Ecoevaluation in manufacturing. Course material (2008)
6. Oberg, J.: Why the Mars Probe went off course. IEEE Spectrum, December (1999)

Chapter 15
Projects

15.1 Project 1

15.1.1 Step 1

The aim of the first part of the project, based on Sect. 2.7, is to practice identification of various elements in the model. Count the different elements in the models shown in Fig. 15.1 and complete Table 15.1 with the number of faces, edges, vertices and inner contours. Remember, these are all Eulerian objects, with genus 0 and multiplicity 1. The two contours in object P6 should be considered part of the same object, they have a common face.

To identify the inner contours, or hole-loops, look at the objects. Every face which has two or more separate boundaries has one or more inner contours.

15.1.2 Step 2

The intention with the second step, based on Sect. 2.7.3, is for you to become familiar with the Euler–Poincaré equation and Euler operators.

Use the objects shown in Figs. 15.2, 15.3, 15.4 and 15.5.

Complete Table 15.2:

In the last column put the result of the equation:

$$\underline{v} - \underline{e} + \underline{f} - \underline{h} + 2\underline{g} - 2\underline{m}$$

This should be zero for every object. In the last row, put the sum of the columns (except the last). Is the equation for the totals also zero?

Use the matrix in Sect. 2.7.3.1 to determine the number of Euler operators of each type needed to create the objects and fill in Table 15.3. Note that a negative number of any operation implies the application of one of the inverse operations: kev, kfe, kbfv, kemh or kgb.

I. Stroud and H. Nagy, *Solid Modelling and CAD Systems*,
DOI: 10.1007/978-0-85729-259-9_15, © Springer-Verlag London Limited 2011

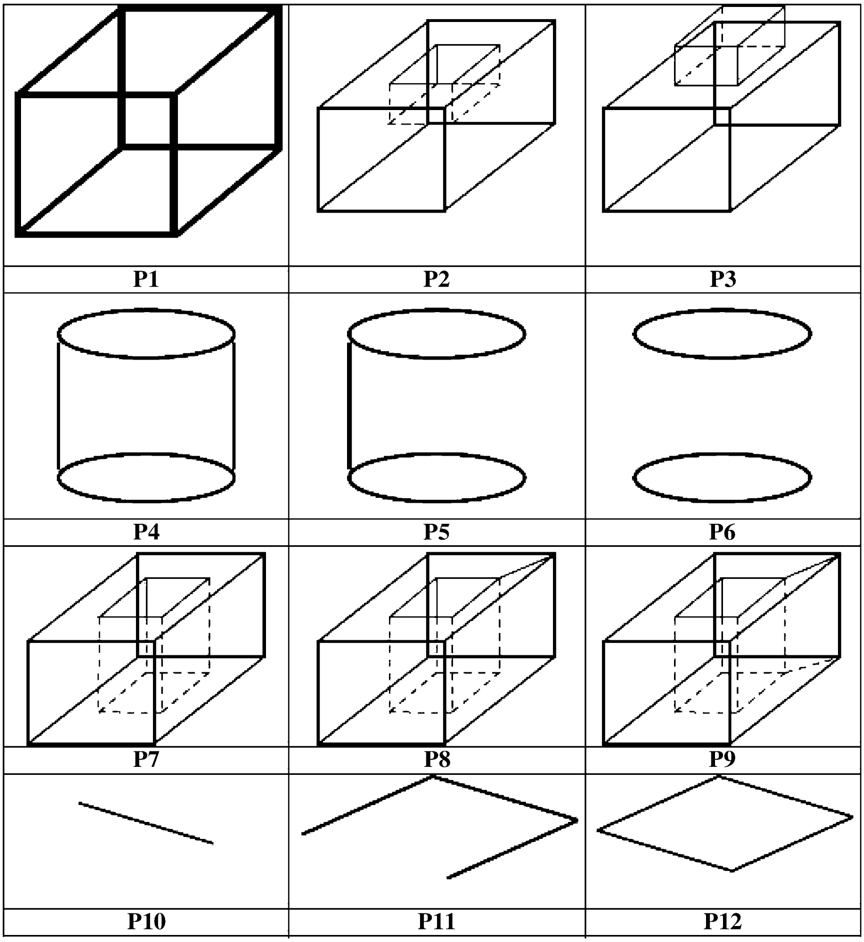


Fig. 15.1 Simple objects for counting

How many operations of each type are needed to convert object 1 into object 2?

15.1.3 Step 3

Propose a stepwise method and a Boolean operation based method for the following operations:

1. Slot operation
2. Perimeter operation
3. Hole making operation

Table 15.1 Number of elements for given objects

Object	Vertices	Edges	Faces	Inner contours
P1				
P2				
P3				
P4				
P5				
P6				
P7				
P8				
P9				
P10				
P11				
P12				

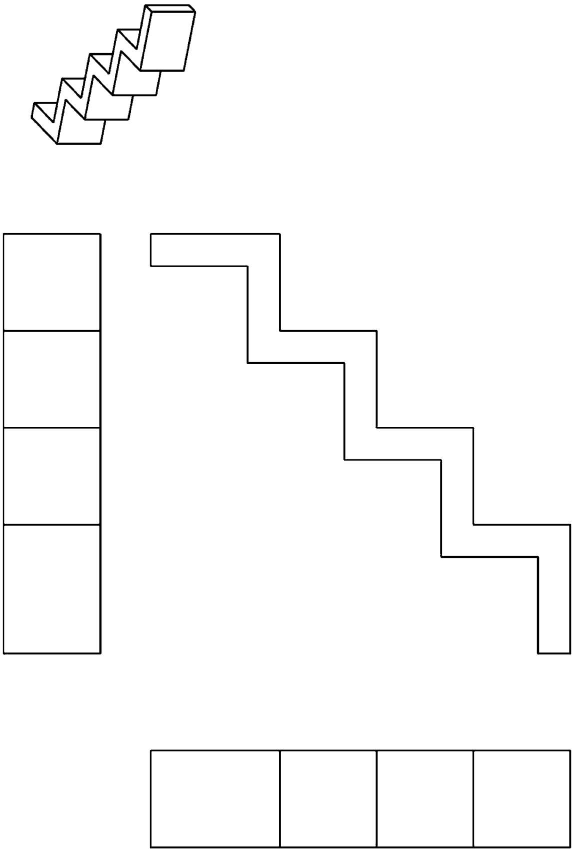


Fig. 15.2 Object 1—Euler wiggle

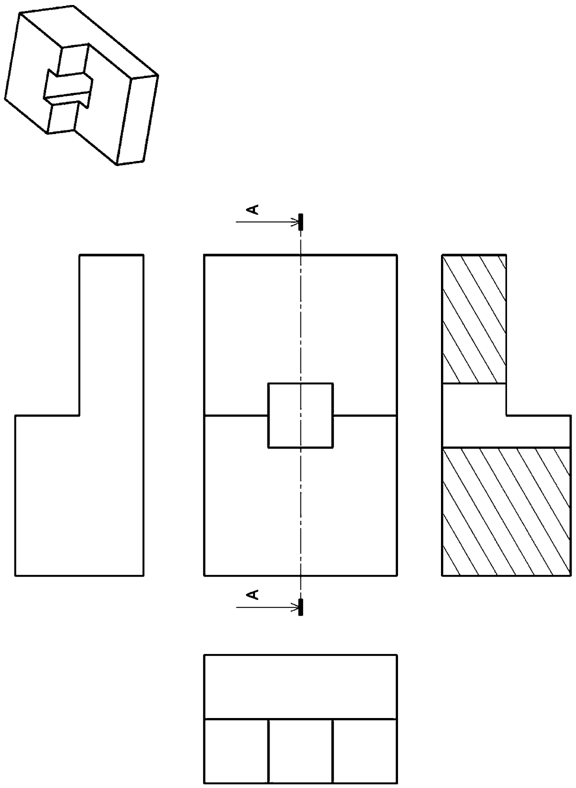


Fig. 15.3 Object 2—Euler stepped block

Determine the input parameters.
Determine the error conditions.

15.1.3.1 Slot Operation

The slot operation makes a slot in an object, simulating the shape formed by passing a tool through the top of the part (Fig. 15.6).

What are the parameters of the operation for defining the slot shape? What are the error conditions that might occur?

Count the elements in the two objects above and identify the Euler operators needed to change one into the other.

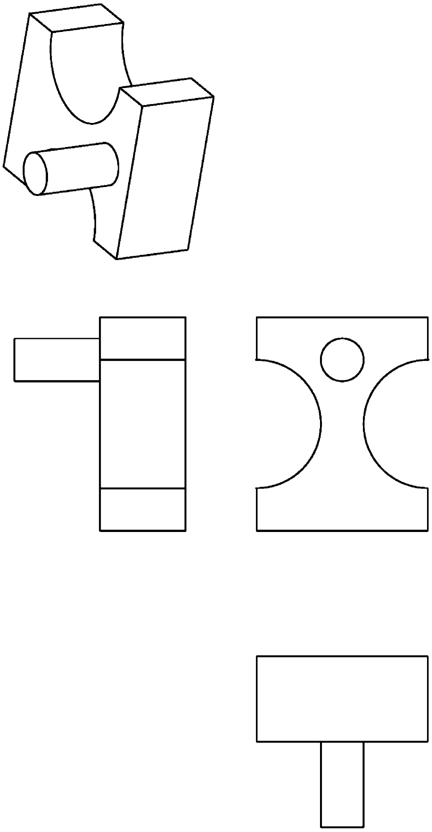


Fig. 15.4 Object 3—Euler curved shape

How would you define the operation to create a slot in a face using Boolean operations? How would you cope with making a slot in odd shaped objects like those in Fig. 15.7?

15.1.3.2 Perimeter Operation

This operation is to create a perimeter wall around a face as in the examples shown in Fig. 15.8.

As above, determine the input parameters and the possible error conditions. Calculate the number of Euler operators needed for the two examples.

How would the operation work on a curved face? Use as example a half-cylindrical face.

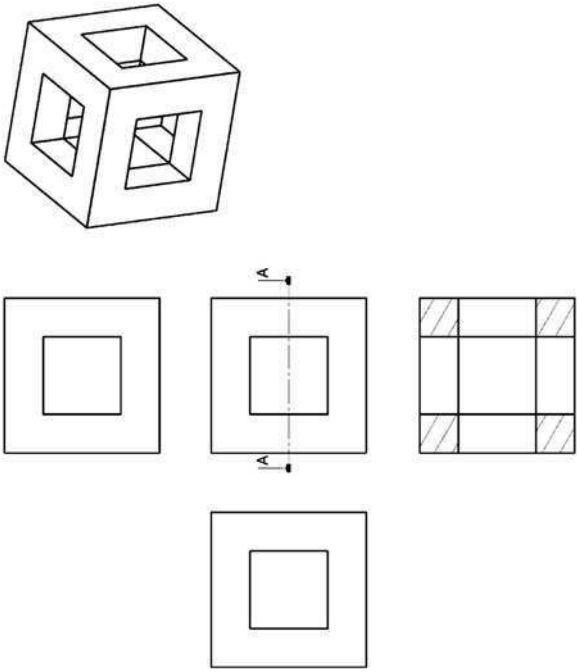


Fig. 15.5 Object 4—Euler hole block

Table 15.2 Number of elements for given objects

Object	$\underline{v}$	$\underline{e}$	$\underline{f}$	$\underline{h}$	$\underline{g}$	$\underline{m}$	eqn
1							
2							
3							
4							
Total							

Table 15.3 Number of operations to make given objects

Object	mev	mfe	mbfv	mgb	mekh
1					
2					
3					
4					
Total					

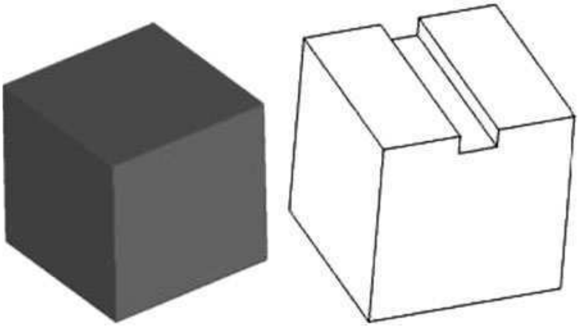


Fig. 15.6 Slot making on a rectangular block

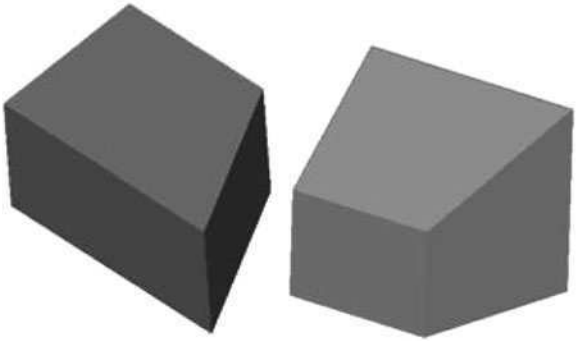


Fig. 15.7 Slot making on odd shaped blocks

15.1.3.3 Hole Making Operation

In your opinion, what should the input parameters and error conditions be for the operation? Sketch some examples and determine the Euler operators needed for them. Suggest a sequence of operations for making a simple hole through a rectangular block. How would the sequence change if you are able to use the Euler operator: MHGKF$(0, -0, -1, 1, 1, 0)$?

15.2 Project 2

This exercise is about lofting which is a technique for creating extruded shapes and using them. The basis for the exercise consists of four sections. The first is on the plane defined by $Y = -150$, the second is on the plane $Y = -100$, the third is in a plane centred at the origin but rotated 45° about the X-axis ($Y = -Z$). The final section is in the plane $Y = 100$. See Fig. 15.9. The top and middle rows show the sections, while the bottom image shows the sections from the side.

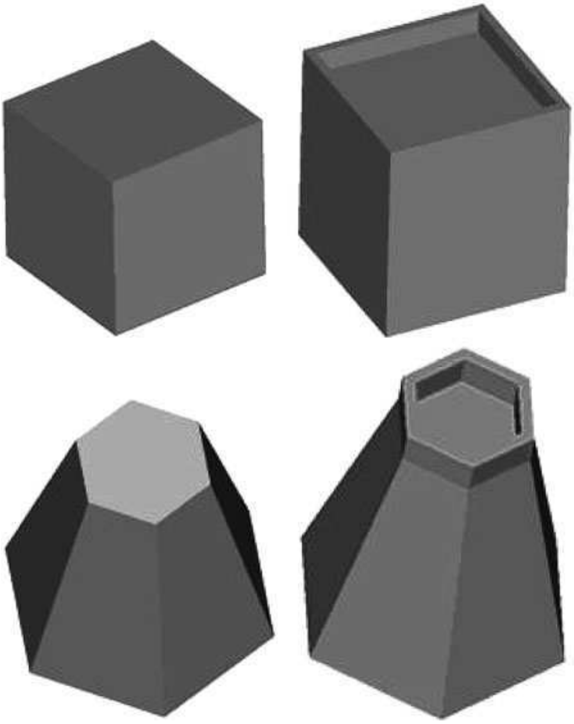


Fig. 15.8 Making a perimeter wall around a face

The first section has a Bézier curve with control points:

(−42, −150, −10) (−32, −150, −10) (−10, −150, 0) (0, −150, 0)

The second section is on the plane $Y = -100$. The curved top section is a Bézier curve passing through the point (0, −100, −2).

The third section is on a plane angled at 45 around the X-axis and passing through the origin.

The final section is on the plane $Y = 100$.

15.2.1 Step 1

First, interpolate the top curve for section 2, that is, a quadratic Bézier curve passing through the points.

(−42, −100, −10) (0, −100, −2) (42, −100, −10)

Next, write down the coordinates of the points in each section. In order to have the same number of elements in each section you should split the appropriate edges to have five points for each section.

This gives you a matrix of points, shown in Table 15.4.

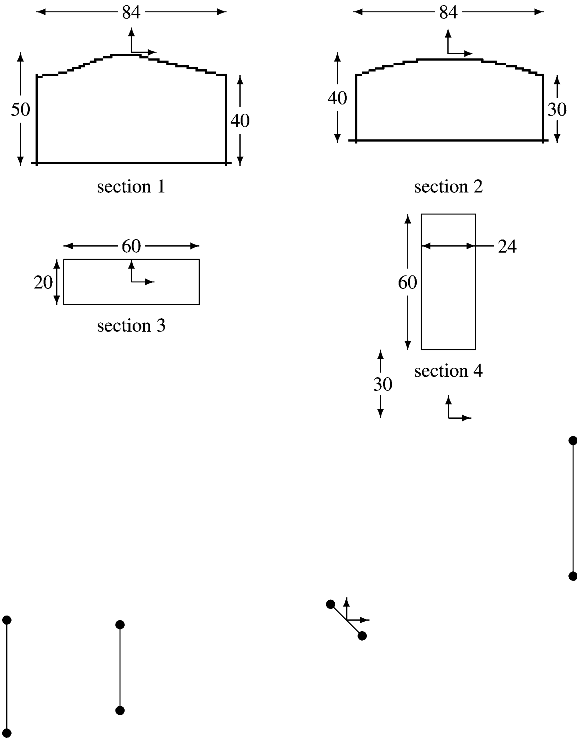


Fig. 15.9 Sections for interpolation

Table 15.4 Points for interpolation

	c1	c2	c3	c4	c5
Section 1	p_{11}	p_{12}	p_{13}	p_{14}	p_{15}
Section 2	p_{21}	p_{22}	p_{23}	p_{24}	p_{25}
Section 3	p_{31}	p_{32}	p_{33}	p_{34}	p_{35}
Section 4	p_{41}	p_{42}	p_{43}	p_{44}	p_{45}

Finally interpolate the five side curves for the sections using the matrix method, described in Sect. 5.5.9. The first curve passes through points $p_{11}, p_{21}, p_{31}, p_{41}$, the second passes through the points $p_{12}, p_{22}, p_{32}, p_{42}$, and so on. Use the parameters 0, 0.2, 0.6 and 1.0 for the estimated parameter values to simplify the calculations.

The basic matrix has the form:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ a & b & c & d \\ e & f & g & h \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The inverse is of the form:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ i & j & k & l \\ m & n & o & p \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

15.2.2 Step 2

Figure 15.10 illustrates how the control points of the section curves are used to create the surface control points.

The points p0, p1, p2 are the control points of a curve in section 1, p3, p4, p5 for section 2, and so on. Interpolating p0, p3, p6 and p9 gives the first row of the surface control points. Interpolating p1, p4, p7 and p10 gives the second row of control points of the surface and interpolating p2, p5, p8 and p11 gives the last row.

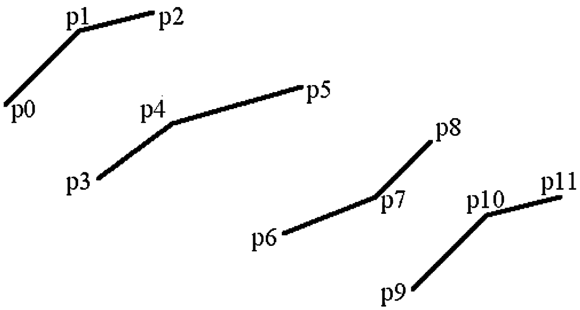
Subdivide the top curve of section 2 at $t = 0.5$ using the de Casteljau method described in Sect. 5.5.8. This gives two half curves instead of the original interpolated curve. This is important because one of these has to have its degree raised.

