FLOW VISUALIZATION

Visualization has become an essential part of scientific and engineering practice to help analyze the massive data fields being generated from supercomputer simulations and laboratory observations. A *field* is any physical quantity, such as density or force, for which a value is defined at every point of a given spatial domain. Data fields can consist of discretized scalar, vector, or tensor quantities or any combination thereof. Examples include scalar intensity fields from medical scanner magnetic resonance imaging (MRI), computerized tomography (CT), velocity vector fields from computational fluid dynamics simulations ("flow fields"), and stress tensor fields from structural mechanics.

For visualization, the abstract physical parameters are mapped into visual parameters such as shape, structure, color, and texture, so that the scientist or engineer can perceive meaningful patterns and understand the underlying physical process. To achieve these mappings, we can link the field quantities directly to the visual primitives, we can derive geometric objects (curves, surfaces, or solids) from data fields, or we can extract topological structures.

In this article, we will discuss the area of *flow visualization,* which is the application of visualization techniques to steady and unsteady (time-varying) flow fields. The focus will be on visualization techniques for vector, tensor, and time-varying scalar flow datasets. Our main source of data is computational fluid dynamics (CFD), and many of the physical concepts and analogies underlying the visualization techniques are derived from this domain. Also, fluid dynamics has a long and rich experimental tradition, in which visualization plays a major role. Experimental visualization in fluid dynamics has been (and still is) a strong inspiration to research in scientific visualization. For information on general volume visualization see VOLUME VISUALIZATION. For an overall text on visualization which includes vector field visualization, see Ref. 1.

We will first describe the main characteristic of the data fields, and then we will describe some basic operations on data fields such as transformations, interpolations, and gradient computations. We will then describe various visualization techniques for flow fields, such as arrow plots, stream curves and surface generation, texture-based flow rendering, vector field topology, tensor field visualization, and feature tracking.

DATA FIELDS: PROPERTIES AND REPRESENTATIONS

Basic Field Types

A three-dimensional field can be represented analytically by a global function $f(x, y, z)$, defined over a bounded spatial domain in R^3. A field value s at every point (x, y, z) of the domain can be found by evaluating: $s = f(x, y, z)$. This is usually not the case with the discrete numerical fields that are more common in science and engineering, where data values are known only at a large but finite number of data points. Such discrete fields are usually generated by numerical computer simulations and data sensing systems. (For simplicity, we will assume in the rest of this article that the data fields have been generated by numerical simulations, but most discussions will apply to measured data fields as well.)

Physical models often cannot be solved analytically. Thus, discrete methods such as finite-element, finite-difference, or finite-volume methods are often used to numerically solve systems of partial differential equations. These methods are based on defining a computational grid. Approximate equations are specified, resulting in a system of equations that can be solved numerically at each grid node.

The domain of a simulation may have two or three spatial dimensions. It may also be variable in time. The data points (or grid points) thus are two-dimensional (2-D) (x, y) or three-dimensional (3-D) (x, y, z) coordinate positions. The data fields may contain any combination of scalar quantities (e.g., pressure, density, or temperature), vector quantities (e.g., force or velocity), or tensor quantities (e.g., stress or deformation) at each data point. The data values may be constant, or they may vary as a function of time. Time-dependent fields are important for highly dynamic phenomena such as fluid flow.

Grid Types

There are many types of computational grids, depending on the simulation technique, the domain, and the application. A grid consists of nodes and cells. The nodes are points defined in the simulation domain, and the cells are simple spatial elements connecting the nodes: triangles or quadrangles in 2-D, tetrahedra or hexahedra in 3-D. The cells must fill the whole domain, but may not intersect or overlap, and adjacent cells must have common edges and faces. Grids can be classified according to their geometry, their topology, and their cell shape. Three of the most important types are shown in Fig. 1. The simplest type is the regular orthogonal (or Cartesian) grid [Fig. 1(a)]. This type of grid has a regular geometry and topology; the nodes are spaced in a regular array, and the cells are all unit cubes. The grid lines connecting the nodes are straight and orthogonal. Every node can be referenced by an integer index vector $\boldsymbol{i}(i, j, k)$. Adjacent nodes can be found by incrementing any of the index vector components. Many operations on this type of grid (such as searching the grid cell which contains a given point) are very simple, but grid density is constant throughout the domain, and the shape of the domain must be rectangular.

The second type of grid is the structured, curvilinear grid [Fig. 1(b)]. This type has a regular topology (the adjacency pattern for each internal node is the same), with the nodes again referenced by a 3-D index vector $\boldsymbol{i}(i, j, k)$, and adjacent nodes can be found by incrementing index values. The cells are usually hexahedra, with a deformed-brick shape. The geometry of each cell is irregular, and the cell faces are nonplanar quadrangles. The cell size of a curvilinear grid can be highly variable, and thus the resolution of the simulation can

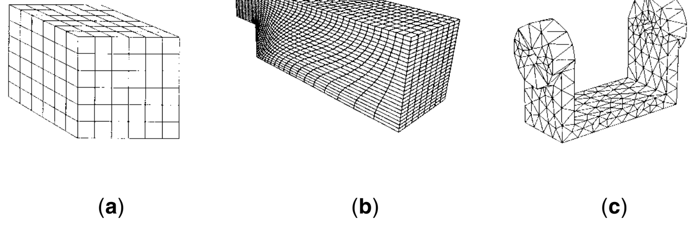
(a) (b) (c)

Figure 1. Three types of grids.

be higher in areas of strong variation. Also, the curvilinear shape can be made to conform to the boundary of a curved object, such as an airplane wing. This type of grid is common in finite-volume CFD simulations.

The third type of grid is the unstructured grid [Fig. 1(c)], where the topology and geometry are both irregular. The nodes do not have a fixed adjacency pattern, and adjacency information cannot be derived from a spatial index, but has to be stored explicitly. The cells are usually triangles in 2-D or tetrahedra in 3-D. Cell size can be varied according to the amount of detail desired, and they can be used to model a complex geometry. Unstructured grids are often used in finite-element analysis. Due to the simple cell geometry, calculations on a single cell are simple.

There are many more variations of grids: staggered grids, hybrid (mixed-type) grids, multiblock grids, moving grids, and multiresolution grids. In this article, we will concentrate mainly on static 3-D Cartesian and structured curvilinear grids.

A numerical solution will generally produce a discrete data field, consisting of a combination of scalar, vector, or tensor quantities, given at every grid point. These datasets can be very large, with as many as 10^4 to 10^6 nodes and with 10 or more variables defined at every node. This results in a size of 10 Mbytes to 100 Mbytes for constant (time-independent) fields and several gigabytes for time-dependent fields.

BASIC OPERATIONS IN GRIDS

Now that we have defined some characteristics of the fields we are working with, we go on to describe the following basic operations that can be performed on these grids:

- Interpolation
- Grid transformation
- Grid traversal and point location
- Gradient computation

These operations are the building blocks for more complex algorithms, and they will be described in the following subsections.

Interpolation

The field value at an arbitrary point X in the domain can be found by interpolation between data values at surrounding grid nodes (since the underlying assumption is of a physical continuum). Interpolation can be considered as a local approximation function fitted to the data at the grid points. Piecewise constant or linear interpolations are often used as a "minimum" assumption about the intermediary field. With piecewise constant (zero order) interpolation, the field value at a point X in a cell is taken either equal to the nearest grid node (nearest-neighbor interpolation) or as an average of the surrounding grid nodes. In these cases, the resulting field is discontinuous. With linear (first-order) interpolation, a linear variation of the field is assumed between the data values at the surrounding grid nodes. An example of trilinear interpolation is shown in Fig. 2. If the field values are uniquely defined in a face common to two neighboring cells, then the resulting field is C^0 continuous. Higher-order interpolations using quadratic or cubic basis functions provide higher orders of continuity, but these are far less common, because they make more assumptions on the field between the nodes, and they are much more expensive to calculate.

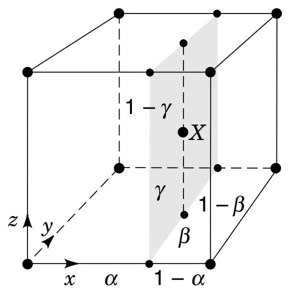

Figure 2. Trilinear interpolation.

Grid Transformation

Curvilinear grids are widely used in CFD because of their flexibility for modeling physical boundaries. However, it is more difficult to perform common mathematical operations (such as interpolation and point location) on these grids. Therefore, transformations are used to convert the physical space (P space or $\mathscr{P}$) to a computational space (C-space or $\mathscr{C}$) (see Fig. 3). The transformation between the two domains can be performed in both directions, and positions and vector values are transformed (scalar values are not transformed since they are independent of the underlying spatial grid).

For some curvilinear grids, it is possible to define a global transformation which maps an entire grid to the new domain. This is the case for grids with simple parameterized geometries (e.g., cylindrical or spherical grids) or for grids that are defined by a transfinite parametric mapping. In the general case, a local transformation is defined for each cell.

