Vector Visualization

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)

Divergence

Given a vector field $v: R^3 \rightarrow R^3$, the **divergence** of $v=(v_x,v_y,v_z)$ is the scalar quantity

$$div\, v = \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z}$$

Intuitively, if v is a flow field that transports mass, $div\, v$ characterizes the increase or loss of mass at a given point p in the vector field in unit time.

- positive divergence at p: mass spreads from p outward.
- negative divergence at p: mass gets sucked into p.
- zero divergence at p: mass is transported without getting spread or sucked, i.e. without compression or expansion.

Vector Algorithms (continued)

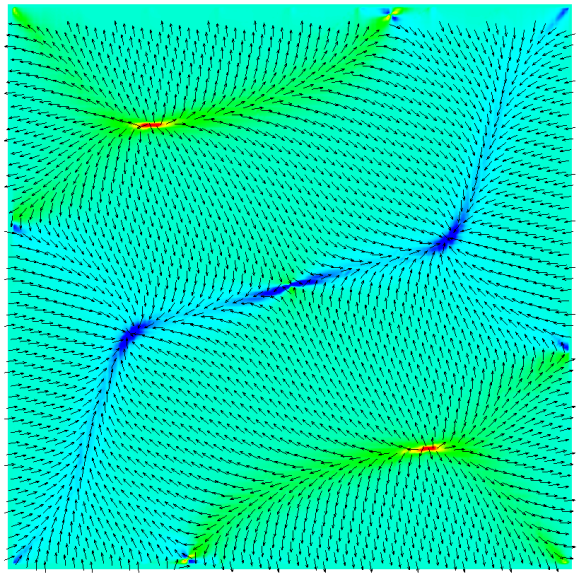
Vorticity

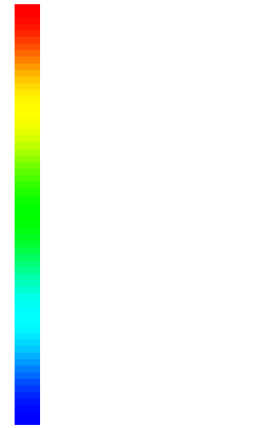
Given a vector field $v:R^3 \rightarrow R^3$, the **vorticity** of $v=(v_x,v_y,v_z)$, also called **curl** or **rotor**, is the vector quantity

$$rot\ v = \left(\frac{\partial v_z}{\partial y} - \frac{\partial v_y}{\partial z}, \frac{\partial v_x}{\partial z} - \frac{\partial v_z}{\partial x}, \frac{\partial v_y}{\partial x} - \frac{\partial v_x}{\partial y} \right)$$

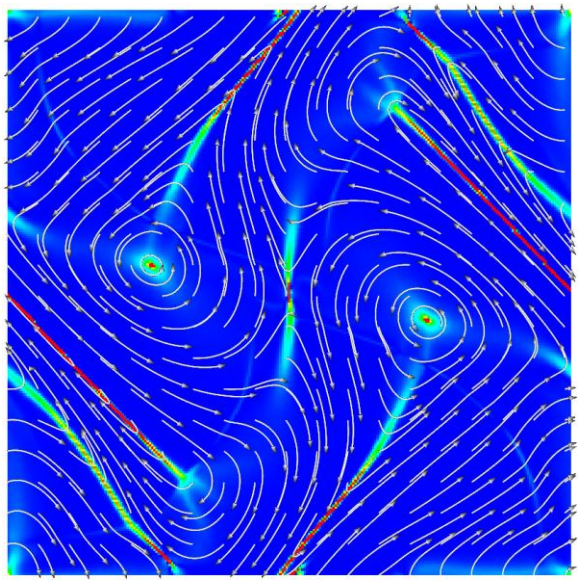
The vorticity $rot\ v$ of v is a vector field that is locally perpendicular to the plane or rotation of v and whose magnitude expresses the speed of angular rotation of v around $rot\ v$. Hence, the vorticity vector characterizes the speed and direction of rotation of a given vector field at every point. Sometimes $rot\ v$ is also denoted as $curl\ v$.

Vector Algorithms (continued)





Divergence of a 2D vector field

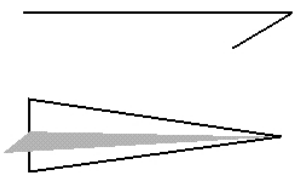


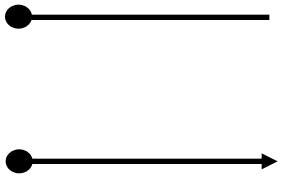
Vorticity of a 2D vector field

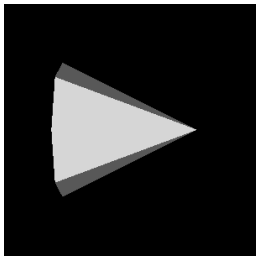
Images courtesy of Alexandru Telea

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.



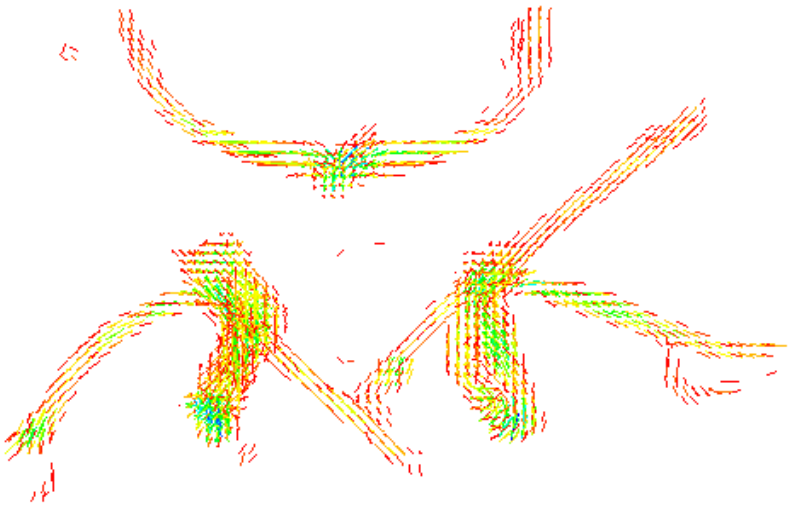


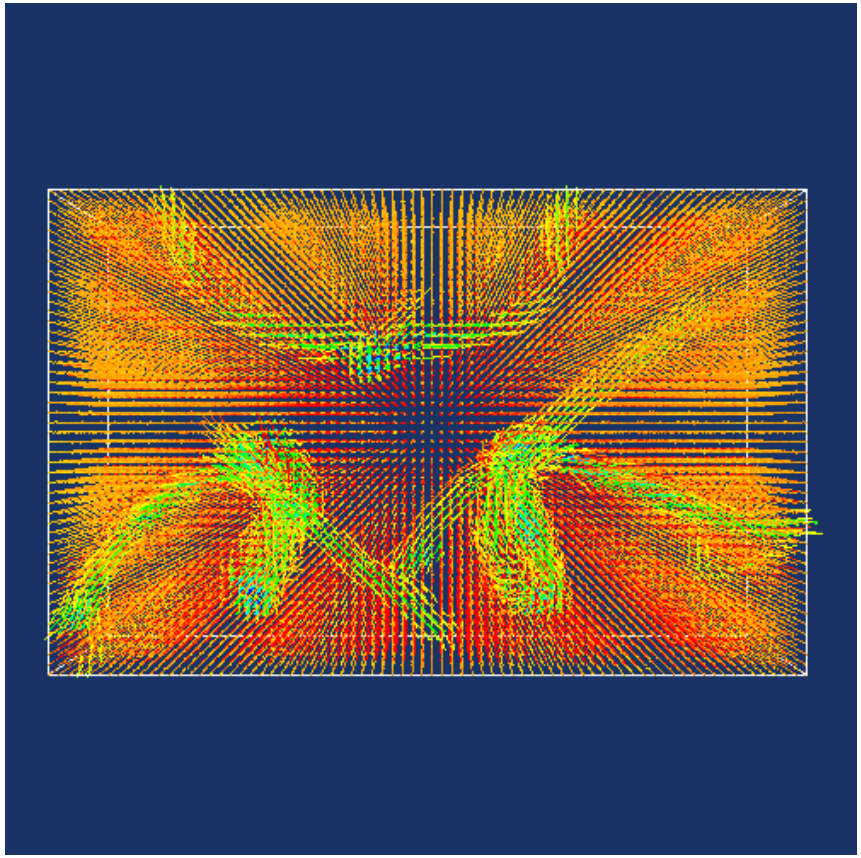


Direction can also be visualized using color coding by using different colors at each ends of the glyph

Vector Algorithms (continued)

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





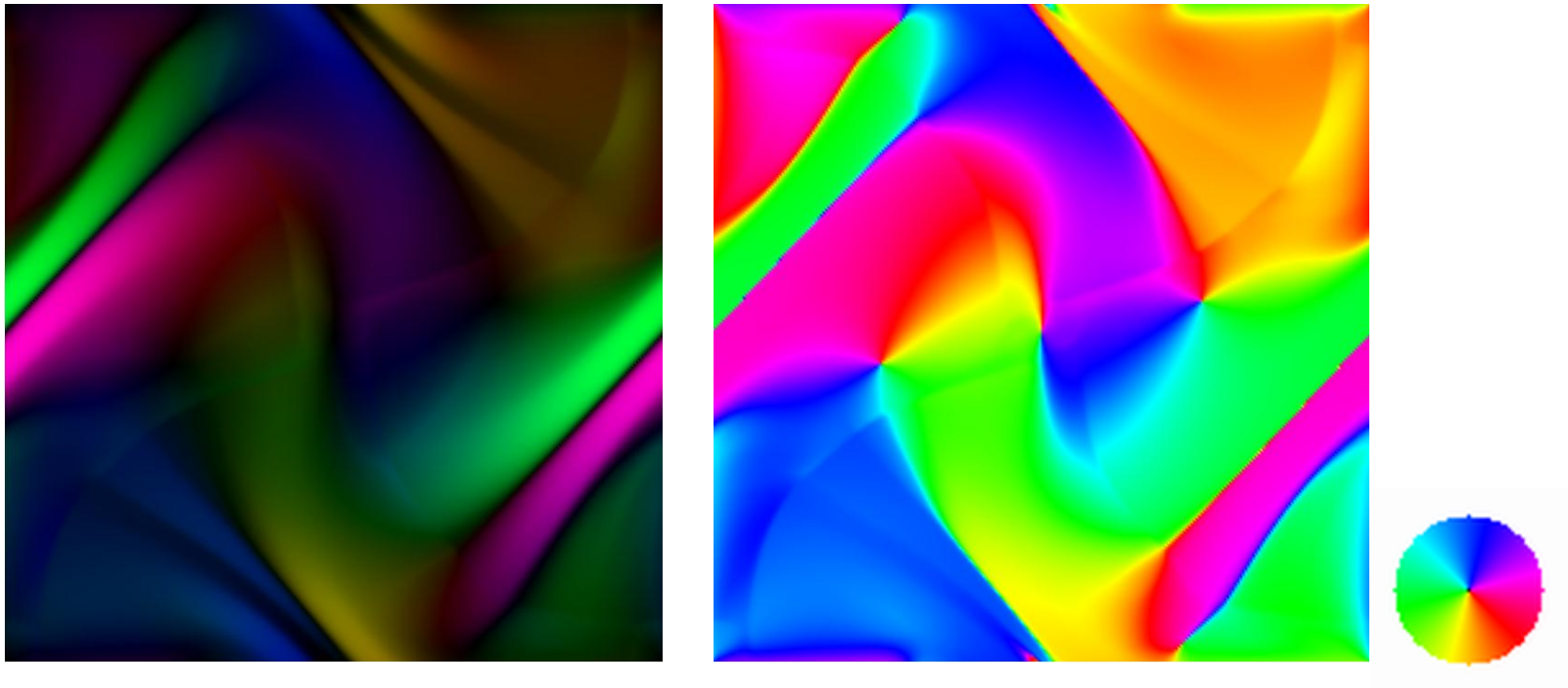
Vector Algorithms (continued)

Vector Color Coding

One of the simplest techniques to produce a visualization is to use **vector color coding**. Similar to scalar color mapping, vector color coding associates a color with every point of a given surface, on which we have defined a vector dataset. The color is used to encode the vector orientation and direction attributes. Every distinct hue corresponds to a different angle of the color wheel: red is 0º, magenta 60º, blue is 120º, cyan is 180º, green is 240º, and yellow is 300º. In addition, the vector magnitude can be encoded as the luminance, i.e. long vectors result in full color whereas shorter vectors tend to be represented as black.