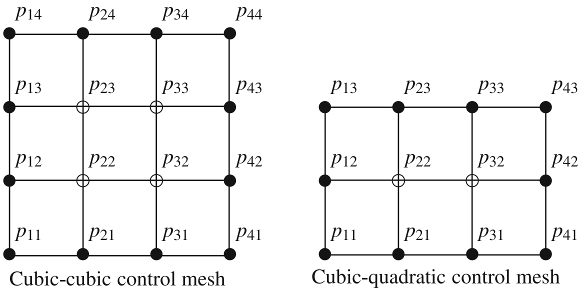
Raise the degree of the appropriate half of the curve, as described in Sect. 5.5.7. One of the subdivided curves corresponds to the cubic Bézier of section 1. Since it is necessary to have the same degree for all curve sections, all sections must be cubic. The half curve is a quadratic and so it is necessary to raise the degree once to have the control points of a cubic Bézier curve with the same shape. The linear curves must have their degrees raised twice.

Determine the control points of the five surfaces which surround the sections.

You need the control points of the section curves in order to find the control points of the surfaces. Three of the surfaces are linear–cubic, so the boundary curves give the surface control points directly. The other two are a cubic–cubic and a cubic–quadratic, with control point meshes as shown in Fig. 15.11.

Fig. 15.10 Curve control points for creating a surface





Cubic-cubic control mesh

Cubic-quadratic control mesh

Fig. 15.11 Control point meshes for Bézier surfaces

The arrangements for the control points of the two more complicated surfaces are shown in Fig. 15.11.

The outside surface control points come from the section control points or the interpolated curve control points. What is missing are the internal points which you get from interpolating the internal control points of the section curves. These points are not the same as the section control points themselves.

15.2.3 Step 3

Now do the same exercise using your CAD system to define the sections and the lofting operation to create the surfaces. Create a solid by adding thickness to the surface object.

1. What happens if the thickness given is 10?
2. Is the resulting object manifold or non-manifold?
3. Reflect the object about the plane $Y = 100$. Is the object tangent continuous? If not, what has to be done to make it tangent continuous?

15.3 Project 3

15.3.1 Step 1

A cube has the corner coordinates:

(40.8494, −34.1506, 68.3013)	(−9.1506, 52.4519, 68.3013)
(−84.1506, 9.1506, 18.3013)	(−34.1506, −77.4519, 18.3013)
(34.1506, 77.4519, −18.3013)	(84.1506, −9.1506, −18.3013)
(9.1506, −52.4519, −68.3013)	(−40.8494, 34.1506, −68.3013)

Create the points in the CAD system, then create the lines between them, finally fill in the edges to create the surfaces. The cube can be created by joining the surfaces and then filling in the cube. This is, at least in part, the opposite operation to creating a shell object from a solid.

As a new part, create a rectangle 50 × 100 on the *XY* plane, as shown in Fig. 15.12. Extrude it 20. On the top face create a rectangle 50 × 20 and extrude this 100 units. You now have an L-shaped block. Determine the transformation matrix needed to move this onto the top of the tilted block.

Calculate the normal vector from two adjacent edges of the cube. This and the two vectors above form a set of coordinates to which the L-block is to be aligned.

To align the block with this set of coordinates, it is necessary to rotate the L-block and then translate it to the corner point. Use the vector $(0, 0, 1)$ as the vector to align with calculated normal vector. Calculate the rotation first and then add the translation. The translation vector is the vector to one of the points on the top surface of the cube.

In an assembly assemble the two objects. First fix the cube as the grounded object and then create constraints to fix the L-shaped rigidly to the top of the block (Fig. 15.13).

Which transformations are needed? Give the answer in terms of the constraints, plane–plane, etc. It is not necessary to give the exact transformation matrices apart from that asked for above.

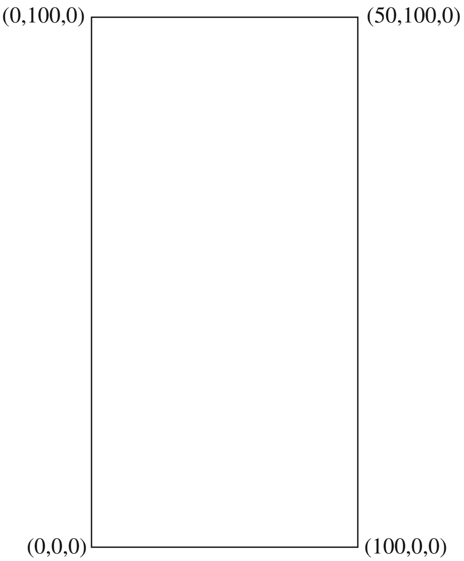


Fig. 15.12 Position of L-shape base rectangle

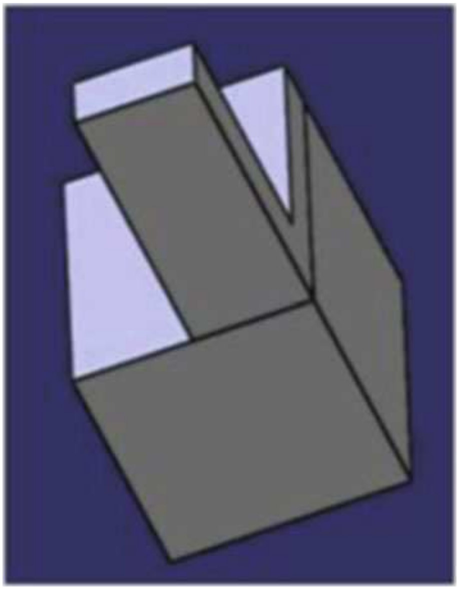


Fig. 15.13 Aligned L-block and cube

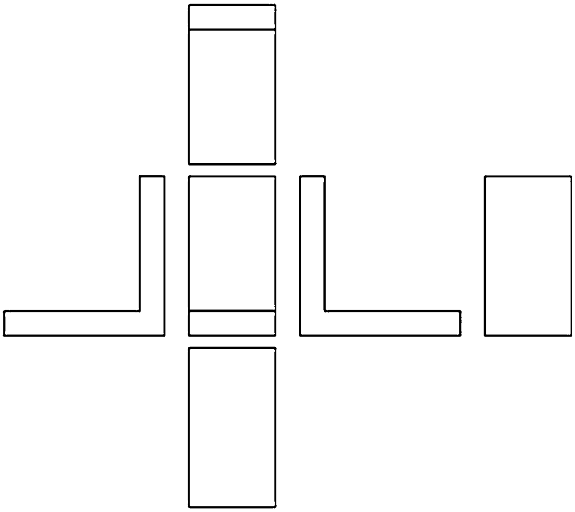


Fig. 15.14 L-block for facetting

15.3.2 *Step 2*

Now subdivide the L-block into facets. The faces are shown in Fig. 15.14.

Determine the transformation matrix to transform these facets onto the plane $Y = 500$. How do you decide which facets are potentially visible from the point (0, 1,000, 0)?

15.3.3 Step 3

Output the product in the STEP format. Identify the transformation specifications for the assembly. What transformations can be represented in this way?

15.4 Project 4

This project concerns the design and manufacture of a hypothetical part. The exercise should be done with a CAD system and you should note the decisions you have made and copy sketches, if you made any, for the project report.

The part to be designed has to take fluids from two places to which it is clamped with flanges and output the liquid at a third point, to which it is also attached with a flange. There should be one support bracket around the centre of gravity of the part. The exact position is not critical because the attachment method will be designed afterwards. There should also be a triangular reinforcement plate with corner points at around: (300, 950, 0) (−300, 950, 0) and (0, 440, 0).

15.4.1 Step 1

Design the holes. One input is positioned at $(-500, 1{,}000, 0)$ and has circular cross-section, radius 25, normal direction $(1, 0, 0)$. The second input is symmetrically placed at $(500, 1{,}000, 0)$, but has radius 20 and normal direction $(-1, 0, 0)$. The output is positioned at $(0, -250, 0)$ and has radius 35, normal direction $(0, -1, 0)$. There should be an outlet for a safety valve somewhere around the plane $y = 400$. This should be horizontal, that is, in the plane $z = 0$.

Create a new product. Insert a new part and build the hole in the new part. You can later use a Boolean operation to subtract the hole from the body.

Once the hole has been designed, design a second new part around that so that it will enclose the hole. This casing should have at least radius 55 at the outlet, radius 40 around the inlet with radius 25 and at least radius 35 around the inlet radius 20. Where the tubes meet there should be at least a thickness of 50 outside the hole cavity. Add the reinforcement plate and the bracket. Also add flanges, thickness 20, around the inlets and outlets, radius 80 with a pattern of eight holes, diameter 12, with a pattern radius of 65. Finally, subtract the hole to make the part.

What are the design features of the part?

15.4.2 Step 2

Now adapt the part for manufacture. How would you make the part for production runs of one part, five parts and one thousand parts. For each of these you should note the production method and the manufacturing features for the production methods. Do you need to change the shape of the part or modify the shape of any of the subparts?

What tolerances do you need to set on which parts of the object?

15.4.3 Step 3

Finally, analyse your work with respect to the CAD system. What couldn't you do in the CAD system? Did you need to use pencil and paper sketches? Look at the history tree of the part. For each operation, write down whether it corresponds exactly to a design feature, exactly to a manufacturing feature or to neither of these.

15.5 Project 5

This project is about features and their role in design and manufacturing. The CAD part is to provide a focus.

Make an assembly like the one shown in Fig.15.15. The project is not about copying a real object but about the elements for the design and manufacture.

The object shown is a frame with three axles linked by gears. The frame is approximately 160 mm long, 110 mm wide and 120 mm high. For the project I would like you to make a model of the frame, the three axes and their gears, and the flywheel. It is not necessary to make an accurate model of the object shown,

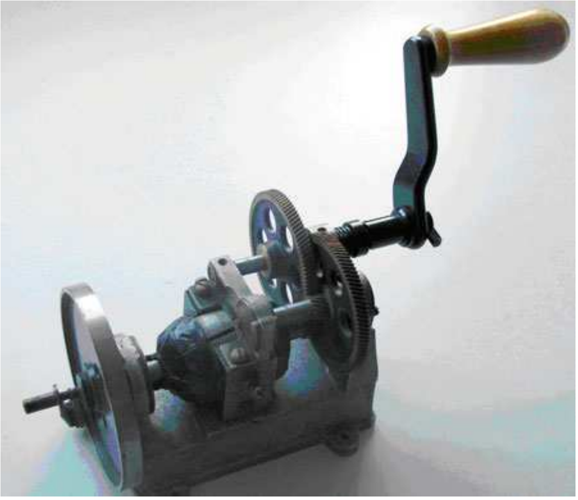


Fig. 15.15 Example object for project

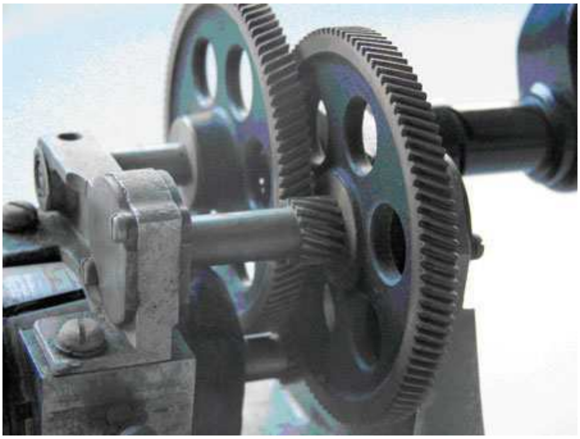


Fig. 15.16 Gears in example object

understand the mechanism and make a model of something similar. The project is about your understanding of features and their role in design and manufacture. The gears are shown in more detail in Fig. 15.16.

It is not necessary to make angled gears, for the purposes of the exercise a model of straight gears is enough. There is a question, though, about what the model of a gearwheel should be. What do you think is a valid model of a gearwheel?

Tasks:

1. Create the assembly. Note any extra sketches or support material you used.
2. What are the "design features" in your design? These are the elements around which the design is built. Use the feature list in Appendix C, but ignore the parameters.
3. What are the "manufacturing features" in your design? Again, use Appendix C as a guide for feature types. These are the elements that are machined and depend on the manufacturing method. Indicate how you would manufacture the elements of the assembly if you manufacture one piece or five thousand. Are the methods the same?
4. Finally, look at the CAD system history tree and say which modelling operations correspond to manufacturing features.

15.6 Project 6

The aim of this project is to examine design problems and to compare them with the available design tools. There is little guidance, because it is for you to consider the whole design chain. This cannot be done if a solution is provided. However, it is important to look at the design activity in terms of the tools available, hence this project.

The project is based on Chap. 11 as a basis for detailed design. It is not absolutely necessary to produce a detailed design but, in the past, students have found this a fun-part. Students should perform a web-search for examples of similar products and note the characteristics of these, or describe existing products they might know of, to show how their design is different. It is expected that students identify the main elements of the product with their estimated dimensions, down to the level of detail necessary, and the connections between them, in terms of physical connections, type of exchange (e.g. gas, fluid, electricity, and so on). The project should produce a product specification, a high-level product solution and an embodiment model. If elements of the project are to be purchased or subcontracted, please explain why. If there are such elements, please explain to what level of detail you would model these in the design.

The following are some examples for consideration, other ideas may be acceptable. Note, also, the information that you are passing between the activities in the design process and analyse which information is needed for different stages.

15.6.1 Transport Vehicle

The transport vehicle is a small vehicle for transporting goods round large sites and manufacturing facilities. An example of a product needs specification might be:

Maximum load: 750 kg.
Product variations/options:
With or without handling crane for loading and unloading.
Two size variants: 3 or 4 m.
Possibility to attach a trailer.
Electric motor.
Sales price: 15,000 CHF or equivalent (please suggest a sum).

This is an example of an initial vague specification, which should be converted to a more formal list of requirements, such as that shown in Table 15.5.

These are just simple examples of what you might produce, you are expected to use your engineering knowledge to provide a more realistic version.

Table 15.5 Transport vehicle elements

	Cart 3 m	Cart 4 m	Trailer
Price			
Variants	With crane Without crane	With crane Without crane	
Weight (kg)	750	1,000	500
Motor	Size?	Size?	Size?
Accumulators	6	8	6
Power	?	?	?

The next step is to identify product components in the solution. Produce these as a list together with a note about which ones you develop and which ones are to be bought or outsourced. Then perform the embodiment task, noting the linkages and constraints between the elements.

If more time is given for the exercise then the detailed design would follow. The exercise should be treated as a game, students should use their imagination.

15.6.2 Breakfast Machine

When I started using the breakfast machine as an example, in 1998, I had not seen such a product but since then I have found that these things exist. The aim, here, is not to copy the commercial product but to solve a design problem with an idea and some simple constraints.

This machine is to make breakfast for people with little time. The aim is to have a machine which can make coffee, boil eggs, squeeze fruit and make toast. The size of the base of the machine should be about 400 mm × 600 mm, maximum, but the height is not fixed. The aim would be to set up the machine in the evening, before going to bed, so that it would switch itself on automatically in the morning.

15.6.3 Vending Machine

A manufacturer would like to make machines to sell a variety of different products. The machine should be able to provide drinks in cans, drinks in cartons, sandwiches, small packets of biscuits, chocolate, chewing-gum and bags of crisps, for example. The manufacturer would like to be able to have enough products to sell during a 12-h period. You might like to consider what difference there might be for a machine selling higher value products, DVDs, for example.

15.6.4 Snowmobile

Design of a snowmobile, jet-ski or motorised scooter for one or two people.

15.6.5 Vacuum Cleaner

Design of a well-known, common product.

15.6.6 Washing Machine

Design a washing machine which can handle a maximum of 5 k of clothes.

15.6.7 Pinball Machine

Design of a pinball machine or other games machine. This should be based on a theme, for example Star Wars, Indiana Jones, or a university game with professors as hazards or to shoot at.

15.6.8 Motorized Wheelchair

Design of a motorised wheelchair for people with limited mobility. The maximum weight of the person should be 150 kg. Plan for a capacity of 6 h of autonomy without recharging. Consider the problems of different terrains and access problems, and note formally your conclusions.

Appendix A
Euler Operators

This appendix contains two lists of Euler operators for reference. The first list contains all possible Euler operators, 99 in all. The second list contains those which are not combinations of others, and was, I believe, first presented by Sue Pavey. The names are indicative only.

