Fundamental Algorithms

Fundamental Algorithms

Overview

This chapter introduces some basic techniques for visualizing different types of scientific data sets. We will categorize visualization methods into classes distinguished by data type and learn more about how to visualize these kinds of data. Combinations of these techniques can then be used depending on the type of data you want to visualize.

Categorization (continued)

Four different categories can be used to classify visualization techniques based on the type of data they operate on:

- Scalar algorithms
- Vector algorithms
- Tensor algorithms
- Modeling algorithms

Algorithms can also be classified by the type of data they process which can be ambiguous. For example, volume visualization techniques are nowadays applied to scalar, vector, and tensor data sets.

Categorization (continued)

Scalar algorithms

Scalar algorithms operate on scalar data. For example, the generation of contour lines of temperature on a weather map.

Categorization (continued)

Vector algorithms

Vector algorithms operate on vector data. Showing oriented arrows of airflow (direction and magnitude) is an example of vector visualization. There are more advanced techniques, such as topological analysis or line integral convolution (LIC) approaches.

Categorization (continued)

Tensor algorithms

Tensor algorithms operate on tensor matrices. An example of a tensor algorithm is to show the components of stress or strain in a material using oriented icons. This is possible due to the fact that stress/strain tensors are symmetric, i.e. the eigenvalues of the describing matrices exist and are real. Other techniques for visualizing tensors, which also exploit this property, are hyperstreamlines.

Scalar Algorithms

Color Mapping

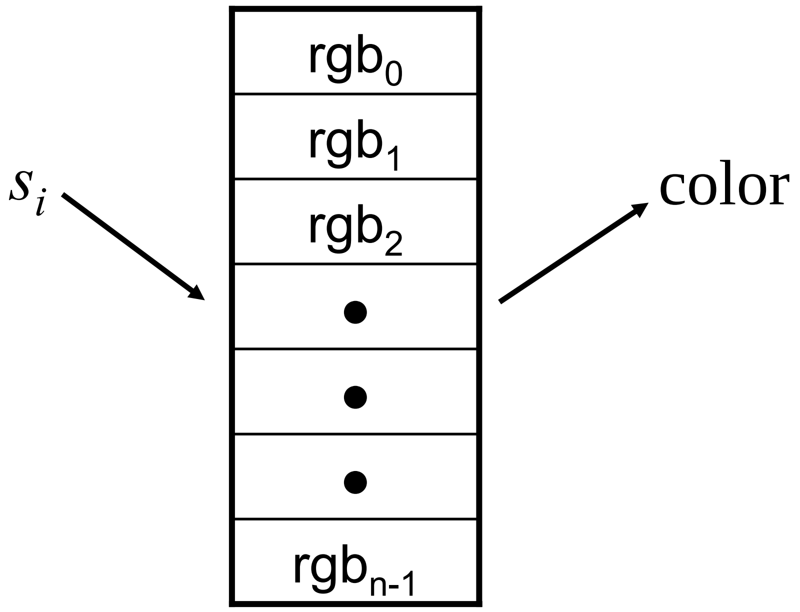
Color mapping is a common scalar visualization technique that maps scalar data to colors, and displays the colors on the computer system. The scalar mapping is implemented by indexing into a *color lookup table*. Scalar values then serve as indices into this lookup table.

The lookup table holds an array of colors. Associated with the table is a minimum and maximum scalar range into which the scalars are mapped. Scalar values greater than the maximum are clamped to the maximum color, scalar values less than the minimum are clamped to the minimum value.

Scalar Algorithms (continued)

Then, for each scalar value s_i, the index i into the color able with n entries is given as:

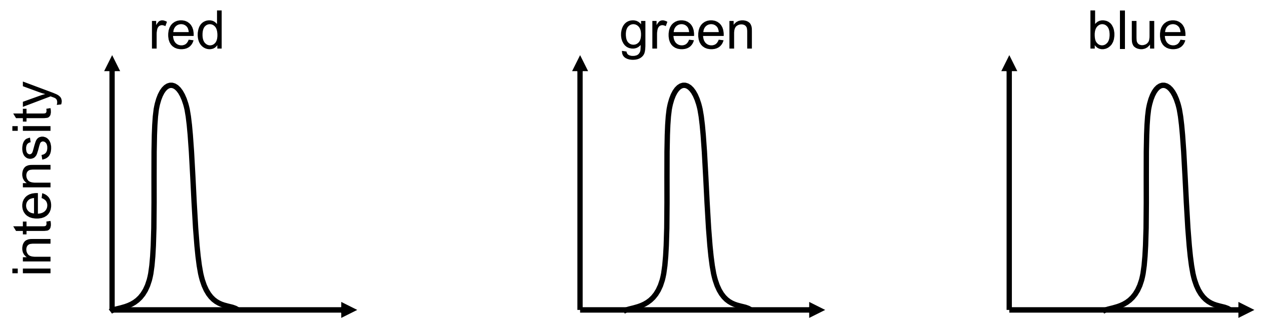
$$
\begin{aligned}
s_i < \min \quad &: \quad i = 0 \\
s_i > \max \quad &: \quad i = n - 1 \\
\text{otherwise}: \quad & i = n \cdot \left(\frac{s_i - \min}{\max - \min} \right)
\end{aligned}
$$



Scalar Algorithms (continued)

Transfer Functions

A more general form of the lookup table is called *transfer function*. A transfer function is any expression that maps scalar values into a color specification. For example, a function can be used to map scalar values into separate intensity values for the red, green, and blue components.



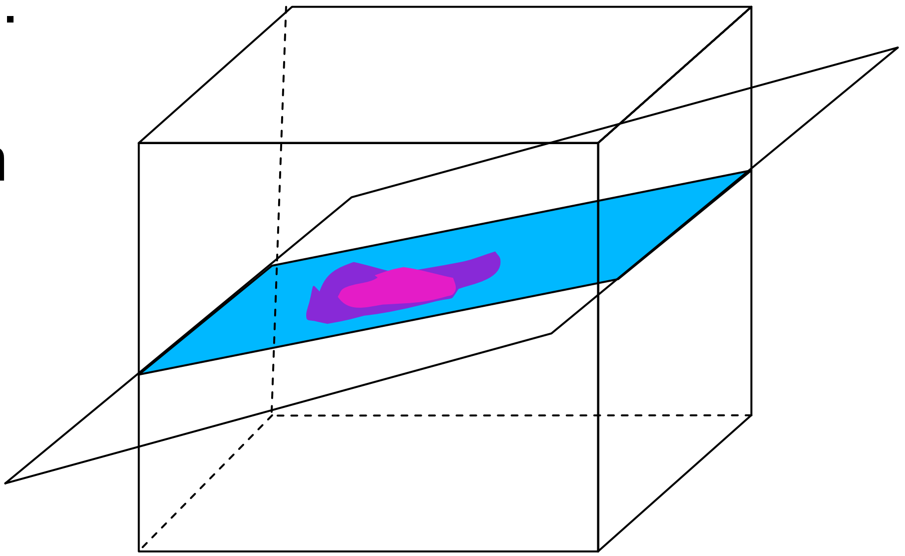
Scalar Algorithms (continued)

We can also use transfer functions to map scalar data into other information such as local transparency. This will be discussed later when we talk about volume rendering. A lookup table is a discrete sampling of a transfer function. We can create a lookup table from any transfer function by sampling the transfer function at a set of discrete points.

Scalar Algorithms (continued)

Color mapping is a one-dimensional visualization technique. It maps one piece of information (i.e. a scalar value) into a color specification. However, the display of color information is not limited to one-dimensional displays. Often we use color information mapped onto 1-D, 2-D, or 3-D objects.

This is a simple way to increase the information content of our visualization. In 3-D, cutting planes can be used to visualize the data inside.

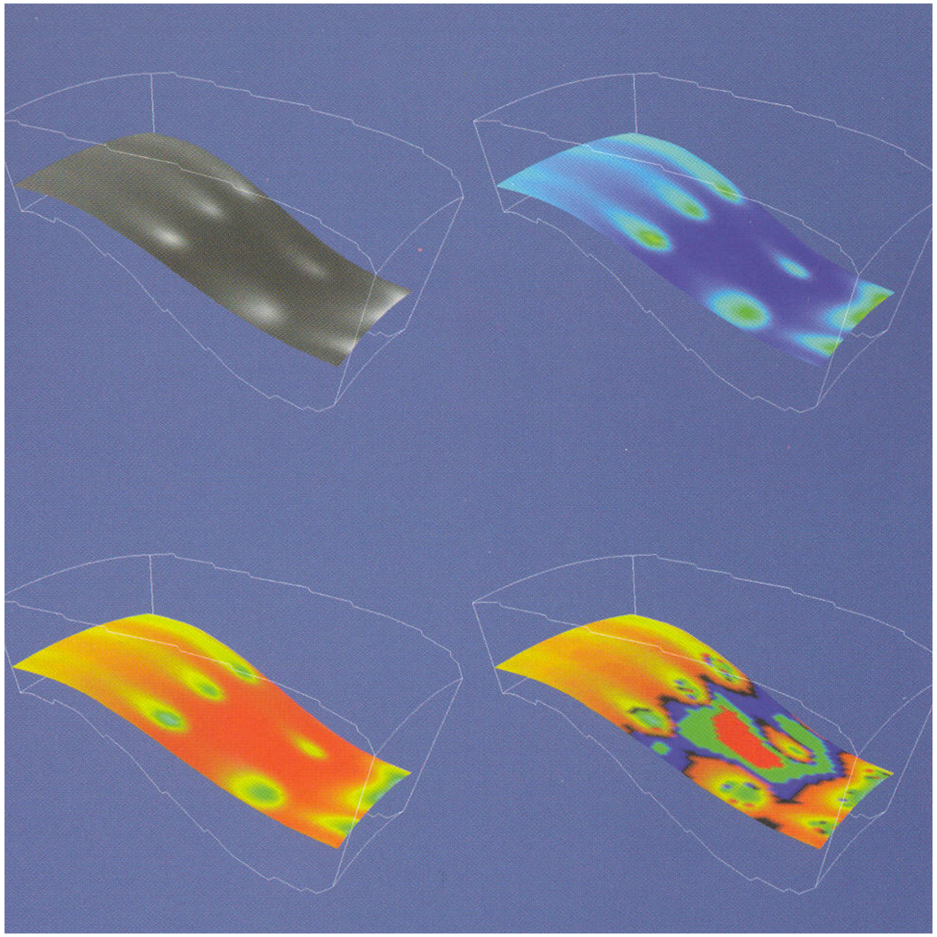


Scalar Algorithms (continued)

The key to color mapping for visualization is to choose the lookup table entries carefully. Designing lookup tables is as much art as it is science. From a practical point of view, tables should accentuate important features, while minimizing less important or extraneous details. It is also desirable to use palettes that inherently contain scaling information. For example, a color rainbow scale from blue to red is often used to represent temperature scale, since many people associate blue with cold temperatures and red with hot temperatures.

Scalar Algorithms (continued)

Examples:



Scalar Algorithms (continued)

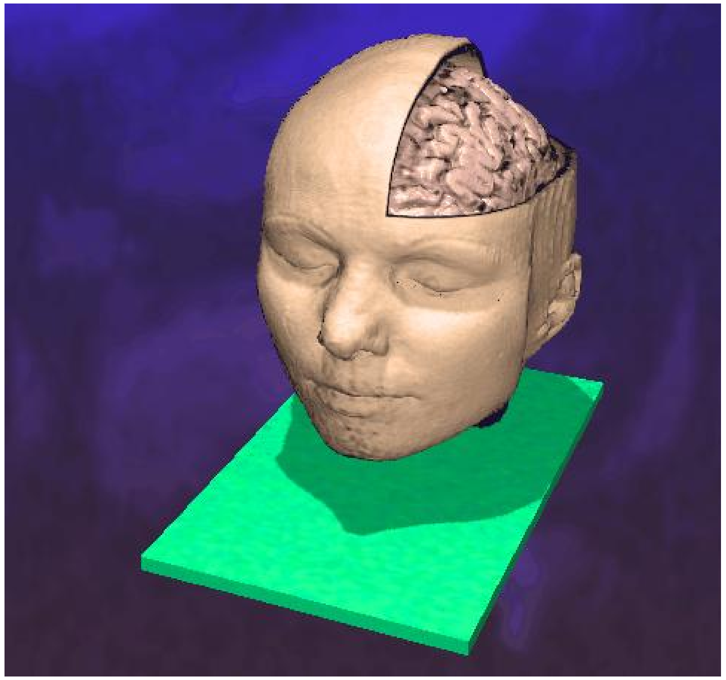
Contouring

A natural extension to color mapping is contouring. When we see a surface colored with data values, the eye often separates similarly colored areas into distinct regions. When we contour data, we are effectively constructing the boundary between these regions. These boundaries correspond to contour lines (2-D) or surfaces (3-D) of constant scalar value.

Examples of 2-D contour displays include weather maps annotated with lines of constant temperature (isotherms), or topological maps drawn with lines of constant elevation.

Scalar Algorithms (continued)

Three-dimensional contours are called *isosurfaces*, and can be approximated by many polygonal primitives. Examples of isosurfaces include constant medical image intensity corresponding to body tissues such as skin, bone, or other organs. (The corresponding isovalue for the same tissue, however, is not necessarily constant among several different scans.) Other abstract isosurfaces such as surfaces of constant pressure or temperature in fluid flow also may be created.