Vector Algorithms (continued)

Example



Orientation and magnitudeOrientation only

Images courtesy of Alexandru Telea

Vector Algorithms (continued)

Warping

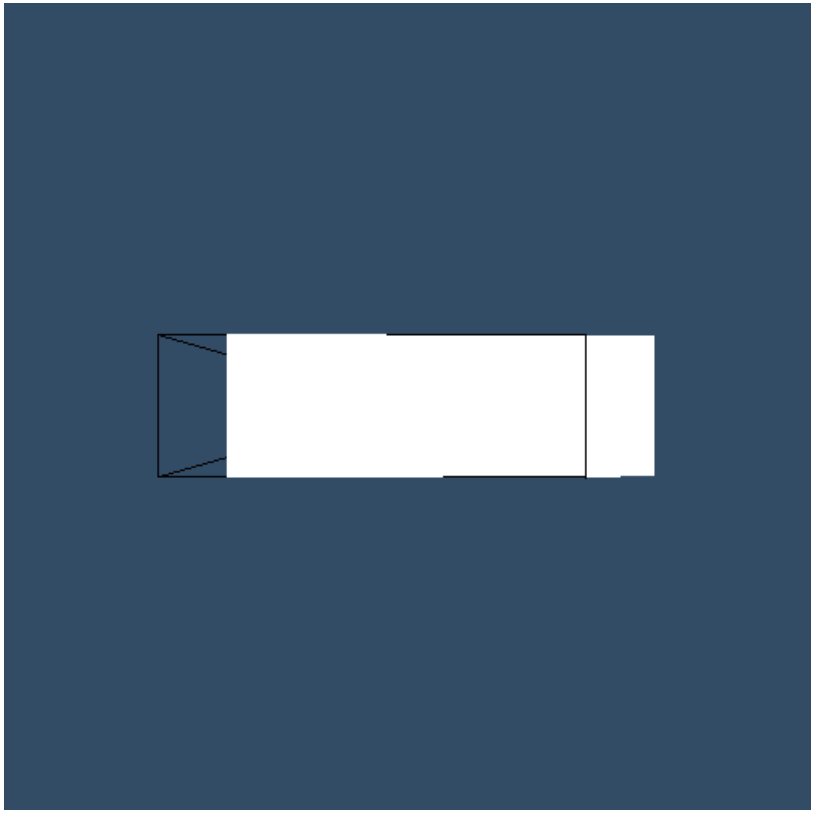
Vector data is often associated with motion. The motion is in the form of velocity or displacement. An effective technique for displaying such vector data is to “warp” or deform geometry according to the vector field. For example, imagine representing the displacement of a structure under load by deforming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

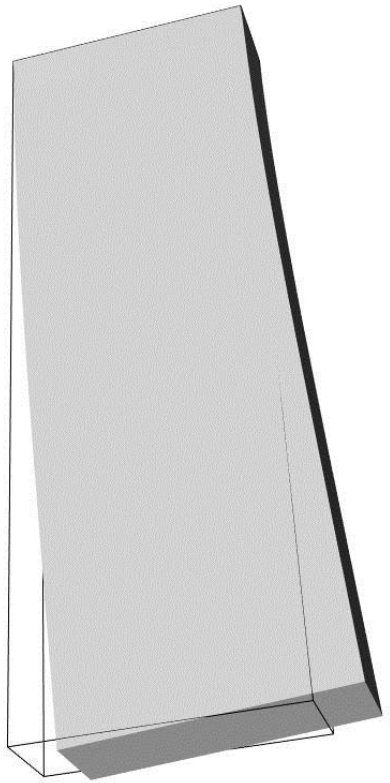
The warping technique should – as usual – be applied with the application in mind.

Vector Algorithms (continued)

Example

The motion of a vibrating beam

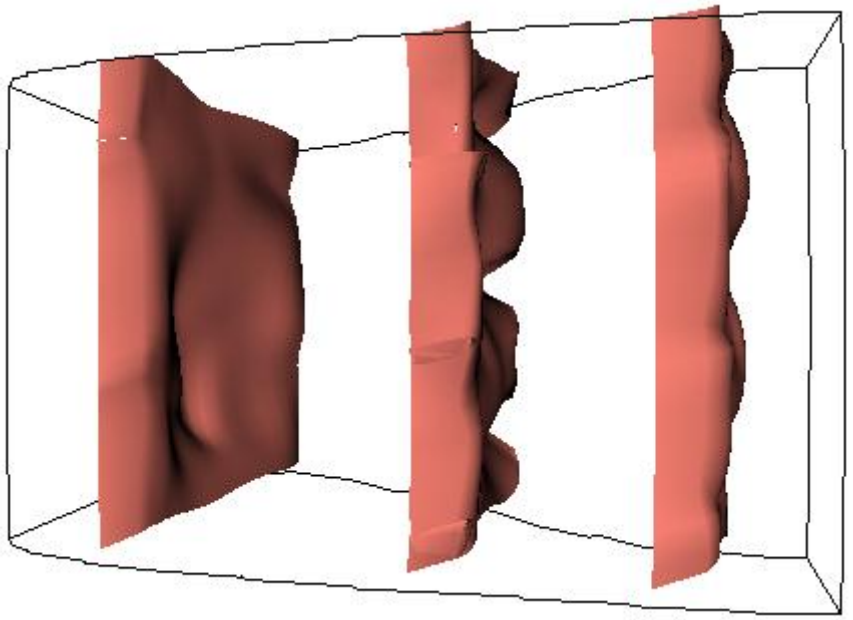




Vector Algorithms (continued)

Example

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

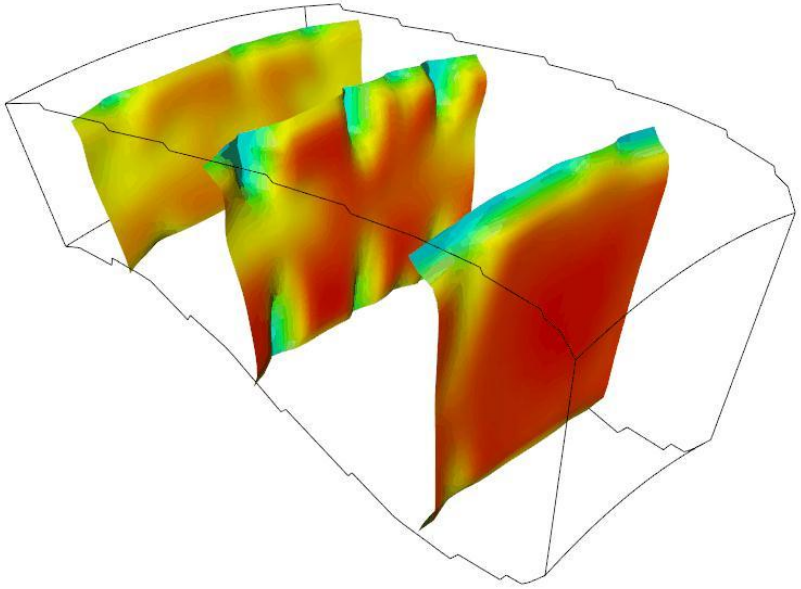


Note: scaling might be required

Vector Algorithms (continued)

Combination of techniques

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



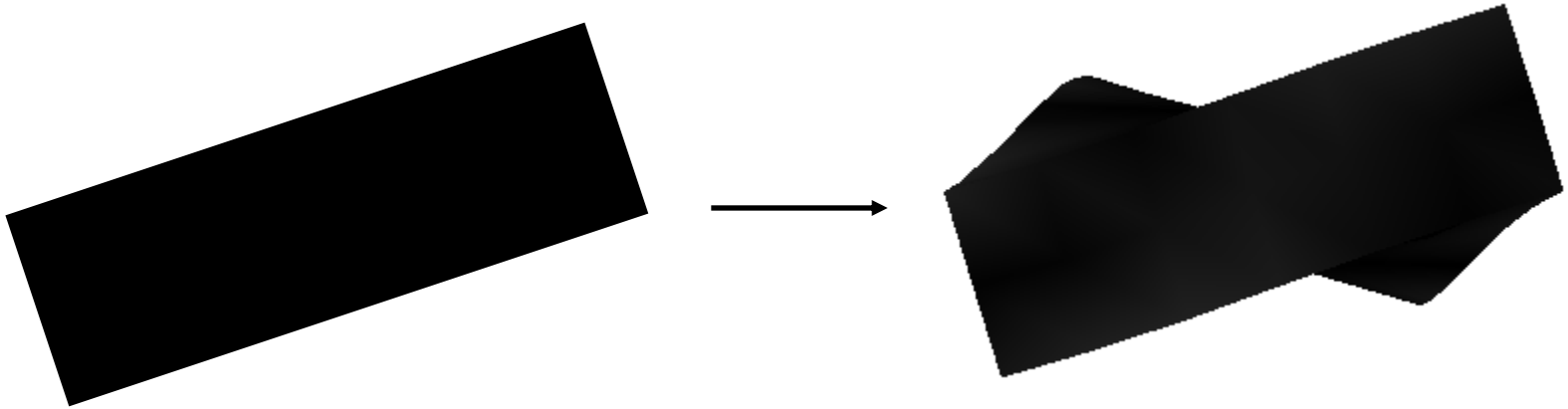
Vector Algorithms (continued)

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)

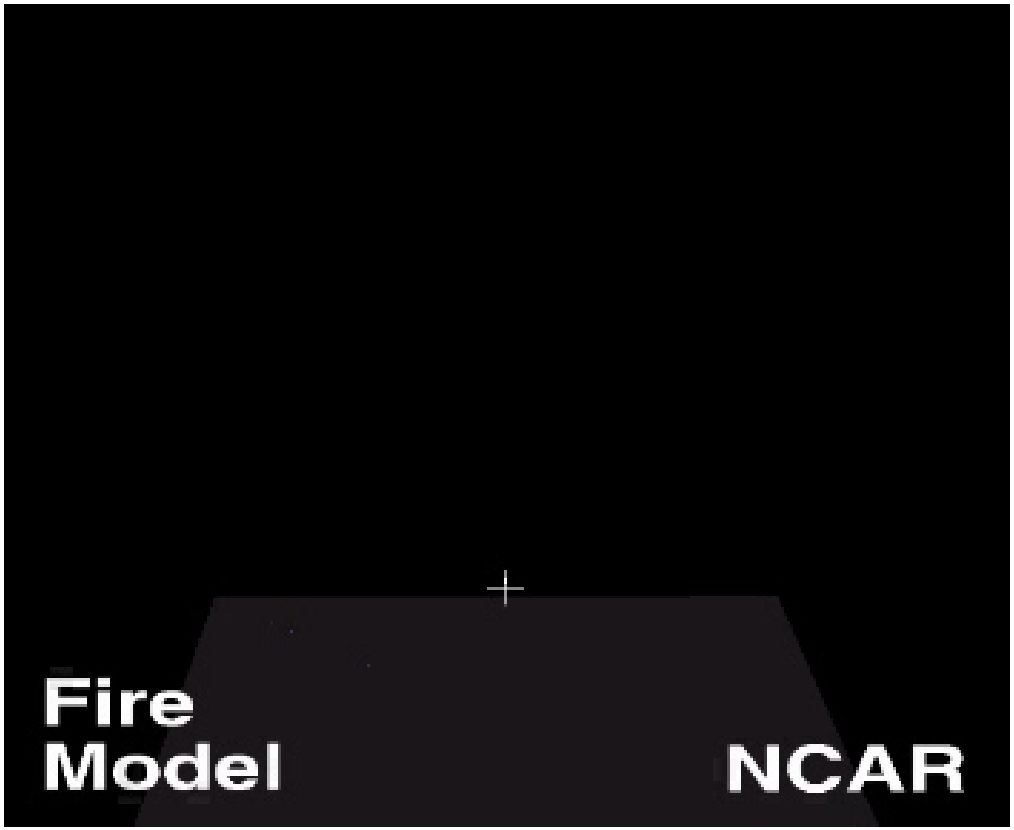
The eye tends to trace out a path by connecting the bubbles, giving the observer a qualitative understanding of the fluid flow in that area. The bubbles may be displayed as an animation over time (giving the illusion of motion) or as a multiple exposure sequence (giving the appearance of a path).