A.1 Euler Operators: Full List

1	(−1, −1, −1, −1, −1, −1)	kvefhgb
2	(0, 0, −1, −1, −1, −1)	kfhgb
3	(+1, +1, −1, −1, −1, −1)	mvekfhgb
4	(−1, 0, 0, −1, −1, −1)	kvhgb
5	(0, +1, 0, −1, −1, −1)	mekhgb
6	(−1, +1, +1, −1, −1, −1)	mefkvhgb
7	(0, −1, −1, 0, −1, −1)	kefgb
8	(+1, 0, −1, 0, −1, −1)	mvkfgb
9	(−1, −1, 0, 0, −1, −1)	kvegb
10	(0, 0, 0, 0, −1, −1)	kgb
11	(+1, +1, 0, 0, −1, −1)	mvekgb
12	(−1, 0, +1, 0, −1, −1)	mfkvgb
13	(0, +1, +1, 0, −1, −1)	mefkgb
14	(+1, −1, −1, +1, −1, −1)	mvhkefgb
15	(0, −1, 0, +1, −1, −1)	mhkegb
16	(+1, 0, 0, +1, −1, −1)	mvhkgb
17	(−1, −1, +1, +1, −1, −1)	mfhkvegb
18	(0, 0, +1, +1, −1, −1)	mfhkgb
19	(+1, +1, +1, +1, −1, −1)	mvefhkgb
20	(−1, +1, −1, −1, 0, −1)	mekvfhb
21	(−1, +1, +1, −1, −1, −1)	mefkvhgb
22	(0, +1, −1, 0, 0, −1)	mekfb
23	(−1, +1, 0, 0, 0, −1)	mekvb
24	(−1, −1, −1, +1, 0, −1)	mhkvefb
25	(0, 0, −1, +1, 0, −1)	mhkfb
26	(+1, +1, −1, +1, 0, −1)	mvehkfb
27	(−1, 0, 0, +1, 0, −1)	mhkvb
28	(0, +1, 0, +1, 0, −1)	mehkb
29	(−1, +1, +1, +1, 0, −1)	mefhkvb
30	(−1, +1, −1, +1, +1, −1)	mehgkvfb
31	(+1, −1, −1, −1, −1, 0)	mvkefhg
32	(0, −1, 0, −1, −1, 0)	kehg
33	(+1, 0, 0, −1, −1, 0)	mvkhg
34	(−1, −1, +1, −1, −1, 0)	mfkvehg
35	(0, 0, +1, −1, −1, 0)	mfkhg
36	(+1, +1, +1, −1, −1, 0)	mvefkhg
37	(+1, −1, 0, 0, −1, 0)	mvkeg
38	(0, −1, +1, 0, −1, 0)	mfkeg
39	(+1, 0, +1, 0, −1, 0)	mvfkg
40	(+1, −1, +1, +1, −1, 0)	mvfhkeg

41	(−1, −1, −1, −1, 0, 0)	kvefh
42	(0, 0, −1, −1, 0, 0)	kfh
43	(+1, +1, −1, −1, 0, 0)	mvekfh
44	(−1, 0, 0, −1, 0, 0)	kvh
45	(0, +1, 0, −1, 0, 0)	mekh
46	(−1, +1, +1, −1, 0, 0)	mefkvh
47	(0, −1, −1, 0, 0, 0)	kfe
48	(+1, 0, −1, 0, 0, 0)	mvkf
49	(−1, −1, 0, 0, 0, 0)	kev
50	(0, 0, 0, 0, 0, 0)	Null operator
51	(+1, +1, 0, 0, 0, 0)	mev
52	(−1, 0, +1, 0, 0, 0)	mfkv
53	(0, +1, +1, 0, 0, 0)	mfe
54	(+1, −1, −1, +1, 0, 0)	mvhkef
55	(0, −1, 0, +1, 0, 0)	mhke
56	(+1, 0, 0, +1, 0, 0)	mvh
57	(−1, −1, +1, +1, 0, 0)	mfhkve
58	(0, 0, +1, +1, 0, 0)	mfh
59	(+1, +1, +1, +1, 0, 0)	mvefh
60	(−1, +1, −1, −1, +1, 0)	megkvfh
61	(−1, 0, −1, 0, +1, 0)	mgkvf
62	(0, +1, −1, 0, +1, 0)	megkf
63	(−1, +1, 0, 0, +1, 0)	megkv
64	(−1, −1, −1, +1, +1, 0)	mhgkvef
65	(0, 0, −1, +1, +1, 0)	mhgkf
66	(+1, +1, −1, +1, +1, 0)	mvehgkf
67	(−1, 0, 0, +1, +1, 0)	mhgkv
68	(0, +1, 0, +1, +1, 0)	mehg
69	(−1, +1, +1, +1, +1, 0)	mefhgkv
70	(+1, −1, +1, −1, −1,+1)	mvfbkehg

71	(+1, −1, −1, −1, 0, +1)	mvbkefh
72	(0, −1, 0, −1, 0, +1)	mbkeh
73	(+1, 0, 0, −1, 0, +1)	mvbkh
74	(−1, −1, +1, −1, 0, +1)	mfbkveh
75	(0, 0, +1, −1, 0, +1)	mfbkh
76	(+1, +1, +1, −1, 0, +1)	mvefbkh
77	(+1, −1, 0, 0, 0, +1)	mvbke
78	(0, −1, +1, 0, 0, +1)	mfbke
79	(+1, 0, +1, 0, 0, +1)	mbfv
80	(+1, −1, +1, +1, 0, +1)	mvfhbke
81	(−1, −1, −1, −1, +1, +1)	mgbkvefh
82	(0, 0, −1, −1, +1, +1)	mgbkfh
83	(+1, +1, −1, −1, +1, +1)	mvegbkfh
84	(−1, 0, 0, −1, +1, +1)	mgbkvh
85	(0, +1, 0, −1, +1, +1)	megbkh
86	(−1, +1, +1, −1, +1, +1)	mefgbkvh
87	(0, −1, −1, 0, +1, +1)	mgbkef
88	(+1, 0, −1, 0, +1, +1)	mvgbkf
89	(−1, −1, 0, 0, +1, +1)	mgbkve
90	(0, 0, 0, 0, +1, +1)	mgb
91	(+1, +1, 0, 0, +1, +1)	mvegb
92	(−1, 0, +1, 0, +1, +1)	mfgbkv
93	(0, +1, +1, 0, +1, +1)	mefgb
94	(+1, −1, −1, +1, +1, +1)	mvhgbkef
95	(0, −1, 0, +1, +1, +1)	mhgbke
96	(+1, 0, 0, +1, +1, +1)	mvhgb
97	(−1, −1, +1, +1, +1, +1)	mfhgbkve
98	(0, 0, +1, +1, +1, +1)	mfhgb
99	(+1, +1, +1, +1, +1, +1)	mvefhgb

A.2 Euler Operators: Shortened List

1	(0, 0, 0, 0, −1, −1)	kgb
2	(−1, 0, −1, 0, 0, −1)	kvfb
3	(0, +1, −1, 0, 0, −1)	mekfb
4	(−1, +1, 0, 0, 0, −1)	mekvb
5	(0, 0, −1, +1, 0, −1)	mhkfb
6	(−1, 0, 0, +1, 0, −1)	mhkvb
7	(0, +1, 0, +1, 0, −1)	mehkb
8	(0, −1, 0, −1, −1, 0)	kehg
9	(+1, 0, 0, −1, −1, 0)	mvkhg

10	(0, 0, +1, −1, −1, 0)	mfkhg
11	(+1, −1, 0, 0, −1, 0)	mvkeg
12	(0, −1, +1, 0, −1, 0)	mfkeg
13	(+1, 0, +1, 0, −1, 0)	mvfkg
14	(0, 0, −1, −1, 0, 0)	kfh
15	(−1, 0, 0, −1, 0, 0)	kvh
16	(0, +1, 0, −1, 0, 0)	mekh
17	(0, −1, −1, 0, 0, 0)	kfe
18	(+1, 0, −1, 0, 0, 0)	mvkf

19	(−1, −1, 0, 0, 0, 0)	kev
20	(0, 0, 0, 0, 0, 0)	Null operator
21	(+1, +1, 0, 0, 0, 0)	mev
22	(−1, 0, +1, 0, 0, 0)	mfkv
23	(0, +1, +1, 0, 0, 0)	mfe
24	(0, −1, 0, +1, 0, 0)	mhke
25	(+1, 0, 0, +1, 0, 0)	mvh
26	(0, 0, +1, +1, 0, 0)	mfh
27	(−1, 0, −1, 0, +1, 0)	mgkvf
28	(0, +1, −1, 0, +1, 0)	megkf
29	(−1, +1, 0, 0, +1, 0)	megkv
30	(0, 0, −1, +1, +1, 0)	mhgkf
31	(−1, 0, 0, +1, +1, 0)	mhgkv
32	(0, +1, 0, +1, +1, 0)	mehg
33	(0, −1, 0, −1, 0, +1)	mbkeh
34	(+1, 0, 0, −1, 0, +1)	mvbkh
35	(0, 0, +1, −1, 0, +1)	mfbkh
36	(+1, −1, 0, 0, 0, +1)	mvbke
37	(0, −1, +1, 0, 0, +1)	mfbke
38	(+1, 0, +1, 0, 0, +1)	mbfv
39	(0, 0, 0, 0, +1, +1)	mgb

Appendix B
Data Exchange Format Examples

This appendix contains a few examples of data exchange files for a simple rectangular block 10 × 20 × 30, centred about the origin. The first output is in IGES format, from CATIA. The second and third files are STEP files from CATIA and Solidworks respectively. The reason for having two files is because there is a slight difference in philosophy. In CATIA the entities are grouped together, hence the line numbers are not sequential. In the Solidworks variant the line numbers are sequential and so the entities are output as they come. The final two formats are STL and VRML to show common graphical approximative formats. The two formats are simple to understand and use only planar geometry, hence they can be handled simply, as in Rapid Prototyping systems. In STL the facets are output separately, with multiple common vertices. In the VRML version shown there is a vertex list to which the face definitions refer. In this case it is easier to identify and recreate closed structures. VRML, though, can output separate facets just as easily.

B.1 IGES

The following shows the IGES format.

```
START RECORD GO HERE.                                                   S      1
1H,,1H;,20HCNEXT - IGES PRODUCT,13HBLK102030.igs,44HIBM CATIA IGES - CATG      1
IA Version 5 Release 15 ,27HCATIA Version 5 Release 15 ,32,75,6,75,15,5HG      2
PART4,1.0,2,2HMM,1000,1.0,15H20071114.070402,0.001,10000.0,6Hstroud,3HSTG      3
I,11,0,15H20071114.070402,;                                             G      4
     108       1       0       0       0       0       0       001010001D      1
     108       0       0       1       0                               0D      2
     110       2       0       0       0       0       0       001010001D      3
     110       0       0       1       0                               0D      4
     110       3       0       0       0       0       0       001010001D      5
     110       0       0       1       0                               0D      6
     110       4       0       0       0       0       0       001010001D      7
     110       0       0       1       0                               0D      8
     110       5       0       0       0       0       0       001010001D      9
     110       0       0       1       0                               0D     10
     102       6       0       0       0       0       0       001010001D     11
     102       0       0       1       0                               0D     12
     142       7       0       0       0       0       0       001010001D     13
     142       0       0       1       0                               0D     14
     144       8       0       0   10000       0       0       000000000D     15
     144       0       0       1       0                               0D     16
     108       9       0       0       0       0       0       001010001D     17
     108       0       0       1       0                               0D     18
     110      10       0       0       0       0       0       001010001D     19
     110       0       0       1       0                               0D     20
     110      11       0       0       0       0       0       001010001D     21
     110       0       0       1       0                               0D     22
     110      12       0       0       0       0       0       001010001D     23
     110       0       0       1       0                               0D     24
     110      13       0       0       0       0       0       001010001D     25
     110       0       0       1       0                               0D     26
     102      14       0       0       0       0       0       001010001D     27
     102       0       0       1       0                               0D     28
     142      15       0       0       0       0       0       001010001D     29
     142       0       0       1       0                               0D     30
     144      16       0       0   10000       0       0       000000000D     31
     144       0       0       1       0                               0D     32
     108      17       0       0       0       0       0       001010001D     33
     108       0       0       1       0                               0D     34
     110      18       0       0       0       0       0       001010001D     35
     110       0       0       1       0                               0D     36
     110      19       0       0       0       0       0       001010001D     37
     110       0       0       1       0                               0D     38
     110      20       0       0       0       0       0       001010001D     39
     110       0       0       1       0                               0D     40
     110      21       0       0       0       0       0       001010001D     41
     110       0       0       1       0                               0D     42
     102      22       0       0       0       0       0       001010001D     43
     102       0       0       1       0                               0D     44
     142      23       0       0       0       0       0       001010001D     45
     142       0       0       1       0                               0D     46
     144      24       0       0   10000       0       0       000000000D     47
     144       0       0       1       0                               0D     48
     108      25       0       0       0       0       0       001010001D     49
     108       0       0       1       0                               0D     50
     110      26       0       0       0       0       0       001010001D     51
     110       0       0       1       0                               0D     52
     110      27       0       0       0       0       0       001010001D     53
     110       0       0       1       0                               0D     54
     110      28       0       0       0       0       0       001010001D     55
     110       0       0       1       0                               0D     56
     110      29       0       0       0       0       0       001010001D     57
     110       0       0       1       0                               0D     58
     102      30       0       0       0       0       0       001010001D     59
     102       0       0       1       0                               0D     60
     142      31       0       0       0       0       0       001010001D     61
     142       0       0       1       0                               0D     62
     144      32       0       0   10000       0       0       000000000D     63
     144       0       0       1       0                               0D     64
     108      33       0       0       0       0       0       001010001D     65
     108       0       0       1       0                               0D     66
     110      34       0       0       0       0       0       001010001D     67
     110       0       0       1       0                               0D     68
     110      35       0       0       0       0       0       001010001D     69
     110       0       0       1       0                               0D     70
     110      36       0       0       0       0       0       001010001D     71
     110       0       0       1       0                               0D     72
     110      37       0       0       0       0       0       001010001D     73
     110       0       0       1       0                               0D     74
     102      38       0       0       0       0       0       001010001D     75
     102       0       0       1       0                               0D     76
     142      39       0       0       0       0       0       001010001D     77
     142       0       0       1       0                               0D     78
     144      40       0       0   10000       0       0       000000000D     79
     144       0       0       1       0                               0D     80
     108      41       0       0       0       0       0       001010001D     81
     108       0       0       1       0                               0D     82
     110      42       0       0       0       0       0       001010001D     83
     110       0       0       1       0                               0D     84
     110      43       0       0       0       0       0       001010001D     85
     110       0       0       1       0                               0D     86
     110      44       0       0       0       0       0       001010001D     87
     110       0       0       1       0                               0D     88
     110      45       0       0       0       0       0       001010001D     89
     110       0       0       1       0                               0D     90
     102      46       0       0       0       0       0       001010001D     91
     102       0       0       1       0                               0D     92
     142      47       0       0       0       0       0       001010001D     93
     142       0       0       1       0                               0D     94
     144      48       0       0   10000       0       0       000000000D     95
     144       0       0       1       0                               0D     96
108,-1.0,0.0,0.0,5.0,0,-5.0,10.0,0.0,1.0,0,0;                          1P      1
110,-5.0,10.0,15.0,-5.0,-10.0,15.0,0,0;                                3P      2
110,-5.0,-10.0,15.0,-5.0,-10.0,-15.0,0,0;                              5P      3
110,-5.0,-10.0,-15.0,-5.0,10.0,-15.0,0,0;                              7P      4
110,-5.0,10.0,-15.0,-5.0,10.0,15.0,0,0;                                9P      5
102,4,3,5,7,9,0,0;                                                    11P      6
142,0,1,0,11,2,0,0;                                                   13P      7
144,1,1,0,13,0,0;                                                     15P      8
108,0.0,1.0,0.0,-10.0,0,5.0,-10.0,0.0,1.0,0,0;                        17P      9
110,-5.0,-10.0,15.0,5.0,-10.0,15.0,0,0;                               19P     10
110,5.0,-10.0,15.0,5.0,-10.0,-15.0,0,0;                               21P     11
110,5.0,-10.0,-15.0,-5.0,-10.0,-15.0,0,0;                             23P     12
110,-5.0,-10.0,-15.0,-5.0,-10.0,15.0,0,0;                             25P     13
102,4,19,21,23,25,0,0;                                                27P     14
142,0,17,0,27,2,0,0;                                                  29P     15
144,17,1,0,29,0,0;                                                    31P     16
108,1.0,0.0,0.0,5.0,0,5.0,-10.0,0.0,1.0,0,0;                          33P     17
110,5.0,-10.0,15.0,5.0,10.0,15.0,0,0;                                 35P     18
110,5.0,10.0,15.0,5.0,10.0,-15.0,0,0;                                 37P     19
110,5.0,10.0,-15.0,5.0,-10.0,-15.0,0,0;                               39P     20
110,5.0,-10.0,-15.0,5.0,-10.0,15.0,0,0;                               41P     21
102,4,35,37,39,41,0,0;                                                43P     22
142,0,33,0,43,2,0,0;                                                  45P     23
144,33,1,0,45,0,0;                                                    47P     24
108,0.0,-1.0,0.0,-10.0,0,-5.0,10.0,0.0,1.0,0,0;                       49P     25
110,5.0,10.0,15.0,-5.0,10.0,15.0,0,0;                                 51P     26
110,-5.0,10.0,15.0,-5.0,10.0,-15.0,0,0;                               53P     27
```

```
110,-5.0,10.0,-15.0,5.0,10.0,-15.0,0,0;                                   55P     28
110,5.0,10.0,-15.0,5.0,10.0,15.0,0,0;                                     57P     29
102,4,51,53,55,57,0,0;                                                    59P     30
142,0,49,0,59,2,0,0;                                                      61P     31
144,49,1,0,61,0,0;                                                        63P     32
108,0.0,0.0,1.0,-15.0,0,0.0,0.0,-15.0,1.0,0,0;                            65P     33
110,-5.0,10.0,-15.0,-5.0,-10.0,-15.0,0,0;                                 67P     34
110,-5.0,-10.0,-15.0,5.0,-10.0,-15.0,0,0;                                 69P     35
110,5.0,-10.0,-15.0,5.0,10.0,-15.0,0,0;                                   71P     36
110,5.0,10.0,-15.0,-5.0,10.0,-15.0,0,0;                                   73P     37
102,4,67,69,71,73,0,0;                                                    75P     38
142,0,65,0,75,2,0,0;                                                      77P     39
144,65,1,0,77,0,0;                                                        79P     40
108,0.0,0.0,1.0,15.0,0,0.0,0.0,15.0,1.0,0,0;                              81P     41
110,-5.0,10.0,15.0,5.0,10.0,15.0,0,0;                                     83P     42
110,5.0,10.0,15.0,5.0,-10.0,15.0,0,0;                                     85P     43
110,5.0,-10.0,15.0,-5.0,-10.0,15.0,0,0;                                   87P     44
110,-5.0,-10.0,15.0,-5.0,10.0,15.0,0,0;                                   89P     45
102,4,83,85,87,89,0,0;                                                    91P     46
142,0,81,0,91,2,0,0;                                                      93P     47
144,81,1,0,93,0,0;                                                        95P     48
S      1G      4D     96P     48                                            T      1
```