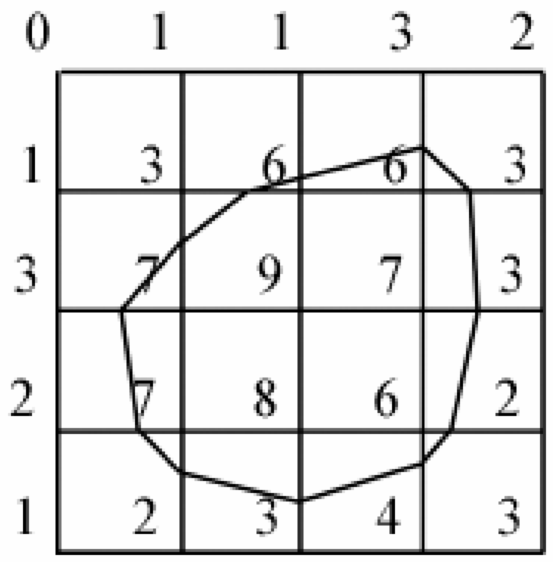


Scalar Algorithms (continued)

First, we will focus on 2-D contours and how to generate such an isocontour for a given isovalue. Consider a regular grid with scalar values assigned to the grid nodes. Contouring always begins by selecting a scalar value (the isovalue or contourvalue) that corresponds to the contour lines or surface generated. Assuming linear interpolation on the regular grid, we can identify those locations on the edges of the regular grid where the data assumes the isovalue. For example, if an edge has scalar values 10 and 0 at its two end points, and if we are trying to generate a contour line of value 5, then the contour passes through the midpoint of that edge.

Scalar Algorithms (continued)

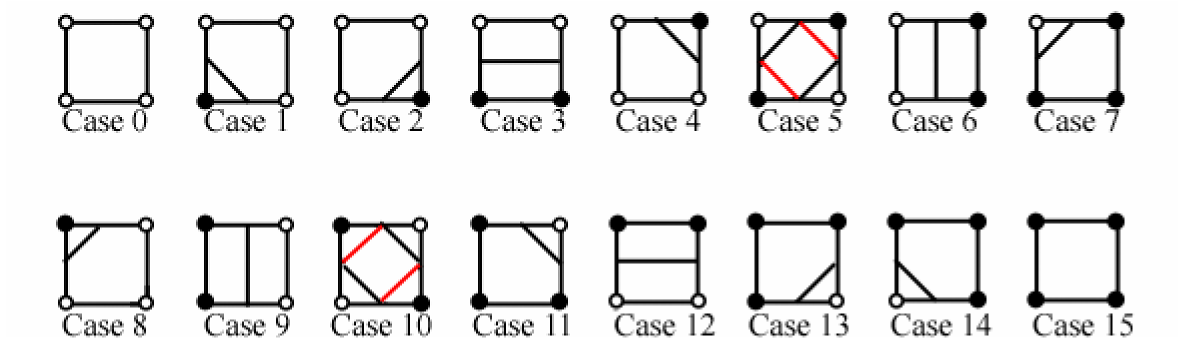
Once the points on all edges are generated, we can connect these points into contours using a few different approaches. One approach detects an edge intersection, i.e. the contour passes through an edge, and then tracks this contour as it moves across cell boundaries. We now that if a contour edge enters a cell, it must exit a cell as well. The contour is tracked until it closes back on itself, or exits a data set boundary.



Scalar Algorithms (continued)

Marching Squares

Another approach uses a divide and conquer technique, treating cells independently. This *marching squares* algorithm assumes that a contour can only pass through a cell in a finite number of ways due to the linear interpolation used. A case table is constructed that enumerates all possible topological states of a cell, given combinations of scalar values at the cell points.

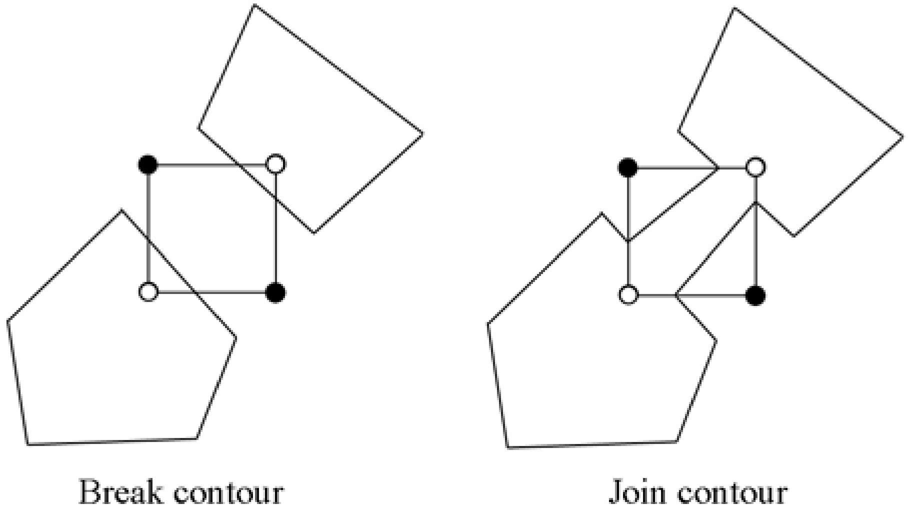


(dark vertices indicate scalar value is above isovalue)

Scalar Algorithms (continued)

Ambiguities in Marching Squares

While trying this algorithm on different configurations we realize that some cases may be ambiguous. That is the situation for the squares 5 and 10.



As you can see on the previous picture we are not able to take a decision on the interpretation of this kind of situation. However, these exceptions do not imply any real error because the edges keep closed.

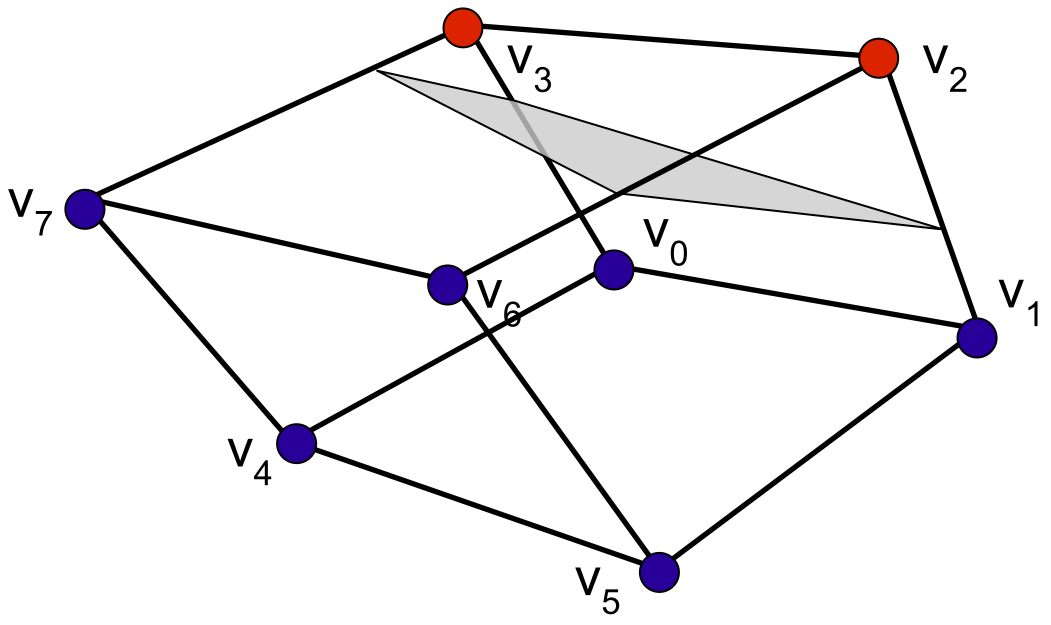
Scalar Algorithms (continued)

Marching Cubes

Lorensen and Cline introduced Marching Cubes in 1987.
[William E. Lorensen, Harvey E. Cline, „Marching Cubes: A High Resolution 3D Surface Construction Algorithm“, ACM Computer Graphics Vol. 21 No. 4 (SIGGRAPH 1987 Proceedings)]

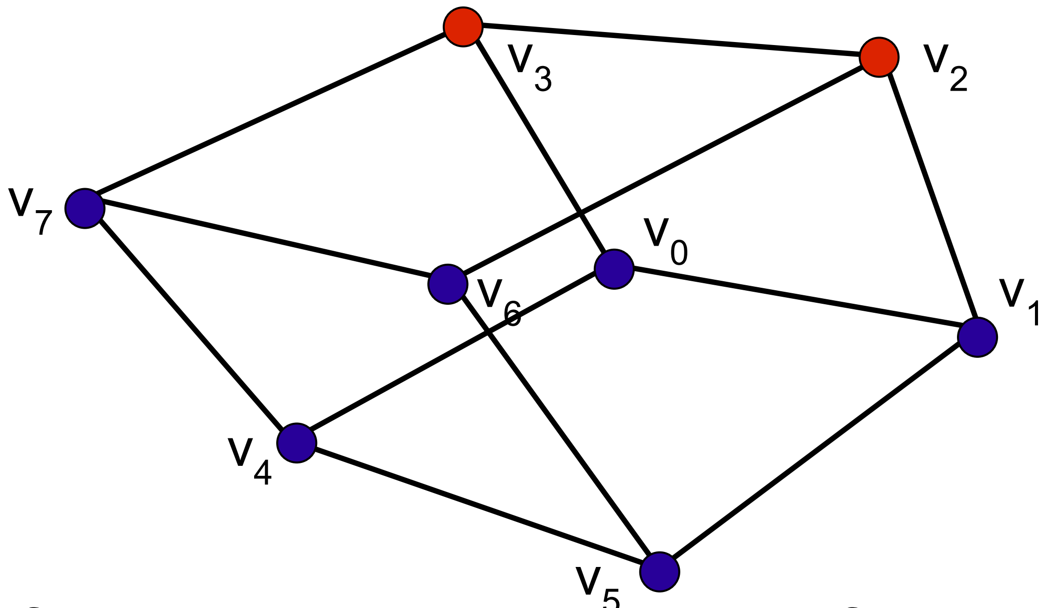
Marching Cubes (MC) is an efficient method for extracting isosurfaces from scalar data set defined on a regular grid. Similar to marching squares, surface segment is computed for each cell of the grid that approximates the isosurface.

Scalar Algorithms (continued)



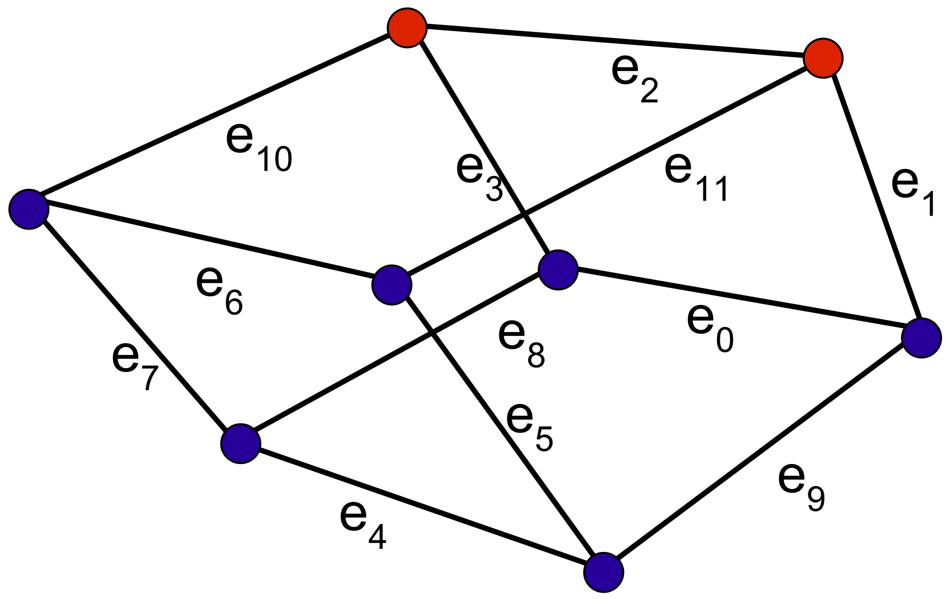
Since the triangulation inside the cell only depends on whether edges exist that intersect the isosurface, we again focus on checking if an edge has values at its vertices in such a way, that one is smaller and one is larger than the isovalue.

Scalar Algorithms (continued)



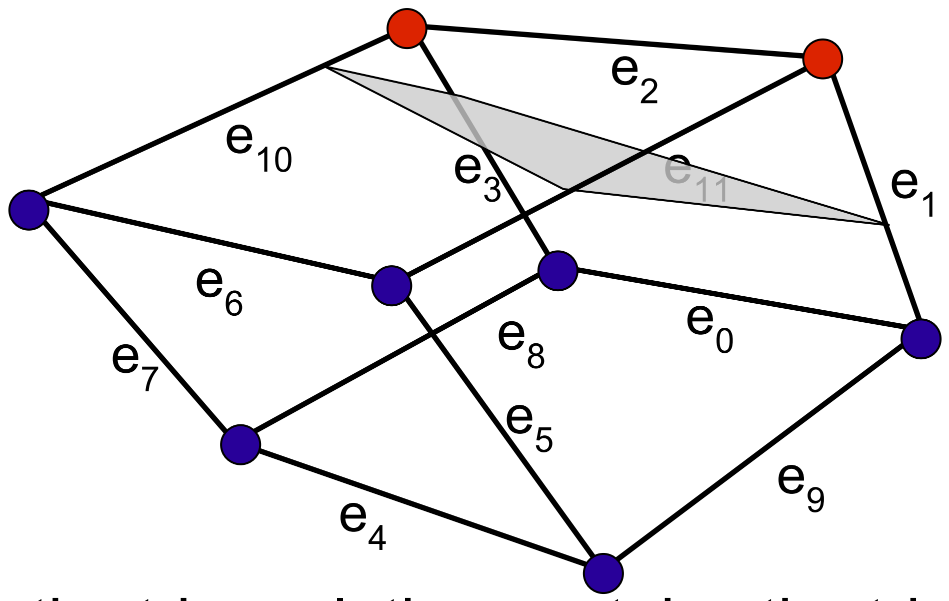
In order to find the edges that are part of the isosurface a lookup table can be used. In order to find the correct entry in this table the vertices are numbered. By setting the corresponding bit for each value larger than the isovalue we get the index referring to the lookup table. There are 256 different combinations possible.