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



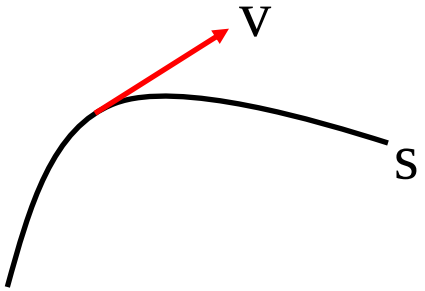
Vector Algorithms (continued)

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 mass-less 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

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

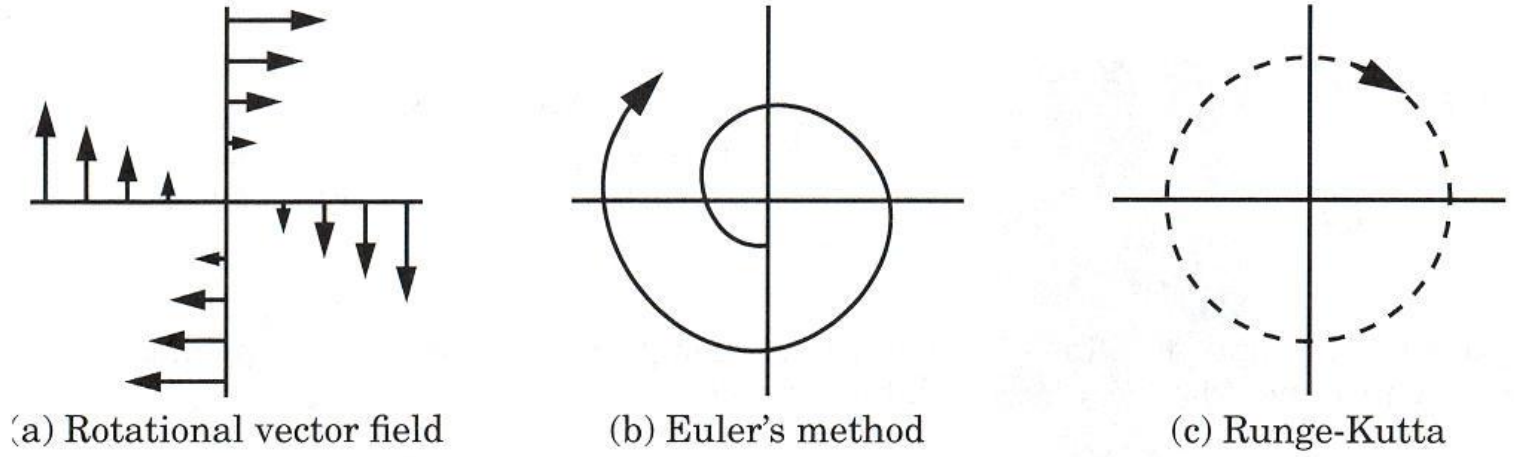
where x_i is the position and Δt the step size.

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*:

0					
c_2	a_{21}				
c_3	a_{31}	a_{32}			
:	:	:			
c_s	a_{s1}	a_{s2}	…	$a_{s,s-1}$	
	b_1	b_2	…	b_{s-1}	b_s

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.

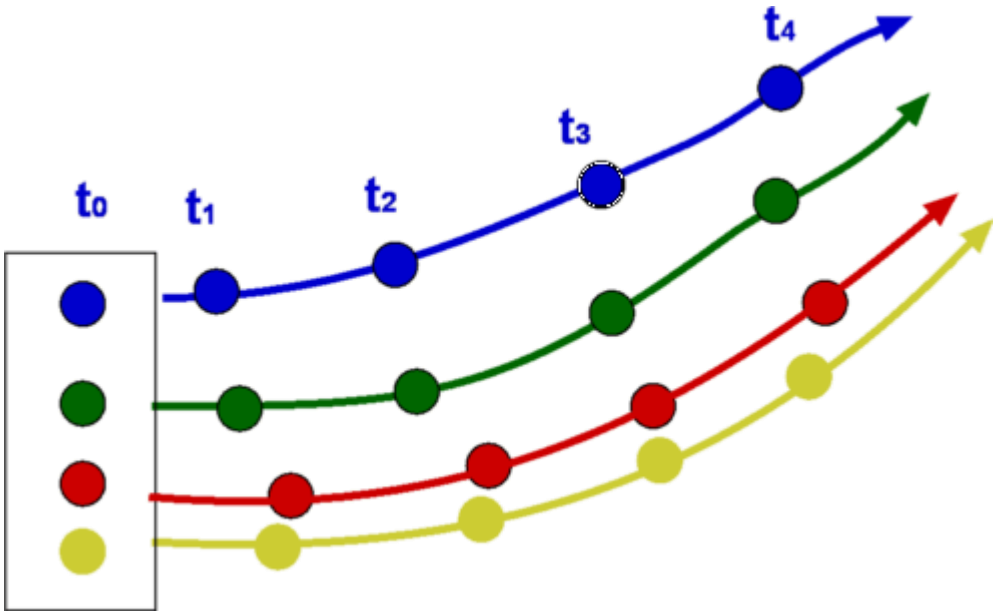
Different type types of integral curves exist:

- Pathlines
- Streaklines
- Streamlines

Vector Algorithms (continued)

Pathline

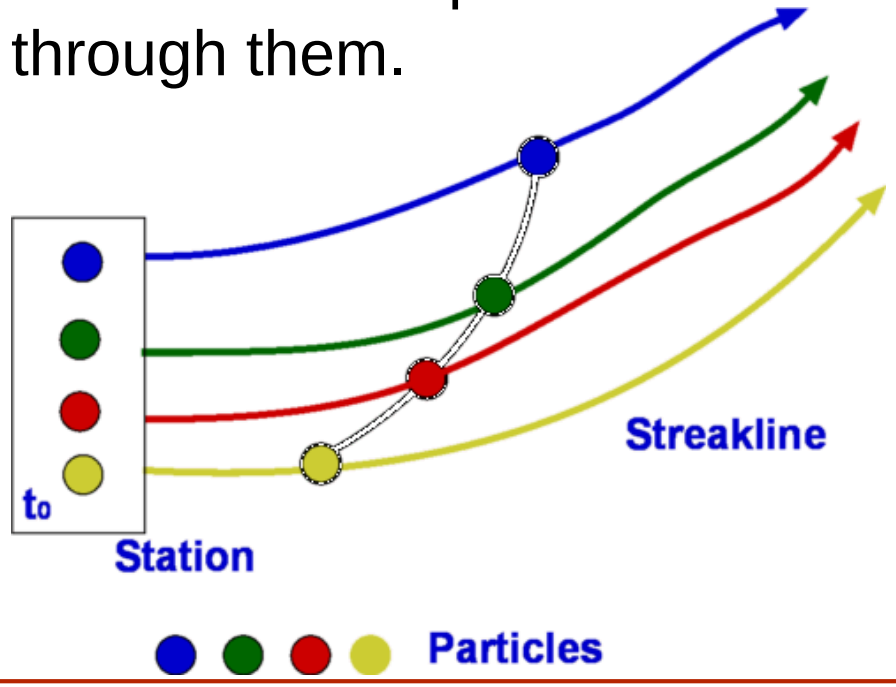
A pathline is the line traced by a given particle. This is generated by injecting a dye into the fluid and following its path by photography or other means



Vector Algorithms (continued)

Streakline

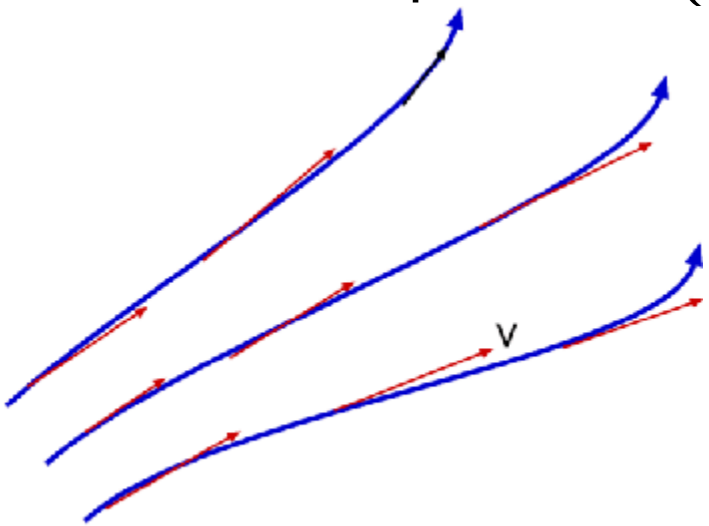
A streakline concentrates on fluid particles that have gone through a fixed station or point. At some instant of time the position of all these particles are marked and a line is drawn through them.

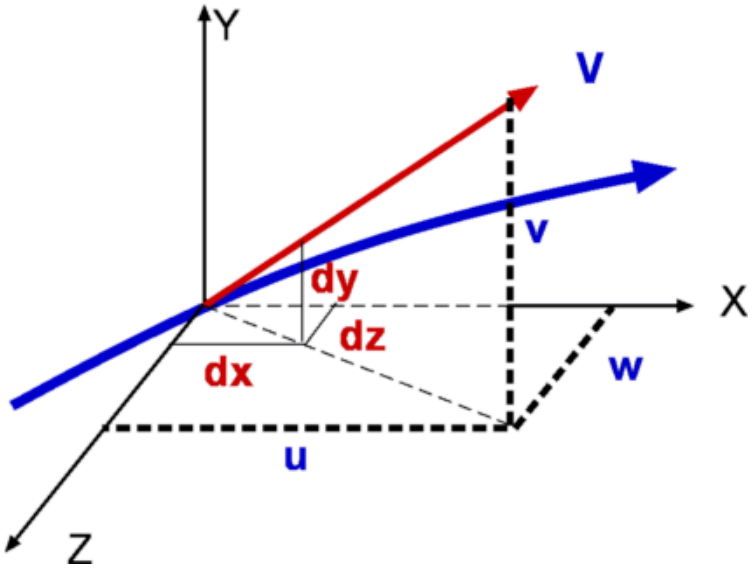


Vector Algorithms (continued)