B.2 VDA-FS

This is a data exchange method developed by the German automobile industry for exchanging complex geometry.

```
SLDWORKS = HEADER / 20                                                  00000001
************************************************************************00000002
VDAFS VERSION     :  2.0                                                00000003
----------------------------- SENDER DATA ------------------------------00000004
SENDER COMPANY    :   EPFL                                              00000005
CONTACT PERSON    :  Ian Stroud                                         00000006
TELEPHONE NO.     :  UNKNOWN                                            00000007
ADDRESS           :  UNKNOWN                                            00000008
SENDING SYSTEM    :  SolidWorks-2007078                                 00000009
SENDING DATE      :  11/15/2007                                         00000010
FILE NAME         :  blk102030vda.vda                                   00000011
------------------------------ PART DATA -------------------------------00000012
PROJECT NAME      :  UNKNOWN                                            00000013
OBJECT CODE       :  NONE                                               00000014
VARIANT           :  NONE                                               00000015
CONFIDENTIALITY   :  UNCLASSIFIED                                       00000016
DATE EFFECTIVE    :  11/15/2007                                         00000017
---------------------------- RECEIVER DATA -----------------------------00000018
COMPANY NAME      :  UNKNOWN                                            00000019
RECEIVER NAME     :  UNKNOWN                                            00000020
************************************************************************00000021
CV1 = CURVE / 1, 0.0000000000000000E+00, 2.0000000000000000E-02,        00000022
  2,                                                                    00000023
    -5.0000000000000000E+00, 3.4694469519536142E-15,                    00000024
    1.0000000000000000E+01, -2.0000000000000000E+01,                    00000025
    1.5000000000000000E+01, 0.0000000000000000E+00                      00000026
CV2 = CURVE / 1, 0.0000000000000000E+00, 2.0000000000000000E-02,        00000027
  2,                                                                    00000028
    -5.0000000000000000E+00, 3.4694469519536142E-15,                    00000029
    1.0000000000000000E+01, -2.0000000000000000E+01,                    00000030
    -1.5000000000000000E+01, 0.0000000000000000E+00                     00000031
CV3 = CURVE / 1, 0.0000000000000000E+00, 2.9999999999999999E-02,        00000032
  2,                                                                    00000033
    -4.9999999999999964E+00, 0.0000000000000000E+00,                    00000034
    -1.0000000000000000E+01, 0.0000000000000000E+00,                    00000035
    1.5000000000000000E+01, -3.0000000000000000E+01                     00000036
CV4 = CURVE / 1, -2.0000000000000000E-02, 0.0000000000000000E+00,       00000037
  2,                                                                    00000038
    5.0000000000000000E+00, 0.0000000000000000E+00,                     00000039
    -1.0000000000000000E+01, 2.0000000000000000E+01,                    00000040
    -1.5000000000000000E+01, 0.0000000000000000E+00                     00000041
CV5 = CURVE / 1, 0.0000000000000000E+00, 2.9999999999999999E-02,        00000042
  2,                                                                    00000043
    -5.0000000000000000E+00, 0.0000000000000000E+00,                    00000044
    1.0000000000000000E+01, 0.0000000000000000E+00,                     00000045
    1.5000000000000000E+01, -3.0000000000000000E+01                     00000046
CV6 = CURVE / 1, 0.0000000000000000E+00, 2.9999999999999999E-02,        00000047
  2,                                                                    00000048
    5.0000000000000000E+00, 0.0000000000000000E+00,                     00000049
    -1.0000000000000000E+01, 0.0000000000000000E+00,                    00000050
    1.5000000000000000E+01, -3.0000000000000000E+01                     00000051
CV7 = CURVE / 1, -1.0000000000000000E-02, 0.0000000000000000E+00,       00000052
  2,                                                                    00000053
    5.0000000000000000E+00, -1.0000000000000000E+01,                    00000054
    1.0000000000000000E+01, 0.0000000000000000E+00,                     00000055
    -1.5000000000000000E+01, 0.0000000000000000E+00                     00000056
CV8 = CURVE / 1, -2.0000000000000000E-02, 0.0000000000000000E+00,       00000057
  2,                                                                    00000058
    5.0000000000000000E+00, 0.0000000000000000E+00,                     00000059
    -1.0000000000000000E+01, 2.0000000000000000E+01,                    00000060
    1.5000000000000000E+01, 0.0000000000000000E+00                      00000061
CV9 = CURVE / 1, 0.0000000000000000E+00, 9.9999999999999967E-03,        00000062
  2,                                                                    00000063
    -4.9999999999999964E+00, 9.9999999999999964E+00,                    00000064
    -1.0000000000000000E+01, 0.0000000000000000E+00,                   00000065
    -1.5000000000000000E+01, 0.0000000000000000E+00                    00000066
CV10 = CURVE / 1, 0.0000000000000000E+00, 9.9999999999999967E-03,      00000067
  2,                                                                   00000068
    -4.9999999999999964E+00, 9.9999999999999964E+00,                   00000069
    -1.0000000000000000E+01, 0.0000000000000000E+00,                   00000070
    1.5000000000000000E+01, 0.0000000000000000E+00                     00000071
CV11 = CURVE / 1, 0.0000000000000000E+00, 2.9999999999999999E-02,      00000072
  2,                                                                   00000073
    5.0000000000000000E+00, 0.0000000000000000E+00,                    00000074
    1.0000000000000000E+01, 0.0000000000000000E+00,                    00000075
    1.5000000000000000E+01, -3.0000000000000000E+01                    00000076
CV12 = CURVE / 1, -1.0000000000000000E-02, 0.0000000000000000E+00,     00000077
  2,                                                                   00000078
    5.0000000000000000E+00, -1.0000000000000000E+01,                   00000079
    1.0000000000000000E+01, 0.0000000000000000E+00,                    00000080
    1.5000000000000000E+01, 0.0000000000000000E+00                     00000081
SR13 = SURF / 1, 1, 0.0000000000000000E+00, 1.0000000000000000E+00,   00000082
 0.0000000000000000E+00, 1.0000000000000000E+00,                       00000083
4, 4, -4.9999999999999964E+00, -2.6020852139652106E-15,                00000084
 -2.6020852139652106E-15, 2.6020852139652106E-15,                      00000085
 0.0000000000000000E+00, 0.0000000000000000E+00,                       00000086
 0.0000000000000000E+00, 0.0000000000000000E+00,                       00000087
 0.0000000000000000E+00, 0.0000000000000000E+00,                       00000088
 0.0000000000000000E+00, 0.0000000000000000E+00,                       00000089
 8.6736173798840355E-16, 0.0000000000000000E+00,                       00000090
 0.0000000000000000E+00, 0.0000000000000000E+00,                       00000091
 -1.0000000000000000E+01, 1.9999999999999996E+01,                      00000092
 0.0000000000000000E+00, -1.7347234759768071E-15,                      00000093
 0.0000000000000000E+00, 0.0000000000000000E+00,                       00000094
 0.0000000000000000E+00, 0.0000000000000000E+00,                       00000095
 0.0000000000000000E+00, 0.0000000000000000E+00,                       00000096
 0.0000000000000000E+00, 0.0000000000000000E+00,                       00000097
 1.7347234759768071E-15, 0.0000000000000000E+00,                       00000098
 0.0000000000000000E+00, 0.0000000000000000E+00,                       00000099
 -1.5000000000000000E+01, 0.0000000000000000E+00,                      00000100
 0.0000000000000000E+00, 0.0000000000000000E+00,                       00000101
 2.9999999999999996E+01, 0.0000000000000000E+00,                       00000102
 0.0000000000000000E+00, 0.0000000000000000E+00,                       00000103
 0.0000000000000000E+00, 0.0000000000000000E+00,                       00000104
 0.0000000000000000E+00, 0.0000000000000000E+00,                       00000105
 0.0000000000000000E+00, 0.0000000000000000E+00,                       00000106
 0.0000000000000000E+00, 0.0000000000000000E+00                        00000107
SR14 = SURF / 1, 1, 0.0000000000000000E+00, 1.0000000000000000E+00,   00000108
 0.0000000000000000E+00, 1.0000000000000000E+00,                       00000109
4, 4, -4.9999999999999964E+00, 0.0000000000000000E+00,                 00000110
 0.0000000000000000E+00, 8.6736173798840355E-16,                       00000111
 9.9999999999999964E+00, 0.0000000000000000E+00,                       00000112
 0.0000000000000000E+00, -2.6020852139652106E-15,                      00000113
 0.0000000000000000E+00, 0.0000000000000000E+00,                       00000114
 0.0000000000000000E+00, 2.6020852139652106E-15,                       00000115
 0.0000000000000000E+00, 0.0000000000000000E+00,                       00000116
 0.0000000000000000E+00, -1.7347234759768071E-15,                      00000117
 -1.0000000000000000E+01, 0.0000000000000000E+00,                      00000118
 0.0000000000000000E+00, 1.7347234759768071E-15,                       00000119
 0.0000000000000000E+00, 0.0000000000000000E+00,                       00000120
 0.0000000000000000E+00, 0.0000000000000000E+00,                       00000121
 0.0000000000000000E+00, 0.0000000000000000E+00,                       00000122
 0.0000000000000000E+00, 0.0000000000000000E+00,                       00000123
 1.7347234759768071E-15, 0.0000000000000000E+00,                       00000124
 0.0000000000000000E+00, 0.0000000000000000E+00,                       00000125
 -1.5000000000000000E+01, 2.9999999999999996E+01,                      00000126
 0.0000000000000000E+00, 0.0000000000000000E+00,                       00000127
 0.0000000000000000E+00, 0.0000000000000000E+00,                       00000128
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000129
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000130
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000131
 0.0000000000000000E+00, -3.4694469519536142E-15,                       00000132
 0.0000000000000000E+00, 0.0000000000000000E+00                         00000133
SR15 = SURF / 1, 1, 0.0000000000000000E+00, 1.0000000000000000E+00,     00000134
 0.0000000000000000E+00, 1.0000000000000000E+00,                        00000135
4, 4, -5.0000000000000000E+00, 0.0000000000000000E+00,                  00000136
 0.0000000000000000E+00, 8.6736173798840355E-16,                        00000137
 9.9999999999999982E+00, 0.0000000000000000E+00,                        00000138
 0.0000000000000000E+00, -2.6020852139652106E-15,                       00000139
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000140
 0.0000000000000000E+00, 2.6020852139652106E-15,                        00000141
 -8.6736173798840355E-16, 0.0000000000000000E+00,                       00000142
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000143
 1.0000000000000000E+01, 0.0000000000000000E+00,                        00000144
 0.0000000000000000E+00, -1.7347234759768071E-15,                       00000145
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000146
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000147
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000148
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000149
 -1.7347234759768071E-15, 0.0000000000000000E+00,                       00000150
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000151
 1.5000000000000000E+01, -2.9999999999999996E+01,                       00000152
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000153
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000154
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000155
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000156
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000157
 0.0000000000000000E+00, 3.4694469519536142E-15,                        00000158
 0.0000000000000000E+00, 0.0000000000000000E+00                         00000159
SR16 = SURF / 1, 1, 0.0000000000000000E+00, 1.0000000000000000E+00,     00000160
 0.0000000000000000E+00, 1.0000000000000000E+00,                        00000161
4, 4, -5.0000000000000000E+00, 9.9999999999999982E+00,                  00000162
 0.0000000000000000E+00, -8.6736173798840355E-16,                       00000163
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000164
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000165
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000166
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000167
 8.6736173798840355E-16, 0.0000000000000000E+00,                        00000168
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000169
 -1.0000000000000000E+01, 0.0000000000000000E+00,                       00000170
 0.0000000000000000E+00, 1.7347234759768071E-15,                        00000171
 1.9999999999999996E+01, 0.0000000000000000E+00,                        00000172
 0.0000000000000000E+00, -5.2041704279304213E-15,                       00000173
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000174
 0.0000000000000000E+00, 5.2041704279304213E-15,                        00000175
 -1.7347234759768071E-15, 0.0000000000000000E+00,                       00000176
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000177
 1.5000000000000000E+01, 0.0000000000000000E+00,                        00000178
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000179
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000180
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000181
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000182
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000183
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000184
 0.0000000000000000E+00, 0.0000000000000000E+00                         00000185
SR17 = SURF / 1, 1, 0.0000000000000000E+00, 1.0000000000000000E+00,     00000186
 0.0000000000000000E+00, 1.0000000000000000E+00,                        00000187
4, 4, -5.0000000000000000E+00, 9.9999999999999982E+00,                  00000188
 0.0000000000000000E+00, -8.6736173798840355E-16,                       00000189
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000190
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000191
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000192
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000193
 8.6736173798840355E-16, 0.0000000000000000E+00,                        00000194
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000195
 -1.0000000000000000E+01, 0.0000000000000000E+00,                       00000196
 0.0000000000000000E+00, 1.7347234759768071E-15,                        00000197
 1.9999999999999996E+01, 0.0000000000000000E+00,                        00000198
 0.0000000000000000E+00, -5.2041704279304213E-15,                       00000199
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000200
 0.0000000000000000E+00, 5.2041704279304213E-15,                        00000201
 -1.7347234759768071E-15, 0.0000000000000000E+00,                       00000202
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000203
 -1.5000000000000000E+01, 0.0000000000000000E+00,                       00000204
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000205
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000206
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000207
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000208
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000209
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000210
 0.0000000000000000E+00, 0.0000000000000000E+00                         00000211
SR18 = SURF / 1, 1, 0.0000000000000000E+00, 1.0000000000000000E+00,     00000212
 0.0000000000000000E+00, 1.0000000000000000E+00,                        00000213
4, 4, 5.0000000000000000E+00, 0.0000000000000000E+00,                   00000214
 0.0000000000000000E+00, -8.6736173798840355E-16,                       00000215
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000216
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000217
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000218
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000219
 -8.6736173798840355E-16, 0.0000000000000000E+00,                       00000220
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000221
 -1.0000000000000000E+01, 0.0000000000000000E+00,                       00000222
 0.0000000000000000E+00, 1.7347234759768071E-15,                        00000223
 1.9999999999999996E+01, 0.0000000000000000E+00,                        00000224
 0.0000000000000000E+00, -5.2041704279304213E-15,                       00000225
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000226
 0.0000000000000000E+00, 5.2041704279304213E-15,                        00000227
 -1.7347234759768071E-15, 0.0000000000000000E+00,                       00000228
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000229
 -1.5000000000000000E+01, 2.9999999999999996E+01,                       00000230
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000231
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000232
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000233
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000234
 0.0000000000000000E+00, 0.0000000000000000E+00,                        00000235
 0.0000000000000000E+00, -3.4694469519536142E-15,                       00000236
 0.0000000000000000E+00, 0.0000000000000000E+00                         00000237
CN19 = CONS / SR13, CV1, 2.0000000000000000E-02,                        00000238
 0.0000000000000000E+00,                                                00000239
1,                                                                      00000240
-2.0000000000000000E-02, 0.0000000000000000E+00,                        00000241
  2,                                                                    00000242
    0.0000000000000000E+00, 1.0000000000000000E+00,                     00000243
    1.0000000000000000E+00, 0.0000000000000000E+00                      00000244
CN20 = CONS / SR13, CV3, 2.9999999999999999E-02,                        00000245
 0.0000000000000000E+00,                                                00000246
1,                                                                      00000247
-2.9999999999999999E-02, 0.0000000000000000E+00,                        00000248
  2,                                                                    00000249
    0.0000000000000000E+00, 0.0000000000000000E+00,                     00000250
    0.0000000000000000E+00, 1.0000000000000000E+00                      00000251
CN21 = CONS / SR13, CV2, 0.0000000000000000E+00,                        00000252
 2.0000000000000000E-02,                                                00000253
1,                                                                      00000254
0.0000000000000000E+00, 2.0000000000000000E-02,                         00000255
  2,                                                                    00000256
```

```
    1.0000000000000000E+00, -1.0000000000000000E+00,                    00000257
    0.0000000000000000E+00, 0.0000000000000000E+00                      00000258
CN22 = CONS / SR13, CV5, 0.0000000000000000E+00,                        00000259
 2.9999999999999999E-02,                                                00000260
1,                                                                      00000261
0.0000000000000000E+00, 2.9999999999999999E-02,                         00000262
  2,                                                                    00000263
    1.0000000000000000E+00, 0.0000000000000000E+00,                     00000264
    1.0000000000000000E+00, -1.0000000000000000E+00                     00000265
CN23 = CONS / SR18, CV4, -2.0000000000000000E-02,                       00000266
 0.0000000000000000E+00,                                                00000267
1,                                                                      00000268
-2.0000000000000000E-02, 0.0000000000000000E+00,                        00000269
  2,                                                                    00000270
    0.0000000000000000E+00, 0.0000000000000000E+00,                     00000271
    0.0000000000000000E+00, 1.0000000000000000E+00                      00000272
CN24 = CONS / SR18, CV6, 0.0000000000000000E+00,                        00000273
 2.9999999999999999E-02,                                                00000274
1,                                                                      00000275
0.0000000000000000E+00, 2.9999999999999999E-02,                         00000276
  2,                                                                    00000277
    1.0000000000000000E+00, -1.0000000000000000E+00,                    00000278
    0.0000000000000000E+00, 0.0000000000000000E+00                      00000279
CN25 = CONS / SR14, CV10, 9.9999999999999967E-03,                       00000280
 0.0000000000000000E+00,                                                00000281
1,                                                                      00000282
-9.9999999999999967E-03, 0.0000000000000000E+00,                        00000283
  2,                                                                    00000284
    1.0000000000000000E+00, 0.0000000000000000E+00,                     00000285
    1.0000000000000000E+00, -1.0000000000000000E+00                     00000286
CN26 = CONS / SR14, CV6, 2.9999999999999999E-02,                        00000287
 0.0000000000000000E+00,                                                00000288
1,                                                                      00000289
-2.9999999999999999E-02, 0.0000000000000000E+00,                        00000290
  2,                                                                    00000291
    0.0000000000000000E+00, 1.0000000000000000E+00,                     00000292
    1.0000000000000000E+00, 0.0000000000000000E+00                      00000293
CN27 = CONS / SR14, CV9, 0.0000000000000000E+00,                        00000294
 9.9999999999999967E-03,                                                00000295
1,                                                                      00000296
0.0000000000000000E+00, 9.9999999999999967E-03,                         00000297
  2,                                                                    00000298
    0.0000000000000000E+00, 0.0000000000000000E+00,                     00000299
    0.0000000000000000E+00, 1.0000000000000000E+00                      00000300
CN28 = CONS / SR14, CV3, 0.0000000000000000E+00,                        00000301
 2.9999999999999999E-02,                                                00000302
1,                                                                      00000303
0.0000000000000000E+00, 2.9999999999999999E-02,                         00000304
  2,                                                                    00000305
    1.0000000000000000E+00, -1.0000000000000000E+00,                    00000306
    0.0000000000000000E+00, 0.0000000000000000E+00                      00000307
CN29 = CONS / SR15, CV12, 0.0000000000000000E+00,                       00000308
 -1.0000000000000000E-02,                                               00000309
1,                                                                      00000310
0.0000000000000000E+00, 1.0000000000000000E-02,                         00000311
  2,                                                                    00000312
    0.0000000000000000E+00, 0.0000000000000000E+00,                     00000313
    0.0000000000000000E+00, 1.0000000000000000E+00                      00000314
CN30 = CONS / SR15, CV5, 2.9999999999999999E-02,                        00000315
 0.0000000000000000E+00,                                                00000316
1,                                                                      00000317
-2.9999999999999999E-02, 0.0000000000000000E+00,                        00000318
  2,                                                                    00000319
    1.0000000000000000E+00, -1.0000000000000000E+00,                    00000320
    0.0000000000000000E+00, 0.0000000000000000E+00                      00000321
CN31 = CONS / SR15, CV7, -1.0000000000000000E-02,                       00000322
 0.0000000000000000E+00,                                                00000323
1,                                                                      00000324
-1.0000000000000000E-02, 0.0000000000000000E+00,                        00000325
  2,                                                                    00000326
    1.0000000000000000E+00, 0.0000000000000000E+00,                     00000327
    1.0000000000000000E+00, -1.0000000000000000E+00                     00000328
CN32 = CONS / SR15, CV11, 0.0000000000000000E+00,                       00000329
 2.9999999999999999E-02,                                                00000330
1,                                                                      00000331
0.0000000000000000E+00, 2.9999999999999999E-02,                         00000332
  2,                                                                    00000333
    0.0000000000000000E+00, 1.0000000000000000E+00,                     00000334
    1.0000000000000000E+00, 0.0000000000000000E+00                      00000335
CN33 = CONS / SR18, CV8, 0.0000000000000000E+00,                        00000336
 -2.0000000000000000E-02,                                               00000337
1,                                                                      00000338
0.0000000000000000E+00, 2.0000000000000000E-02,                         00000339
  2,                                                                    00000340
    1.0000000000000000E+00, 0.0000000000000000E+00,                     00000341
    1.0000000000000000E+00, -1.0000000000000000E+00                     00000342
CN34 = CONS / SR18, CV11, 2.9999999999999999E-02,                       00000343
 0.0000000000000000E+00,                                                00000344
1,                                                                      00000345
-2.9999999999999999E-02, 0.0000000000000000E+00,                        00000346
  2,                                                                    00000347
    0.0000000000000000E+00, 1.0000000000000000E+00,                     00000348
    1.0000000000000000E+00, 0.0000000000000000E+00                      00000349
CN35 = CONS / SR17, CV9, 9.9999999999999967E-03,                        00000350
 0.0000000000000000E+00,                                                00000351
1,                                                                      00000352
-9.9999999999999967E-03, 0.0000000000000000E+00,                        00000353
  2,                                                                    00000354
    1.0000000000000000E+00, -1.0000000000000000E+00,                    00000355
    0.0000000000000000E+00, 0.0000000000000000E+00                      00000356
CN36 = CONS / SR17, CV4, 0.0000000000000000E+00,                        00000357
 -2.0000000000000000E-02,                                               00000358
1,                                                                      00000359
0.0000000000000000E+00, 2.0000000000000000E-02,                         00000360
  2,                                                                    00000361
    1.0000000000000000E+00, 0.0000000000000000E+00,                     00000362
    1.0000000000000000E+00, -1.0000000000000000E+00                     00000363
CN37 = CONS / SR17, CV7, 0.0000000000000000E+00,                        00000364
 -1.0000000000000000E-02,                                               00000365
1,                                                                      00000366
0.0000000000000000E+00, 1.0000000000000000E-02,                         00000367
  2,                                                                    00000368
    0.0000000000000000E+00, 1.0000000000000000E+00,                     00000369
    1.0000000000000000E+00, 0.0000000000000000E+00                      00000370
CN38 = CONS / SR17, CV2, 2.0000000000000000E-02,                        00000371
 0.0000000000000000E+00,                                                00000372
1,                                                                      00000373
-2.0000000000000000E-02, 0.0000000000000000E+00,                        00000374
  2,                                                                    00000375
    0.0000000000000000E+00, 0.0000000000000000E+00,                     00000376
    0.0000000000000000E+00, 1.0000000000000000E+00                      00000377
CN39 = CONS / SR16, CV12, -1.0000000000000000E-02,                      00000378
 0.0000000000000000E+00,                                                00000379
1,                                                                      00000380
-1.0000000000000000E-02, 0.0000000000000000E+00,                        00000381
  2,                                                                    00000382
    1.0000000000000000E+00, -1.0000000000000000E+00,                    00000383
    1.0000000000000000E+00, 0.0000000000000000E+00                      00000384
```

```
CN40 = CONS / SR16, CV8, -2.0000000000000000E-02,                       00000385
 0.0000000000000000E+00,                                                00000386
1,                                                                      00000387
-2.0000000000000000E-02, 0.0000000000000000E+00,                        00000388
  2,                                                                    00000389
    1.0000000000000000E+00, 0.0000000000000000E+00,                     00000390
    0.0000000000000000E+00, 1.0000000000000000E+00                      00000391
CN41 = CONS / SR16, CV1, 0.0000000000000000E+00,                        00000392
 2.0000000000000000E-02,                                                00000393
1,                                                                      00000394
0.0000000000000000E+00, 2.0000000000000000E-02,                         00000395
  2,                                                                    00000396
    0.0000000000000000E+00, 0.0000000000000000E+00,                     00000397
    1.0000000000000000E+00, -1.0000000000000000E+00                     00000398
CN42 = CONS / SR16, CV10, 0.0000000000000000E+00,                       00000399
 9.9999999999999967E-03,                                                00000400
1,                                                                      00000401
0.0000000000000000E+00, 9.9999999999999967E-03,                         00000402
  2,                                                                    00000403
    0.0000000000000000E+00, 1.0000000000000000E+00,                     00000404
    0.0000000000000000E+00, 0.0000000000000000E+00                      00000405
FC43 = FACE / SR16, 1,                                                  00000406
  4,                                                                    00000407
    CN41, 0.0000000000000000E+00, 2.0000000000000000E-02,               00000408
    CN42, 0.0000000000000000E+00, 9.9999999999999967E-03,               00000409
    CN40, -2.0000000000000000E-02, 0.0000000000000000E+00,              00000410
    CN39, -1.0000000000000000E-02, 0.0000000000000000E+00               00000411
FC44 = FACE / SR17, 1,                                                  00000412
  4,                                                                    00000413
    CN38, -2.0000000000000000E-02, 0.0000000000000000E+00,              00000414
    CN37, 0.0000000000000000E+00, 1.0000000000000000E-02,               00000415
    CN36, 0.0000000000000000E+00, 2.0000000000000000E-02,               00000416
    CN35, -9.9999999999999967E-03, 0.0000000000000000E+00               00000417
FC45 = FACE / SR15, 1,                                                  00000418
  4,                                                                    00000419
    CN32, 0.0000000000000000E+00, 2.9999999999999999E-02,               00000420
    CN31, -1.0000000000000000E-02, 0.0000000000000000E+00,              00000421
    CN30, -2.9999999999999999E-02, 0.0000000000000000E+00,              00000422
    CN29, 0.0000000000000000E+00, 1.0000000000000000E-02                00000423
FC46 = FACE / SR18, 1,                                                  00000424
  4,                                                                    00000425
    CN24, 0.0000000000000000E+00, 2.9999999999999999E-02,               00000426
    CN23, -2.0000000000000000E-02, 0.0000000000000000E+00,              00000427
    CN34, -2.9999999999999999E-02, 0.0000000000000000E+00,              00000428
    CN33, 0.0000000000000000E+00, 2.0000000000000000E-02                00000429
FC47 = FACE / SR14, 1,                                                  00000430
  4,                                                                    00000431
    CN28, 0.0000000000000000E+00, 2.9999999999999999E-02,               00000432
    CN27, 0.0000000000000000E+00, 9.9999999999999967E-03,               00000433
    CN26, -2.9999999999999999E-02, 0.0000000000000000E+00,              00000434
    CN25, -9.9999999999999967E-03, 0.0000000000000000E+00               00000435
FC48 = FACE / SR13, 1,                                                  00000436
  4,                                                                    00000437
    CN22, 0.0000000000000000E+00, 2.9999999999999999E-02,               00000438
    CN21, 0.0000000000000000E+00, 2.0000000000000000E-02,               00000439
    CN20, -2.9999999999999999E-02, 0.0000000000000000E+00,              00000440
    CN19, -2.0000000000000000E-02, 0.0000000000000000E+00               00000441
SLDWORKS = END                                                          00000442
```