Scalar Algorithms (continued)



Similarly, the edges are numbered. By generating a bit mask like before using the marked edges we can use the resulting value to point to another lookup table for the triangulation. In the above case the marked edges are 1, 3, 10, and 11.

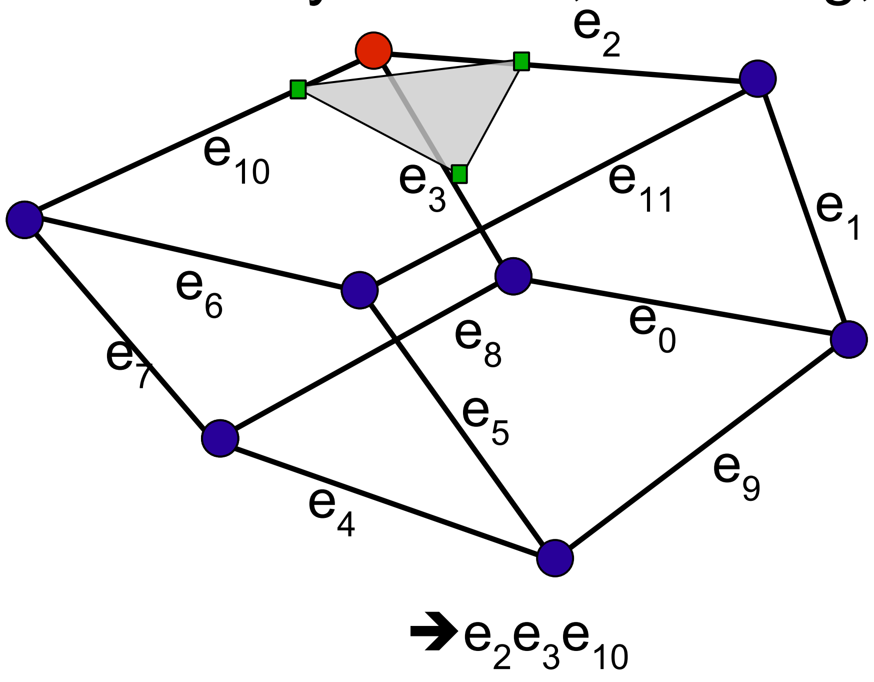
Scalar Algorithms (continued)

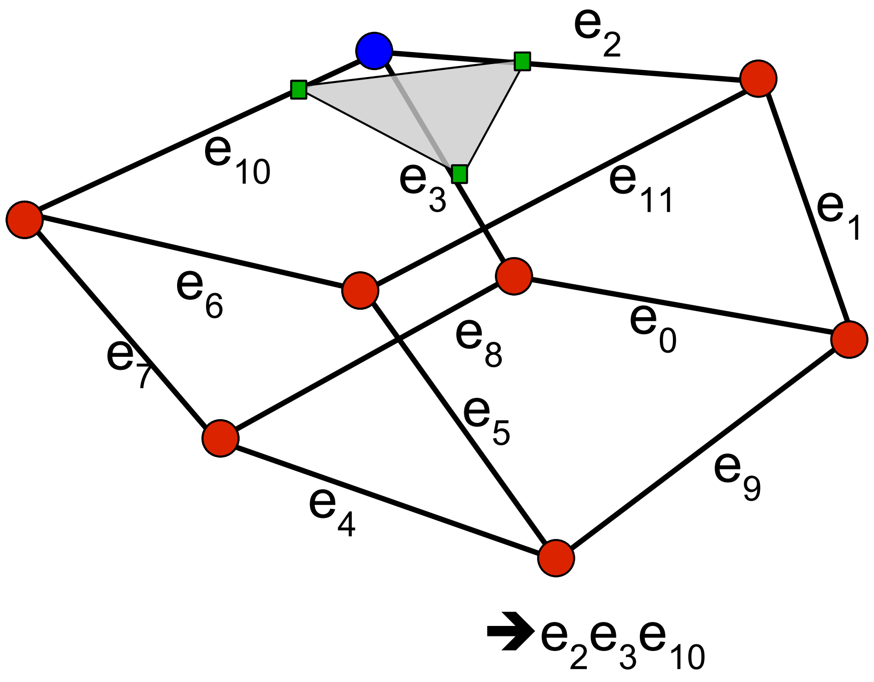


The table for the triangulation contains the triangulations for all 256 cases. Each entry has a list of triangles; the vertices refer to the interpolated points of intersection with the isosurface. The triangles should be oriented mathematically positively for correct backface culling. In our example, the triangulation is $e_1e_3e_{11};e_3e_{10}e_{11}$.

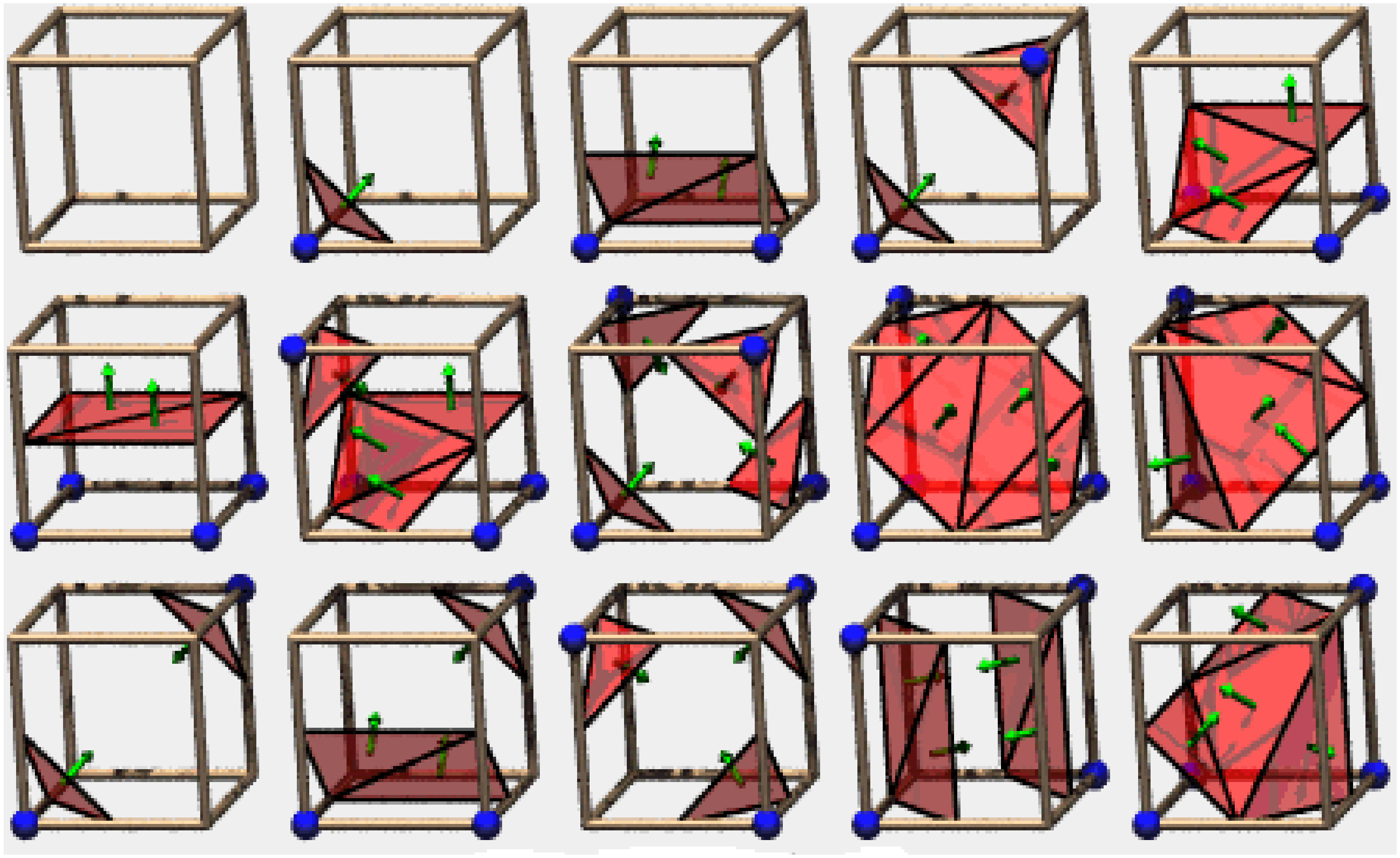
Scalar Algorithms (continued)

Even though there are 256 possible configurations which have to be triangulated, only 15 of them need to be stored. The remaining ones can be derived from these 16 cases by rotation, mirroring, or inversion.





Scalar Algorithms (continued)



All 15 basic cases needed for Marching Cubes

Scalar Algorithms (continued)

Computation of normal vectors

The quality of the resulting representation of the extracted isosurface can be improved by computing the normal vectors of all vertices. We can exploit the fact, that the gradient of the scalar function

$$\nabla f(x,y,z) = \begin{pmatrix} \frac{\partial f}{\partial x}(x,y,z) \\ \frac{\partial f}{\partial y}(x,y,z) \\ \frac{\partial f}{\partial z}(x,y,z) \end{pmatrix}$$

is always orthogonal to the isosurface. Marching Cubes approximates the gradient at the vertices of the grid as

$$\nabla f(x,y,z) = \begin{pmatrix} \frac{D(i+1,j,j)-D(i-1,j,k)}{\Delta x} \\ \frac{D(i,j+1,j)-D(i,j-1,k)}{\Delta y} \\ \frac{D(i,j,j+1)-D(i,j,k-1)}{\Delta z} \end{pmatrix}$$

and interpolates linearly to determine the gradient at the intersection.

Scalar Algorithms (continued)

Problems with Marching Cubes

After the article about Marching Cubes was published it turned out that the isosurfaces extracted using Marching Cubes can contain holes under certain circumstances due to ambiguities in the case table. Several follow-up papers exist to fix several issues with Marching Cubes.

Variants

There are variants of Marching Cubes for triangles and tetrahedra as well.

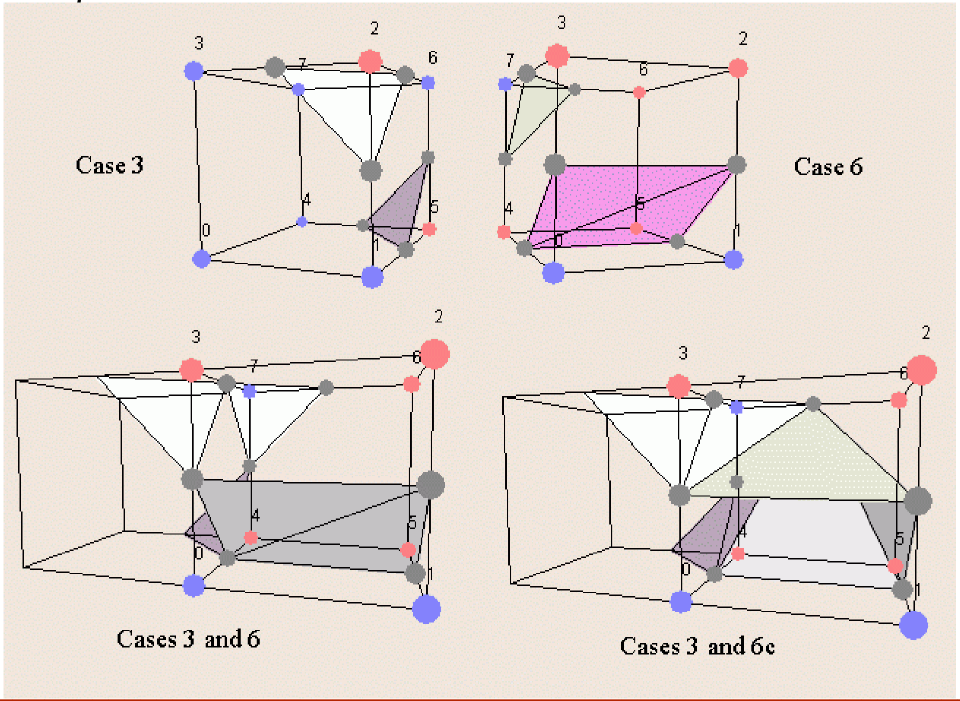
Scalar Algorithms (continued)

Coping with the Ambiguities

The original set of cases for creating triangles within the cells to generate an isosurface can create holes in some cases. This is basically due to ambiguities, i.e. there are more than one way to generate triangles for some cases. Hence, by introducing additional cases to our case table we can cope with the ambiguities. Obviously, in order to decide which configuration to use, we also have to look at the neighboring cells.

Scalar Algorithms (continued)

Consider the following two cells. The original marching cubes solution (left) would change the topology of the resulting isosurface, i.e. create holes. By introducing an additional case, this can be fixed.