Point Transformation. Points can be transformed from $\mathscr{C}$ to $\mathscr{P}$ by mapping the corner nodes of a cubic cell in $\mathscr{C}$ to the corner nodes of a curvilinear cell in $\mathscr{P}$ and by interpolating all the points in between (2). Let point $\boldsymbol{\xi} = (\xi, \eta, \zeta)$ be a point in $\mathscr{C}$, whose coordinates may be split into an integer part $I = (i, j, k)$ and a fractional part $\alpha = (\alpha, \beta, \gamma)$, with $0 \leq \alpha, \beta, \gamma \leq 1$. In addition, let $\boldsymbol{c}(I)$ be the coordinates of grid node $I = (i, j, k)$. Now, we can transform $\boldsymbol{\xi}$ in $\mathscr{C}$ to $\boldsymbol{x}$ in $\mathscr{P}$ by interpolating

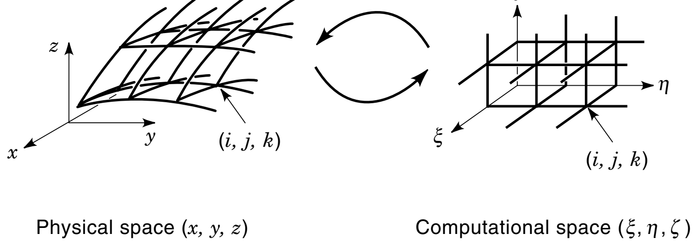

Figure 3. Transformation between $\mathscr{P}$ and $\mathscr{C}$.

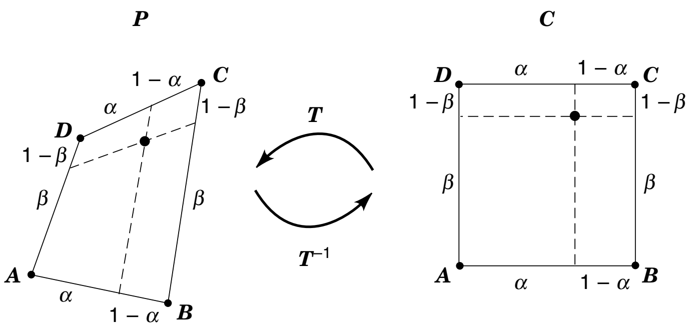

Figure 4. Transformation through interpolation.

(see previous section) the $\mathscr{P}$ coordinates of the corner nodes $I(i, j, k)$, using the local offsets $\boldsymbol{A} = (\alpha, \beta, \gamma)$:

$$\boldsymbol{x} = \mathscr{T}(\boldsymbol{\xi}) = \mathscr{I}_{\text{tri}}(\boldsymbol{A}, c(I)) \tag{1}$$

Figure 4 illustrates this principle for a 2-D cell.

The inverse transformation involves finding $\boldsymbol{A}$ given some point X. This is more complex, since an explicit expression for $\boldsymbol{A}$ cannot be given; instead, a Newton–Raphson iteration could be used to find their values.

Vector Transformation. A vector $\boldsymbol{v}_c$ in $\mathscr{C}$ is transformed to $\boldsymbol{v}_p$ in $\mathscr{P}$ using the equation

$$\boldsymbol{v}_p = \boldsymbol{J} \cdot \boldsymbol{v}_c \tag{2}$$

Similarly, a vector $\boldsymbol{v}_p$ in $\mathscr{P}$ is transformed to $\boldsymbol{v}_c$ in $\mathscr{C}$ with

$$\boldsymbol{v}_c = \boldsymbol{J}^{-1} \cdot \boldsymbol{v}_p \tag{3}$$

Here, $\boldsymbol{J}$, called the *metric Jacobian,* is a matrix representing the cell deformation. This matrix contains the partial derivatives of the transformation $\mathscr{T}$:

$$\boldsymbol{J} = \begin{pmatrix} x_\xi & x_\eta & x_\zeta \\ y_\xi & y_\eta & y_\zeta \\ z_\xi & z_\eta & z_\zeta \end{pmatrix} \tag{4}$$

where x_ξ is short for $\partial x/\partial\xi$, and so on. These derivatives may also be considered groupwise, since the columns $(\boldsymbol{j}_1 \mid \boldsymbol{j}_2 \mid \boldsymbol{j}_3)$ of $\boldsymbol{J}$ are in fact the partial derivatives $\partial\boldsymbol{x}/\partial\xi$, $\partial\boldsymbol{x}/\partial\eta$, $\partial\boldsymbol{x}/\partial\zeta$. The calculation of transformation Jacobians may be done in two ways: Jacobians may be calculated directly using continuous derivatives, or they may be approximated using finite differences. Since most of the grids are not given in analytical form, finite differences are used. For a full discussion of the various finite-difference methods, please see Ref. 3.

Grid Traversal and Point Location

Grid traversal is a problem that occurs in visualization techniques such as ray casting and particle tracing. A subproblem of grid traversal is *point location,* which may be defined as the process of finding which cell contains a given point. In ray casting, a ray traverses a grid containing a scalar field, which is sampled at subsequent positions. The value of a sample is determined by interpolating the field values at the corners of the current cell containing that position. Therefore, it must be determined which cell is the current one. In particle tracing, a particle traverses a grid containing a velocity field, which is sampled at subsequent positions visited by the particle. Again, the value of a sample is determined by interpolating the field values at the corners of the current cell, so that here, too, it must be determined which cell is the current one.

We can distinguish between *global* and *incremental* point location. In global point location, a given point in a grid must be found without a previous, known cell. In a curvilinear grid, this is not an easy task. As with all search algorithms, it is possible to use a simple brute-force algorithm which searches all grid cells one-by-one, but this is clearly very expensive. Auxiliary data structures can be used to speed up this search (4,5).

Fortunately, in many visualization techniques there is a previous known position in a previous known cell. Starting from there, a new position is to be found. This is called *incremental point location.* Two possible approaches for this problem are *stencil walk* (6) and *tetrahedrization* (7). The stencil walk approach is a recursive algorithm that begins with a guess at an initial point in computation space. That point is transformed to physical space, and the difference vector between that and the target point is calculated. This vector is then transformed back to computational space and added to the previous point, resulting in a new guess. This process is repeated until the right cell has been found. In the tetrahedrization approach, the hexahedral grid cells are broken up into tetrahedra. A line from the previous known position to the new position is drawn. This line intersects the faces of adjacent tetrahedral, thereby identifying adjacent cells in which containment tests can be performed to find the new point.

Gradient Computation

Gradient quantities play an important role in visualization in two ways: Either they are visualized directly, or they are used as part of another visualization technique. Gradient quantities are typically derived from quantities given in the data field, using the nabla operator $\nabla = (\partial/\partial x, \partial/\partial y, \partial/\partial z)$, which is applied in one of the following ways:

- The *gradient* of a scalar field f is a vector field: $\nabla f = (\partial u/\partial x, \partial v/\partial y, \partial w/\partial z)$.
- The *gradient* of a vector field $\boldsymbol{v}$ is a (second-order) tensor field:

$$\nabla\boldsymbol{v} = \begin{pmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ w_x & w_y & w_z \end{pmatrix}$$

  where $\boldsymbol{v} = (u, v, w)$ denote the vector components, and u_x is short for $\partial u/\partial x$, and so on.
- The *divergence* of a vector field $\boldsymbol{v}$ is a scalar field: $\nabla \cdot \boldsymbol{v} = \partial u/\partial x + \partial v/\partial y + \partial w/\partial z$.
- The *rotation* (or curl) of a vector field $\boldsymbol{v}$ is a vector field:

$$\nabla \times \boldsymbol{v} = \begin{pmatrix} \dfrac{\partial}{\partial x} \\ \dfrac{\partial}{\partial y} \\ \dfrac{\partial}{\partial z} \end{pmatrix} \times \begin{pmatrix} u \\ v \\ w \end{pmatrix}$$

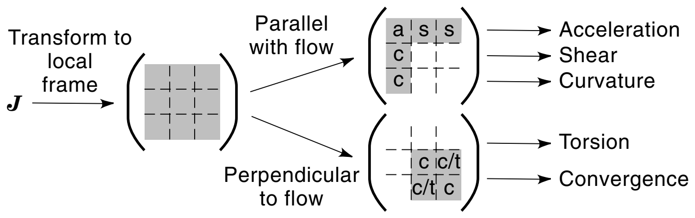

Figure 5. Decomposition.

In fluid flows, the rotation of a velocity field $\boldsymbol{v}$ is called the *vorticity* $\boldsymbol{\omega}$.

A useful application of gradient quantities is in flow fields, where meaningful flow properties may be derived from the velocity gradient. The velocity gradient tensor $\boldsymbol{J}$, calculated as shown above, may be decomposed into two components:

$$\boldsymbol{J} = \boldsymbol{J}_s + \boldsymbol{J}_a = \frac{1}{2}(\boldsymbol{J} + \boldsymbol{J}^{\mathrm{T}}) + \frac{1}{2}(\boldsymbol{J} - \boldsymbol{J}^{\mathrm{T}}) \tag{5}$$

Here, the symmetrical tensor $\boldsymbol{J}_s$ represents the deformation of an infinitesimal fluid element, and the antisymmetrical tensor $\boldsymbol{J}_a$ represents its rotation.