B.3 STEP: CATIA

In this STEP variant, output from CATIA, the elements are grouped together.

```
ISO-10303-21;
HEADER;
FILE_DESCRIPTION(('CATIA V5 STEP Exchange'),'2;1');

FILE_NAME('\\\\Icap2ksrv\\licp\\LICP-staff\\stroud\\doxtrans\\book2
\\BLK102030.stp','2007-11-14T06:04:41+00:00',('none'),('none'),'CATIA Version 5
Release 15 (IN-10)','CATIA V5 STEP AP203','none');

FILE_SCHEMA(('CONFIG_CONTROL_DESIGN'));

ENDSEC;
/* file written by CATIA V5R15 */
DATA;
#5=PRODUCT('PART4','','',(#2)) ;
#1=APPLICATION_CONTEXT('configuration controlled 3D design of mechanical parts
and assemblies') ;
#14=PRODUCT_DEFINITION(' ',' ',#6,#3) ;
#16=SECURITY_CLASSIFICATION(' ',' ',#15) ;
#15=SECURITY_CLASSIFICATION_LEVEL('unclassified') ;
#47=CARTESIAN_POINT(' ',(0.,0.,0.)) ;
#52=CARTESIAN_POINT('Axis2P3D Location',(-5.,10.,0.)) ;
#57=CARTESIAN_POINT('Line Origine',(-5.,10.,0.)) ;
#61=CARTESIAN_POINT('Vertex',(-5.,10.,-15.)) ;
#63=CARTESIAN_POINT('Vertex',(-5.,10.,15.)) ;
#66=CARTESIAN_POINT('Line Origine',(-5.,0.,-15.)) ;
#70=CARTESIAN_POINT('Vertex',(-5.,-10.,-15.)) ;
#73=CARTESIAN_POINT('Line Origine',(-5.,-10.,0.)) ;
#77=CARTESIAN_POINT('Vertex',(-5.,-10.,15.)) ;
#80=CARTESIAN_POINT('Line Origine',(-5.,0.,15.)) ;
#92=CARTESIAN_POINT('Axis2P3D Location',(5.,-10.,0.)) ;
#97=CARTESIAN_POINT('Line Origine',(-1.7763568394E-015,-10.,-15.)) ;
#101=CARTESIAN_POINT('Vertex',(5.,-10.,-15.)) ;
#104=CARTESIAN_POINT('Line Origine',(5.,-10.,0.)) ;
#108=CARTESIAN_POINT('Vertex',(5.,-10.,15.)) ;
#111=CARTESIAN_POINT('Line Origine',(-1.7763568394E-015,-10.,15.)) ;
#123=CARTESIAN_POINT('Axis2P3D Location',(5.,-10.,0.)) ;
#128=CARTESIAN_POINT('Line Origine',(5.,0.,-15.)) ;
#132=CARTESIAN_POINT('Vertex',(5.,10.,-15.)) ;
#135=CARTESIAN_POINT('Line Origine',(5.,10.,0.)) ;
#139=CARTESIAN_POINT('Vertex',(5.,10.,15.)) ;
#142=CARTESIAN_POINT('Line Origine',(5.,0.,15.)) ;
#154=CARTESIAN_POINT('Axis2P3D Location',(-5.,10.,0.)) ;
#159=CARTESIAN_POINT('Line Origine',(-1.7763568394E-015,10.,-15.)) ;
#164=CARTESIAN_POINT('Line Origine',(-1.7763568394E-015,10.,15.)) ;
#176=CARTESIAN_POINT('Axis2P3D Location',(0.,0.,-15.)) ;
#188=CARTESIAN_POINT('Axis2P3D Location',(0.,0.,15.)) ;
#53=DIRECTION('Axis2P3D Direction',(-1.,0.,0.)) ;
#54=DIRECTION('Axis2P3D XDirection',(0.,-1.,0.)) ;
#58=DIRECTION('Vector Direction',(0.,0.,1.)) ;
#67=DIRECTION('Vector Direction',(0.,-1.,0.)) ;
#74=DIRECTION('Vector Direction',(0.,0.,1.)) ;
#81=DIRECTION('Vector Direction',(0.,-1.,0.)) ;
#93=DIRECTION('Axis2P3D Direction',(0.,1.,0.)) ;
#94=DIRECTION('Axis2P3D XDirection',(-1.,0.,0.)) ;
#98=DIRECTION('Vector Direction',(-1.,0.,0.)) ;
#105=DIRECTION('Vector Direction',(0.,0.,1.)) ;
#112=DIRECTION('Vector Direction',(-1.,0.,0.)) ;
#124=DIRECTION('Axis2P3D Direction',(1.,0.,0.)) ;
#125=DIRECTION('Axis2P3D XDirection',(0.,1.,0.)) ;
#129=DIRECTION('Vector Direction',(0.,1.,0.)) ;
#136=DIRECTION('Vector Direction',(0.,0.,1.)) ;
#143=DIRECTION('Vector Direction',(0.,1.,0.)) ;
#155=DIRECTION('Axis2P3D Direction',(0.,-1.,0.)) ;
#156=DIRECTION('Axis2P3D XDirection',(1.,0.,0.)) ;
```

```
#160=DIRECTION('Vector Direction',(1.,0.,0.)) ;
#165=DIRECTION('Vector Direction',(1.,0.,0.)) ;
#177=DIRECTION('Axis2P3D Direction',(0.,0.,1.)) ;
#178=DIRECTION('Axis2P3D XDirection',(1.,0.,0.)) ;
#189=DIRECTION('Axis2P3D Direction',(0.,0.,1.)) ;
#190=DIRECTION('Axis2P3D XDirection',(1.,0.,0.)) ;
#48=AXIS2_PLACEMENT_3D(' ',#47,$,$) ;
#55=AXIS2_PLACEMENT_3D('Plane Axis2P3D',#52,#53,#54) ;
#95=AXIS2_PLACEMENT_3D('Plane Axis2P3D',#92,#93,#94) ;
#126=AXIS2_PLACEMENT_3D('Plane Axis2P3D',#123,#124,#125) ;
#157=AXIS2_PLACEMENT_3D('Plane Axis2P3D',#154,#155,#156) ;
#179=AXIS2_PLACEMENT_3D('Plane Axis2P3D',#176,#177,#178) ;
#191=AXIS2_PLACEMENT_3D('Plane Axis2P3D',#188,#189,#190) ;
#40=PRODUCT_DEFINITION_SHAPE(' ',' ',#14) ;
#31=APPROVAL_PERSON_ORGANIZATION(#25,#21,#19) ;
#25=PERSON_AND_ORGANIZATION(#22,#23) ;
#22=PERSON(' ',' ',' ',$,$,$) ;
#23=ORGANIZATION(' ',' ',' ') ;
#21=APPROVAL(#20,' ') ;
#20=APPROVAL_STATUS('not_yet_approved') ;
#19=APPROVAL_ROLE('APPROVER') ;
#13=DATE_AND_TIME(#11,#12) ;
#12=LOCAL_TIME(7,4,39.,#10) ;
#10=COORDINATED_UNIVERSAL_TIME_OFFSET(0,0,.AHEAD.) ;
#86=ORIENTED_EDGE('',*,*,#65,.F.) ;
#87=ORIENTED_EDGE('',*,*,#72,.T.) ;
#88=ORIENTED_EDGE('',*,*,#79,.T.) ;
#89=ORIENTED_EDGE('',*,*,#84,.F.) ;
#117=ORIENTED_EDGE('',*,*,#79,.F.) ;
#118=ORIENTED_EDGE('',*,*,#103,.T.) ;
#119=ORIENTED_EDGE('',*,*,#110,.T.) ;
#120=ORIENTED_EDGE('',*,*,#115,.F.) ;
#148=ORIENTED_EDGE('',*,*,#110,.F.) ;
#149=ORIENTED_EDGE('',*,*,#134,.T.) ;
#150=ORIENTED_EDGE('',*,*,#141,.T.) ;
#151=ORIENTED_EDGE('',*,*,#146,.F.) ;
#170=ORIENTED_EDGE('',*,*,#141,.F.) ;
#171=ORIENTED_EDGE('',*,*,#163,.T.) ;
#172=ORIENTED_EDGE('',*,*,#65,.T.) ;
#173=ORIENTED_EDGE('',*,*,#168,.F.) ;
#182=ORIENTED_EDGE('',*,*,#163,.F.) ;
#183=ORIENTED_EDGE('',*,*,#134,.F.) ;
#184=ORIENTED_EDGE('',*,*,#103,.F.) ;
#185=ORIENTED_EDGE('',*,*,#72,.F.) ;
#194=ORIENTED_EDGE('',*,*,#84,.T.) ;
#195=ORIENTED_EDGE('',*,*,#115,.T.) ;
#196=ORIENTED_EDGE('',*,*,#146,.T.) ;
#197=ORIENTED_EDGE('',*,*,#168,.T.) ;
#51=CLOSED_SHELL('Closed Shell',(#91,#122,#153,#175,#187,#199)) ;
#59=VECTOR('Line Direction',#58,1.) ;
#68=VECTOR('Line Direction',#67,1.) ;
#75=VECTOR('Line Direction',#74,1.) ;
#82=VECTOR('Line Direction',#81,1.) ;
#99=VECTOR('Line Direction',#98,1.) ;
#106=VECTOR('Line Direction',#105,1.) ;
#113=VECTOR('Line Direction',#112,1.) ;
#130=VECTOR('Line Direction',#129,1.) ;
#137=VECTOR('Line Direction',#136,1.) ;
#144=VECTOR('Line Direction',#143,1.) ;
#161=VECTOR('Line Direction',#160,1.) ;
#166=VECTOR('Line Direction',#165,1.) ;
#201=ADVANCED_BREP_SHAPE_REPRESENTATION('NONE',(#200),#46) ;
#49=SHAPE_REPRESENTATION(' ',(#48),#46) ;
#91=ADVANCED_FACE('',(#90),#56,.T.) ;
```

```
#122=ADVANCED_FACE('',(#121),#96,.F.) ;
#153=ADVANCED_FACE('',(#152),#127,.T.) ;
#175=ADVANCED_FACE('',(#174),#158,.F.) ;
#187=ADVANCED_FACE('',(#186),#180,.F.) ;
#199=ADVANCED_FACE('',(#198),#192,.T.) ;
#4=APPLICATION_PROTOCOL_DEFINITION('international
standard','config_control_design',1994,#1) ;
#32=APPROVAL_DATE_TIME(#13,#21) ;
#200=MANIFOLD_SOLID_BREP('PartBody',#51) ;
#11=CALENDAR_DATE(2007,14,11) ;
#30=CC_DESIGN_APPROVAL(#21,(#16,#6,#14)) ;
#18=CC_DESIGN_DATE_AND_TIME_ASSIGNMENT(#13,#17,(#16)) ;
#29=CC_DESIGN_DATE_AND_TIME_ASSIGNMENT(#13,#28,(#14)) ;
#17=DATE_TIME_ROLE('classification_date') ;
#28=DATE_TIME_ROLE('creation_date') ;
#27=CC_DESIGN_PERSON_AND_ORGANIZATION_ASSIGNMENT(#25,#26,(#16)) ;
#33=CC_DESIGN_PERSON_AND_ORGANIZATION_ASSIGNMENT(#25,#34,(#6)) ;
#35=CC_DESIGN_PERSON_AND_ORGANIZATION_ASSIGNMENT(#25,#36,(#6,#14)) ;
#37=CC_DESIGN_PERSON_AND_ORGANIZATION_ASSIGNMENT(#25,#38,(#5)) ;
#26=PERSON_AND_ORGANIZATION_ROLE('classification_officer') ;
#34=PERSON_AND_ORGANIZATION_ROLE('design_supplier') ;
#36=PERSON_AND_ORGANIZATION_ROLE('creator') ;
#38=PERSON_AND_ORGANIZATION_ROLE('design_owner') ;
#39=CC_DESIGN_SECURITY_CLASSIFICATION(#16,(#6)) ;
#202=SHAPE_REPRESENTATION_RELATIONSHIP(' ',' ',#49,#201) ;
#3=DESIGN_CONTEXT(' ',#1,'design') ;
#65=EDGE_CURVE('',#62,#64,#60,.T.) ;
#72=EDGE_CURVE('',#62,#71,#69,.T.) ;
#79=EDGE_CURVE('',#71,#78,#76,.T.) ;
#84=EDGE_CURVE('',#64,#78,#83,.T.) ;
#103=EDGE_CURVE('',#71,#102,#100,.F.) ;
#110=EDGE_CURVE('',#102,#109,#107,.T.) ;
#115=EDGE_CURVE('',#78,#109,#114,.F.) ;
#134=EDGE_CURVE('',#102,#133,#131,.T.) ;
#141=EDGE_CURVE('',#133,#140,#138,.T.) ;
#146=EDGE_CURVE('',#109,#140,#145,.T.) ;
#163=EDGE_CURVE('',#133,#62,#162,.F.) ;
#168=EDGE_CURVE('',#140,#64,#167,.F.) ;
#85=EDGE_LOOP('',(#86,#87,#88,#89)) ;
#116=EDGE_LOOP('',(#117,#118,#119,#120)) ;
#147=EDGE_LOOP('',(#148,#149,#150,#151)) ;
#169=EDGE_LOOP('',(#170,#171,#172,#173)) ;
#181=EDGE_LOOP('',(#182,#183,#184,#185)) ;
#193=EDGE_LOOP('',(#194,#195,#196,#197)) ;
#90=FACE_OUTER_BOUND('',#85,.T.) ;
#121=FACE_OUTER_BOUND('',#116,.T.) ;
#152=FACE_OUTER_BOUND('',#147,.T.) ;
#174=FACE_OUTER_BOUND('',#169,.T.) ;
#186=FACE_OUTER_BOUND('',#181,.T.) ;
#198=FACE_OUTER_BOUND('',#193,.T.) ;
#45=UNCERTAINTY_MEASURE_WITH_UNIT(LENGTH_MEASURE(0.005),#
41,'distance_accuracy_value','CONFUSED CURVE UNCERTAINTY') ;
#60=LINE('Line',#57,#59) ;
#69=LINE('Line',#66,#68) ;
#76=LINE('Line',#73,#75) ;
#83=LINE('Line',#80,#82) ;
#100=LINE('Line',#97,#99) ;
#107=LINE('Line',#104,#106) ;
#114=LINE('Line',#111,#113) ;
#131=LINE('Line',#128,#130) ;
#138=LINE('Line',#135,#137) ;
#145=LINE('Line',#142,#144) ;
#162=LINE('Line',#159,#161) ;
#167=LINE('Line',#164,#166) ;
```

```
#2=MECHANICAL_CONTEXT(' ',#1,'mechanical') ;
#24=PERSONAL_ADDRESS(' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',(#22),' ')
;
#56=PLANE('Plane',#55) ;
#96=PLANE('Plane',#95) ;
#127=PLANE('Plane',#126) ;
#158=PLANE('Plane',#157) ;
#180=PLANE('Plane',#179) ;
#192=PLANE('Plane',#191) ;
#43=PLANE_ANGLE_MEASURE_WITH_UNIT(PLANE_ANGLE_MEASURE(0.0174532925199),#42) ;
#7=PRODUCT_CATEGORY('part',$) ;
#9=PRODUCT_CATEGORY_RELATIONSHIP(' ',' ',#7,#8) ;
#6=PRODUCT_DEFINITION_FORMATION_WITH_SPECIFIED_SOURCE('',' ',#5,.NOT_KNOWN.) ;
#8=PRODUCT_RELATED_PRODUCT_CATEGORY('detail',$,(#5)) ;
#50=SHAPE_DEFINITION_REPRESENTATION(#40,#49) ;
#62=VERTEX_POINT('',#61) ;
#64=VERTEX_POINT('',#63) ;
#71=VERTEX_POINT('',#70) ;
#78=VERTEX_POINT('',#77) ;
#102=VERTEX_POINT('',#101) ;
#109=VERTEX_POINT('',#108) ;
#133=VERTEX_POINT('',#132) ;
#140=VERTEX_POINT('',#139) ;
#41=(LENGTH_UNIT()NAMED_UNIT(*)SI_UNIT(.MILLI.,.METRE.)) ;
#42=(NAMED_UNIT(*)PLANE_ANGLE_UNIT()SI_UNIT($,.RADIAN.)) ;
#44=(NAMED_UNIT(*)SI_UNIT($,.STERADIAN.)SOLID_ANGLE_UNIT()) ;
#46=(GEOMETRIC_REPRESENTATION_CONTEXT(3)GLOBAL_UNCERTAINTY_ASSIGNED_CONTEXT((#
45))GLOBAL_UNIT_ASSIGNED_CONTEXT((#41,#42,#44))REPRESENTATION_CONTEXT(' ',' '))
;
ENDSEC;
END-ISO-10303-21;
```