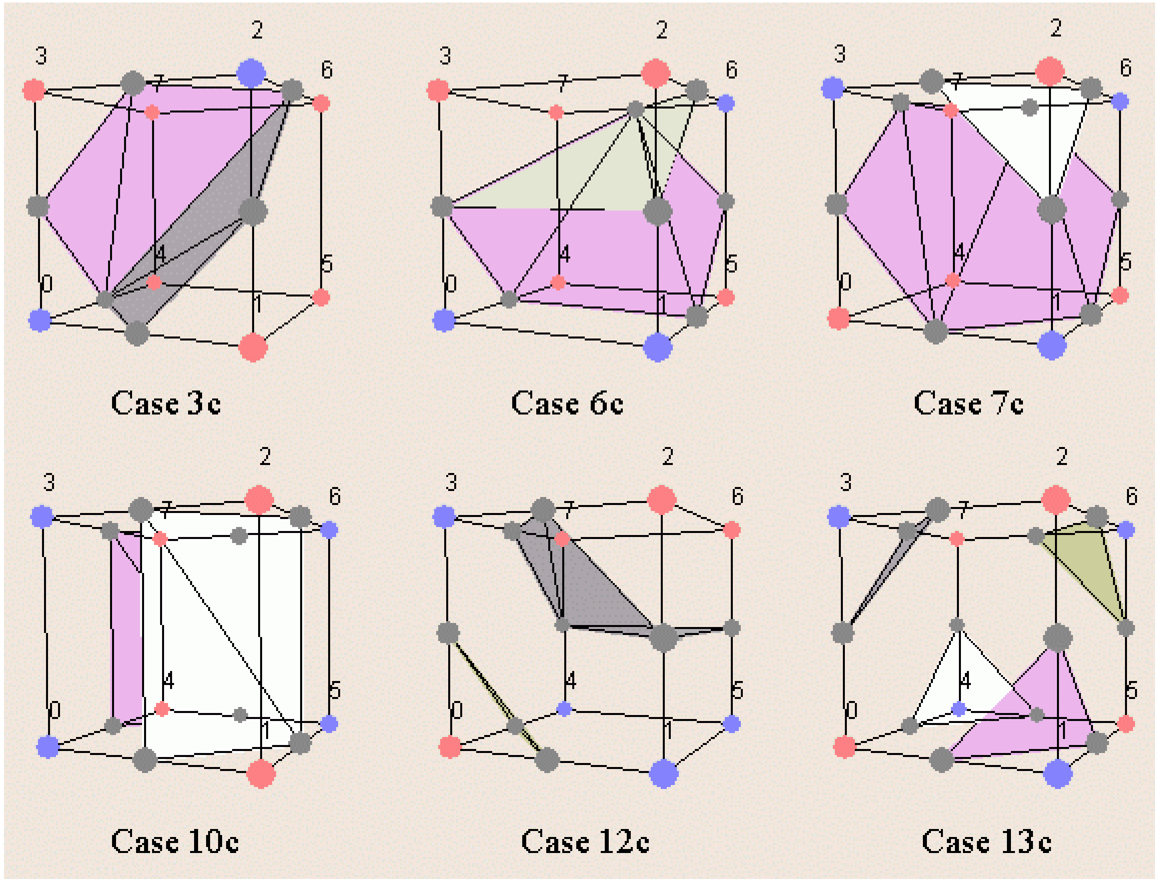


Scalar Algorithms (continued)

To cope with these topology errors (as holes in the 3D model), 6 cases have been added to the marching cubes cases. These cases have to be used as complementary cases. For instance, in the previous picture, you have to use the case 6c instead of the standard complementary of the case 6. The list of new cases is shown on the next slide.

Scalar Algorithms (continued)

Additional cases:



Scalar Algorithms (continued)

Scalar generation

The two techniques – color mapping and contouring – are simple, effective methods to display scalar information. It is natural to turn to these techniques first when visualizing data. However, often our data is not in a form convenient to these techniques. The data may not be single-valued, i.e. a scalar, or it may be a mathematical or other complex relationship. That is part of the fun and creative challenge of visualization: we must tap our creative resources to convert data into a form we can visualize.

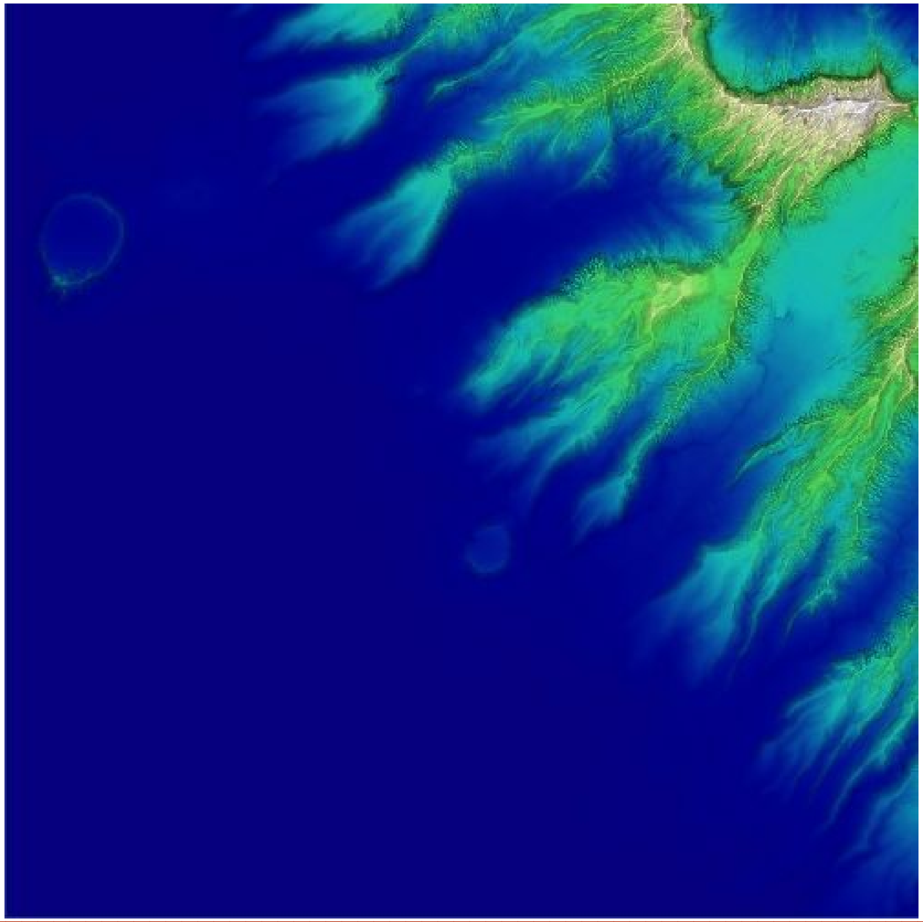
Scalar Algorithms (continued)

Example

Consider a terrain data set. We assume that the data is given as x-y-z coordinates, where x and y represents the coordinates in the plane, and z represents the elevation above sea level. Our desired visualization is to color the terrain according to elevation. This requires creating a colormap – possibly using white for high altitude, blue for sea level and below, and various shades of green and brown corresponding to elevation between sea level and high altitude. We also need scalars to index into the colormap. The obvious choice here is to extract the z-coordinate. That is, scalars are simply the z-coordinate.

Scalar Algorithms (continued)

The resulting visualization may look like this:



Scalar Algorithms (continued)

Similarly, other types of data sets can be converted to the scalar data format. For example, by computing the lengths of the vectors a vector data set can be converted to a scalar data sets. This approach, however, should only be used if such a conversion makes sense for the application.

Vector Algorithms

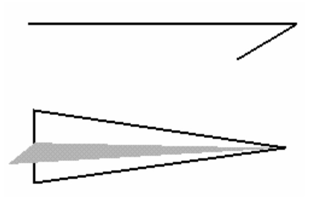
Vector data is a three-dimensional representation of direction and magnitude. Vector data often results from the study of fluid flow, or when examining derivatives, i.e. rate of change, of some quantity.

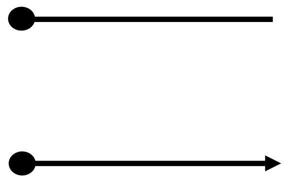
Different visualization techniques are available for vector data sets, for example:

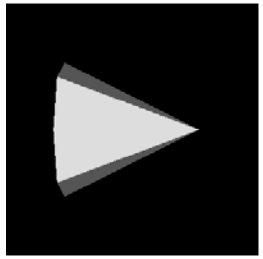
- Hedgehogs and oriented glyphs
- Warping
- Displacement plots
- Time animation
- Streamlines

Vector Algorithms (continued)

A simple vector visualization technique is to draw an oriented, scaled line for each vector. The line begins at the point with which the vector is associated and is oriented in the direction of the vector components. Typically, the resulting line must be scaled up or down to control the size of its visual representation. This technique is often referred to as *hedgehog* because of the bristly result.



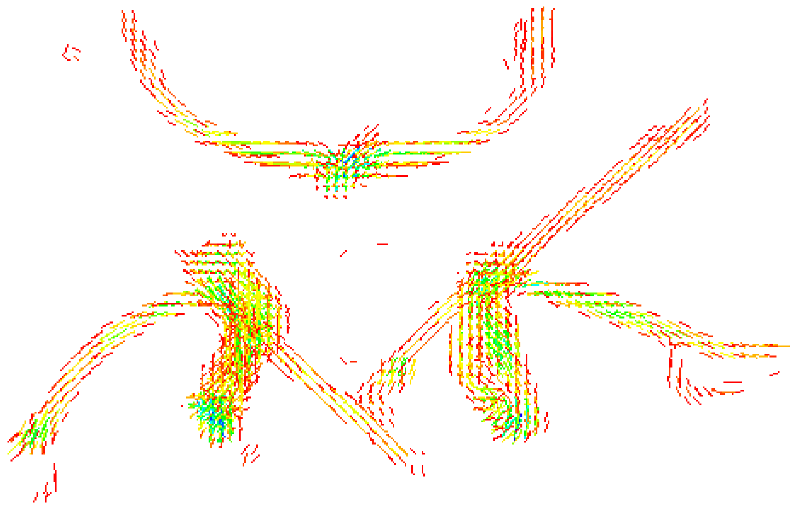


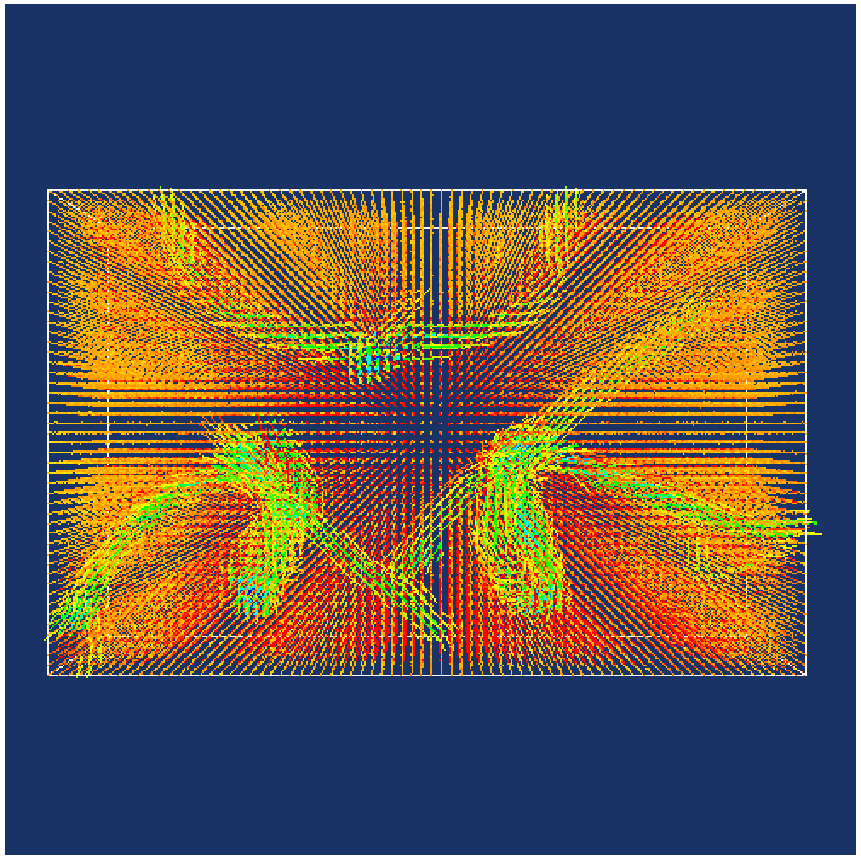


Direction can also be visualized using color coding by using different colors at each ends of the glyph

Vector Algorithms (continued)

The problem with glyphs is that it easily results in clutter.





Vector Algorithms (continued)

Warping

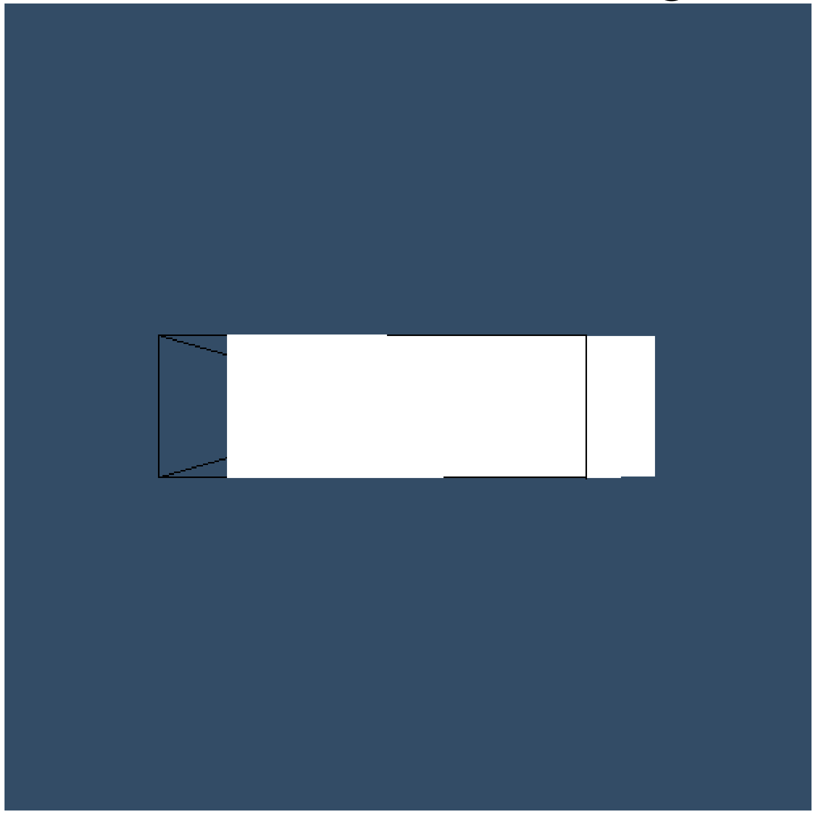
Vector data is often associated with motion. The motion is in the form of velocity or displacement. An effective technique for displaying such vector data is to “warp” or deform geometry according to the vector field. For example, imagine representing the displacement of a structure under load by deforming the structure.