An alternative decomposition of the velocity gradient tensor, given in Ref. 8, is based on a local coordinate frame with the x axis parallel to the local velocity vector, and the other two axes are defined as a Frenet frame. This allows us to determine several useful flow properties both parallel with the flow and perpendicular to it, as shown in Fig. 5.

As the x axis is defined parallel with the direction of the flow, the *acceleration* in the direction of the flow simply becomes the element u_x of the velocity gradient tensor. *Torsion* around the velocity axis is given by the x component of the rotation $\boldsymbol{\omega}$. The *curvature* at a point of a streamline may be visualized using the osculating circle, as shown in Fig. 6(a).

The other two properties may be visualized with a plane perpendicular to the flow. *Shear* in the direction of the flow is represented by the change of *orientation* of this plane. Figure 6(b) shows a reference plane and the changed orientation caused by the local flow. On the other hand, *convergence/divergence* of the flow is represented by the change of *shape* of the plane, as shown in Fig. 6(c).

VISUALIZATION

In what follows, we discuss a variety of visualization techniques for vector, tensor, and time-varying flow datasets. The visualization techniques discussed can be divided in three groups:

- *Global (or Direct) Techniques.* A qualitative visualization of a whole data set, or a large subset of it, at a low level of abstraction. Scalar fields can be visualized globally using direct volume rendering or isosurface extraction (1). One simple method to visualize a vector field is to reduce the vector to a scalar value, such as the magnitude of the vector. Directional information is lost, but this can still be useful for many purposes. Also, the scalar (magnitude) field can be used as an enhancement to many of the techniques listed below, using color mapping, or by using thresholding to select parts of the dataset. Examples of global techniques for vector fields are arrow plots or "hedgehogs" (see section entitled "Arrows and Hedgehogs") and texture-based visualization (see section entitled "Texture-Based Vector Field Visualization").
- *Geometric Techniques.* Generation and visualization of geometric objects such as curves, surfaces, and solids, of which the shape is directly related to the data field. Definitions, generation, and visualization of flow curves will be described in the sections entitled "Flow Curves," "Integral Curve Generation," "Curve Generation," and "Curve Visualization." Flow surfaces will be discussed in the sections entitled "Surface Definitions," and "Stream Surface Generation."
- *Feature Extraction and Tracking.* High-level entities (features) are extracted from large datasets, resulting in representations that are directly related to the concepts of the application. Examples of flow features are vortices and shock waves. In feature-based visualization the most relevant information is selected, which can lead to a large reduction of the data. Features are characterized by quantitiative measures, thus emphasizing quantification for precise evaluation and comparison. An example is the extraction of flow field topology, discussed in the section entitled "Flow Field Topology." In time-dependent flow simulations, the dynamics of features is studied by tracking their evolution in time, thus extracting the temporal behavior and important events. This topic will be discussed in the section entitled "Feature Extraction and Tracking."

Arrows and Hedgehogs

Vector fields can be directly displayed with arrow plots or hedgehogs (oriented lines anchored by a point). The arrow/line is drawn from the location of the vector and the direction of the vector determines the direction of the arrow or line

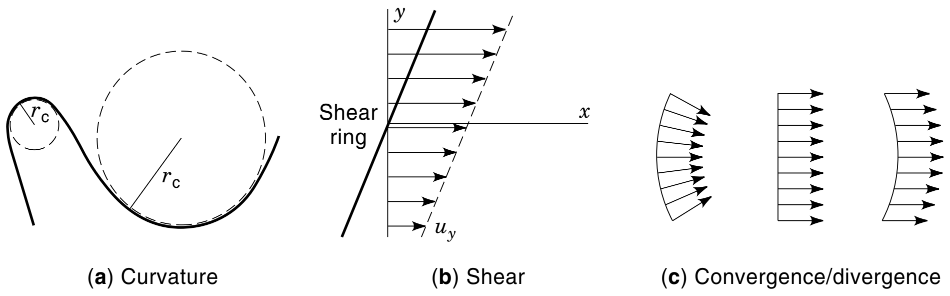

Figure 6. Three components of the velocity gradient.

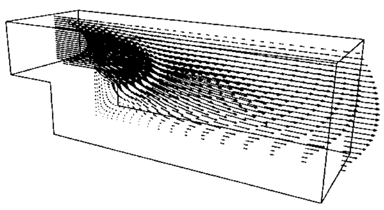

Figure 7. Cluttered vector plot.

segment. The arrow is also scaled and/or colored by the magnitude of the vector. Thresholding can be used to restrict the arrow plots to certain portions of the datasets to avoid clutter. These techniques work reasonably well with 2-D vector fields, but in 3-D the arrows are ambiguous and the images are cluttered so that very little useful information is displayed (Fig. 7). Other types of glyphs or icons can also be used (9). Figure 8 shows one type of icon to interrogate and visualize different variables in the flow field (8). Related global visualization methods are the texture-based techniques described in the section entitled "Texture-Based Vector Field Visualization." The effect of a vector field can also be seen by placing a geometric object, such as a plane, in certain locations in the field and "warping" the plane according to the vectors at that location (1,10).

Flow Curves

A *field line* or *tangent curve* is a curve that is everywhere tangent to a vector field. The different types of flow curves and their definitions are listed below.

- *Streamline.* A tangent curve in a steady velocity field. The curve satisfies the equations $dx/u = dy/v = dz/w$, where (u, v, w) are the velocity components in the x, y, and z direction of the domain.
- *Streak Line.* A line joining the positions at one instant of all particles that have been released from a single point.
- *Particle Path.* A trajectory curve of a single fluid particle moving in the flow. This curve is identical to an integral curve, obtained by stepwise integration of the velocity vector field.

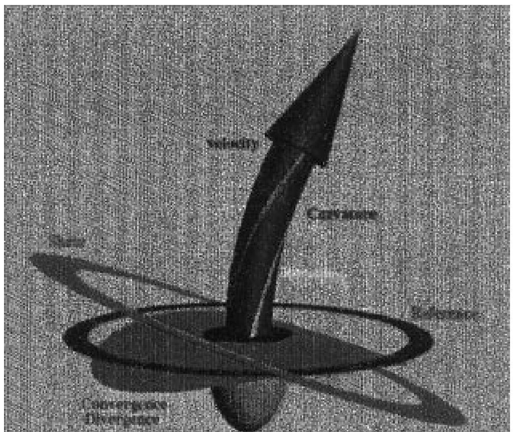

Figure 8. Flow probe.

- *Time Line.* A line connecting all particles that have been simultaneously released in a flow from positions on a straight line, perpendicular to the flow direction. The straight line moves and deforms with the flow due to local velocity variations.
- *Vorticity Line.* A field line of a vorticity vector field.
- *Hyperstreamline.* A field line of an eigenvector (usually with the largest magnitude) of a tensor field (11).

Most of the definitions are based on the notion of *particle advection,* or particles moving in a flow. Streaklines and time lines have been derived from experiments. Streamlines, vorticity lines, and hyperstreamlines are mathematical abstractions, but they are all based on the idea of field lines. In a steady (stationary, time-independent) flow, streamlines, streak lines, and particle paths are identical (2). In an unsteady (instationary, time-dependent) flow, these curves are all different, but they can all be generated in a straightforward way using integral curve algorithms.

Curve Integration Methods (Integral Curve Generation)

Most of the curves described above (streamlines, pathlines, etc.) are based upon the same principle: They are generated by integrating a vector field. The only thing that distinguishes these curves is the underlying vector field used to calculate them: For streamlines, pathlines, streaklines and timelines, a velocity field $\boldsymbol{v}$ is used; for vorticity lines a vorticity field ($\boldsymbol{\omega} = \nabla \times \boldsymbol{v}$) is used, and for hyperstreamlines a tensor field is used.

All the curves are generated with the same basic algorithm: Starting in some specified initial position, a stepwise, numerical integration is performed, yielding a sequence of positions through which a curve may be fitted. This is described by the equation

$$\boldsymbol{x}(t) = \int_t \boldsymbol{v}(\boldsymbol{x})\,dt \tag{6}$$

where t denotes time, $\boldsymbol{x}$ the current position, and $\boldsymbol{v}(\boldsymbol{x})$ the vector field. The initial condition for the equation is provided by the initial position x_0. The solution is a sequence of positions $(\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots)$.