B.4 STEP: Solidworks

In the Solidworks output file the line numbers are sequential and the entities output as they come.

```
ISO-10303-21;
HEADER;
FILE_DESCRIPTION (( 'STEP AP203' ),
    '1' );
FILE_NAME ('blk102030b.STEP',
    '2007-11-14T07:10:46',
    ( 'LICPPC78' ),
    ( 'EPFL' ),
    'SwSTEP 2.0',
    'SolidWorks 2006104',
    '' );
FILE_SCHEMA (( 'CONFIG_CONTROL_DESIGN' ));
ENDSEC;

DATA;
#1 = ORIENTED_EDGE ( 'NONE', *, *, #2, .F. ) ;
#2 = EDGE_CURVE ( 'NONE', #98, #105, #37, .T. ) ;
#3 = VERTEX_POINT ( 'NONE', #42 ) ;
#4 = ORIENTED_EDGE ( 'NONE', *, *, #92, .F. ) ;
#5 = EDGE_CURVE ( 'NONE', #3, #10, #44, .T. ) ;
#6 = EDGE_CURVE ( 'NONE', #105, #10, #33, .T. ) ;
#7 = ORIENTED_EDGE ( 'NONE', *, *, #6, .T. ) ;
#8 = ORIENTED_EDGE ( 'NONE', *, *, #2, .T. ) ;
#9 = ORIENTED_EDGE ( 'NONE', *, *, #5, .F. ) ;
#10 = VERTEX_POINT ( 'NONE', #31 ) ;
#11 = EDGE_LOOP ( 'NONE', ( #8, #7, #9, #4 ) ) ;
#12 = ORIENTED_EDGE ( 'NONE', *, *, #80, .F. ) ;
#13 = ORIENTED_EDGE ( 'NONE', *, *, #97, .F. ) ;
#14 = ORIENTED_EDGE ( 'NONE', *, *, #104, .F. ) ;
#15 = EDGE_LOOP ( 'NONE', ( #14, #12, #13, #109 ) ) ;
#16 = ORIENTED_EDGE ( 'NONE', *, *, #24, .T. ) ;
#17 = ORIENTED_EDGE ( 'NONE', *, *, #92, .T. ) ;
#18 = ORIENTED_EDGE ( 'NONE', *, *, #100, .T. ) ;
#19 = EDGE_LOOP ( 'NONE', ( #18, #17, #21, #16 ) ) ;
#20 = ADVANCED_FACE ( 'NONE', ( #50 ), #49, .F. ) ;
#21 = ORIENTED_EDGE ( 'NONE', *, *, #84, .T. ) ;
#22 = DIRECTION ( 'NONE',  ( -1.000000000000000000, -1.734723475976807600E-016,
0.0000000000000000000 ) ) ;
#23 = ADVANCED_FACE ( 'NONE', ( #47 ), #38, .T. ) ;
#24 = EDGE_CURVE ( 'NONE', #78, #90, #39, .T. ) ;
#25 = VECTOR ( 'NONE', #32, 1000.000000000000000 ) ;
#26 = CARTESIAN_POINT ( 'NONE',  ( 5.000000000000000900, 10.00000000000000200, -
15.00000000000000000 ) ) ;
#27 = AXIS2_PLACEMENT_3D ( 'NONE', #46, #45, #48 ) ;
#28 = CARTESIAN_POINT ( 'NONE',  ( 4.999999999999997300, -10.00000000000000000,
-15.00000000000000000 ) ) ;
#29 = LINE ( 'NONE', #28, #59 ) ;
#30 = CARTESIAN_POINT ( 'NONE',  ( -5.000000000000000900, -10.00000000000000200,
15.00000000000000000 ) ) ;
#31 = CARTESIAN_POINT ( 'NONE',  ( -5.000000000000000900, 10.00000000000000200,
-15.00000000000000000 ) ) ;
#32 = DIRECTION ( 'NONE',  ( -1.000000000000000000, 0.0000000000000000000,
0.0000000000000000000 ) ) ;
#33 = LINE ( 'NONE', #26, #25 ) ;
#34 = DIRECTION ( 'NONE',  ( 0.0000000000000000000, 0.0000000000000000000, -
1.000000000000000000 ) ) ;
#35 = VECTOR ( 'NONE', #34, 1000.000000000000000 ) ;
#36 = CARTESIAN_POINT ( 'NONE',  ( 5.000000000000000900, 10.00000000000000200,
15.00000000000000000 ) ) ;
#37 = LINE ( 'NONE', #36, #40 ) ;
#38 = PLANE ( 'NONE',  #73 ) ;
#39 = LINE ( 'NONE', #41, #53 ) ;
#40 = VECTOR ( 'NONE', #74, 1000.000000000000000 ) ;
#41 = CARTESIAN_POINT ( 'NONE',  ( 4.999999999999997300, -10.00000000000000000,
15.00000000000000000 ) ) ;
#42 = CARTESIAN_POINT ( 'NONE',  ( -5.000000000000000900, 10.00000000000000200,
15.00000000000000000 ) ) ;
#43 = CARTESIAN_POINT ( 'NONE',  ( -5.000000000000000900, 10.00000000000000200,
15.00000000000000000 ) ) ;
#44 = LINE ( 'NONE', #43, #35 ) ;
#45 = DIRECTION ( 'NONE',  ( 0.0000000000000000000, 0.0000000000000000000,
1.000000000000000000 ) ) ;
#46 = CARTESIAN_POINT ( 'NONE',  ( 5.000000000000000900, 10.00000000000000200, -
15.00000000000000000 ) ) ;
#47 = FACE_OUTER_BOUND ( 'NONE', #19, .T. ) ;
#48 = DIRECTION ( 'NONE',  ( 1.000000000000000000, 0.0000000000000000000,
0.0000000000000000000 ) ) ;
#49 = PLANE ( 'NONE',  #27 ) ;
#50 = FACE_OUTER_BOUND ( 'NONE', #15, .T. ) ;
#51 = CARTESIAN_POINT ( 'NONE',  ( -5.000000000000000900, -10.00000000000000200,
-15.00000000000000000 ) ) ;
#52 = DIRECTION ( 'NONE',  ( 1.000000000000000000, 1.734723475976807300E-016,
0.0000000000000000000 ) ) ;
#53 = VECTOR ( 'NONE', #52, 1000.000000000000000 ) ;
#54 = DIRECTION ( 'NONE',  ( 0.0000000000000000000, -1.000000000000000000,
0.0000000000000000000 ) ) ;
#55 = VECTOR ( 'NONE', #54, 1000.000000000000000 ) ;
#56 = CARTESIAN_POINT ( 'NONE',  ( -5.000000000000000900, -10.00000000000000200,
15.00000000000000000 ) ) ;
#57 = LINE ( 'NONE', #56, #55 ) ;
#58 = DIRECTION ( 'NONE',  ( 1.000000000000000000, 1.734723475976807300E-016,
0.0000000000000000000 ) ) ;
#59 = VECTOR ( 'NONE', #58, 1000.000000000000000 ) ;
#60 = DIRECTION ( 'NONE',  ( -1.734723475976807600E-016, 1.000000000000000000,
0.0000000000000000000 ) ) ;
#61 = CARTESIAN_POINT ( 'NONE',  ( 4.999999999999997300, -10.00000000000000000,
15.00000000000000000 ) ) ;
#62 = AXIS2_PLACEMENT_3D ( 'NONE', #61, #60, #22 ) ;
#63 = PLANE ( 'NONE',  #62 ) ;
#64 = DIRECTION ( 'NONE',  ( 1.000000000000000000, 0.0000000000000000000,
0.0000000000000000000 ) ) ;
#65 = FACE_OUTER_BOUND ( 'NONE', #86, .T. ) ;
#66 = DIRECTION ( 'NONE',  ( 0.0000000000000000000, 0.0000000000000000000, -
1.000000000000000000 ) ) ;
#67 = VECTOR ( 'NONE', #66, 1000.000000000000000 ) ;
#68 = CARTESIAN_POINT ( 'NONE',  ( -5.000000000000000900, -10.00000000000000200,
15.00000000000000000 ) ) ;
#69 = LINE ( 'NONE', #68, #67 ) ;
#70 = DIRECTION ( 'NONE',  ( 1.000000000000000000, 0.0000000000000000000,
0.0000000000000000000 ) ) ;
#71 = DIRECTION ( 'NONE',  ( 0.0000000000000000000, 0.0000000000000000000,
1.000000000000000000 ) ) ;
#72 = CARTESIAN_POINT ( 'NONE',  ( 5.000000000000000900, 10.00000000000000200,
15.00000000000000000 ) ) ;
#73 = AXIS2_PLACEMENT_3D ( 'NONE', #72, #71, #70 ) ;
#74 = DIRECTION ( 'NONE',  ( 0.0000000000000000000, 0.0000000000000000000, -
1.000000000000000000 ) ) ;
#75 = VERTEX_POINT ( 'NONE', #51 ) ;
#76 = ORIENTED_EDGE ( 'NONE', *, *, #77, .F. ) ;
#77 = EDGE_CURVE ( 'NONE', #78, #75, #69, .T. ) ;
#78 = VERTEX_POINT ( 'NONE', #30 ) ;
#79 = ORIENTED_EDGE ( 'NONE', *, *, #80, .T. ) ;
#80 = EDGE_CURVE ( 'NONE', #75, #102, #29, .T. ) ;
#81 = ORIENTED_EDGE ( 'NONE', *, *, #89, .F. ) ;
#82 = ORIENTED_EDGE ( 'NONE', *, *, #24, .F. ) ;
#83 = ORIENTED_EDGE ( 'NONE', *, *, #84, .F. ) ;
#84 = EDGE_CURVE ( 'NONE', #3, #78, #57, .T. ) ;
#85 = ADVANCED_FACE ( 'NONE', ( #65 ), #63, .F. ) ;
```

```
#86 = EDGE_LOOP ( 'NONE', ( #87, #79, #81, #82 ) ) ;
#87 = ORIENTED_EDGE ( 'NONE', *, *, #77, .T. ) ;
#88 = CLOSED_SHELL ( 'NONE', ( #101, #20, #106, #23, #93, #85 ) ) ;
#89 = EDGE_CURVE ( 'NONE', #90, #102, #115, .T. ) ;
#90 = VERTEX_POINT ( 'NONE', #116 ) ;
#91 = MANIFOLD_SOLID_BREP ( 'NONE', #88 ) ;
#92 = EDGE_CURVE ( 'NONE', #98, #3, #112, .T. ) ;
#93 = ADVANCED_FACE ( 'NONE', ( #121 ), #120, .F. ) ;
#94 = EDGE_LOOP ( 'NONE', ( #95, #96, #76, #83 ) ) ;
#95 = ORIENTED_EDGE ( 'NONE', *, *, #5, .T. ) ;
#96 = ORIENTED_EDGE ( 'NONE', *, *, #97, .T. ) ;
#97 = EDGE_CURVE ( 'NONE', #10, #75, #127, .T. ) ;
#98 = VERTEX_POINT ( 'NONE', #151 ) ;
#99 = ORIENTED_EDGE ( 'NONE', *, *, #100, .F. ) ;
#100 = EDGE_CURVE ( 'NONE', #90, #98, #132, .T. ) ;
#101 = ADVANCED_FACE ( 'NONE', ( #152 ), #145, .F. ) ;
#102 = VERTEX_POINT ( 'NONE', #149 ) ;
#103 = ORIENTED_EDGE ( 'NONE', *, *, #104, .T. ) ;
#104 = EDGE_CURVE ( 'NONE', #102, #105, #143, .T. ) ;
#105 = VERTEX_POINT ( 'NONE', #124 ) ;
#106 = ADVANCED_FACE ( 'NONE', ( #123 ), #135, .F. ) ;
#107 = EDGE_LOOP ( 'NONE', ( #108, #103, #1, #99 ) ) ;
#108 = ORIENTED_EDGE ( 'NONE', *, *, #89, .T. ) ;
#109 = ORIENTED_EDGE ( 'NONE', *, *, #6, .F. ) ;
#110 = VECTOR ( 'NONE', #122, 1000.000000000000000 ) ;
#111 = CARTESIAN_POINT ( 'NONE',  ( 5.000000000000000900, 10.00000000000000200,
15.00000000000000000 ) ) ;
#112 = LINE ( 'NONE', #111, #110 ) ;
#113 = DIRECTION ( 'NONE',  ( 0.0000000000000000000, 0.0000000000000000000, -
1.000000000000000000 ) ) ;
#114 = CARTESIAN_POINT ( 'NONE',  ( 4.999999999999997300, -10.00000000000000000,
15.00000000000000000 ) ) ;
#115 = LINE ( 'NONE', #114, #117 ) ;
#116 = CARTESIAN_POINT ( 'NONE',  ( 4.999999999999997300, -10.00000000000000000,
15.00000000000000000 ) ) ;
#117 = VECTOR ( 'NONE', #113, 1000.000000000000000 ) ;
#118 = CARTESIAN_POINT ( 'NONE',  ( -5.000000000000000900, -
10.00000000000000200, 15.00000000000000000 ) ) ;
#119 = AXIS2_PLACEMENT_3D ( 'NONE', #118, #64, #133 ) ;
#120 = PLANE ( 'NONE',  #119 ) ;
#121 = FACE_OUTER_BOUND ( 'NONE', #94, .T. ) ;
#122 = DIRECTION ( 'NONE',  ( -1.000000000000000000, 0.0000000000000000000,
0.0000000000000000000 ) ) ;
#123 = FACE_OUTER_BOUND ( 'NONE', #107, .T. ) ;
#124 = CARTESIAN_POINT ( 'NONE',  ( 5.000000000000000900, 10.00000000000000200,
-15.00000000000000000 ) ) ;
#125 = DIRECTION ( 'NONE',  ( 1.734723475976806800E-016, 1.000000000000000000,
0.0000000000000000000 ) ) ;
#126 = VECTOR ( 'NONE', #125, 1000.000000000000000 ) ;
#127 = LINE ( 'NONE', #136, #134 ) ;
#128 = DIRECTION ( 'NONE',  ( 0.0000000000000000000, -1.000000000000000000,
0.0000000000000000000 ) ) ;
#129 = CARTESIAN_POINT ( 'NONE',  ( 5.000000000000000900, 10.00000000000000200,
15.00000000000000000 ) ) ;
#130 = AXIS2_PLACEMENT_3D ( 'NONE', #129, #128, #150 ) ;
#131 = VECTOR ( 'NONE', #157, 1000.000000000000000 ) ;
#132 = LINE ( 'NONE', #137, #131 ) ;
#133 = DIRECTION ( 'NONE',  ( 0.0000000000000000000, 0.0000000000000000000, -
1.000000000000000000 ) ) ;
#134 = VECTOR ( 'NONE', #144, 1000.000000000000000 ) ;
#135 = PLANE ( 'NONE',  #140 ) ;
#136 = CARTESIAN_POINT ( 'NONE',  ( -5.000000000000000900, -
10.00000000000000200, -15.00000000000000000 ) ) ;
#137 = CARTESIAN_POINT ( 'NONE',  ( 4.999999999999997300, -10.00000000000000000,
15.00000000000000000 ) ) ;
#138 = DIRECTION ( 'NONE',  ( -1.000000000000000000, 1.734723475976806800E-016,
0.0000000000000000000 ) ) ;
#139 = CARTESIAN_POINT ( 'NONE',  ( 4.999999999999997300, -10.00000000000000000,
15.00000000000000000 ) ) ;
#140 = AXIS2_PLACEMENT_3D ( 'NONE', #139, #138, #160 ) ;
#141 = DIRECTION ( 'NONE',  ( 0.0000000000000000000, 0.0000000000000000000,
1.000000000000000000 ) ) ;
#142 = CARTESIAN_POINT ( 'NONE',  ( 4.999999999999997300, -10.00000000000000000,
-15.00000000000000000 ) ) ;
#143 = LINE ( 'NONE', #142, #126 ) ;
#144 = DIRECTION ( 'NONE',  ( 0.0000000000000000000, -1.000000000000000000,
0.0000000000000000000 ) ) ;
#145 = PLANE ( 'NONE',  #130 ) ;
#146 = DIRECTION ( 'NONE',  ( 1.000000000000000000, 0.0000000000000000000,
0.0000000000000000000 ) ) ;
#147 = CARTESIAN_POINT ( 'NONE',  ( 0.0000000000000000000,
0.0000000000000000000, 0.0000000000000000000 ) ) ;
#148 = AXIS2_PLACEMENT_3D ( 'NONE', #147, #141, #146 ) ;
#149 = CARTESIAN_POINT ( 'NONE',  ( 4.999999999999997300, -10.00000000000000000,
-15.00000000000000000 ) ) ;
#150 = DIRECTION ( 'NONE',  ( 0.0000000000000000000, 0.0000000000000000000, -
1.000000000000000000 ) ) ;
#151 = CARTESIAN_POINT ( 'NONE',  ( 5.000000000000000900, 10.00000000000000200,
15.00000000000000000 ) ) ;
#152 = FACE_OUTER_BOUND ( 'NONE', #11, .T. ) ;
#153 = ADVANCED_BREP_SHAPE_REPRESENTATION ( 'blk102030b', ( #91, #148 ), #154 )
;
#154 =( GEOMETRIC_REPRESENTATION_CONTEXT ( 3 )
GLOBAL_UNCERTAINTY_ASSIGNED_CONTEXT ( ( #155 ) ) GLOBAL_UNIT_ASSIGNED_CONTEXT (
( #159, #158, #156 ) ) REPRESENTATION_CONTEXT ( 'NONE', 'WORKASPACE' ) );
#155 = UNCERTAINTY_MEASURE_WITH_UNIT (LENGTH_MEASURE( 1.000000000000000100E-005
), #159, 'distance_accuracy_value', 'NONE');
#156 =( NAMED_UNIT ( * ) SI_UNIT ( $, .STERADIAN. ) SOLID_ANGLE_UNIT ( ) );
#157 = DIRECTION ( 'NONE',  ( 1.734723475976806800E-016, 1.000000000000000000,
0.0000000000000000000 ) ) ;
#158 =( NAMED_UNIT ( * ) PLANE_ANGLE_UNIT ( ) SI_UNIT ( $, .RADIAN. ) );
#159 =( LENGTH_UNIT ( ) NAMED_UNIT ( * ) SI_UNIT ( .MILLI., .METRE. ) );
#160 = DIRECTION ( 'NONE',  ( -1.734723475976806800E-016, -1.000000000000000000,
0.0000000000000000000 ) ) ;
#161 = SHAPE_DEFINITION_REPRESENTATION ( #167, #153 ) ;
#162 = PERSON ( 'UNSPECIFIED', 'UNSPECIFIED', 'UNSPECIFIED', ('UNSPECIFIED'),
('UNSPECIFIED'), ('UNSPECIFIED') ) ;
#163 = PRODUCT ( 'blk102030b', 'blk102030b', '', ( #164 ) ) ;
#164 = MECHANICAL_CONTEXT ( 'NONE', #166, 'mechanical' ) ;
#165 = APPLICATION_PROTOCOL_DEFINITION ( 'international standard',
'config_control_design', 1994, #166 ) ;
#166 = APPLICATION_CONTEXT ( 'configuration controlled 3d designs of mechanical
parts and assemblies' ) ;
#167 = PRODUCT_DEFINITION_SHAPE ( 'NONE', 'NONE',  #230 ) ;
#168 = PRODUCT_RELATED_PRODUCT_CATEGORY ( 'detail', '', ( #163 ) ) ;
#169 = CC_DESIGN_SECURITY_CLASSIFICATION ( #215, ( #183 ) ) ;
#170 = PERSON_AND_ORGANIZATION_ROLE ( 'creator' ) ;
#171 = PERSON_AND_ORGANIZATION ( #162, #237 ) ;
#172 = CC_DESIGN_APPROVAL ( #180, ( #183 ) ) ;
#173 = APPROVAL_DATE_TIME ( #174, #180 ) ;
#174 = DATE_AND_TIME ( #175, #176 ) ;
#175 = CALENDAR_DATE ( 2007, 14, 11 ) ;
#176 = LOCAL_TIME ( 8, 10, 46.00000000000000000, #177 ) ;
#177 = COORDINATED_UNIVERSAL_TIME_OFFSET ( 1, 0, .AHEAD. ) ;
#178 = APPROVAL_PERSON_ORGANIZATION ( #182, #180, #179 ) ;
#179 = APPROVAL_ROLE ( '' ) ;
#180 = APPROVAL ( #181, 'UNSPECIFIED' ) ;
#181 = APPROVAL_STATUS ( 'not_yet_approved' ) ;
```

```
#182 = PERSON_AND_ORGANIZATION ( #162, #237 ) ;
#183 = PRODUCT_DEFINITION_FORMATION_WITH_SPECIFIED_SOURCE ( 'ANY', '', #163,
.NOT_KNOWN. ) ;
#184 = CC_DESIGN_DATE_AND_TIME_ASSIGNMENT ( #186, #185, ( #230 ) ) ;
#185 = DATE_TIME_ROLE ( 'creation_date' ) ;
#186 = DATE_AND_TIME ( #187, #188 ) ;
#187 = CALENDAR_DATE ( 2007, 14, 11 ) ;
#188 = LOCAL_TIME ( 8, 10, 46.00000000000000000, #189 ) ;
#189 = COORDINATED_UNIVERSAL_TIME_OFFSET ( 1, 0, .AHEAD. ) ;
#190 = CC_DESIGN_PERSON_AND_ORGANIZATION_ASSIGNMENT ( #192, #191, ( #230 ) ) ;
#191 = PERSON_AND_ORGANIZATION_ROLE ( 'creator' ) ;
#192 = PERSON_AND_ORGANIZATION ( #162, #237 ) ;
#193 = CC_DESIGN_APPROVAL ( #227, ( #230 ) ) ;
#194 = APPROVAL_DATE_TIME ( #221, #227 ) ;
#195 = CC_DESIGN_APPROVAL ( #203, ( #215 ) ) ;
#196 = APPROVAL_DATE_TIME ( #197, #203 ) ;
#197 = DATE_AND_TIME ( #198, #199 ) ;
#198 = CALENDAR_DATE ( 2007, 14, 11 ) ;
#199 = LOCAL_TIME ( 8, 10, 46.00000000000000000, #200 ) ;
#200 = COORDINATED_UNIVERSAL_TIME_OFFSET ( 1, 0, .AHEAD. ) ;
#201 = APPROVAL_PERSON_ORGANIZATION ( #205, #203, #202 ) ;
#202 = APPROVAL_ROLE ( '' ) ;
#203 = APPROVAL ( #204, 'UNSPECIFIED' ) ;
#204 = APPROVAL_STATUS ( 'not_yet_approved' ) ;
#205 = PERSON_AND_ORGANIZATION ( #162, #237 ) ;
#206 = CC_DESIGN_DATE_AND_TIME_ASSIGNMENT ( #208, #207, ( #215 ) ) ;
#207 = DATE_TIME_ROLE ( 'classification_date' ) ;
#208 = DATE_AND_TIME ( #209, #210 ) ;
#209 = CALENDAR_DATE ( 2007, 14, 11 ) ;
#210 = LOCAL_TIME ( 8, 10, 46.00000000000000000, #211 ) ;
#211 = COORDINATED_UNIVERSAL_TIME_OFFSET ( 1, 0, .AHEAD. ) ;
#212 = CC_DESIGN_PERSON_AND_ORGANIZATION_ASSIGNMENT ( #214, #213, ( #215 ) ) ;
#213 = PERSON_AND_ORGANIZATION_ROLE ( 'classification_officer' ) ;
#214 = PERSON_AND_ORGANIZATION ( #162, #237 ) ;
#215 = SECURITY_CLASSIFICATION ( '', '', #216 ) ;
#216 = SECURITY_CLASSIFICATION_LEVEL ( 'unclassified' ) ;
#217 = CC_DESIGN_PERSON_AND_ORGANIZATION_ASSIGNMENT ( #219, #218, ( #183 ) ) ;
#218 = PERSON_AND_ORGANIZATION_ROLE ( 'design_supplier' ) ;
#219 = PERSON_AND_ORGANIZATION ( #162, #237 ) ;
#220 = CC_DESIGN_PERSON_AND_ORGANIZATION_ASSIGNMENT ( #171, #170, ( #183 ) ) ;
#221 = DATE_AND_TIME ( #222, #223 ) ;
#222 = CALENDAR_DATE ( 2007, 14, 11 ) ;
#223 = LOCAL_TIME ( 8, 10, 46.00000000000000000, #224 ) ;
#224 = COORDINATED_UNIVERSAL_TIME_OFFSET ( 1, 0, .AHEAD. ) ;
#225 = APPROVAL_PERSON_ORGANIZATION ( #229, #227, #226 ) ;
#226 = APPROVAL_ROLE ( '' ) ;
#227 = APPROVAL ( #228, 'UNSPECIFIED' ) ;
#228 = APPROVAL_STATUS ( 'not_yet_approved' ) ;
#229 = PERSON_AND_ORGANIZATION ( #162, #237 ) ;
#230 = PRODUCT_DEFINITION ( 'UNKNOWN', '', #183, #231 ) ;
#231 = DESIGN_CONTEXT ( 'detailed design', #233, 'design' ) ;
#232 = APPLICATION_PROTOCOL_DEFINITION ( 'international standard',
'config_control_design', 1994, #233 ) ;
#233 = APPLICATION_CONTEXT ( 'configuration controlled 3d designs of mechanical
parts and assemblies' ) ;
#234 = CC_DESIGN_PERSON_AND_ORGANIZATION_ASSIGNMENT ( #236, #235, ( #163 ) ) ;
#235 = PERSON_AND_ORGANIZATION_ROLE ( 'design_owner' ) ;
#236 = PERSON_AND_ORGANIZATION ( #162, #237 ) ;
#237 = ORGANIZATION ( 'UNSPECIFIED', 'UNSPECIFIED', '' ) ;
ENDSEC;
END-ISO-10303-21;
```