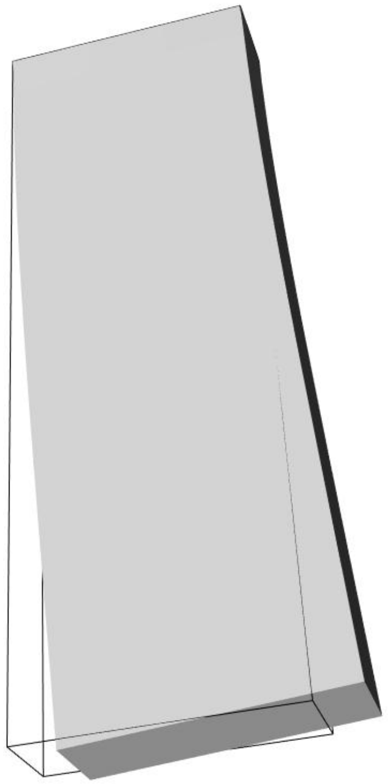
The warping technique should – as usual – be applied with the application in mind.

Vector Algorithms (continued)

Example

The motion of a vibrating beam

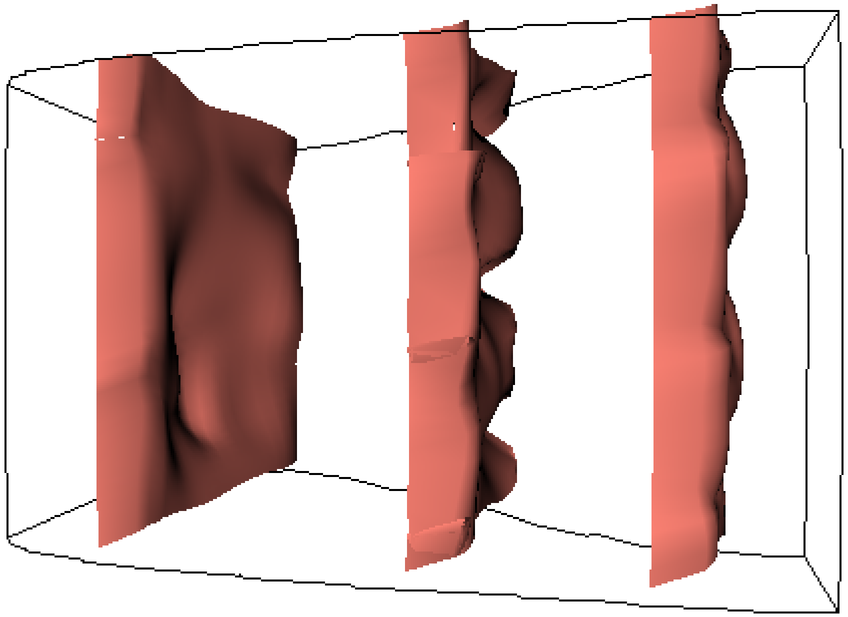




Vector Algorithms (continued)

Example

Warped planes in a structured grid data set. The planes are warped according to flow momentum.

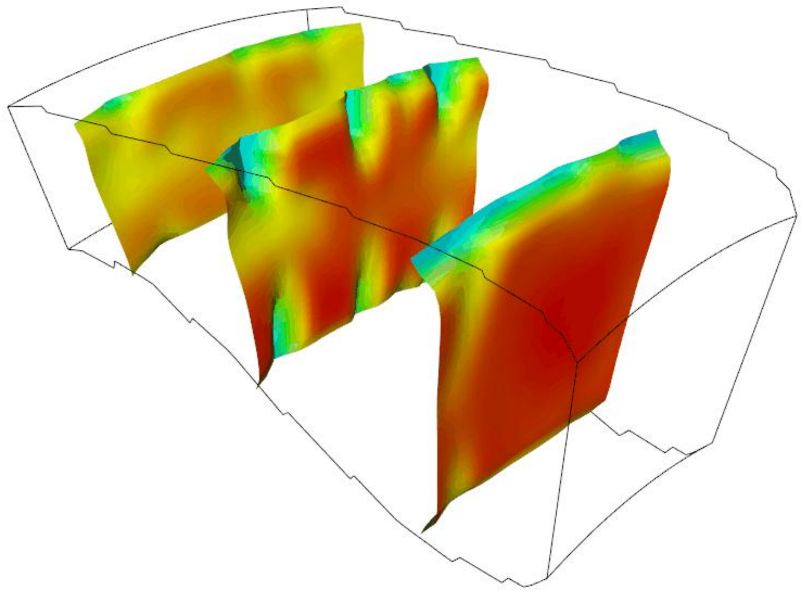


Note: scaling might be required

Vector Algorithms (continued)

Combination of techniques

We can also combine scalar and vector techniques by using a colormap:



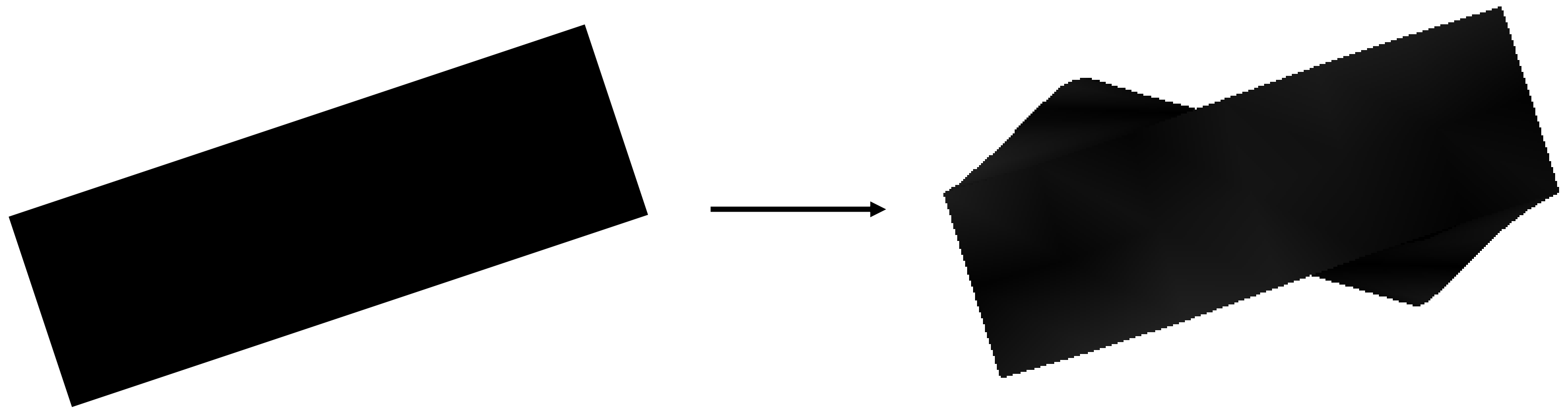
Vector Algorithms (continued)

Displacement plots

Vector displacement on the surface of an object can be visualized with displacement plots. A displacement plot shows the motion of an object in the direction perpendicular to its surface. The object motion is caused by an applied vector field. In a typical application the vector field is a displacement or strain field. A useful application of this technique is the study of vibration.

Vector Algorithms (continued)

In order to move an object's surface in normal direction using vector data, the vectors have to be converted into scalars by computing the dot product between the vector and the normal.



Vector Algorithms (continued)

Example

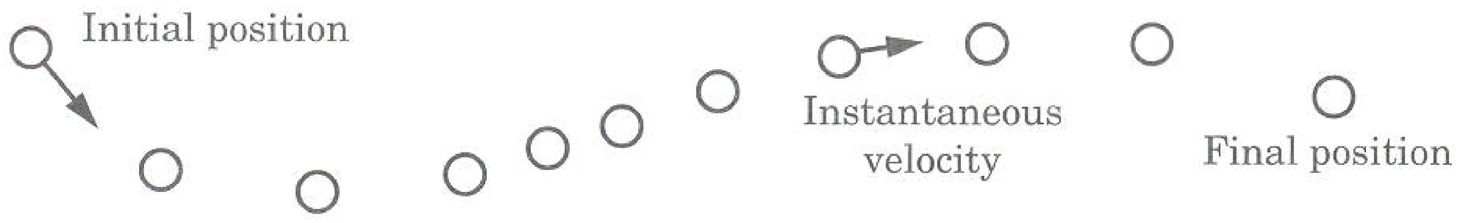
Displacement plot with color map

Vector Algorithms (continued)

Time animation

The idea is to move points (mass less particles) along the vector field. Basically, the particle is *advected* at every point in direction of the vector at that location (if necessary interpolation needs to be used), i.e. $v = dx/dt$.

Beginning with a sphere S centered about some point, we move S repeatedly to generate the bubbles below:



Vector Algorithms (continued)

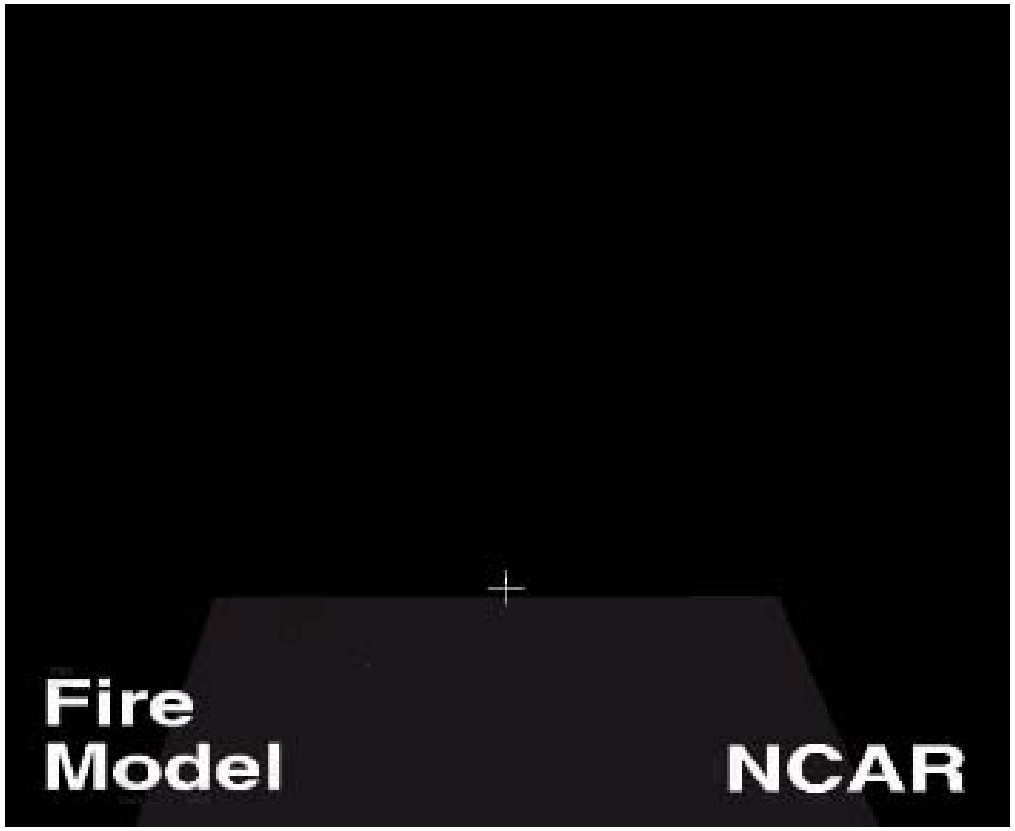
The eye tends to trace out a path by connecting the bubbles, giving the observer a qualitative understanding of the fluid flow in that area. The bubbles may be displayed as an animation over time (giving the illusion of motion) or as a multiple exposure sequence (giving the appearance of a path).

The choice of step size is a critical parameter in constructing accurate visualization of particle paths in a vector field. By taking large steps we are likely to jump over changes in the velocity. Using smaller steps we will end in a different position.

Vector Algorithms (continued)

Example

Particle advection for fire simulation



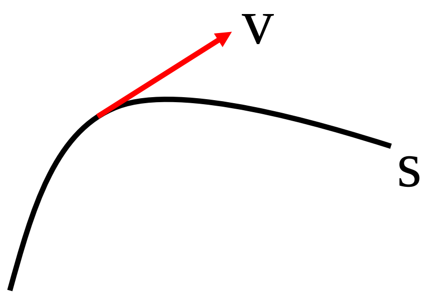
Vector Algorithms (continued)

Tracing particles

In order to determine the locations of a particle previously represented as a bubble, the particle needs to be traced throughout the vector field.

Since we are considering a massless particle, the particle basically follows the integral curve, i.e.

$$s'(x,t) = \vec{v}(s(x,t))$$



The initial position is user-defined.

Vector Algorithms (continued)

Although this form cannot be solved analytically for most real world data, its solution can be approximated using numerical integration techniques. Accurate numerical integration is a topic beyond the scope of this class, but it is known that the accuracy of the integration is a function of the step size. Since the path is an integration throughout the data set, the accuracy of the cell interpolation functions, as well as the accuracy of the original vector data, plays an important role in realizing accurate solutions.