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

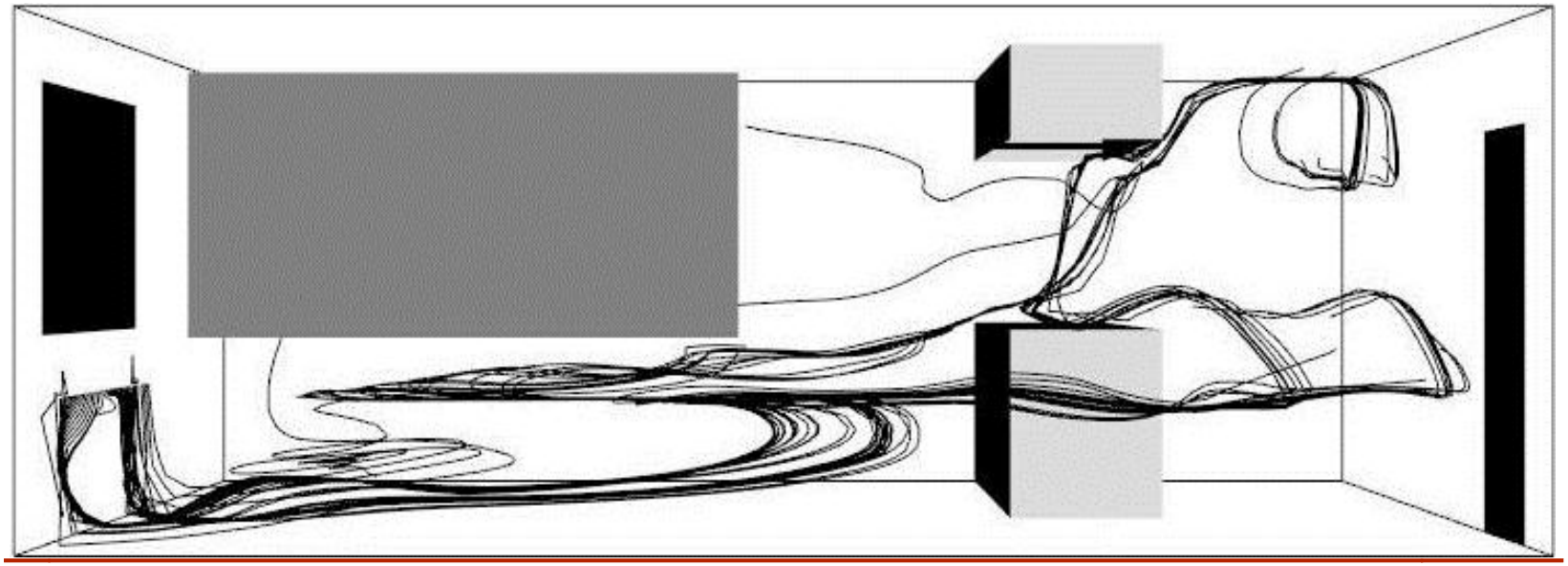




Vector Algorithms (continued)

Example

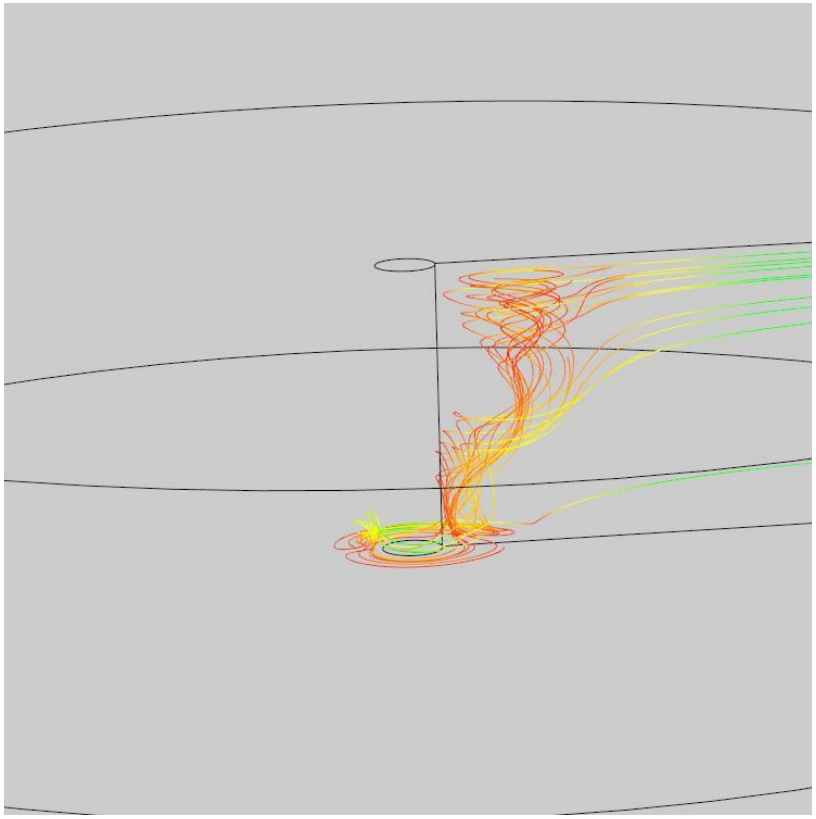
Flow velocity computed for a small kitchen (side view). Forty streamlines start along the rake positioned under the window. Some eventually travel over the hot stove and are convected upwards.



Vector Algorithms (continued)

Example

Flow around NASA's tapered cylinder



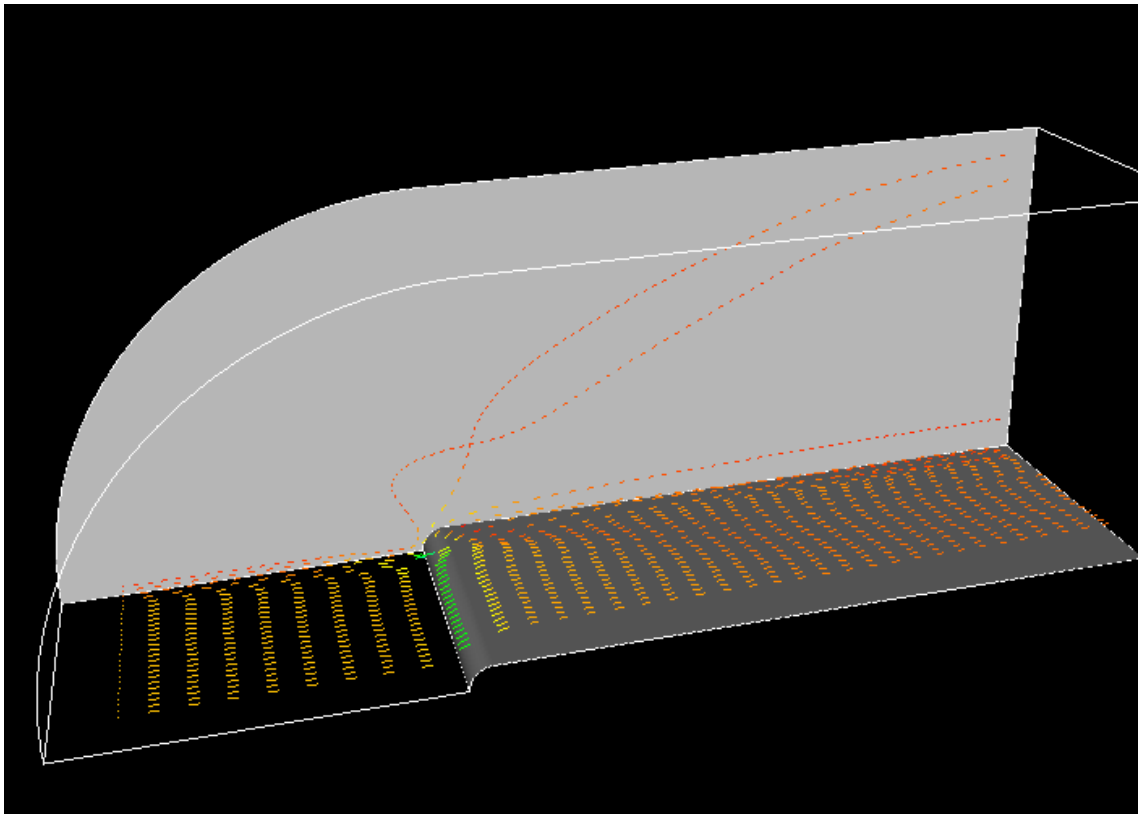
Vector Algorithms (continued)

Many enhancements of streamlines exist. Lines can be colored according to velocity magnitude to indicate speed of flow. 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



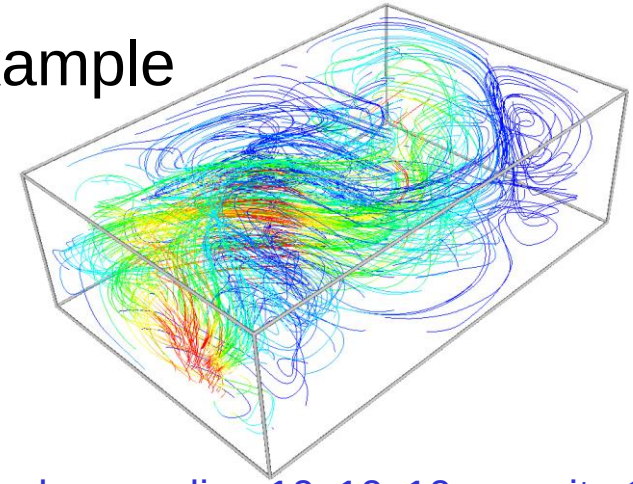
Vector Algorithms (continued)

Choosing an appropriate sampling strategy that solves the coverage, density, and continuity issues well is more critical when tracing streamlines in 3D datasets as compared to 2D datasets.

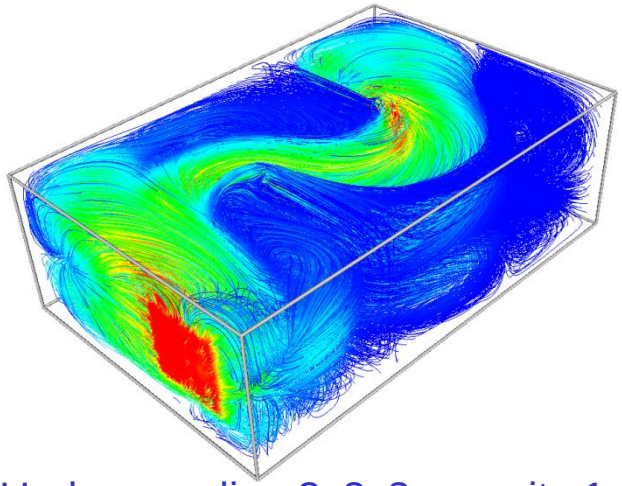
Similar to the using glyphs, streamline visualizations can get cluttered if the parameters for placement and opacity are not chosen properly.

Vector Algorithms (continued)

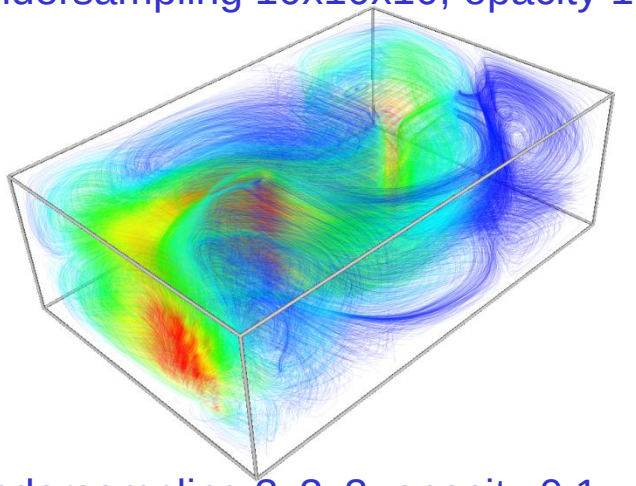
Example



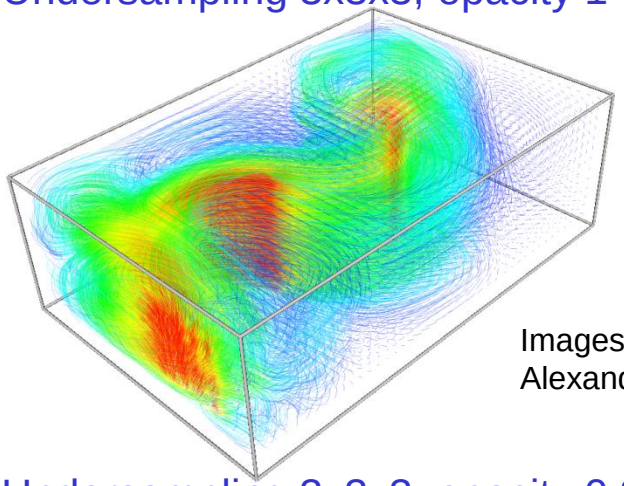
Undersampling 10x10x10, opacity 1



Undersampling 3x3x3, opacity 1



Undersampling 3x3x3, opacity 0.1



Undersampling 3x3x3, opacity 0.3

Images courtesy of Alexandru Telea

Vector Algorithms (continued)

Previously, we showed how to create simple vector glyphs and how to integrate particles through a vector field to create streamlines. Now we extend these concepts to integrate more data into our visualization.