B.5 STL

STL is a commonly used format which has many drawbacks.

```
solid CATIA STL
  facet normal  0.000000e+000  0.000000e+000  1.000000e+000
    outer loop
      vertex -5.000000e+000 -1.000000e+001  1.500000e+001
      vertex  5.000000e+000 -1.000000e+001  1.500000e+001
      vertex -5.000000e+000  1.000000e+001  1.500000e+001
    endloop
  endfacet
  facet normal  0.000000e+000  0.000000e+000  1.000000e+000
    outer loop
      vertex -5.000000e+000  1.000000e+001  1.500000e+001
      vertex  5.000000e+000 -1.000000e+001  1.500000e+001
      vertex  5.000000e+000  1.000000e+001  1.500000e+001
    endloop
  endfacet
  facet normal  0.000000e+000  0.000000e+000 -1.000000e+000
    outer loop
      vertex  5.000000e+000 -1.000000e+001 -1.500000e+001
      vertex -5.000000e+000 -1.000000e+001 -1.500000e+001
      vertex  5.000000e+000  1.000000e+001 -1.500000e+001
    endloop
  endfacet
  facet normal  0.000000e+000  0.000000e+000 -1.000000e+000
    outer loop
      vertex  5.000000e+000  1.000000e+001 -1.500000e+001
      vertex -5.000000e+000 -1.000000e+001 -1.500000e+001
      vertex -5.000000e+000  1.000000e+001 -1.500000e+001
    endloop
  endfacet
  facet normal  0.000000e+000  1.000000e+000  0.000000e+000
    outer loop
      vertex  5.000000e+000  1.000000e+001 -1.500000e+001
      vertex -5.000000e+000  1.000000e+001 -1.500000e+001
      vertex  5.000000e+000  1.000000e+001  1.500000e+001
    endloop
  endfacet
  facet normal  0.000000e+000  1.000000e+000  0.000000e+000
    outer loop
      vertex  5.000000e+000  1.000000e+001  1.500000e+001
      vertex -5.000000e+000  1.000000e+001 -1.500000e+001
      vertex -5.000000e+000  1.000000e+001  1.500000e+001
    endloop
  endfacet
  facet normal  1.000000e+000  0.000000e+000  0.000000e+000
    outer loop
      vertex  5.000000e+000 -1.000000e+001 -1.500000e+001
      vertex  5.000000e+000  1.000000e+001 -1.500000e+001
      vertex  5.000000e+000 -1.000000e+001  1.500000e+001
    endloop
  endfacet
  facet normal  1.000000e+000  0.000000e+000  0.000000e+000
    outer loop
      vertex  5.000000e+000 -1.000000e+001  1.500000e+001
      vertex  5.000000e+000  1.000000e+001 -1.500000e+001
      vertex  5.000000e+000  1.000000e+001  1.500000e+001
    endloop
  endfacet
  facet normal  0.000000e+000 -1.000000e+000  0.000000e+000
    outer loop
      vertex -5.000000e+000 -1.000000e+001 -1.500000e+001
      vertex  5.000000e+000 -1.000000e+001 -1.500000e+001
      vertex -5.000000e+000 -1.000000e+001  1.500000e+001
    endloop
  endfacet
  facet normal  0.000000e+000 -1.000000e+000  0.000000e+000
    outer loop
      vertex -5.000000e+000 -1.000000e+001  1.500000e+001
      vertex  5.000000e+000 -1.000000e+001 -1.500000e+001
      vertex  5.000000e+000 -1.000000e+001  1.500000e+001
    endloop
  endfacet
  facet normal -1.000000e+000  0.000000e+000  0.000000e+000
    outer loop
      vertex -5.000000e+000  1.000000e+001 -1.500000e+001
      vertex -5.000000e+000 -1.000000e+001 -1.500000e+001
      vertex -5.000000e+000  1.000000e+001  1.500000e+001
    endloop
  endfacet
  facet normal -1.000000e+000  0.000000e+000  0.000000e+000
    outer loop
      vertex -5.000000e+000  1.000000e+001  1.500000e+001
      vertex -5.000000e+000 -1.000000e+001 -1.500000e+001
      vertex -5.000000e+000 -1.000000e+001  1.500000e+001
    endloop
  endfacet
endsolid CATIA STL
```

B.6 VRML

VRML is another format which has drawbacks in being approximative.

```
#VRML V1.0 ascii
Separator {
MaterialBinding {
value OVERALL
}
Material {
ambientColor [
 0.796078 0.823529 0.937255
]
diffuseColor [
 0.796078 0.823529 0.937255
]
emissiveColor [
 0.063686 0.065882 0.074980
]
specularColor [
 0.756275 0.782353 0.890392
]
shininess [
 0.550000
]
transparency [
 0.000000
]
}
Coordinate3 {
point [
 -0.005000 -0.010000 -0.015000,
 -0.005000 -0.010000 0.015000,
 -0.005000 0.010000 -0.015000,
 -0.005000 0.010000 0.015000,
  0.005000 -0.010000 -0.015000,
  0.005000 -0.010000 0.015000,
  0.005000 0.010000 -0.015000,
  0.005000 0.010000 0.015000
]
}
IndexedFaceSet {
coordIndex [
 3, 2, 1, -1,
 1, 2, 0, -1,
 7, 6, 3, -1,
 3, 6, 2, -1,
 5, 4, 7, -1,
 7, 4, 6, -1,
 1, 0, 5, -1,
 5, 0, 4, -1,
 6, 4, 2, -1,
 2, 4, 0, -1,
 5, 7, 1, -1,
 1, 7, 3, -1
]
}
}
```

Appendix C
Machining Feature Summary

The following is a summary of machining features present in STEP-NC (ISO 14649). It was written in order to try and find simple parameters in order to extrapolate the machining experiments for eco-evaluation in the NEXT project. The reason for including it in this book is to provide a summary of machining features for some proposed projects.

STEP-NC is an interesting topic in its own right. It started as work to update the current control standard, with G and M codes, to remove the need for post-processors and to pass CAD data directly to the machine tool. The CAD data is represented as a set of features that are for manufacturing. It is expected that this data is supplied by a process planner who has decided on the manufacturing method and supplies high-level information about the elements to be manufactured. This is the sort of input currently supplied to a CAM system for generating toolpaths. With STEP-NC the intention is to let the machine tool control unit calculate its own toolpaths, based on knowledge about the needs of the machine tool it controls. The features here are given as an example of manufacturing features as an illustration of an application area. For a full definition of the features it is necessary to look at the ISO standard document, the presentation here differs slightly from that in the standards document.

In Table C.1 there is a summary of the features with some general parameters. This information was used as input for calculating ecological effects as part of the NEXT (next generation of machine tools) project, so exact geometric information was not needed.

Table C.1 Manufacturing feature summary with eco-evaluation parameters

Region projection	Position, area
Region surface list	Position, combined area
Planar_face	Position, area
Closed pocket	Position, plunge strategy, pocketing strategy, pocket area, bottom conditions?
Open pocket	Position, pocket area, bottom conditions
Through closed pocket	Position, pocket area
Open slot	Position, depth, width
Half-open slot (round end)	Position, depth, width
Half-open slot (sq. end)	Position, depth, width, end blend radius
Half-open slot (Woodruff)	Position, depth, width, radius
Closed slot (round end)	Position, depth, width
Closed slot (square end)	Position, depth, width, end blend radius
Closed slot (Woodruff)	Position, depth, width, radius
Loop slot	Position, depth, width, radius
Step	Position, depth, width
Round_hole	Position, radius, depth
Rounded, tapered hole	Position, radius, depth, angle
Outer profile	Position, length, maximum height
Shape profile	Position, Length, maximum height, bottom conditions
Boss	Position, surrounding face area
Spherical_cap	Position, radius
Rounded_end	Position, radius, height
Thread	Position, radius, thread size
Counterbore hole	Position, radius 1, radius 2, depth 1, depth 2
Countersunk hole	Radius 1, radius 2, depth 1, depth 2
Circular pattern	Position, characteristics of repeated element, pattern radius, number of elements in pattern
Rectangular pattern	Position, characteristics of repeated element, pattern width and height, number of elements in pattern
Chamfer	Position, length and depth
Fillet	Position, length and radius

C.1 Region Projection

This is a general description of an area which can be used to represent an arbitrary machining surface which does not fit easily into another category. It is also a component of the region surface list. This is not illustrated here.

C.2 Region Surface List

This can be used to represented a general machining area where the surface information is used directly for calculating tool movements. It consists of a connected set of surfaces (Fig. C.1).

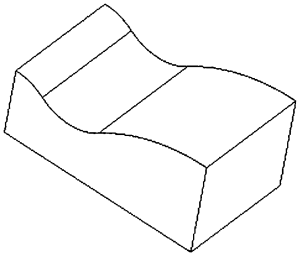


Fig. C.1 Region surface example

C.3 Planar_Face

This is an example of a feature which is seldom recognised as a shape feature but is valid because it corresponds to a manufacturing operation. It is used to define a facing-off of a particular element, for example to create a contact surface within some tolerance. It might also be used to remove a layer of material if machining from a standard-sized rectangular block to get the block down to the correct height (Fig. C.2).

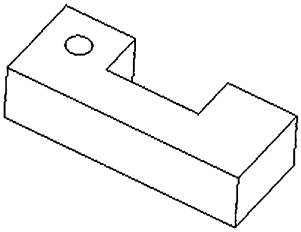


Fig. C.2 Planar face example

C.4 Closed Pocket

In a closed pocket the tool usually has to enter from the top rather than the side, although an exception is when machining is done from a casting. The material is removed downwards in layers. The pocket boundary has an arbitrary shape (Fig. C.3).