Vector Algorithms (continued)

Euler's method

The simples form of numerical integration is Euler's method

$$\vec{x}_{i+1} = \vec{x}_i + \vec{v}(\vec{x}_i) \cdot \Delta t$$

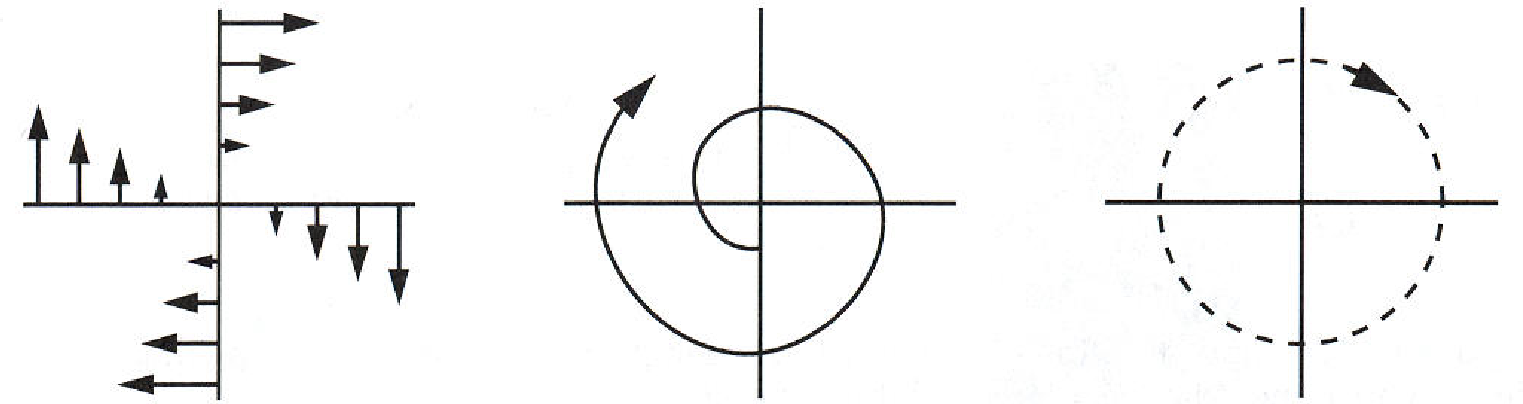
where x_i is the position and Δt the step size.

Euler's method has an error on the order of $O(\Delta t^2)$, which is not accurate enough for some applications.

Vector Algorithms (continued)

Example

Integral curves computed using two different techniques for a rotational vector field



(a) Rotational vector field (b) Euler's method (c) Runge-Kutta

Vector Algorithms (continued)

Runge-Kutta method

The family of explicit Runge-Kutta methods is given by

$$y_{n+1} = y_n + h \sum_{i=1}^{s} b_i k_i,$$

Where

$$\begin{aligned} k_1 &= f(t_n, y_n), \\ k_2 &= f(t_n + c_2 h, y_n + a_{21} h k_1), \\ k_3 &= f(t_n + c_3 h, y_n + a_{31} h k_1 + a_{32} h k_2), \\ &\vdots \\ k_s &= f(t_n + c_s h, y_n + a_{s1} h k_1 + a_{s2} h k_2 + \cdots + a_{s,s-1} h k_{s-1}). \end{aligned}$$

(Note: the above equations have different but equivalent definitions in different texts).

Vector Algorithms (continued)

To specify a particular method, one needs to provide the integer *s* (the number of stages), and the coefficients a_{ij} (for $1 \le j < i \le s$), b_i (for $i = 1, 2, ..., s$) and c_i (for $i = 2, 3, ..., s$). These data are usually arranged in a mnemonic device, known as a *Runge-Kutta tableau*:

$$
\begin{array}{c|ccccc}
0 & & & & & \\
c_2 & a_{21} & & & & \\
c_3 & a_{31} & a_{32} & & & \\
\vdots & \vdots & \vdots & & & \\
c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} & \\
\hline
 & b_1 & b_2 & \cdots & b_{s-1} & b_s
\end{array}
$$

The Runge-Kutta method is consistent if $\sum_{j=1}^{i-1} a_{ij} = c_i \text{ for } i = 2, \ldots, s.$

There are also accompanying requirements if we require the method to have a certain order *p*, meaning that the truncation error is O(*hp*+1). These can be derived from the definition of the truncation error itself. For example, a 2-stage method has order 2 if $b_1 + b_2 = 1$, $b_2c_2 = 1/2$, and $b_2a_{21} = 1/2$.

Vector Algorithms (continued)

Runge-Kutta technique of order 2

Hence, we get the following formula for the Runge-Kutta technique of order 2:

$$\vec{x}_{i+1} = \vec{x}_i + \frac{\Delta t}{2}(\vec{v}(\vec{x}_i) + \vec{v}(\vec{x}_{i+1}))$$

Vector Algorithms (continued)

Streamlines

We have seen that the step size is a design parameter. Hence, we can choose the step size in such a way that a line is formed. For a static vector field, i.e. a vector field that does not change over time, the integral curve results in a streamline.

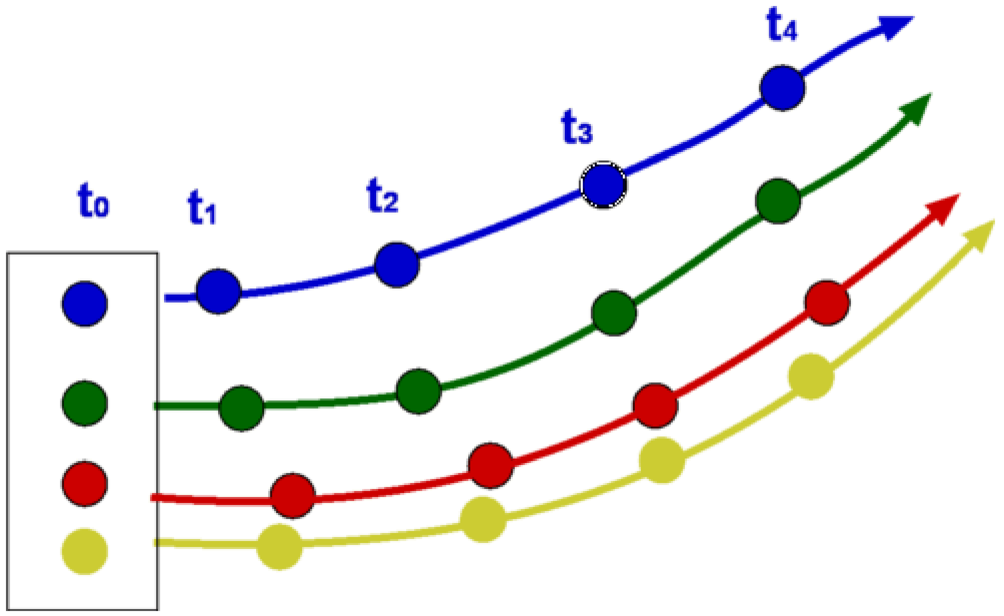
Different type types of integral curves exist:

- Pathlines
- Streaklines
- Streamlines

Vector Algorithms (continued)

Pathline

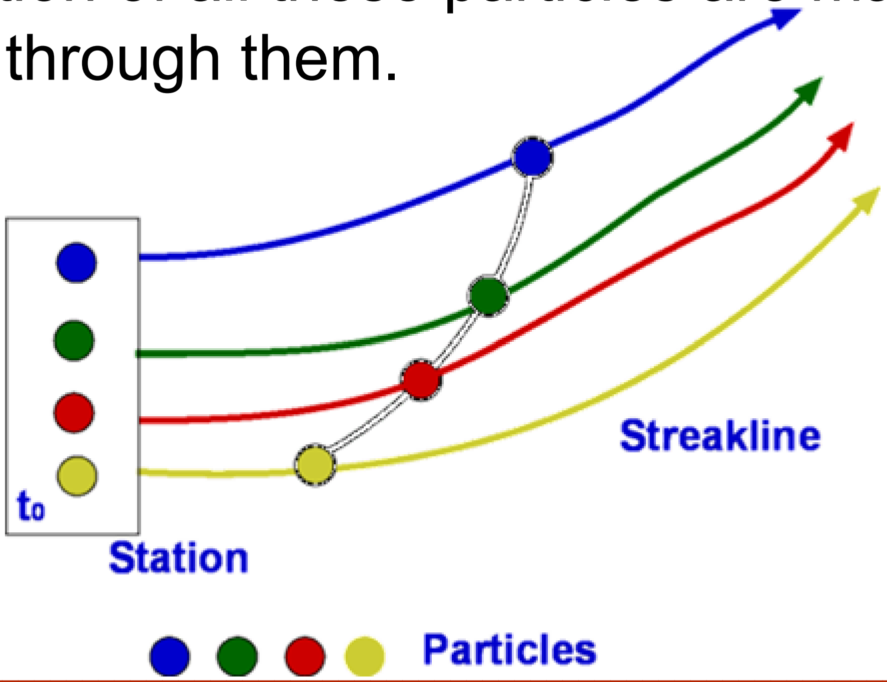
A pathline is the line traced by a given particle. This is generated by injecting a dye into the fluid and following its path by photography or other means



Vector Algorithms (continued)

Streakline

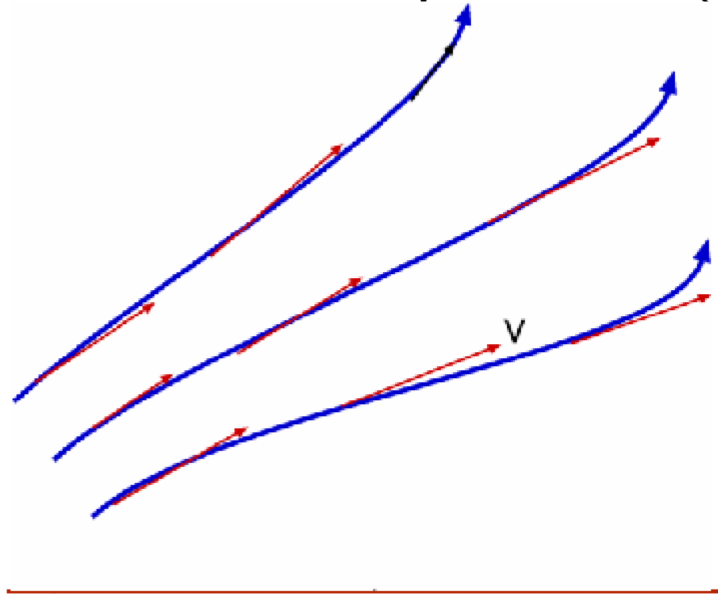
A streakline concentrates on fluid particles that have gone through a fixed station or point. At some instant of time the position of all these particles are marked and a line is drawn through them.

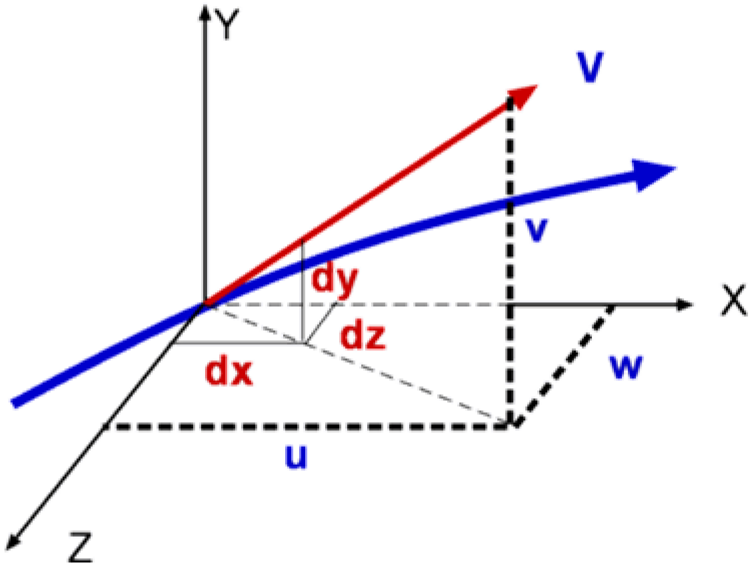


Vector Algorithms (continued)

Streamline

A streamline is one that is drawn tangential to the velocity vector at every point in the flow at a given instant and forms a powerful tool in understanding flows. Thus, it satisfies the equation $s'(x,t) = \vec{v}(s(x,t))$

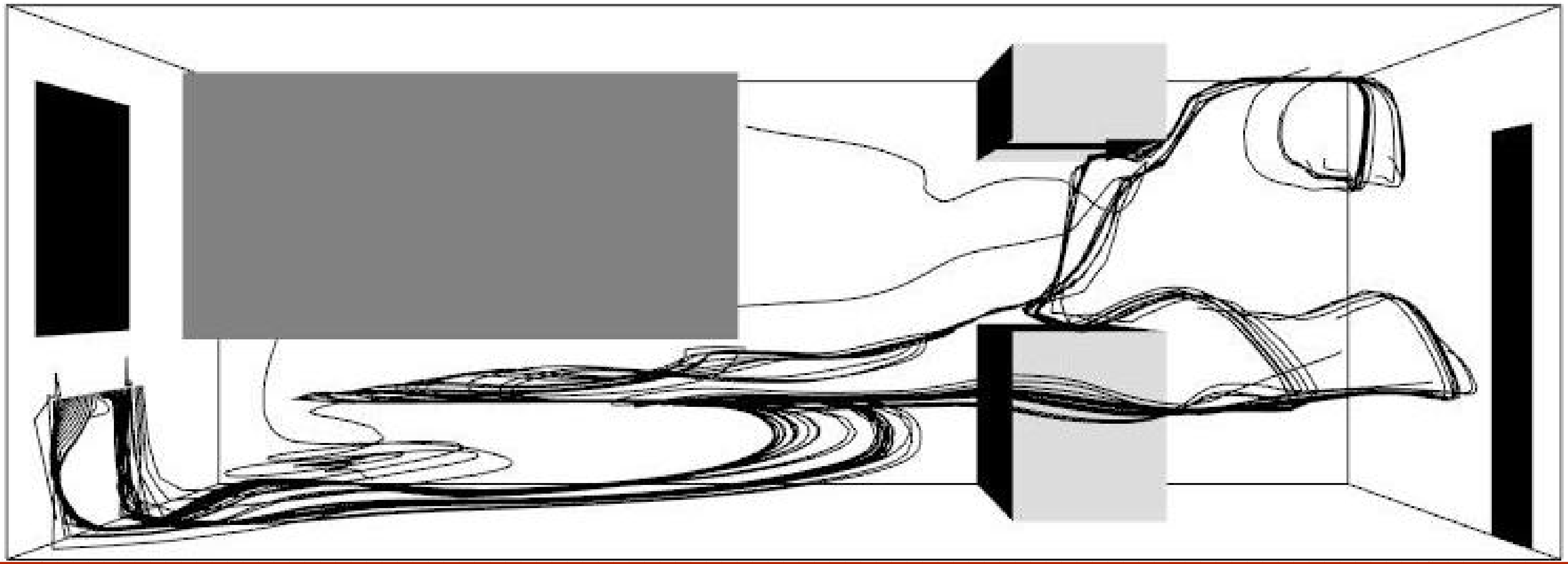




Vector Algorithms (continued)