For the numerical integration, the standard integration methods found in the literature may be applied, such as the first-order Euler method and the second-order Runge-Kutta method (also known as the Heun method). The first-order Euler method is given by

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \Delta t \cdot \boldsymbol{v}(\boldsymbol{x}_n) \tag{7}$$

The second-order Runge–Kutta method uses the first-order Euler method to determine an estimate $\boldsymbol{x}^*_{n+1}$. This is then used to compute $\boldsymbol{x}_{n+1}$ using the equation

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \Delta t \cdot \frac{1}{2}\{\boldsymbol{v}(\boldsymbol{x}_n) + \boldsymbol{v}(\boldsymbol{x}^*_{n+1})\} \tag{8}$$

Generally, the Runge–Kutta technique is used because it is more accurate with an error of $O(\Delta t^3)$ as opposed to the Euler method which has an error on the order of $O(\Delta t^2)$. Higher-order techniques can also be used.

The time step Δt is used for the integration and can be either fixed or adaptable. For animation purposes (e.g., in rendering particles in a velocity field at subsequent positions), a fixed-time step is used since equidistant time intervals are required for a smooth animation (12). If the shape of the integral curve is most important, then adaptive step sizes may lead to more efficient computations and more accurate results. The cell size or path curvature can be used to determine step size. In parts of the grid where cells are small and nodes are closely spaced together, high gradients may occur, so smaller integration steps are better in order not to step over cells and thus miss important data. Furthermore, in regions of high curvature, it is better to space subsequent points of the curve more closely together, while in regions of low curvature the distance between the points may be increased to save computing time (13).

Curve Generation

In fluid dynamics, the local transformation of a structured curvilinear grid cell to a unit cube cell is common practice. Many flow simulations use this transformation to perform calculations in a regular Cartesian grid in computational space C. For integral curve generation (or particle tracing), there are algorithms operating in physical space P, as well as algorithms using the transformation to C. Assuming that a stationary velocity field is defined in P, the general form of the *P-space algorithm* is:

```
find cell containing initial    (point location)
  position
while particle in domain do
      determine velocity at     (interpolation)
        current position
      calculate new             (integration)
        position
      find cell containing      (point location)
        new position
end while
```

The general form of the *C-space algorithm* is:

```
find cell containing initial    (point location)
  position
while particle in domain do
      transform corner          (transform
        velocities of cell       vector)
        from P to C
      determine C-velocity      (interpolation)
        at current position
      calculate new position    (integration)
        in C
      transform C-position      (transform
        to P                     point)
      find cell containing      (point location)
        new position
end while
```

Point location is much simpler in C-space, but in 3-D fields eight vector transformations and one point transformation must be performed. If the vector field is defined in C, then the vector transformations are not necessary, and the C-space algorithm is obviously the best choice. If the vector field is defined in P, then a different transformation is applied to each cell and continuity of the vector field at the cell faces is lost in C. This can lead to errors, especially when a cell face is crossed in the integration step, at a sharp discontinuity of the grid [see Fig. 9(a)]. A typical result of a C-space algorithm is shown in Fig. 9(b).

With P-space algorithms, point location is done directly in P, using either tetrahedral decomposition, or the stencil walk algorithm. Continuity of the vector field is retained, and no errors occur even at sharp discontinuities of the grid [Fig. 9(c)]. The more complex point location in the curvilinear grid is easily compensated by savings on the transformations, and thus these algorithms are more efficient (3,13).

Integral curves start from a user-specified initial point or *seed point*. As the information carried by a single integral curve is local, selection of these seed points is crucial. Important features of the data field may be overlooked by improper seed point selection. Interactive selection is not always possible due to the expensive curve computations. Therefore, algorithmic selection techniques have been developed to locate seed points in areas of special interest (14). For a global view, a large number of seed points can be placed throughout the whole field. An advanced method for doing this is described in Ref. 15.

Particle tracing in instationary or time-dependent fields is different because velocity is sampled both in space and time during integration. This means that a time stamp must be kept for each particle. The particle position is located in the grid cell, and its time stamp is located in an interval between two time steps. Temporal interpolation between two time steps is performed to determine instantaneous velocity. Special care must be taken with moving grids or time-dependent changes in grid geometry. Also, data management of very large time-dependent vector fields is not a simple task. For example, if a data set consists of a separate velocity field for each time step, interpolation in time requires memory references to two different fields for each particle at each step.

Streamlines are only significant for steady flows, because they are only defined at one instant and do not behave coherently in time-dependent flows. A streakline can be generated by (1) releasing a stream of particles from one given point at regular time intervals and (2) joining these particles by line segments. In time-dependent flow fields, streaklines show time-coherent deformations, and thus they are suitable for visualization (13). A time line at time t is determined by tracing a line of particles that were all released simultaneously at time t_0, and again connecting the same particles at time t. Time lines are also useful for visualization of time-dependent flow fields.

Curve Visualization

Visualization of curves is straightforward, using line drawing. Collections of curves can be used to visualize local variations, such as divergence and rotation.

Figure 10 shows an example of streamlines in a stationary velocity field. Although curves show local flow characteristics, an impression of the global structure of the flow field can be obtained using a large number of streamlines. The direction of motion is not shown by the curves, but this can be added

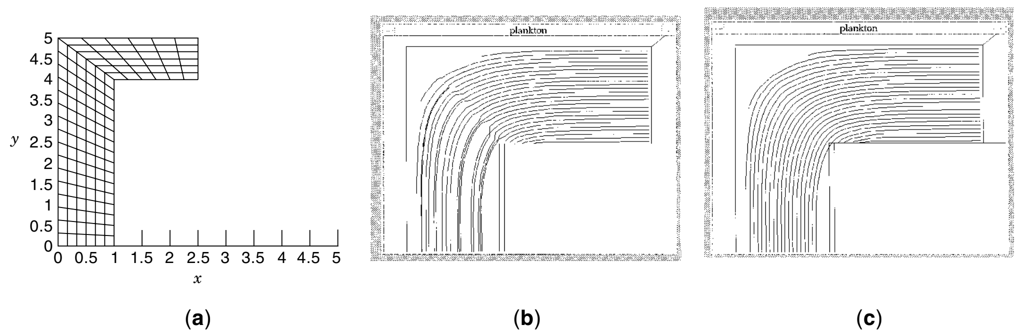

Figure 9. *C*-space and *P*-space algorithms. (a) Grid with sharp discontinuity. (b) Result of *C*-space algorithm. (c) Result of *P*-space algorithm.

by coloring the curve at the inflow or by drawing an arrow on the curve.

Particle animation can provide the "motion" that is missing from a static image of a vector field (16). Particle positions at regular intervals are computed and displayed on the screen. To show accurate velocity of the moving particles, the display update rate must be constant (and not dependent on computation time). Therefore, particle positions are usually precomputed.

Curve rendering and particle animation can be combined (17). Streamlines are generated by calculating positions at constant time intervals. Thus each streamline can be rendered as a series of line segments corresponding to equal time spans. If the color index of these line segments is alternated between different values, then a motion effect along the streamline can be obtained by cyclic changes of the color table of the display system. If a large number of streamlines are precomputed, this technique can be used for interactive exploration.

Playback animation can also be used, either displaying the particles directly at each time step or displaying prerendered images. For stationary flows, *closed-loop* animation can be employed, repeatedly showing a cycle of about 10 frames. The display intensity of a particle varies over a limited life span, starting at zero, increasing to a maximum, and decreasing again to zero. If the birth times of the particles are uniformly spread over the animation time, and wrap around from the last to the first frame, the animation will appear continuous, without any jumps between the cycles.

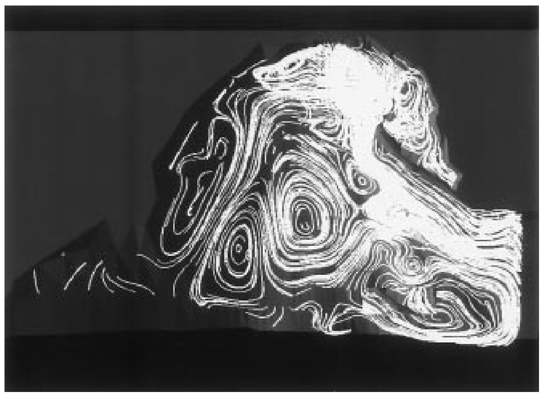

Figure 10. Streamlines in a stationary velocity field. (Data from Delft Hydraulics.)

For unsteady flows, this technique is not feasible, and *open-loop* animation must be employed. A larger number of frames must be precomputed at regular simulation time intervals, and displayed at a constant update rate. For large numbers of particles, playback animation is necessary because of the heavy computational load. For interactive use, particle positions must be computed and displayed in real time, which is possible with a small number of particles in a stationary flow field.

Particle paths are usually smooth curves, reflecting a continuous velocity field. This is true for laminar, convective flow fields. However, in turbulent flows, where small-scale fluctuations of velocity occur, the curves will not be smooth. Special visualization techniques have been devised for turbulent flows (12), showing the jagged, irregular paths dues to random fluctuations in particle motion. Animation shows the erratic motions of particles.