We start with *streamribbons* and *streamsurfaces*.

Streamlines depict particle paths in a vector field. By coloring these lines, or creating local glyphs (such as dashed lines or oriented cones), we can present additional scalar and local information. However, these techniques can convey only elementary information about the vector field. Local information, e.g. flow rotation or derivatives, and global information, e.g. structure of a field such as vortex tubes, is not represented.

Vector Algorithms (continued)

Streamribbons

Streamribbons are a technique that includes local information, such as flow rotation. They are a natural extension of the streamline technique. The ribbon is created by widening the line. It can be constructed by generating two adjacent streamlines and then bridging the lines with a polygonal mesh. This technique works well as long as the streamlines remain relatively close to one another. If separation occurs, so that the streamlines diverge, the resulting ribbon will not accurately represent the flow.

Vector Algorithms (continued)

The streamribbon provides information about important flow parameters: the vector vorticity and flow divergence. *Vorticity* ω is the measure of rotation of the vector field, expressed as a vector quantity: a direction (axis of rotation) and magnitude (amount of rotation). *Streamwise vorticity* Ω is the projection of along the instantaneous velocity vector v. In other words, streamwise vorticity is the rotation of the vector field around the streamline defined as follows:

$$\Omega = \frac{v \cdot \omega}{|v| \cdot |\omega|}$$

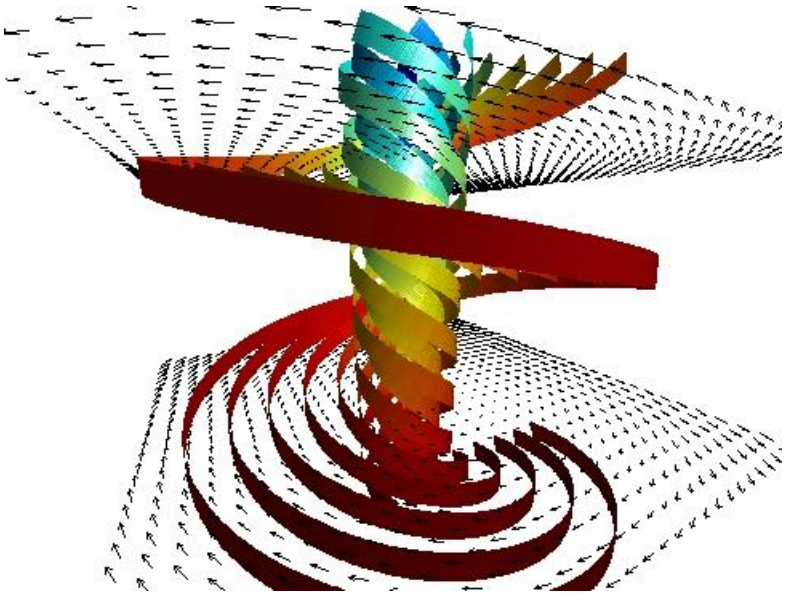
The amount of twisting of the streamribbon approximates the streamwise vorticity.

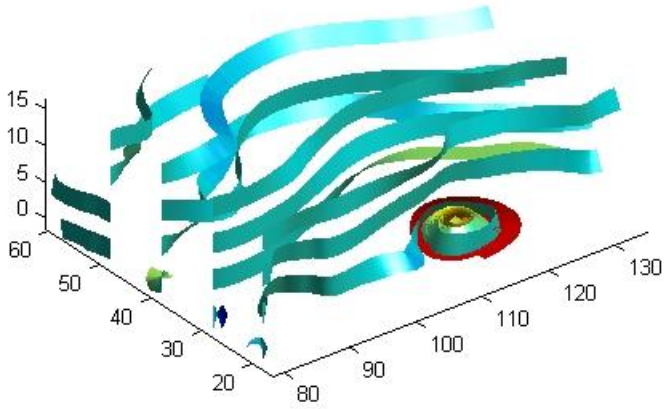
Vector Algorithms (continued)

Flow *divergence* is a measure of the spread of the flow. The changing width of the streamribbon is proportional to the cross-flow divergence of the flow.

Vector Algorithms (continued)

Examples



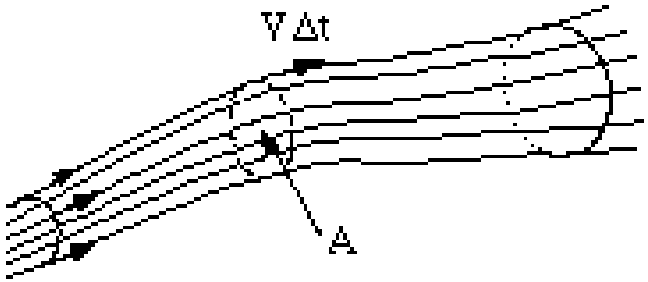


Images courtesy of Mathworks

Vector Algorithms (continued)

Streamtubes

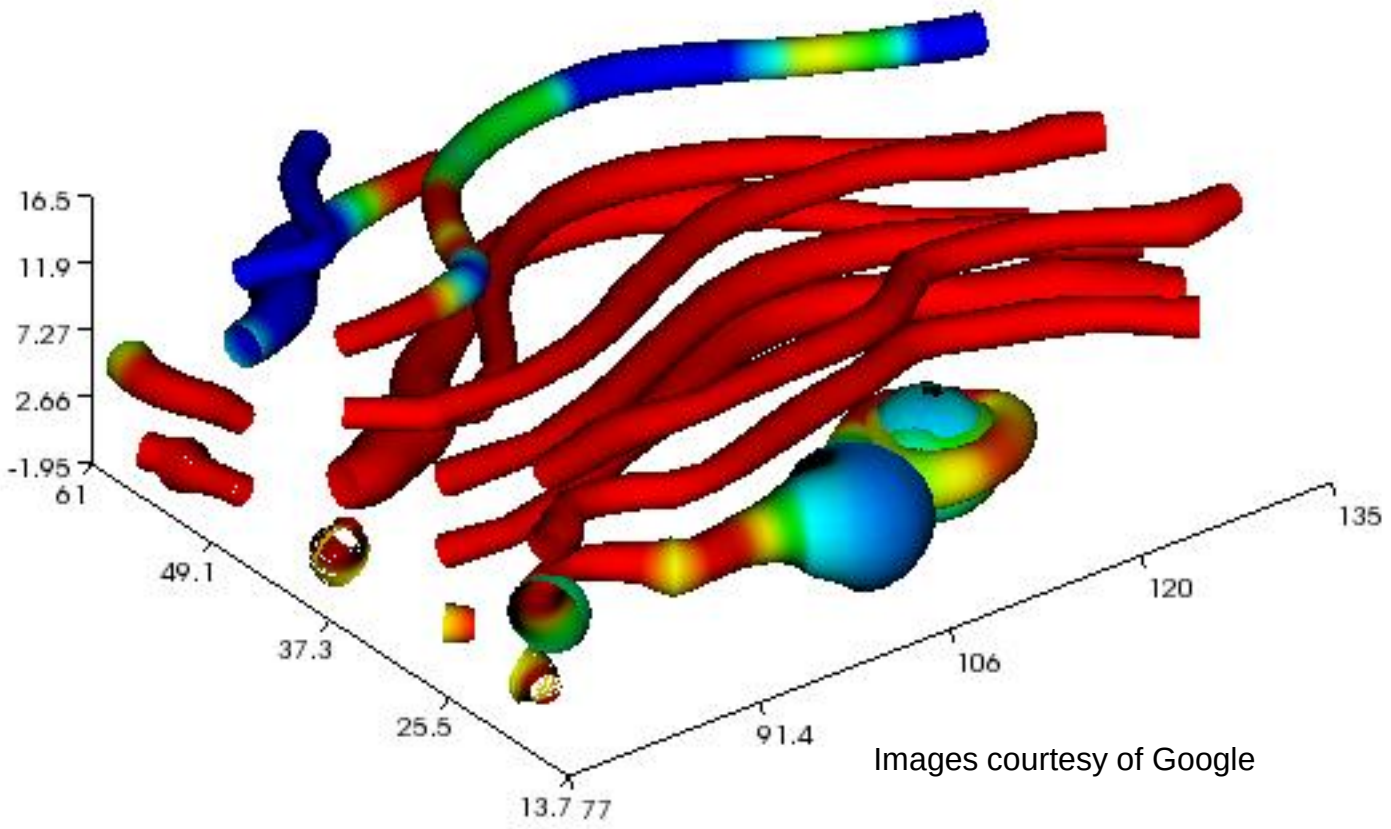
If we now start streamlines around a circular area A perpendicular to the flow we can create a tubular structure representing the vector field:



These streamlines form a tube that is impermeable since the walls of the tube are made up of streamlines, and there can be no flow normal to a streamline (by definition). This tube is called a **streamtube**.

Vector Algorithms (continued)

Example



Images courtesy of Google

Vector Algorithms (continued)

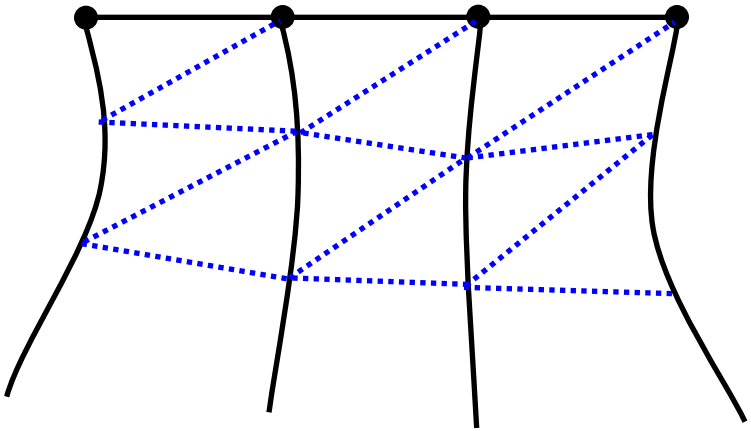
Streamsurfaces

A *streamsurface* is a collection of an infinite number of streamlines passing through a *base curve*. The base curve, or rake, defines the starting points for the streamlines. If the base curve is closed (e.g. a circle) the surface is closed and a *streamtube* results. Thus, streamribbons are specialized types of streamsurfaces with a narrow width compared to length.

Vector Algorithms (continued)

Computing streamsurfaces

In order to compute a streamsurface based on an interpolation, the base curve is split equidistantly. At each split point a streamline is started. The immediate points of the integration process are then connected with triangles.



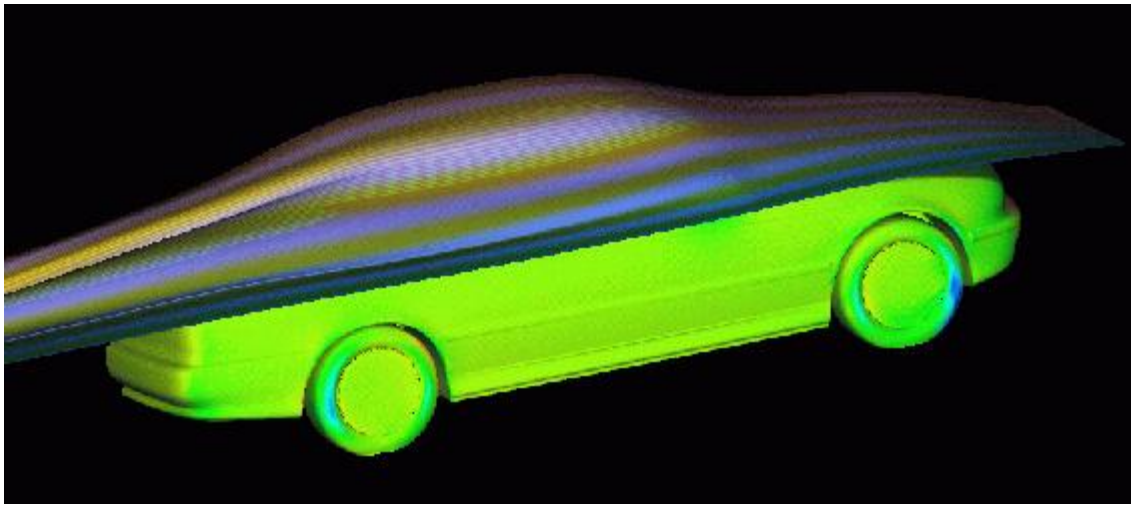
Vector Algorithms (continued)

If two neighboring streamline become closer than a pre-defined threshold, the two streamlines are merged and only one is continued. In the opposite case, where two streamlines diverge so that they are farther away from each other than a pre-defined threshold, an additional streamline is started in the center between those two streamlines to maintain a certain precision of the streamsurface.