Example

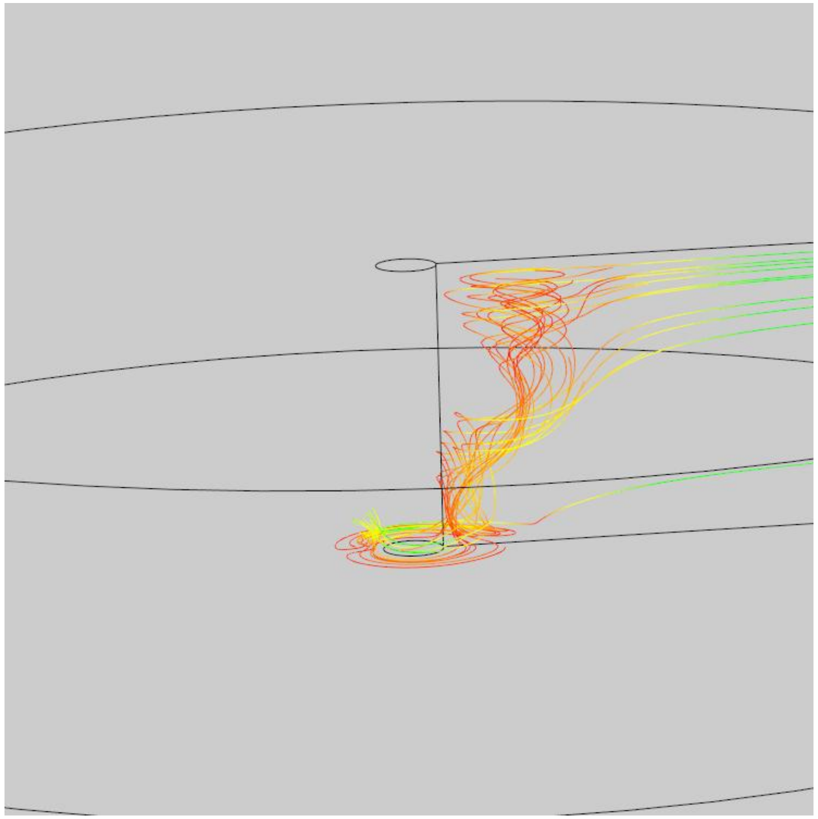
Flow velocity computed for a small kitchen (side view). Forty streamlines start along the rake positioned under the window. Some eventually travel over the hot stove and are convected upwards.



Vector Algorithms (continued)

Example

Flow around NASA’s tapered cylinder



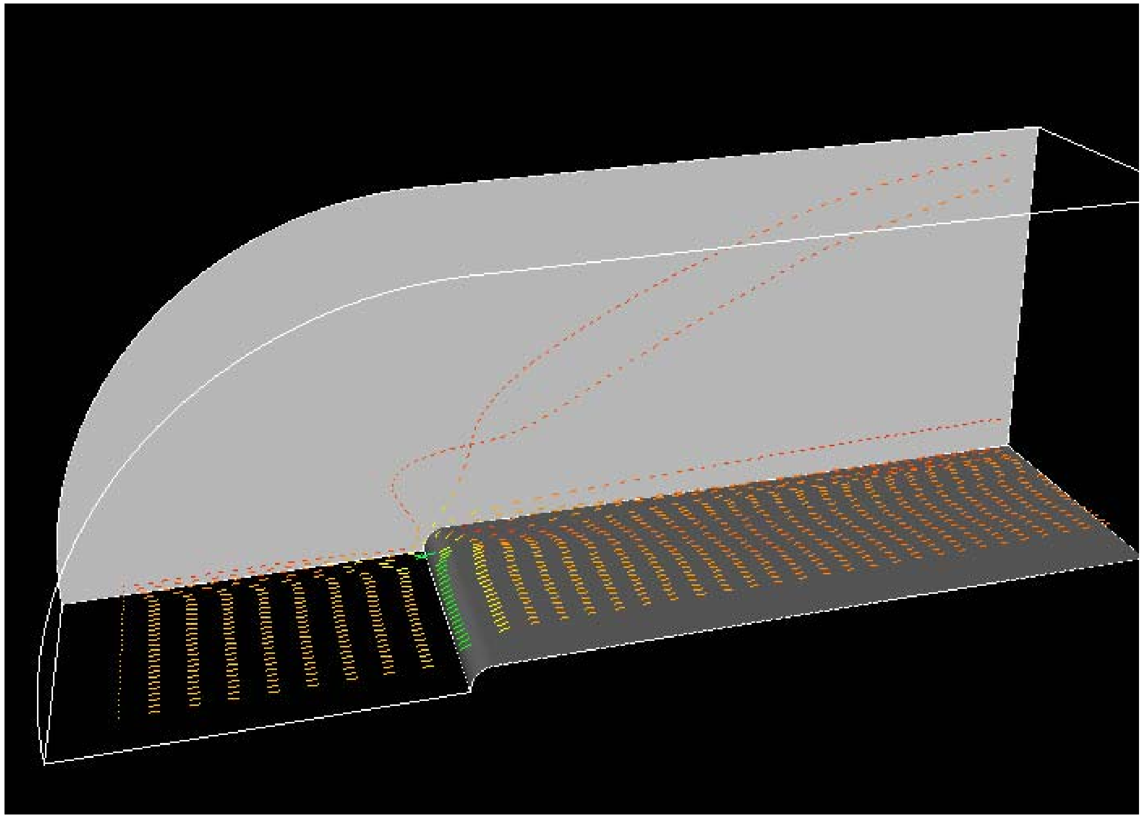
Vector Algorithms (continued)

Many enhancements of streamlines exist. Lines can be colored according to velocity magnitude to indicate speed of low. Other scalar quantities such as temperature or pressure also may be used to color the lines. We also may create constant time dashed lines. Each dash represents a constant time increment. This, in areas of high velocity, the length of the dash will be greater relative to regions of lower velocity.

Vector Algorithms (continued)

Example

NASA’s blunt fin data set



Tensor Algorithms

Dual vector space

The dual vector space to a real vector space V is the vector space of linear functions $f{:}V \rightarrow IR$, denoted V^*.

Tangent bundle

Every smooth manifold M has a tangent bundle TM, which consists of the tangent space TM_p at all points p in M. Since a tangent space TM_p is the set of all tangent vectors to M at p, the tangent bundle is the collection of all tangent vectors, along with the information of the point to which they are tangent.

$$TM = \{(p,v) : p \in M, v \in TM_p\}$$

Tensor Algorithms (continued)

Tensor space

A tensor space of type *(r,s)* can be described as a vector space tensor product between r copies of vector fields and s copies of the dual vector fields, i.e., one-forms. For example,

$$T^{(3,1)} = TM \otimes TM \otimes TM \otimes T^*M$$

is the vector bundle of (3,1)-tensors on a manifold M, where TM is the tangent bundle of M and T^*M is its dual. Tensors of type *(r,s)* form a vector space. This description generalized to any tensor type, and an invertible linear map $J{:}V \rightarrow W$ induces a map $\tilde{J} : V \otimes V^* \rightarrow W \otimes W^*$, where V^* is the dual vector space and J the Jacobian, defined by

$$\tilde{J}(v_1 \otimes v_2^*) = (Jv_1 \otimes (J^T)^{-1} v_2^*) \quad ,$$

where J^T is the pullback map of a form is defined using the transpose of the Jacobian. This definition can be extended similarly to other tensor products of V and V^*. When there is a change of coordinates, then tensors transform similarly, with J the Jacobian of the linear transformation.

Tensor Algorithms (continued)

Tensor

A tensor of order n in m-dimensional space is a mathematical object that has n indices and m^n components and obeys certain transformation rules. Each index of a tensor ranges over the number of dimensions of space. Tensors are generalizations of scalars (that have no indices), vectors (that have exactly one index), and matrices (that have exactly two indices) to an arbitrary number of indices.

Tensors provide a natural and concise mathematical framework for formulating and solving problems in areas of physics such as elasticity, fluid mechanics, and general relativity.

Tensor Algorithms (continued)

Tensors of different orders

The easiest form of a tensor is a tensor of order 0. Since it does not have any indices it is basically a simple scalar.

A tensor of order 1 has one index, i.e. it can represent a vector.

Tensors of order 2 (the ones we deal mostly in this chapter) have 2 indices and 3^2 entries and are usually represented as a matrix.

Tensor Algorithms (continued)

Two common examples for second order tensors in 3-D spaces are stress and strain tensors:

$$\begin{pmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_y & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_z \end{pmatrix} \qquad \begin{pmatrix} \frac{\partial u}{\partial x} & \left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial z}\right) & \left(\frac{\partial u}{\partial z}+\frac{\partial w}{\partial x}\right) \\ \left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial z}\right) & \frac{\partial v}{\partial y} & \left(\frac{\partial v}{\partial z}+\frac{\partial w}{\partial y}\right) \\ \left(\frac{\partial u}{\partial z}+\frac{\partial w}{\partial x}\right) & \left(\frac{\partial v}{\partial z}+\frac{\partial w}{\partial y}\right) & \frac{\partial w}{\partial z} \end{pmatrix}$$

stress tensor   strain tensor

Normal stresses in x-y-z coordinate directions are indicated as σ_x,σ_y,σ_z, shear stresses indicated as τ_{ij}. Material displacements are represented by u, v, w components.

Tensor Algorithms (continued)

According to Linear Algebra, a 3×3 real symmetric matrix can be characterized by three vectors in 3-D called eigenvectors, and three numbers called the eigenvalues of the matrix. The eigenvectors form a 3-D coordinate system whose axes are mutually perpendicular.

In some applications, particularly the study of materials, these aces also are referred to as the principle aces of the tensor and are physically significant. For example, if the tensor is a stress tensor, then the principle axes are the directions of normal stress and no shear stress.

Tensor Algorithms (continued)

Associated with each eigenvector is an eigenvalue. The eigenvalues are often physically significant as well. In the study of vibration, eigenvalues correspond to the resonant frequencies of a structure, and the eigenvectors are associated mode shapes.

Mathematically we can represent eigenvalues and eigenvectors as follows. Given a matrix A, the eigenvector x and eigenvalue λ must satisfy the relation

$$A \cdot x = \lambda x$$

For this equation to hold, the matrix determinant must satisfy

$$\left\| A - \lambda I \right\| = 0$$

Tensor Algorithms (continued)

Expanding this determinant yields a n-th degree polynomial (the characteristic polynomial) in λ whose roots are the eigenvalues. Thus, there are always n eigenvalues, although they may not be distinct (note that this is only guaranteed because the tensor is symmetric!).

Once we determine the eigenvalues, we can substitute each into the equation $\|A - \lambda I\| = 0$ to solve for the associated eigenvectors.

Tensor Algorithms (continued)

We can express the eigenvectors of the 3×3 system as

$$v_i = \lambda_i e_i \text{ , with } i = 1,2,3$$

With e_i a unit vector in the direction of the eigenvector and λ_i the eigenvalues of the system.

If we order the eigenvalues such that

$$\lambda_1 \geq \lambda_2 \geq \lambda_3$$

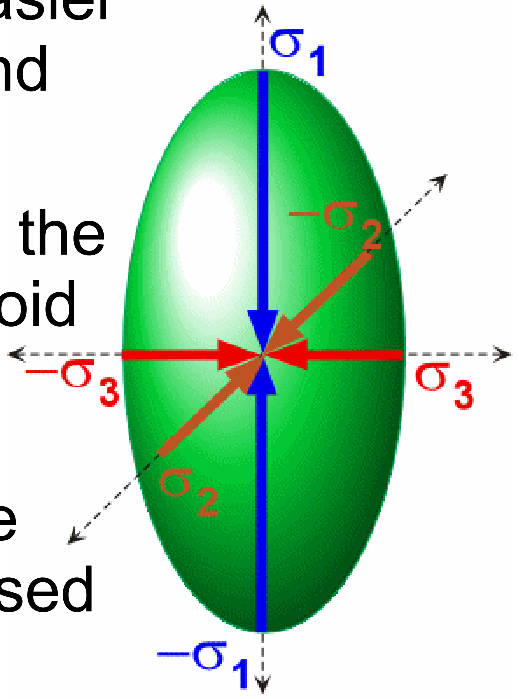
then we refer to the corresponding eigenvectors v_1, v_2, and v_3 as the major, medium, and minor eigenvectors. We use the same terminology for the eigenvalues.

Tensor Algorithms (continued)

Tensor visualization

Due to the high-dimensionality of tensors, almost all visualization techniques try to break a tensor down to a representation that is easier to understand. Usually, the eigenvectors and eigenvalues of the tensor are used.