Surface Definitions

Curves are difficult to visually locate in 3-D because no spatial depth cues are available (rotating the image or viewing the image in stereo can usually help). The curves discussed previously can be made into surfaces for better visualization. The tangent curve can be extended to a *tangent surface,* a surface that is everywhere tangent to the vector direction. In a stationary velocity field, a tangent surface is called a *stream surface.* As the velocity direction is everywhere tangent to the stream surface, the velocity component normal to the surface is everywhere zero. This means that no material flows through a stream surface, so it can be considered as a separation between two independent flow zones. Time lines can be generalized to *time surfaces,* connecting particles that have been simultaneously released from positions on a plane. The other types of curves can be similarly extended to surfaces; we will restrict this discussion to stream surfaces.

Stream Surface Generation

The simplest type of stream surface is a *ribbon,* or a narrow band. Besides local flow direction, it can show the local rotation of the flow. Ribbons can be generated in different ways. First, two adjacent streamlines can be generated from two seed points placed close together, and then a mesh of triangles can be constructed between them. The width of the ribbon depends on the trajectories of both streamlines, and it may become large in a strongly divergent area. A second way is to construct a surface strip of constant width centered around a single streamline. The orientation of the strip is directly linked to the angular velocity of the flow, obtained from the vorticity. From the angular velocity a rotation angle can be found by time integration along the streamline (18). The initial orientation is defined at the seed point, and an incremental rotation is applied in a local coordinate frame at each point on the streamline. The ribbon is constructed by weaving a strip of triangles between the points. The first method can show the vortical behavior of the flow and the divergence by varying the width of the ribbon. The second method shows purely local vortical behavior on the central streamline. In both cases, the surface is not an exact stream surface, and the tangency condition is only true for the constructing streamlines.

A general stream surface can be constructed by generating streamlines from each of a number of points on an initial line segment or *rake.* If for all these streamlines a single constant time step is used, then the lines connecting points of equal time on all streamlines are time lines. Streamlines and time lines thus make a quadrangular mesh (see Fig. 11), which can be easily divided into triangles for visualization.

If the flow is strongly divergent, adjacent streamlines will move too far apart and if there is an object in the flow, the surface must be split. Finally, if there are high-velocity gradients in the flow direction, the mesh will be strongly distorted and unequal-sized and poorly shaped triangles will result.

To solve these problems, an *advancing front* algorithm has been proposed (19). The surface is generated in the transverse direction by adding a strip of triangles to the front. When we use adaptive time steps to compensate for the gradients in the flow direction, all points on the front will move forward by about the same distance. Also, if two adjacent points on the front move too far apart by divergence, a new streamline will be started at the midpoint between them. Conversely, if two points move too close together, one streamline will be terminated. If an object in the flow is detected, the front can be split, and the two parts can move on separately.

A stream surface can also be modeled as an implicit surface $f(\boldsymbol{x}) = C$ (20). The stream surface must satisfy the condition $\nabla f \cdot \boldsymbol{v} = 0$, which means that the normal to the surface (denoted by the gradient ∇f) is perpendicular to the velocity direction. The function f is called the *stream surface function* and are specified at the inflow boundaries of the flow area. All other grid points for the values of f are calculated numerically, either by solving the convection equation or by tracing backwards from each grid point to the inflow boundary. A stream surface is then generated as an isosurface of f [this type of technique can also be used for time surfaces (19)].

Another way to show the depth of a streamline is to use a *stream tube.* Each tube icon is a generalized cylinder, of which

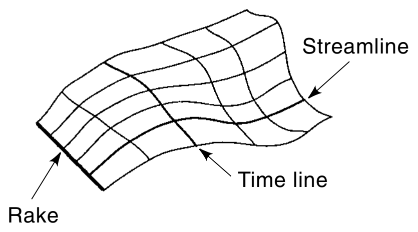

Figure 11. Mesh for a stream surface.

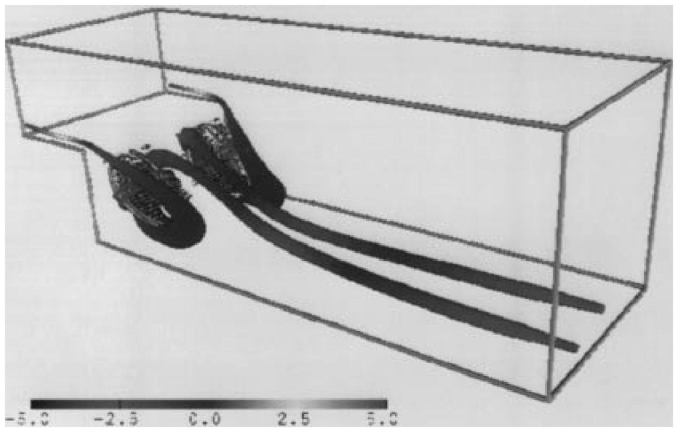

Figure 12. Regions with high normalized helicity density (represented by ellipsoids) and stream tubes through these regions, in a backward-facing step flow.

the axis is defined by two consecutive points on the streamline and by the two direction vectors at these points. The radius of the circular cross section at the end points is bound to the inverse of the square root of velocity magnitude. In this way, a smooth continuous tube is generated, which is an approximation of a constant-flux stream tube; the velocity magnitude can be inferred from the tube diameter. An example of a stream tube is shown in Fig. 12. In this figure, a steady, laminar flow in a backward facing step geometry is visualized. Velocity and pressure data are defined on a $25 \times 37 \times 9$ curvilinear grid. Two streamlines were generated through starting points in these regions, and they were visualized using tubular icons. The streamlines show the characteristic spiraling pattern. The local pressure is bound to an icon parameter that determines the tube's color. (See also section entitled "Feature Extraction and Tracking.") (The simulation was done by the Numerical Mathematics Department, Delft University of Technology, The Netherlands; and the visualization was performed by Theo van Walsum, Department of Technical Informatics, Delft University of Technology, The Netherlands.)

TEXTURE-BASED VECTOR FIELD VISUALIZATION

Surfaces are easy to display using common polygon rendering techniques. The shape of a surface can be very well perceived from the shading derived from the reflection of directional light. With color, an additional scalar variable (such as pressure) can be shown on the surface.

On tangent surfaces, no precise directional information of the vector field is shown, because the true direction of a local tangent vector cannot be derived from the display. One way to improve this is to render tangent lines on the surface. A better way is the use of texture, which gives a complete view of the vector direction on the surface. There are a number of different texture-based synthesis techniques including spot noise, line integral convolution, and texture splats. All of the methods attempt to generate a "feeling of flow" by having a texture perturbed in the direction of flow. The effect is similar to metal shavings on paper lining up in the direction of a magnetic field.

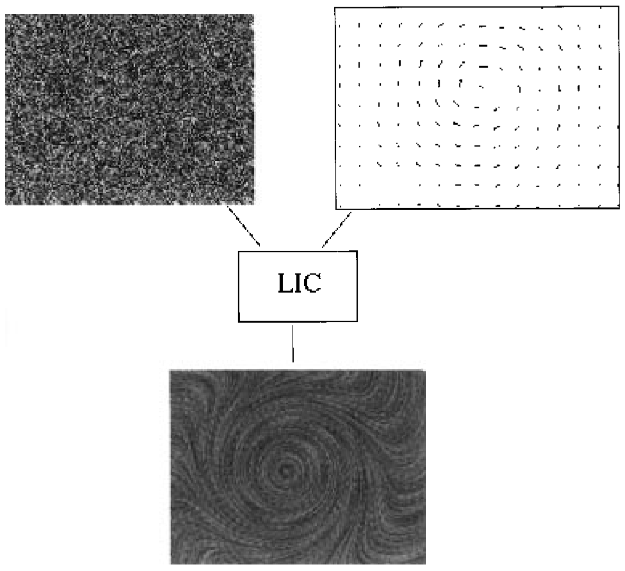

Figure 13. Line integral convolution (LIC) overview. The input to LIC consists of a vector field and a texture.

Line integral convolution (LIC) is one method to produce the textured "flow" effect. The input to the LIC algorithm (21,22) is a vector field and an image texture (see Fig. 13). A local streamline is computed at each pixel (in both directions). The weighted average of the intensities of the pixels of the input image that the streamline passes through is then computed. The image texture is generally white noise (however, any picture or photograph can be used for the image texture, resulting in a warped image in the direction of the vector field). The weighted average is calculated using a convolution filter. The filtering operation causes the noise to be blurred in the direction of the vector field. Animation is possible by applying a phase shift to the filter function, proportional to local velocity magnitude. Texture frames are again precomputed and stored, and they can be interactively viewed in 3D if texture mapping hardware is available. An example is shown in Fig. 13 (23,24). Recently, several extensions to the LIC algorithm have been made to improve rendering in both 2-D and 3-D (see Ref. 25). Figure 14 depicts the surface flow pattern on a rolling delta wing. The flow pattern is generated using the line integral convolution algorithm with enhanced image quality (24). This image depicts the surface flow pattern colored by velocity magnitude. In the color image, low velocity is blue, high velocity is red (25). There are several flow separations and reattachments along the leading edge of the delta wing.