Often times, streamsurfaces are colored in stripes in order to visualize the divergence of the vector field.

Vector Algorithms (continued)

Example



Vector Algorithms (continued)

Streampolygons

Streampolygons try to visualize additional local properties of a vector field, such as strain, displacement, and rotation. The basic idea is to integrate a streamline and use a deformed polygon instead of a single point at each point resulting form the integration step. The deformation of the polygon is based on strain and rotation as defined by the vector field.

Vector Algorithms (continued)

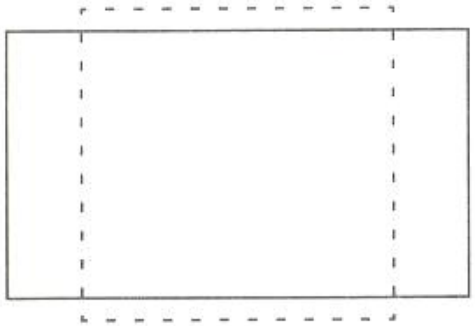
The local strain is expressed as a combination of the partial derivatives:

$$\begin{pmatrix} \frac{\partial u}{\partial x} & \frac{1}{2}\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right) & \frac{1}{2}\left(\frac{\partial u}{\partial z}+\frac{\partial w}{\partial x}\right) \\ \frac{1}{2}\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right) & \frac{\partial v}{\partial y} & \frac{1}{2}\left(\frac{\partial v}{\partial z}+\frac{\partial w}{\partial y}\right) \\ \frac{1}{2}\left(\frac{\partial u}{\partial z}+\frac{\partial w}{\partial x}\right) & \frac{1}{2}\left(\frac{\partial v}{\partial z}+\frac{\partial w}{\partial y}\right) & \frac{\partial w}{\partial z} \end{pmatrix}$$

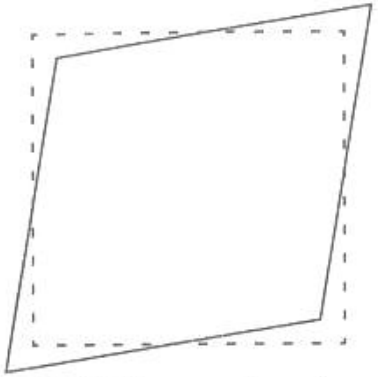
The local rigid body rotation is given by:

$$\begin{pmatrix} \frac{\partial u}{\partial x} & \frac{1}{2}\left(\frac{\partial u}{\partial y}-\frac{\partial v}{\partial x}\right) & \frac{1}{2}\left(\frac{\partial u}{\partial z}-\frac{\partial w}{\partial x}\right) \\ \frac{1}{2}\left(\frac{\partial v}{\partial x}-\frac{\partial u}{\partial y}\right) & \frac{\partial v}{\partial y} & \frac{1}{2}\left(\frac{\partial v}{\partial z}-\frac{\partial w}{\partial y}\right) \\ \frac{1}{2}\left(\frac{\partial w}{\partial x}-\frac{\partial u}{\partial z}\right) & \frac{1}{2}\left(\frac{\partial w}{\partial y}-\frac{\partial v}{\partial z}\right) & \frac{\partial w}{\partial z} \end{pmatrix}$$

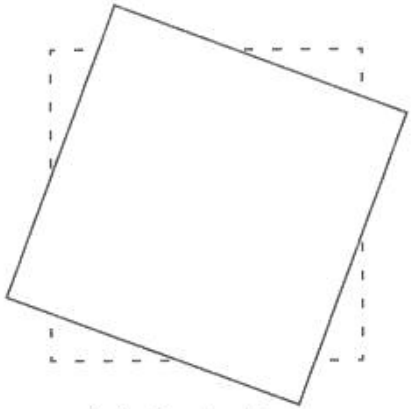
Vector Algorithms (continued)



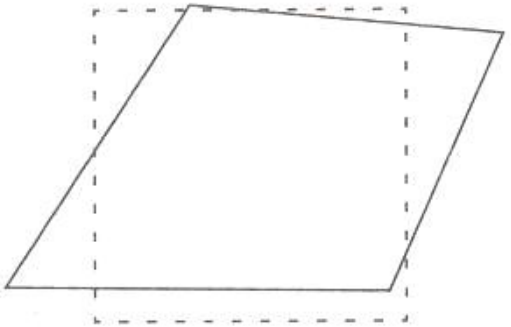
(a) Normal strain



(b) Shear strain



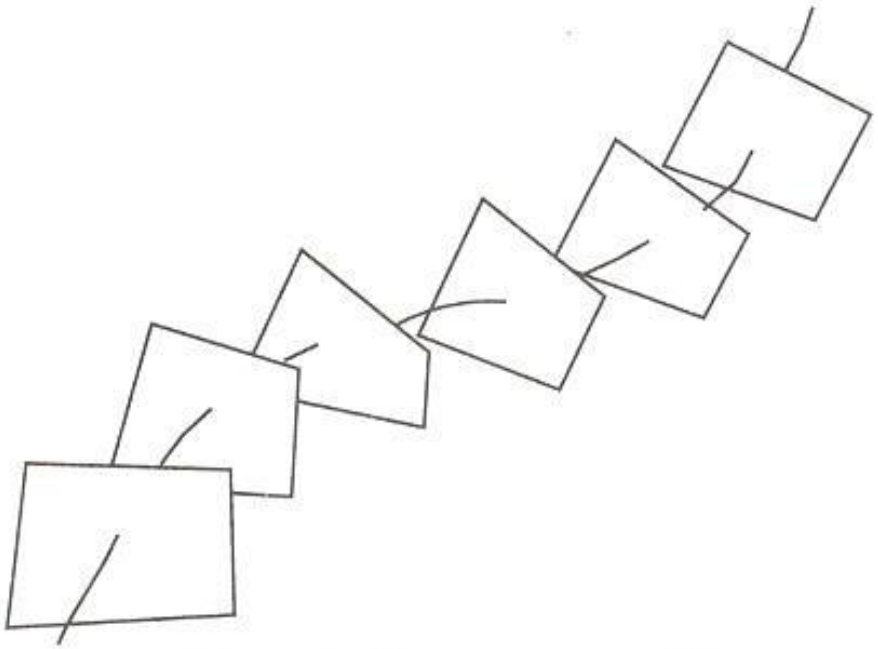
(c) Rotation



(d) Total deformation

Vector Algorithms (continued)

Placing the deformed polygons along the streamline:



Vector Algorithms (continued)

Vector field topology

Vector fields have a complex structure characterized by special features called *critical points*. Critical points are locations in the vector field where the local vector magnitude goes to zero and the vector direction becomes undefined. At these points the vector field either converges or diverges, and/or local circulation around the point occurs. In order to understand critical points better, we take a look at linearly defined vector fields. Since we usually interpolate vector fields linearly, this will result in the most common cases of critical points.

Vector Algorithms (continued)

Critical points

Let *v* be a given vector field $v: W \rightarrow IR^3$ with $W \subset IR^3$ as defined on a face of a tetrahedron. Let further $x_0 \in W$ be a point where the vector field vanishes, i.e. $v(x_0) = 0$. Then x_0 is considered a critical point of the vector field v. Several terms are used synonymously for critical points. These are singularities, singular points, zeros, or equilibrium.

Vector Algorithms (continued)

Linear vector fields

A linear 3-D vector field v can be described by a matrix and a displacement vector. Therefore, a linear map $A: W \rightarrow IR^3$ described by the 3×3 matrix A and a vector $b \in IR^3$ can be found such that it describes the given vector field v, i.e. $v(x) = Ax + b$ for all $x \in W$.

Then, singularities can be found by directly solving the equation $Ax + b = 0$. Obviously, there cannot be more than one singularity located within one triangle when using linear interpolation. For the case $b = 0$ we consider the vector field described by Ax homogenous linear. Without loss of generality we assume homogenous linear vector fields in the further discussion of the theory of vector field topology.

Vector Algorithms (continued)

Classification of critical points

Singularities can be classified using the eigenvalues of the interpolating matrix A regarding their property of attracting or repelling the surrounding flow. If all eigenvalues have negative real parts the singularity is considered a *sink* which attracts the surrounding flow. On the other hand, if all eigenvalues have positive real parts the singularity is a *source* that repels the surrounding flow.

Vector Algorithms (continued)

Computing streamlines

Further analyzing the matrix A leads to a several types of vector fields distinguished by their major properties of the flow, i.e. the behavior of the streamlines within this vector field. In order to compute a streamline, the Cauchy problem has to be solved with initial problem $x(0) = k, k \in IR^3$:

$$\frac{d}{dt}x(t) = Ax(t)$$

Vector Algorithms (continued)

Solution to the Cauchy problem

It can be proven that the solution to the Cauchy problem for a linear vector field can be described by an exponential function:

$$x = e^{tA}k \quad \text{with} \quad e^{A} = \sum_{i=0}^{\infty} \frac{A^i}{i!}$$

Different categories of vector fields can then be distinguished weather the matrix A is diagonalizable, resulting in a different behavior of the streamlines in each case. This leads to three main categories described in the following.

Vector Algorithms (continued)

Type 1 vector fields

The matrix A is diagonalizable, i.e. the eigenvalues λ and μ are real. Thus it is similar to a matrix B, i.e. there exist an invertible matrix P with $B = PAP^{-1}$, of the following structure:

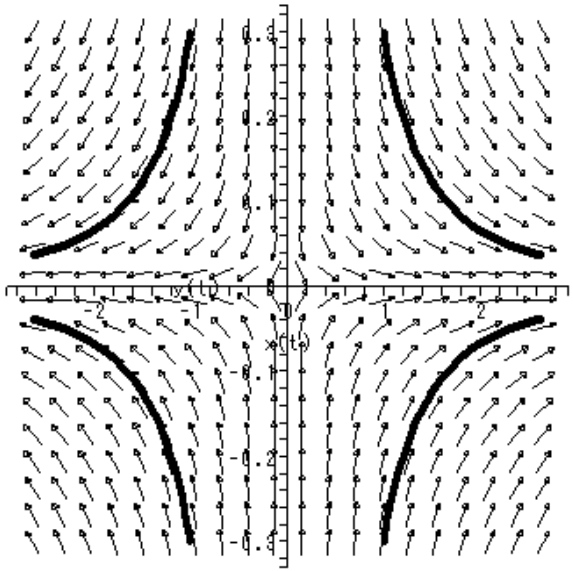
$$B = \begin{pmatrix} \lambda & 0 \\ 0 & \mu \end{pmatrix}$$