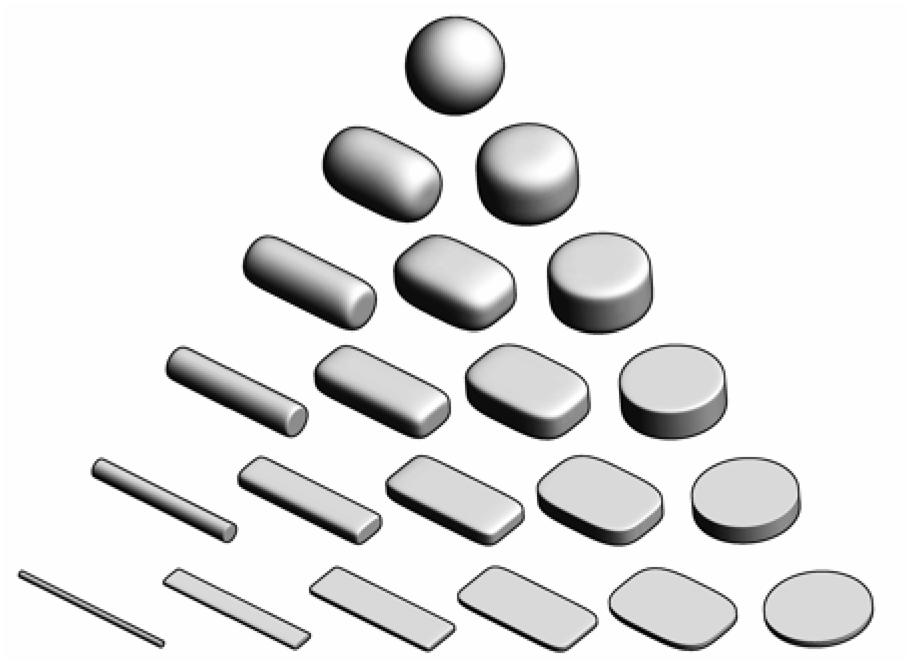
For example, the directions represented by the eigenvectors can be used to define a ellipsoid visualizing the tensor as a glyph. The eigenvalues are then used to scale the ellipsoid accordingly in each direction of the eivenvalue. Other types of glyphs can be used as well, such as quads.



Tensor Algorithms (continued)

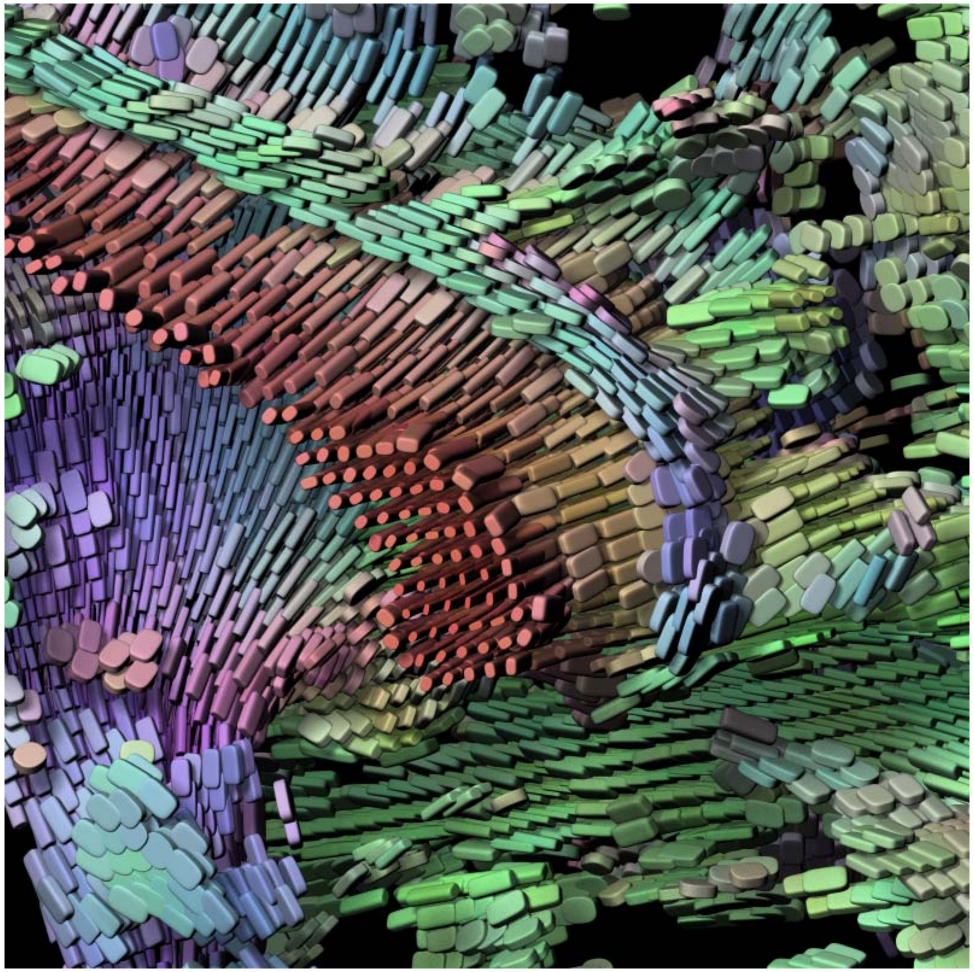
Tensor glyphs

Tensor glyphs were initially often used for DT-MRI images where isotropic and anisotropic tensors occur.



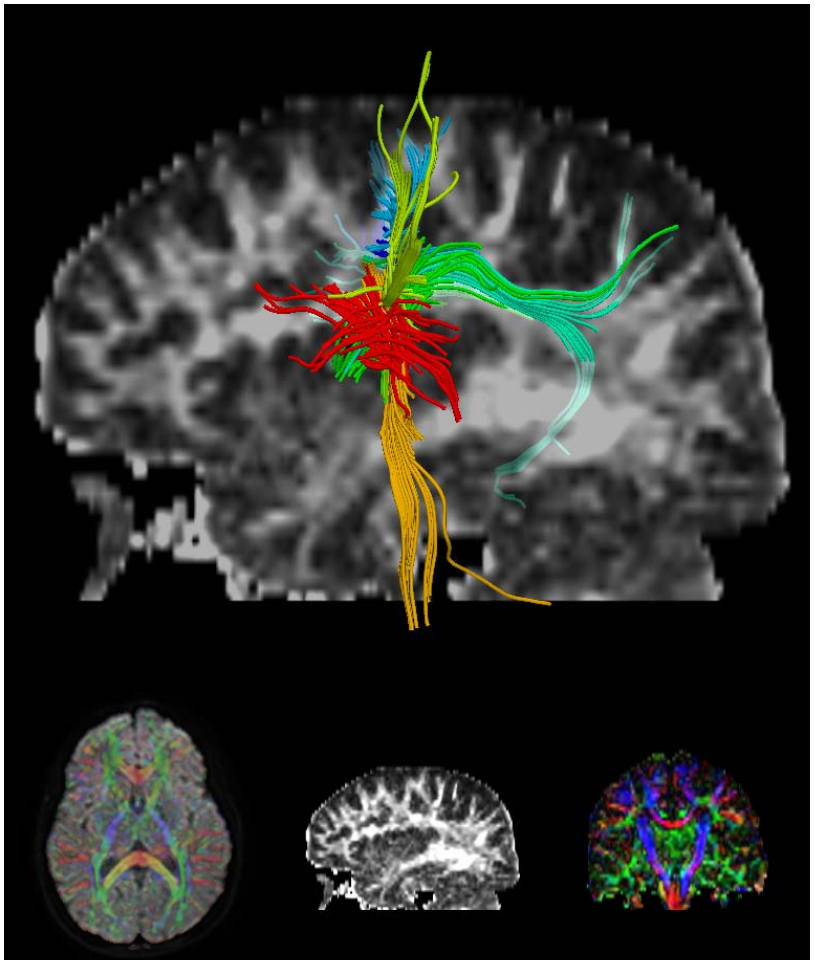
Tensor Algorithms (continued)

Example


Courtesy of Gordon Kindlmann

Tensor Algorithms (continued)

Fiber tracking


Courtesy of MIT Computer Science and AI Lab (CSAIL) Medical Vision Group

Modeling Algorithms

This category basically represents the default category and contains those visualization methods that do not really fit into one of the previously presented techniques.

Examples for modeling algorithms are

- Source objects
- Visualizing mathematical descriptions
- Cutting

Modeling Algorithms (continued)

Source objects

As we have seen in previous examples, source objects begin the visualization pipeline. Source objects are geometry used to support the visualization context or to read in data files. Some examples for source objects are:

- Modeling simple geometry
- Supporting geometry
- Visualizing mathematical descriptions

Modeling Algorithms (continued)

Modeling simple geometry

Spheres, cones, cubes, and other simple geometric objects can be used alone or in combination to model geometry. Often we use real-world applications such as air flow in a room and need to show real-world objects such as furniture, windows, or doors. Real-world objects often can be represented using these simple geometric representations. Alternatively, we may use a reader object to access geometric data defined in data files. These data files may contain more complex geometry, such as that produced by a 3-D CAD (Computer Aided Design) system.

Modeling Algorithms (continued)

Supporting geometry

During the visualization process we may use source objects to create supporting geometry. This may be as simple as three lines to represent a coordinate axis or as complex as tubes around line segments to thicken and enhance their appearance. Another common use us as supplemental input that defines a set of points. For streamlines, the points determine the initial position for generating the streamlines. The probe filter uses the points as the position to compute attribute values, such as scalars, vectors, or tensors.

Modeling Algorithms (continued)

Data attribute creation

Source objects can be used as procedures to create data attributes. For example, we can procedurally create textures and texture coordinates. Another use is to create scalar values over a uniform grid. If the scalar values are generated from a mathematical function, then we can use the visualization technique described here to visualize the function. For example, implicit functions are described by a mathematical formula and can be visualized directly.

Modeling Algorithms (continued)

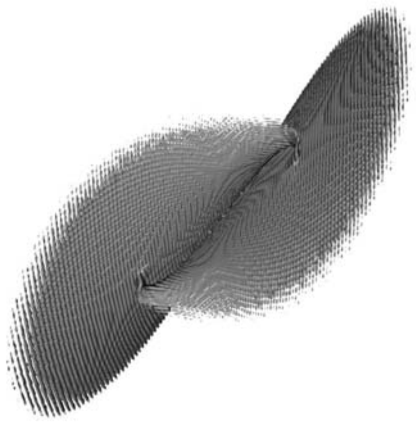
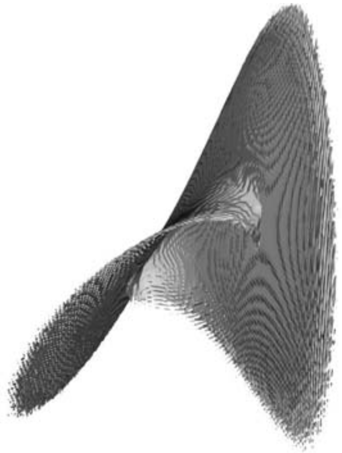
Visualizing mathematical descriptions

Some functions, often discrete or probabilistic in nature, cannot be cast into the form of an implicit function. However, by applying some creative thinking we can often generate scalar values that can be visualized. An interesting example of this is the so-called strange attractor. Strange attractors arise in the study of nonlinear dynamics and chaotic systems. In these systems, the usual types of dynamic motion – equilibrium, periodic motion, or quasi-periodic motion – are not present. Instead the system exhibits chaotic motion. Small perturbations can radically change the behavior of the system.

Modeling Algorithms (continued)

Lorentz attractor

A classical strange attractor was developed by Lorentz in 1963. Lorentz developed a simple model for thermally induced fluid convection in the atmosphere. Convection causes rings of rotating fluid and can be developed from the general Navier-Stokes partial differential equations for fluid flow.

Visualization of an isosurface of the number of visits at each voxel

Modeling Algorithms (continued)

Cutting

Often we want to cut through a data set with a surface and then display the interpolated data values on the surface. We refer to this technique as *data cutting* or simply *cutting*. The data cutting operation requires two pieces of information: a definition for the surface and a data set to cut. The easiest way for the algorithm is to define the cutting surface as an implicit function

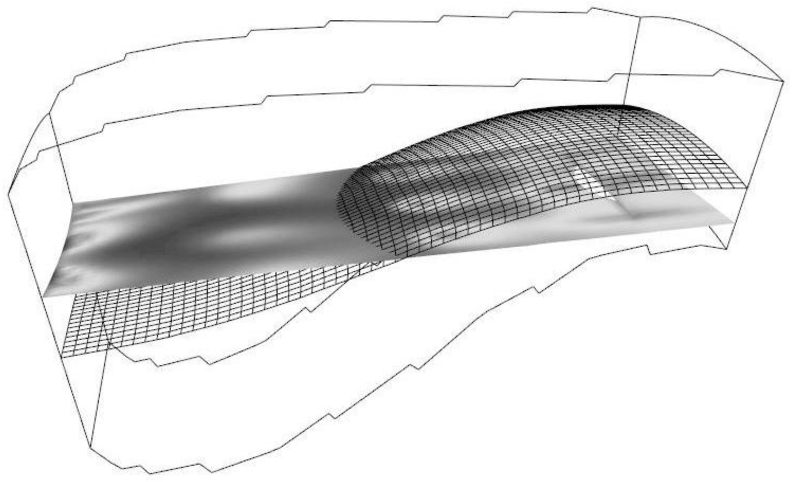
$$F(x, y, z) = 0.$$

Modeling Algorithms (continued)

The cutting algorithm proceeds as follows. For each cell, function values are generated by evaluating $F(x, y, z) = 0$ for each cell point. If all the points evaluate positive or negative, then the surface does not cut the cell. However, if the points evaluate positive and negative, then the surface passes through the cell. We can use the cell contouring operation to generate the isosurface $F(x, y, z) = 0$. Data attribute values can then be computed by interpolating along the cut edges.

Modeling Algorithms (continued)

Examples



Modeling Algorithms (continued)

Cutting can also be used to emulate volume rendering. By introducing a series of cutting planes perpendicular to the camera’s view plane normal and rendering the planes from back to front with an opacity of, for example, 0.05 we get a volume renderer.