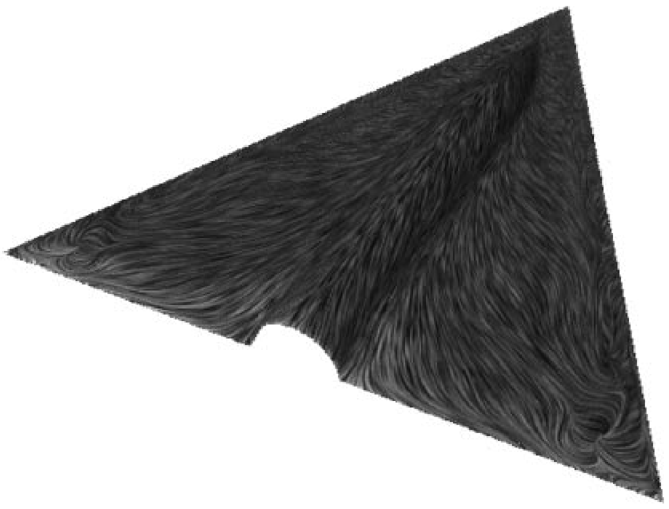

Figure 14. LIC applied to a delta wing simulation. (Data courtesy of Neal Chaderjian, visualization is by David Kao, NASA Ames Research Center.)

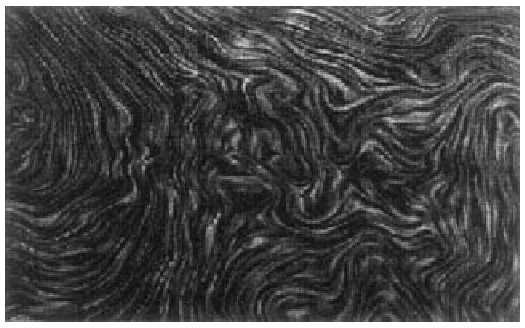

Figure 15. Spot noise applied to a 2-D slice of a 3-D simulation of the flow around a square block. The section behind the block is shown. (Simulation is by R. W. C. P. Verstappen and A. E. P. Veltman, University of Groningen, The Netherlands; Visualization is by W. de Leeuw, Center for Mathematics and Computer Science (CWI), Amsterdam.)

Spot noise (26) is a similar technique by which texture is generated by blending a large number of elementary 2-D shapes (called spots), randomly positioned in a 2-D plane and with random intensity. Local control of the texture is possible by adapting the spot shape to the local values of a 2-D vector field. If the basic spot shape is a circular disk, the deformed spot is an ellipse, with its main axis aligned with the vector direction. The length of the main axis is proportional to vector magnitude, and the area of each spot is kept constant for a given texture. The spots can be bent to adapt better to highly curved and divergent areas in the vector field. A generated 2-D texture is mapped to a 3-D surface and displayed. Figure 15 is a 2-D slice of a 3-D direct simulation of a flow around a square block. The goal of the study is to understand the evolution of vortex shedding and transition to turbulent flow downstream. The grid resolution is 278×208, and the texture resolution is 512×512 pixels, using 40,000 spots. Texture splats (27) is another method to "paint" a vector field. This technique is an extension of the splatting algorithm (28) for scalar fields and is based upon using splats aligned with the vector direction.

FLOW FIELD TOPOLOGY

A vector or tensor field can be characterized by extracting its topology (29–31). The topology can be understood in terms of singular points (critical points in a vector field, degenerate points in a tensor field). These points are connected by integral curves and surfaces, as well as hyperstreamlines, thus building topological skeletons, which divide the flow into separate regions.

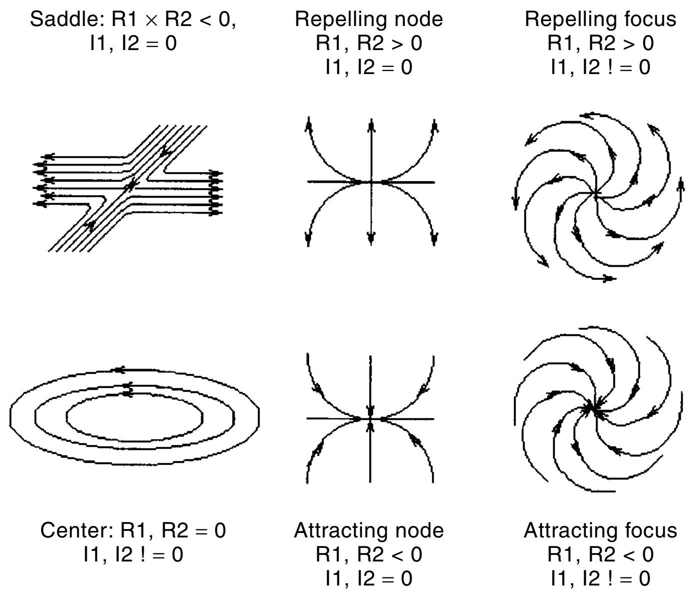

Figure 16. Critical point characterization.

Vector Fields

Critical points are points in *vector field* where the vector magnitude is zero (30,31). A critical point can be classified by the pattern of the field around it—for example, as an attracting or repelling focus, attracting or repelling node, a saddle point, or a center (see Fig. 16). This can be determined from the real and imaginary components of the eigenvalues of the vector gradient tensor (or Jacobian) at the critical point. The real component R determines if the pattern is attracting ($R < 0$), repelling ($R > 0$), or neutral ($R = 0$). For a saddle point, the two real eigenvalues have opposite signs. The imaginary component I describes the circulation around the critical point. If I is not equal to zero, there is a focus or center point. For $I = 0$, there is a node or saddle point. For *no-slip boundaries,* where the velocity is constrained to zero, certain points called *attachment and detachment nodes* are also of interest. At these points the tangential component of the velocity field on the surface goes to zero. A classification for 3-D fields can be found in Ref. 31.

The integral curves and surfaces start at the critical points, at attachment and detachment points, or at the boundary of the field. At critical points, the eigenvectors are used as starting directions. An example of a simple topological skeleton of a 2-D flow around a circular cylinder is shown in Fig. 17. The topological skeleton gives a qualitative summary representation of the flow field.

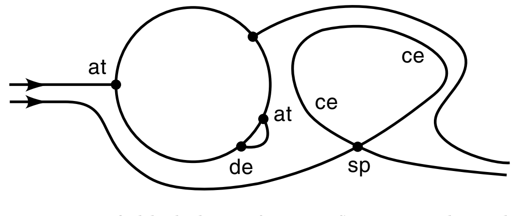

Figure 17. Vector field skeleton for 2-D flow around a cylinder: at, attachment node *s*; de, detachment nodes; ce, center; sp, saddle point.

Second-Order Tensor Fields

Second order tensor fields are common in the study of fluid flow. Second order tensors (3-D) contains a 3 × 3 matrix at each grid location. Examples include the velocity gradient (see section entitled "Basic Operations in Grids"), viscous stress, stress, momentum flux density, and reversible momentum flux density (11). Second-order tensor field visualization is a hard problem and is still an active area of research. Many of the approaches to visualize second-order tensor fields mimic those approaches for vector field (first-order tensor fields). For icons, tensor glyphs or ellipsoids can be used (9,32). The eigenvalues and eigenvectors of a symmetric tensor define the axes and orientation of the ellipsoid centered at that grid location. When the tensor field is defined at every grid location, many small ellipsoids result much like an arrow–vector field. Other local approaches include using *interrogation objects* (33,34) in which a geometric object, such as a plane, is deformed by the tensor field and then rendered.

For symmetric tensor fields, the eigenvalues can be sorted by magnitude and a *hyperstreamline* can be generated by integrating along one of the eigenvector fields. A cylindrical surface can be generated using such a hyperstreamline as a spine curve and by using the other two eigenvectors to define elliptical cross sections along the spine curve. In this way, a "swept ellipse" object is generated along the hyperstreamline, as a variable width tube (11,35). In Fig. 18, four hyperstreamlines which are integrated along the minor principle stress axis are shown. The data are from a point load applied to a semi-infinite domain (Boussinesq problem). A rendering of this dataset using ellipsoids is shown in Ref. 1.

For nonsymmetric tensor fields, the field is first decomposed into a symmetric tensor field and a vector field. Hyperstreamlines are then computed on the symmetric tensor field, and standard vector field visualization techniques can be used to highlight the effect of the vector field.