Due to the structure of the matrix B, a streamline $x(t)$ with initial condition $k = (k_1, k_2)$ can be computed in a vector field described by such a matrix using the following formula:

$$x(t) = \begin{pmatrix} e^{t\lambda} k_1 \\ e^{t\mu} k_2 \end{pmatrix}$$

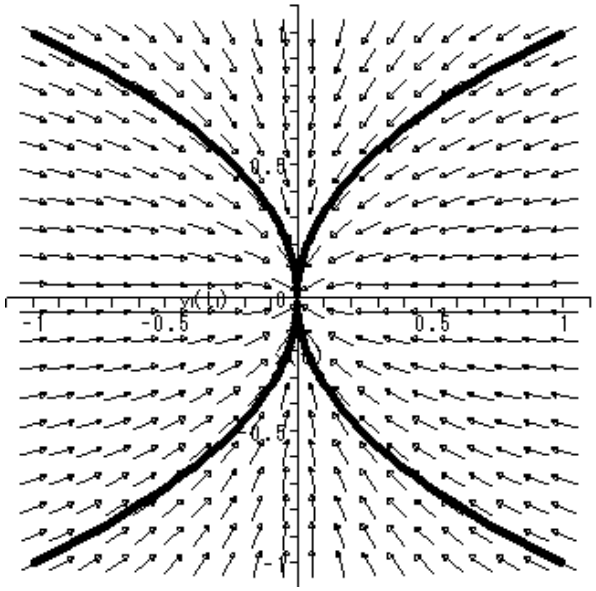
Vector Algorithms (continued)

By computing streamlines we can generate a phase portrait of the different cases of vector fields within this category. Three different types are possible, again distinguished by the eigenvalues of the interpolating matrix A. The first case, where $\lambda > 0 > \mu$, results in a *saddle singularity*:



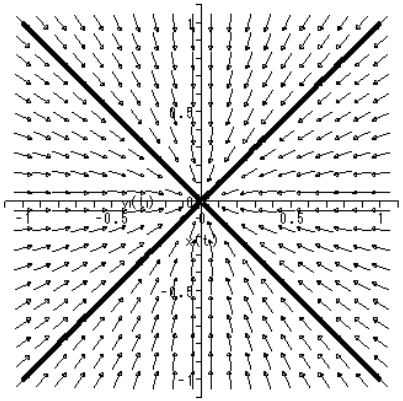
Vector Algorithms (continued)

The second case, described by an eigenvalue configuration of $\lambda < \mu < 0$, describes a *node singularity*:



Vector Algorithms (continued)

The last case with two identical eigenvalues is the *focus singularity*, for example $\lambda = \mu < 0$.



The examples shown here mainly show sinks; however, the same types of singularities occur with sources. The only difference is in the sign of the eigenvalues, i.e. multiplying the eigenvalues by -*1* results in the same singularities as sources.

Vector Algorithms (continued)

Type 2 vector fields

The two eigenvalues of the matrix A have a non-imaginary part, i.e. A is similar to the following matrix:

$$B = \begin{pmatrix} a & -b \\ b & a \end{pmatrix}$$

When substituting the values a and b in the above matrix by introducing new values θ and r:

$$r = \sqrt{a^2 + b^2}$$

$$\theta = \arccos(a/r)$$

the matrix B can be rewritten as follows:

$$B = \begin{pmatrix} a & -b \\ b & a \end{pmatrix} = \begin{pmatrix} r & 0 \\ 0 & r \end{pmatrix} \cdot \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

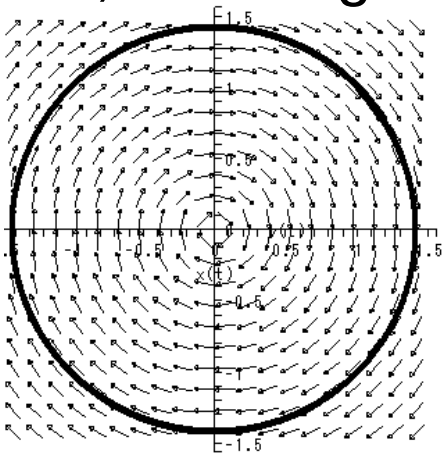
Vector Algorithms (continued)

Obviously, a vector field described by such a matrix has a strong rotational component. Consequently, a streamline $x(t)$ with initial condition $k = (k_1, k_2)$ can be computed using the following formula:

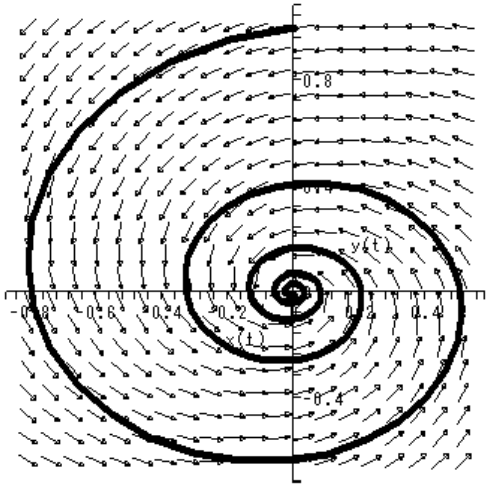
$$x(t) = e^{ta} \cdot \begin{pmatrix} k_1 \cos(tb) - k_2 \sin(tb) \\ k_1 \sin(tb) + k_2 \cos(tb) \end{pmatrix}$$

Vector Algorithms (continued)

For $a=0$, the streamlines describe perfect, concentric circles, resulting in the *center singularity*:



Otherwise, a *spiral singularity* is described with streamlines spiralling around the singularity and then eventually ending up at the singularity itself.



Vector Algorithms (continued)

Type 3 vector fields

The matrix A is not diagonalizable and the two eigenvalues are equal, i.e. $\lambda = \mu$. In this case, A is similar to the following matrix:

$$B = \begin{pmatrix} \lambda & 0 \\ 1 & \lambda \end{pmatrix}$$

By splitting up the matrix B into two components

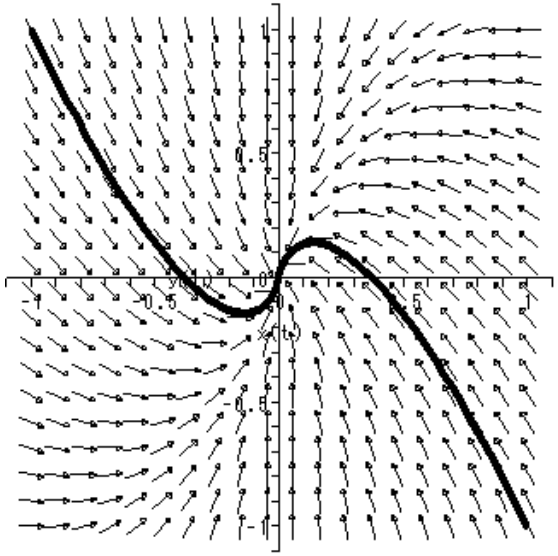
$$B = \begin{pmatrix} \lambda & 0 \\ 1 & \lambda \end{pmatrix} = \lambda \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}$$

it can be easily seen that a streamline with initial condition $k = (k_1, k_2)$ integrated through such a vector field can be described by:

$$x(t) = e^{t\lambda} \cdot \begin{pmatrix} k_1 \\ k_1 t + k_2 \end{pmatrix}$$

Vector Algorithms (continued)

This case resembles an *improper node singularity*:



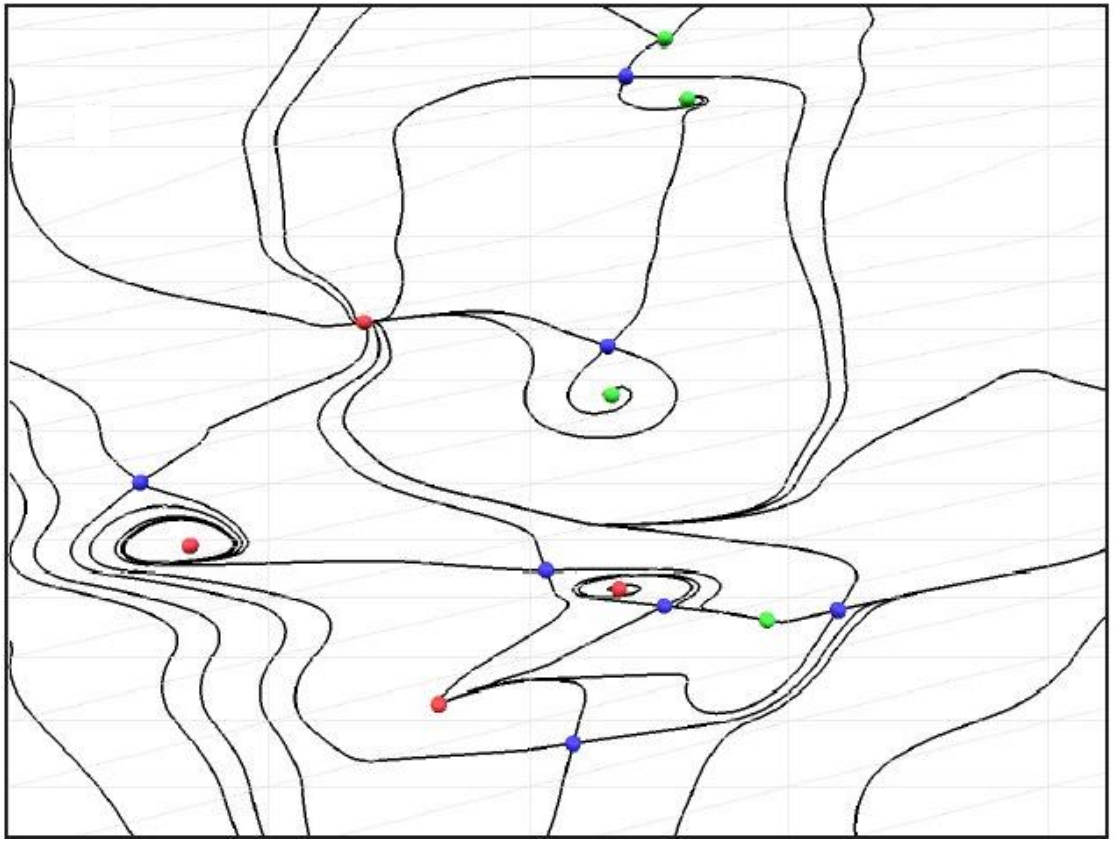
Vector Algorithms (continued)

Topological analysis

The topological graph, or simply topology, of a vector field describes the structure of the flow or phase portrait. Separatrices are used to separate the areas of the flow into regions with similar behavior. Separatrices can be easily computed by integrating streamlines emerging from saddle singularities in direction of the eigenvectors of the interpolating matrix. The topological graph then consists of the singularities and the separatrices.

Vector Algorithms (continued)

Example



Vector Algorithms (continued)

Topological analysis with different interpolation

The topological graph be different when using different types of interpolations. By changing the cell type alone, for example triangulating the cells by splitting up rectangles into two triangles, the topological graph can change. Hence, the interpolation technique used for integrating the streamlines should be chosen with special care!