C.5 Open Pocket

Unlike the closed pocket, the open pocket has at least one open side, so the tool can enter, milling with the side of the tool (Fig. C.4).

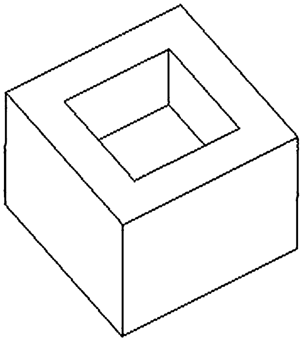


Fig. C.3 Closed pocket example

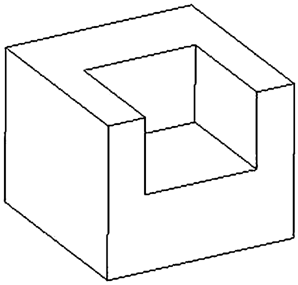


Fig. C.4 Open pocket example

C.6 Through Closed Pocket

A through pocket means that the shape goes right through the object so there is no pocket bottom (Fig. C.5).

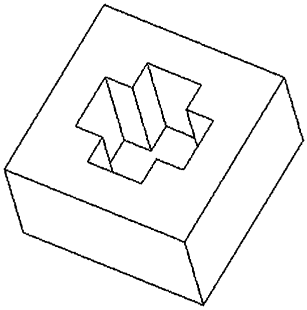


Fig. C.5 Through pocket example

C.7 Open Slot

An open slot differs from a pocket because it is generally the intention to make this with one tool pass, instead of clearing the slot area. With an open slot, the tool passes straight through the object (Fig. C.6).

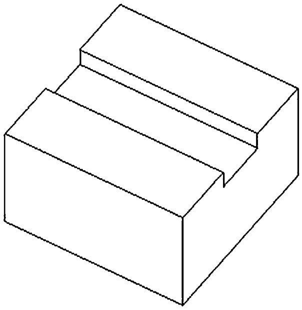


Fig. C.6 Open slot example

C.8 Half Open Slot (Round End)

A half open slot is one in which the tool enters the material but does not pass straight through. With a round end slot the diameter of the rounded end is expected to be the same as the slot width so it can be made by a cylindrical milling tool (Fig. C.7).

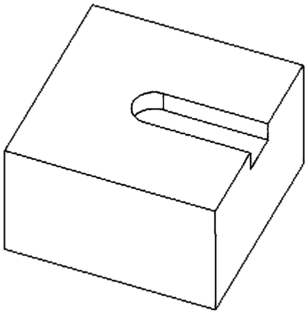


Fig. C.7 Half-open slot with round end example

C.9 Half Open Slot (Square End)

In a half-open square ended slot the end face is planar with rounded corners (Fig. C.8).

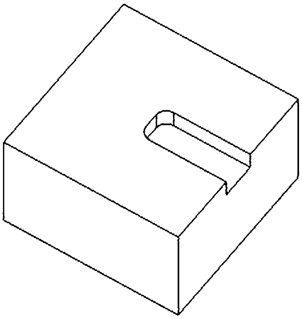


Fig. C.8 Half-open slot with square end example

C.10 Half Open Slot (Woodruff)

As before, the half-open slot has one open end and a differently shaped closed end (Fig. C.9).

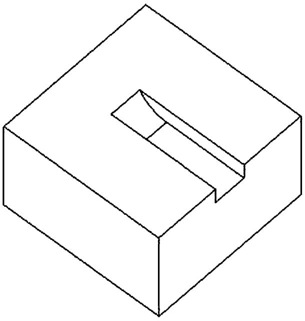


Fig. C.9 Half-open slot with Woodruff end example

C.11 Closed Slot (Round End)

In the closed slot, there is no open end through which the tool can pass, so it is similar to a pocket (Fig. C.10).

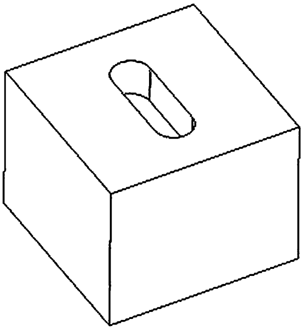


Fig. C.10 Closed slot, round end example

C.12 Closed Slot (Square End)

As with the rounded end closed slot, just with different shape at one or both ends. The end types can also be mixed with, say, a round end and a square end, or a square end and a Woodruff end, for example (Fig. C.11).

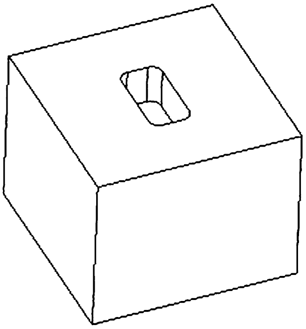


Fig. C.11 Closed slot, square end example

C.13 Closed Slot (Woodruff)

As above, again the Woodruff end provides a different shape possibility (Fig. C.12).

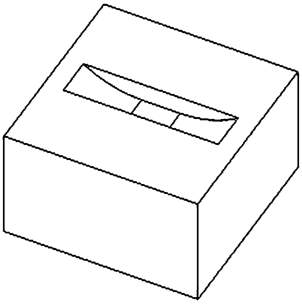


Fig. C.12 Closed slot, Woodruff end example

C.14 Loop Slot

A loop slot can be thought of as special case where the tool moves round in a closed path (Fig. C.13).

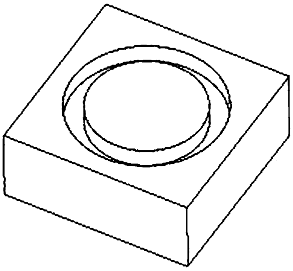


Fig. C.13 Loop slot example

C.15 Step

A step is a double level part of an object, as the name suggests (Fig. C.14).

C.16 Round Hole

Round holes are a special case because they can be made by drilling, say. The round holes may be blind or pass through the object (Fig. C.15).

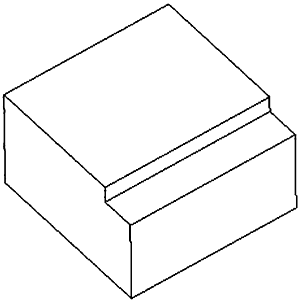


Fig. C.14 Step example

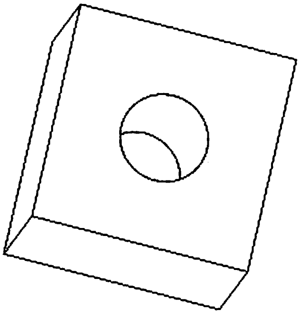


Fig. C.15 Round hole example

C.17 Rounded, Tapered Hole

As above, but the sides of the hole are not straight, suggesting that a conical tool has to be used to produce them (Fig. C.16).

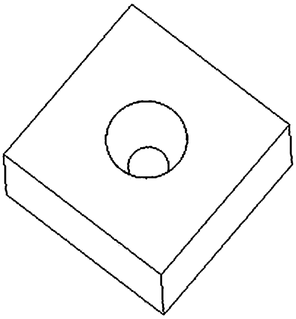


Fig. C.16 Round, tapered hole example

C.18 Outer Profile

An outer profile is the outer shape of an object (Fig. C.17).

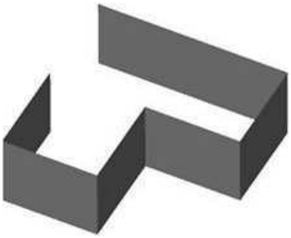


Fig. C.17 Outer profile example

C.19 Shape Profile

A shape profile is a kind of side-on profile where the shape is made by moving a tool up and down while cutting along a path (Fig. C.18).

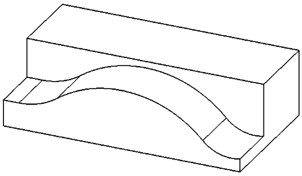


Fig. C.18 Shape profile example

C.20 Boss

A boss is not a shape to be cut but a shape to be left after cutting, a sort of blind area. It can occur in many other features, like pockets or on planar faces (Fig. C.19).

C.21 Spherical Cap

A spherical cap is a shape left on the end of a protrusion after shaping (Fig. C.20).

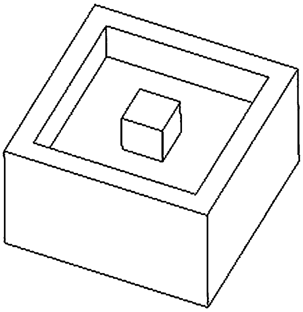


Fig. C.19 Boss example

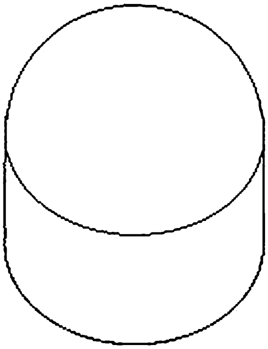


Fig. C.20 Spherical cap example

C.22 Rounded End

Another example of a shaped end after cutting (Fig. C.21).

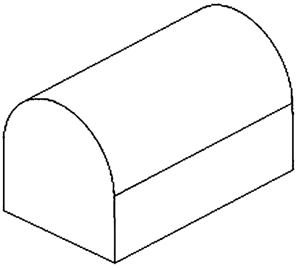


Fig. C.21 Rounded end example

C.23 Thread

A threaded, or tapped hole is usually made in two operations, one to cut the hole and the second to cut the thread. A similar operation is available for an outside thread (Fig. C.22).

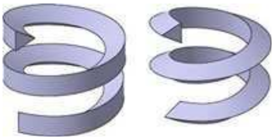


Fig. C.22 Thread example

C.24 Counterbore Hole

This is one example of a complex feature, where there are two concentric cylindrical holes (Fig. C.23).

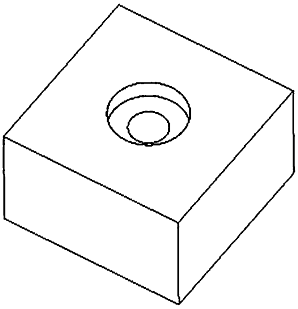


Fig. C.23 Counterbore example

C.25 Countersunk Hole

The other common example of a compound feature, where the cut in is conical rather than cylindrical (Fig. C.24).

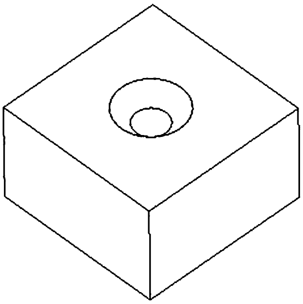


Fig. C.24 Countersink example

C.26 Circular Pattern

Circular patterns are repetitions of a simple feature around a circular path. They can be considered as a feature because the repeated single element forms a group of operations to be performed at the same time (Fig. C.25).

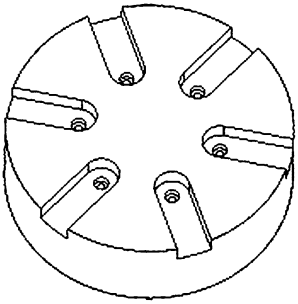


Fig. C.25 Circular pattern example

C.27 Rectangular Pattern

As for the circular pattern, but with a different layout shape of repeated element. This is again a group of operations to be performed at the same time (Fig. C.26).

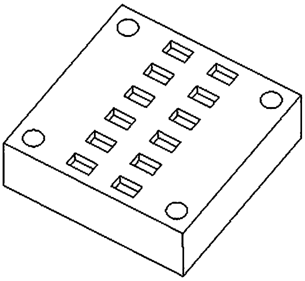


Fig. C.26 Rectangular pattern example

C.28 Chamfer

This is usually a single manufacturing operation, to remove a sharp edge and replace it with an angled face. It is made by running a tool along a path of convex edges (Fig. C.27).

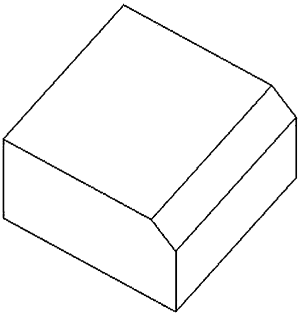


Fig. C.27 Chamfer example

C.29 Fillet

This can be a shape left by using a tool with a rounded end or a specific shape to be made, although cutting a long fillet is time consuming (Fig. C.28).

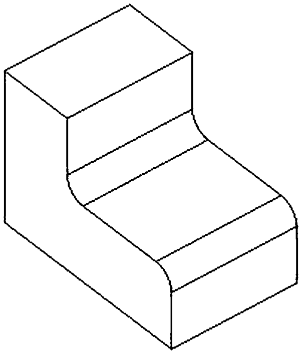


Fig. C.28 Fillet example

Appendix D
Glossary

This appendix is intended to be a summary of some of the common terms that are used in CAD and are used in this book.

2.5D This is used to describe parts which are made with a set of profiles extruded to give the object.

Application programming interface This is an interface which can be used for programming new functions. generally it consists of a set of high-level functions which perform complex operations and low-level functions which can be used for interrogation, for example.

Assembly A collection of associated objects. This might be a complete product or a mechanism, for example. Generally they are represented as a top-level assembly with a set of 'instances' of objects or sub-assemblies.

Boolean operations These are also known as "set operations" or "Boolean set operations". With the classical meaning, these combine two objects using an ADD, SUBTRACT or INTERSECT operation.

CHI See *Computer and Human Interaction*

Computer and human interaction This is a subject used for improving the user-friendliness of computer systems. This is a separate subject which is quite large. You see this as menus, but there is an implicit user model behind the CHI, which is how the system developer expects you to behave. Lack of understanding of the user and application area is a problem in CAD and leads to problems of misunderstandings in how a system should be used.

Concurrent engineering A process whereby several engineering activities are carried out at the same time, or approximately the same time. This technique is intended to save time by advising the designer about potential problems or improvements while the design is being created.

Constraints Connections, particular between elements in assemblies, denoting a relationship. A constraint might be that two planes should be coincident, or two lines should be coincident, for example. These are then taken into account to make automatic modifications to positions of elements.

Curve A geometric entity which defines the shape of an ‘edge’ in a model.

Cylinder representations Although cylinders may seem like trivial objects there have been several ways to model them. Figure D.1 shows some examples. All combinations from 0 to 3 side edges have been used for cylinders. Because these edges lie between faces in the same surface.

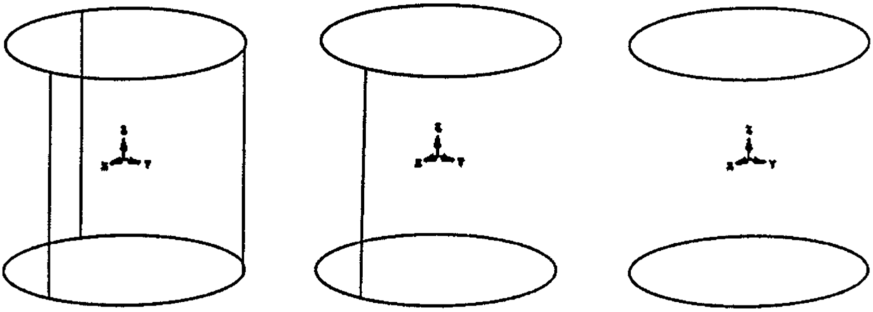


Fig. D.1 Different cylinder representation options

In the original BUILD system [1] three edges were used. This was because faces were not allowed to extend through more than 120 ° because an angle counting method was used as a point-in-face test. Other methods for point-in-face can be used to get round the problem.

CATIA v5, for example, uses two edge cylinders. This means that there are no special case edges (wire-edges) and that cylinders have two faces.ROMULUS, an ancestor of Parasolid from Shape Data, used a cylindrical representation with a single edge. The single edge allowed a ray-test point-in-face to work because the ray always cut through at least one edge.

ACIS, for example, which is a modelling kernel developed by ThreeSpace Ltd. for Spatial Technologies Inc., now part of Dassault Systems, has a minimum cylinder representation with no fake edges.

Data exchange Exchange of model information between applications and systems. Currently the most complete format is the STEP format, but others exist, such as IGES. There are also some exchange formats which are not standard.

Design for X A set of analysis methods for checking a design with respect to some application area. Some examples are design for manufacturing, design for assembly, design for disassembly, design for lifecycle.

DFX See “Design for X”

Early phase design The part of design preceding what is currently termed "CAD" or Computer-Aided Design. This is where the basic design concept is determined as well as many important decisions affecting the subsequent design steps.

Edge A topological entity lying between two faces. The shape of the edge is given by a curve which should lie on the faces adjacent to the edge.

Euler operators Euler operators are basic operations for manipulating the topological elements of a boundary representation data structure.

Euler operations See *Euler operators*

Face A face is a topological entity which defines a bounded portion of a surface. The surface is part of the boundary of an object, lying between material and non-material.

Facetting This is the term used for the subvision of faces into planar sub-faces, often, but not necessarily, triangular. This is done for graphics and also for output as STL.

Fake edges Fake edges are edges between two identical surfaces which are included for algorithmic reasons. See the explanation of cylinders, above.

Graphics user interface The graphics user interface is what application software shows you.

Graphics picking See *hit-testing*.

GUI See *Graphics user interface*.

HCI Human and computer interaction. See *Computer and human interaction*.

History tree A list of the operations that a CAD user has employed to build a model. This is not necessarily, and usually is not, a description of the part, but is really a sort-of recipe for building a model.

Hit testing A method of interactive element selection using a mouse to indicate model elements on the computer screen.

Hole loop An extra face boundary. A face is bounded by sequences of edges. Each sequence is termed a 'loop'. Often one of these is the exterior boundary of the face and the others are internal boundaries. However, in cylinder models with no side edges, one of the two loops bounding the curved face is also a 'hole-loop'.

Layout A set of simple shapes defining the rough position of the main design elements.

Local operation An operation which originally made changes based on local elements, such as edges, faces or vertices. This is opposed to an operation like a Boolean operation which operates on whole objects.

Loop A sequence of edges, closed in a complete model, which forms a boundary of a face.

Non-geometric information This is information such as material, surface finish constraints, etc. which is attached to parts of models.

Non-manifold models Non-manifold models are models which have a material thickness of zero at some point.

Parametrisation Boundary representation algorithms often use parametric characteristics of curves and surfaces.

Persistent naming An idea to identify elements in a model with a name which is adapted during model to provide a permanent identification for model command sequences.

Point in body An interrogation method for solids. This checks whether a given point lies inside an model.

Point in face An interrogation method for solids. This check whether or not a given point lies in the interior of a face.

Product model A complete model of a product comprising the geometric shape and also a variety of information, such as material, colour or surface finish, for example.

Set operations See *Boolean operations*

Sheet models Idealised models, usually of thin-plate objects, which can be changed to full volumetric models by giving them thickness.

Shell A set of connected faces, closed in a complete model. If closed then the shell bounds a portion of material of an object.

Solution catalogue A solution catalogue is a method developed by Sprumont to create a sort of database of solutions to particular design problems which can provide a basis for a complete solution.

Surface An unbounded geometric two-dimensional shape in space.

Transformation An entity defining how an object geometry should be changed (often rotated, scaled or translated).

Vertex A topological entity lying at a position in space. Vertices define the end points of edges.

Wire edge An edge with the same face on both sides. These can occur temporarily during operations but are not usually part of the finished model. An exception is if cylinders are represented with a single edge on the curved face.

Reference

1. Braid, I.C.: Notes on a Geometric Modeller. CAD Group Document 101. Cambridge University Computer Laboratory (1979)

Index