The topological representation of second-order tensor fields is a generalization of the vector field topology described previously. To provide a global view of the field, degenerate points can be identified. *Degenerate points* in a tensor field

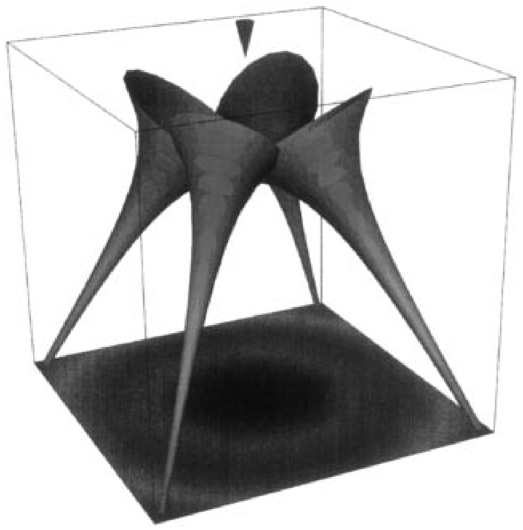

Figure 18. Hyperstreamlines in a compressive stress field (the Boussinesq problem). (From Ref. 1, with permission.)

are points where at least two of the eigenvalues of the tensor are equal to each other. Topological skeletons can be constructed in a similar way as in vector field topology. In this case the field of the tensor's largest eigenvector is used, and hyperstreamlines are generated connecting the degenerate points (11). An example of a tensor topological skeleton is shown in Fig. 19. The skeleton is of the most compressive eigenvector of a stress tensor field. In the original image, color was used to show the magnitude of the compressive force (red = high, blue = low) (36).

FEATURE EXTRACTION AND TRACKING

Scalar time-varying fields (representing flows) are common in many disciplines, such as meteorology or oceanography. Examples include eddy movement, storm-front progression, pollution dispersion, and ozone hole growth. In these cases, standard scalar visualization techniques can be used (with animations to represent time); however, the evolutionary history of the flow is not highlighted.

An effective visualization technique for these fields is to first extract the features of interest and then track them over time. Although each application has its own set of feature definitions, most are based upon some sort of connectivity (i.e., the regions of interest are connected) satisfying threshold and/or vector criteria. Examples include simple threshold intervals on a scalar field, multiple thresholds, vortex tubes using both scalar and vector fields (37), and so on. These features can be extracted using a seed-growing algorithm, which starts with a seed in the region of interest and recursively checks the neighbors for inclusion based upon the defining criteria. The features can be visualized using standard scalar rendering techniques or by drawing vector icons in those regions. An example of using icons to represent the distribution of values within a region is shown in Fig. 12. In this figure, ellipsoid icons are fitted to regions with normalized helicity density at 66% of the global maximum (9).

Once we have defined features, we can characterize the evolutionary events present in continuum scientific simulations as *continuation, creation, dissipation, bifurcation,* and *amalgamation.* These are shown in Fig. 20. For *continuation,* one feature continues from a dataset at time t_i to the next dataset at time t_{i+1}. Rotation or translation of the feature may occur, and its size may remain the same, intensify (become larger—that is, grow), or weaken (become smaller and begin to dissipate). For *creation,* a new feature appears (i.e., cannot be matched to a feature in the previous dataset). For *dissipation,* a feature weakens and disappears into the background. For *bifurcation,* a feature separates in two or more features in the next time step; and for *amalgamation,* two or more features merge from one time step to the next.

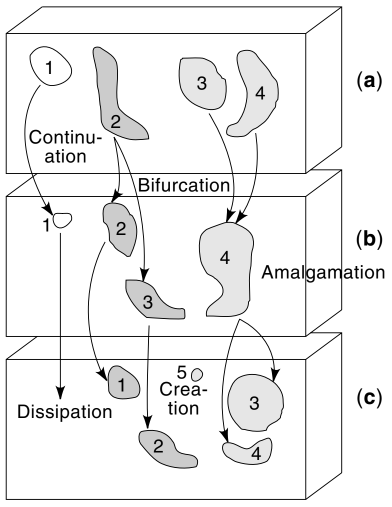

Figure 20. Tracking interactions: continuation, creation, dissipation, bifurcation, and amalgamation.

Matching features from one time step to the next is known as the *correspondence problem* and is a well-studied problem in 2-D computer vision. In Ref. 38 an algorithm is presented which tracks 3-D features in time-varying simulation datasets. Features are matched based upon maximal area overlap and an octree is used to keep the matching hierarchical. A feature f_i from a dataset at time t_i is intersected with the next dataset at time t_{i+1} and a list of candidates is compiled based upon the features from t_{i+1} which overlap with f_i. Using this list of candidates, a best match (which also satisfies a user-defined tolerance) is chosen. An example of the feature tracking algorithm is given in Fig. 21. [This dataset is from a simulation of rotating, stratified turbulence using the quasi-geostrophic (QG) equations, performed by Dr. David G. Dritschel at University of Cambridge (39). The simulation was performed on a 120 × 120 × 60 grid with 1000 timesteps. The variable under investigation is rotation direction (scalar), and the features are defined by their rotational values.]

By isolating regions and extracting them, one can minimize the amount of data to process and thereby reduce *visual clutter.* This is especially useful for vector fields, so that vector visualization techniques can be applied in selective regions.

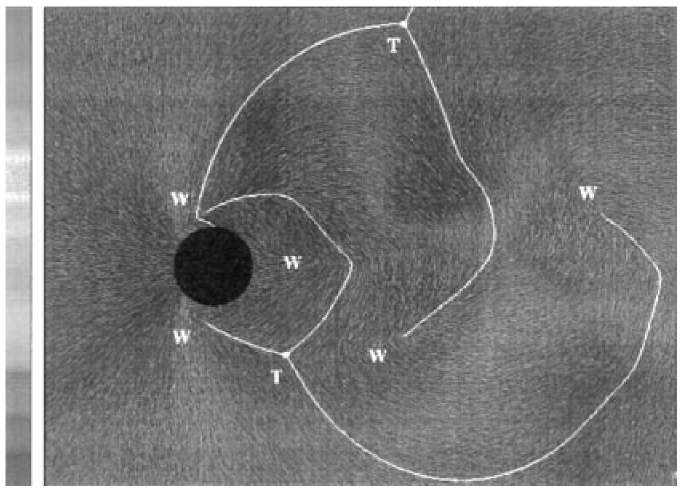

Figure 19. Tensor skeleton is of the most compressive eigenvector of a stress tensor field. (This figure is courtesy of Prof. L. Hesselink and T. Delmarcelle, Stanford University.)

ACKNOWLEDGMENT

The work presented here was done with the help of the members of the Laboratory for Visiometrics and Modeling, the CAIP center at Rutgers University, and Delft University. Special thanks to Dr. N. Zabusky and Simon Wang. Some of the

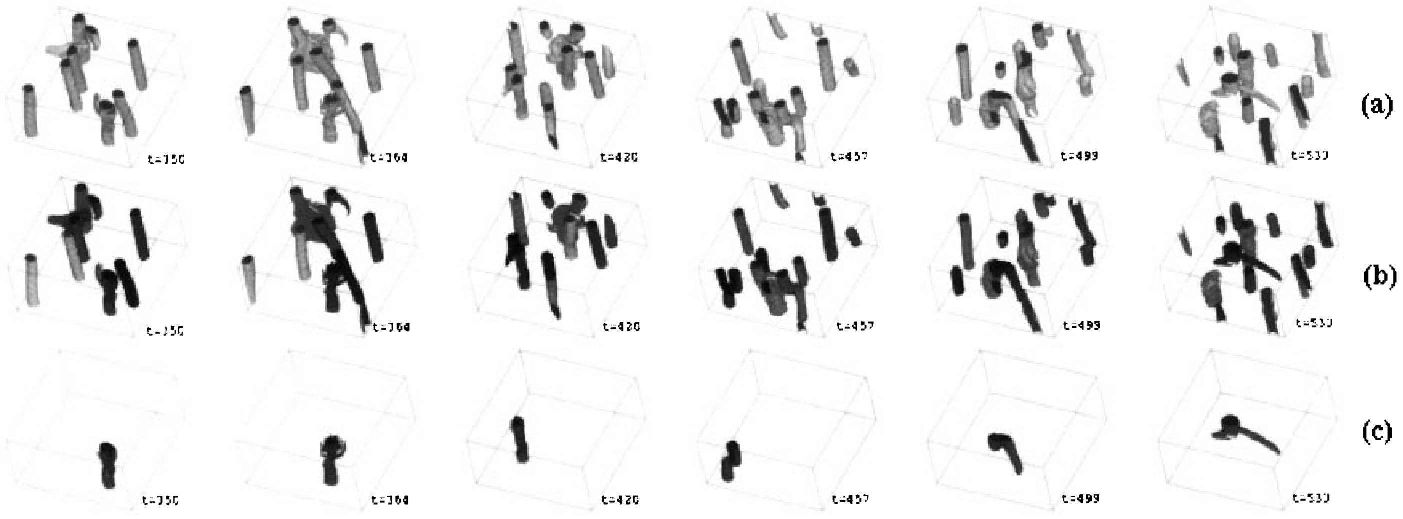

Figure 21. Quasi-geostraphic (QG) simulation, 120 × 120 × 60 resolution. 6/1000 time steps are shown. (a) Standard isosurface rendering without tracking. (b) Each feature is tracked and assigned the same color throughout its lifetime. (c) A particular feature is isolated from all of the timesteps and displayed.

animations mentioned in this article can be seen on the web site http://www.caip.rutgers.edu/vizlab.html. We would also like to thank all the researchers who have made the figures available: In Fig. 10, the data are from Delft Hydraulics, Fig. 15 is courtesy of the Center for Mathematics and Computer Science, Amsterdam (Dr. W. de Leeuw), Fig. 19 is from Professor L. Hesselink of Stanford University, and Fig. 12 is from F. Reinders at Delft University, Technical Informatics, and Fig. 14 is courtesy of D. Kao, NASA Ames Research Center.

BIBLIOGRAPHY

1. W. Schroeder, K. Martin, and B. Lorensen, *The Visualization Toolkit,* Upper Saddle River, NJ: Prentice-Hall, 1997.
2. T. Strid, A. Rizzi, and J. Oppelstrup, Development and use of some flow visualization algorithms, *Computer Graphics and Flow Visualization in CFD,* Brussels, Belgium, 1989, Lecture Series 1989-07, Von Karman Institute for Fluid Dynamics.
3. I. A. Sadarjoen et al., *Particle tracing algorithms for 3D curvilinear grids,* in (C. Nielson, H. Müller, and H. Hagen, eds.) *Scientific Visualization: Overviews, Methodologies, Techniques,* Los Alamitos, CA: IEEE Computer Society Press, 1997, pp. 277–323.
4. H. Neeman, A decomposition algorithm for visualizing irregular grids, *Comput. Graphics,* **24** (5): 49–62, 1990.
5. P. Williams, Interactive direct volume rendering of curvilinear and unstructured data, PhD thesis, University of Illinois, 1992.
6. P. Buning, Numerical algorithms in CFD post-processing, *Computer Graphics and Flow Visualization in CFD,* Brussels, Belgium, 1989, Lecture Series 1989-07, Von Karman Institute for Fluid Dynamics.
7. M. Garrity, Raytracing irregular volume data, *Comput. Graphics,* **24** (5): 35–40, 1990.
8. W. de Leeuw and J. van Wijk, A Probe for Local Flow Field Visualization, in G. Nielson and R. Bergeron (eds.), *Proc. Visualization '93,* Los Alamitos, CA: IEEE Computer Society Press, 1993, pp. 39–45.
9. T. van Walsum et al., Feature extraction and iconic visualization, *IEEE Trans. Vis. Comput. Graphics,* **2**(2): 111–119, 1996.
10. H. Hagen et al., Surface interrogation algorithms, *IEEE Comput. Graphics Appl.,* **11** (3): 36–46, 1991.
11. T. Delmarcelle and L. Hesselink, Visualizing second-order tensor fields with hyperstreamlines, *IEEE Comput. Graphics Appl.,* **4** (13): 25–33, 1993.
12. A. Hin, Visualization of turbulent flows, PhD thesis, Delft University of Technology, Delft, The Netherlands, 1994.
13. D. Kenwright and D. Lane, Optimizing of Time-Dependent Particle Tracing Using Tetrahedral Decomposition, in D. Silver and G. Nielson (eds.), *Proc. Visualization '95,* Los Alamitos, CA: IEEE Computer Society Press, 1995, pp. 321–328.
14. T. van Walsum, Selective visualization techniques for curvilinear grids, PhD thesis, Delft University of Technology, Delft, The Netherlands, 1995.
15. G. Turk and D. Banks, Image-Guided Streamline Placement, in H. Rushmeier (ed.), *SIGGRAPH '96 Conf. Proc.,* Annual Conference Series, ACM SIGGRAPH, Reading, MA: Addison-Wesley, 1996, pp. 453–460.
16. F. Post and T. Walsum, Fluid Flow Visualization, in H. Hagen, H. Müller, and G. Nielson (eds.), *Focus on Scientific Visualization,* New York: Springer-Verlag, 1993, pp. 1–40.
17. A. van Gelder and J. Wilhelms, Interactive Animated Visualization of Flow Fields, in A. Kaufman and W. Lorensen (eds.), *1992 Workshop on Volume Visualization,* New York: ACM Press, 1992, pp. 47–54.
18. H-G Pagendarm and B. Walter, Competent, compact, comparative visualization of a vortical flow field, *IEEE Trans. Vis. Comput. Graphics,* **1**(2): 142–150, 1995.
19. J. Hultquist, Constructing Stream Surfaces in Steady 3D Vector Fields, in A. Kaufman and G. Nielson (eds.), *Proc. Visualization '92,* Los Alamitos, CA: IEEE Computer Society Press, 1992, pp. 171–178.
20. J. van Wijk, Implicit Stream Surfaces, in G. Nielson and R. Bergeron (eds.), *Proc. Visualization '93,* Los Alamitos, CA: IEEE Computer Society Press, 1993, pp. 245–252.
21. B. Cabral and C. Leedom, Imaging vector fields using line integral convolution, *Proc. SIGGRAPH 93,* **27**: 263–272, 1993.
22. L. Forssell and S. Cohen, Using line integral convolution for flow visualization: Curvilinear grids variable speed animation and unsteady flows, *IEEE Comput. Graphics Appl.,* **2** (1): 133–141, 1995.

23. H. Shen and D. Kao, UFLIC: A line integral convolution algorithm for visualizing unsteady flows, *Proc. IEEE Vis. '97 Conf.,* 1997.
24. A. Okada and D. Kao, Enhanced Line Integral Convolution with Flow Feature Detection, *Proc. IS&T/SPIE Electron. Imaging '97,* 1997.
25. *IEEE Visualization '96 and '97 Proc.,* IEEE Computer Society.
26. W. de Leeuw and J. van Wijk, Enhanced Spot Noise for Vector Field Visualization, in D. Silver and G. Nielson (eds.), *Proc. Visualization '95,* Los Alamitos, CA: IEEE Computer Society Press, 1995, pp. 233–239.
27. R. Crawfis and N. Max, Texture Splats for 3D Scalar and Vector Field Visualization, in G. Nielson and R. Bergeron (eds.), *Proc. Visualization '93,* Los Alamitos, CA: IEEE Computer Society Press, 1993, pp. 261–265.
28. L. Westover, Footprint evaluation for volume rendering, *Comput. Graphics,* **24**: 367–376, 1990.
29. J. L. Helman and L. B. Hesselink, Representation and display of vector field topology in fluid flow data sets, *Computer,* **22**(8): 27–36, 1989.
30. J. Helman and L. Hesselink, Visualization of vector field topology in fluid flows, *IEEE Comput. Graphics Appl.,* **11**: 36–46, 1991.
31. A. Globus, C. Levit, and T. Lasinski, A Tool for Visualizing the Topology of Three-Dimensional Vector Fields, in G. M. Nielson and L. Rosenblum (eds.) *Proc. Visualization '91,* Los Alamitos, CA: IEEE Computer Society Press, 1991.
32. R. Haber and D. McNabb, Visualization idioms: A conceptual model for scientific visualization systems, in G. M. Nielson, B. D. Shriver and L. Rosenblum (eds.) *Visualization in Scientific Computing,* Los Alamitos, CA: IEEE Computer Society Press, 1992, pp. 75–93.
33. H. Hagen, S. Hahmann, and H. Weimer, Visualization of deformation tensor fields, in G. Nielson, H. Hagen, and H. Muller (eds.), *Scientific Visualization: Overviews, Methodologies, Techniques,* Los Alamitos, CA: IEEE Computer Society, 1997.
34. E. Boring and A. Pang, Interactive deformations from tensor fields, manuscript, 1998. Available online: http://emerald.ucsc.edu/~edb.
35. R. R. Dickinson, Interactive analysis of the topology of 4D vector fields, *IBM J. Res. Develop.,* **35** (1/2): 59, 1991.
36. T. Delmarcelle and L. Hesselink, The Topology of Symmetric, Second-Order Tensor Fields, in D. Bergeron and A. Kaufman (eds.), *Proc. Visualization '94,* Los Alamitos, CA: IEEE Computer Society Press, 1994, pages 140–147.
37. D. C. Banks and B. A. Singer, A predictor–corrector technique for visualizing unsteady flow, *IEEE Trans. Vis. Comput. Graphics,* **1** (2): 151–163, 1995.
38. D. Silver and X. Wang, Tracking and Visualizing Turbulent 3D Features, *IEEE Trans. Vis. Comput. Graphics,* **3** (2): 1997.
39. D. G. Dritschel and M. H. P. Ambaum, A contour-advective semi-Lagrangian numerical algorithm for simulating fine-scale conservative dynamical fields, *QJRMS,* 1997.

D. Silver
Rutgers University
F. H. Post
I. A. Sadarjoen
Delft University

FLUCTUATIONS, HOT ELECTRON EFFECTS. See Noise, hot carrier effects.

FLUX JUMPS, STABILIZATION AGAINST. See Superconductors, stabilization against flux jumps.

FLUX, MAGNETIC. See Magnetic flux.

FM ANTENNAS. See Television and fm broadcasting antennas.

FMEA. See Failure modes and effects analysis.

FM TRANSMITTERS. See Transmitters for fm broadcasting.