Vector Algorithms (continued)

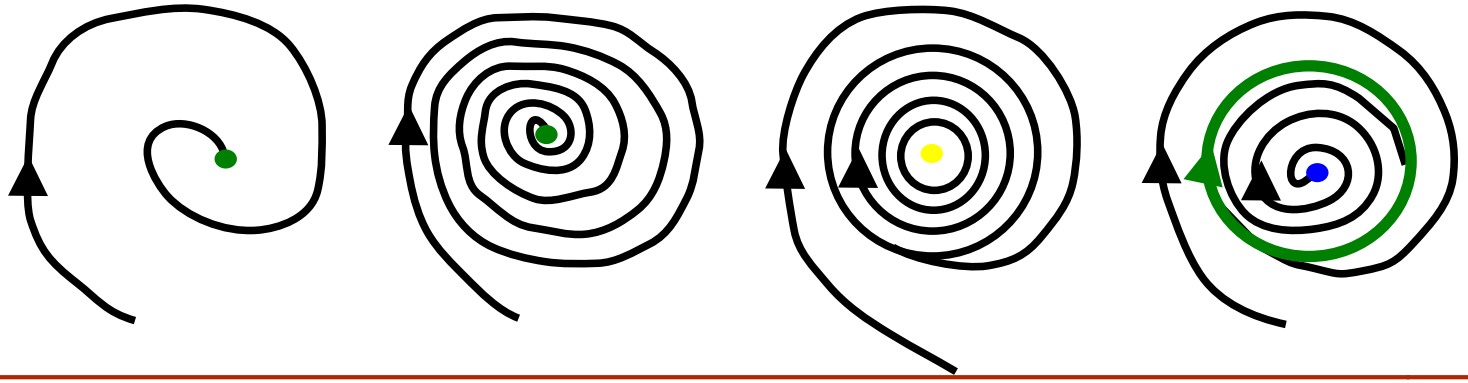
Example

Vector Algorithms (continued)

Closed streamlines

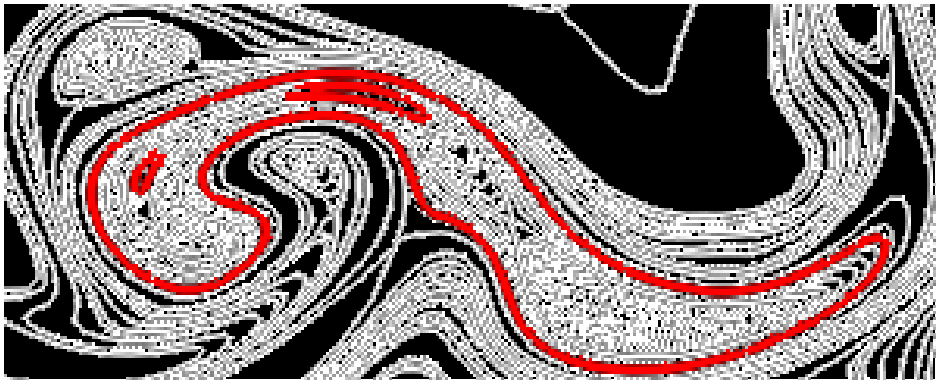
There are more topological features other than separatrices and critical points. The flow can – in the same way as with critical points – be attracted or repelled by a *closed streamline*.

For example, the Hopf bifurcation describes critical point that changes from source to sink resulting in an attracting closed streamline:



Vector Algorithms (continued)

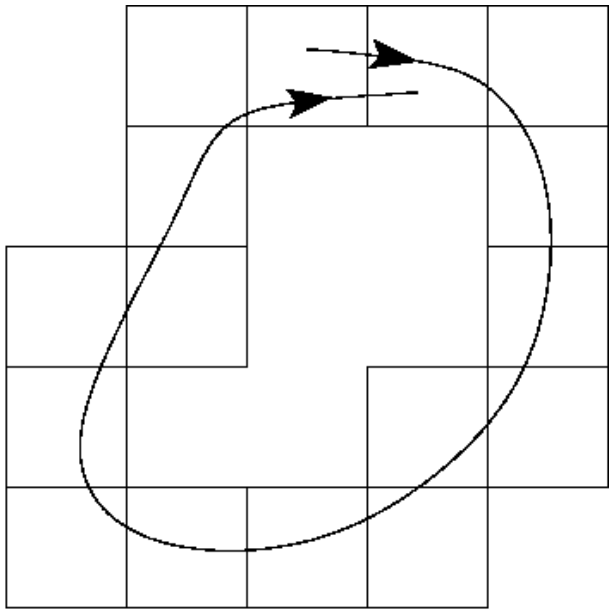
By considering closed streamlines are able to connect parts of the topological skeleton and to complete the topological analysis.



Since closed streamlines are a global feature we cannot identify them by using local properties of the vector field.

Vector Algorithms (continued)

Obviously a closed streamline runs through the same cells of our grid over and over again. The idea for finding a closed streamline is basically to prove that the closed streamline does not leave *cell cycle*.

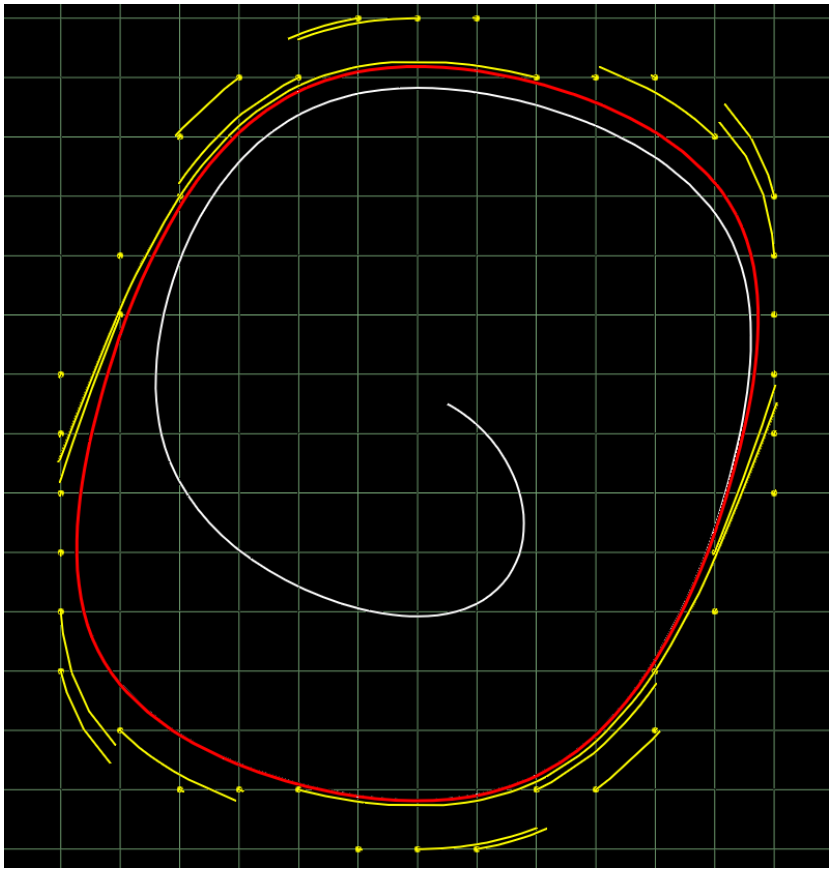


Vector Algorithms (continued)

In order to check if the streamline can leave cell cycle we basically need to start *backward integration* at every point at edge of cell cycle. If the backward integration exists that run toward the streamline then the streamline leaves cell cycle (near the point where the backward integration was started). Otherwise, if such backward integration does not exist then the streamline stays inside the cell cycle forever. The problem, however, is that the number of points at the edge of the cell cycle infinite.

Vector Algorithms (continued)

Therefore, we define potential exits. Potential exits are considered those points that are either vertices of cell cycle or points at edge of cell cycle where the vector field is tangential to that edge. Then, backward integration is only necessary at these potential exits since the integration of two neighboring potential exits cover the entire edge connecting them.



Vector Algorithms (continued)

Example

In a vector field describing a hurricane, the eye of the hurricane is surrounded by a closed streamline if the vector field is projected on to a 2-D plane. Thus, the eye can be identified by finding the closed streamline in the projected vector field.

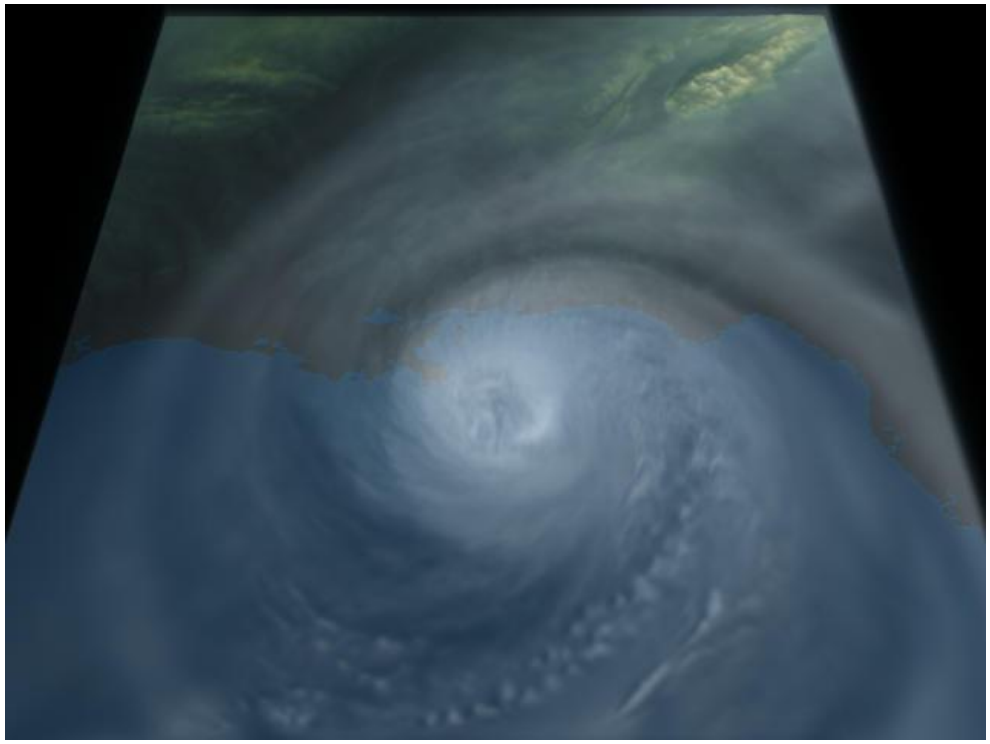
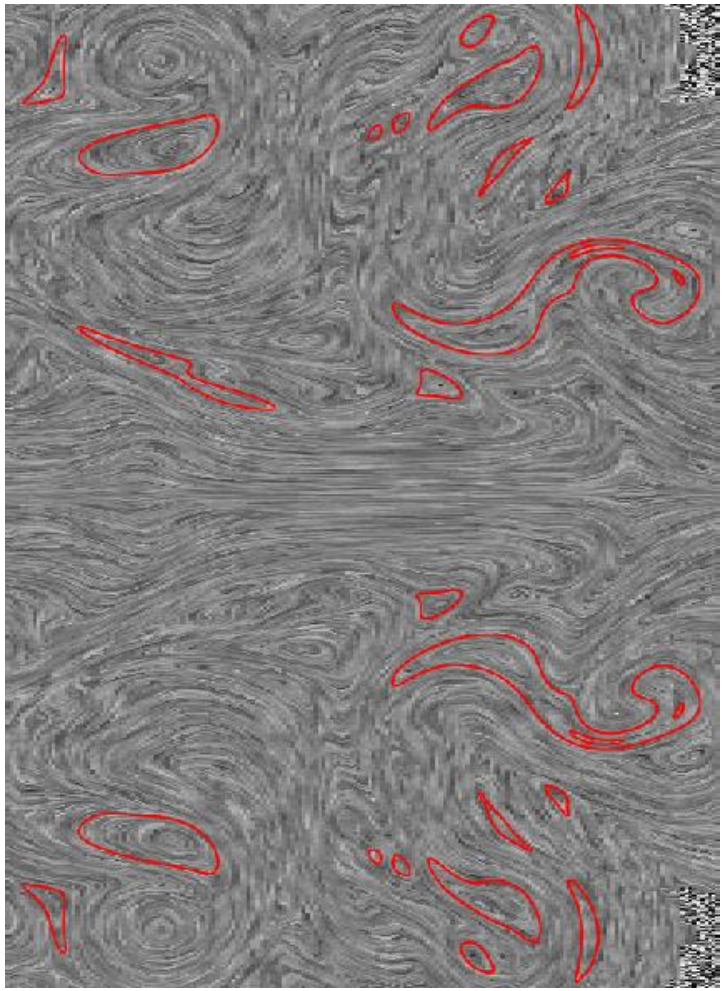


Image courtesy of David Bock, NCSA

Vector Algorithms (continued)

Example

In combustion processes it is important that the gas stays in an area for a certain amount of time for the gas to burn completely. Closed streamlines are a hint for recirculation zones, i.e. areas where the gas stays for longer period of time. Hence, closed streamlines indicate areas with better combustion.



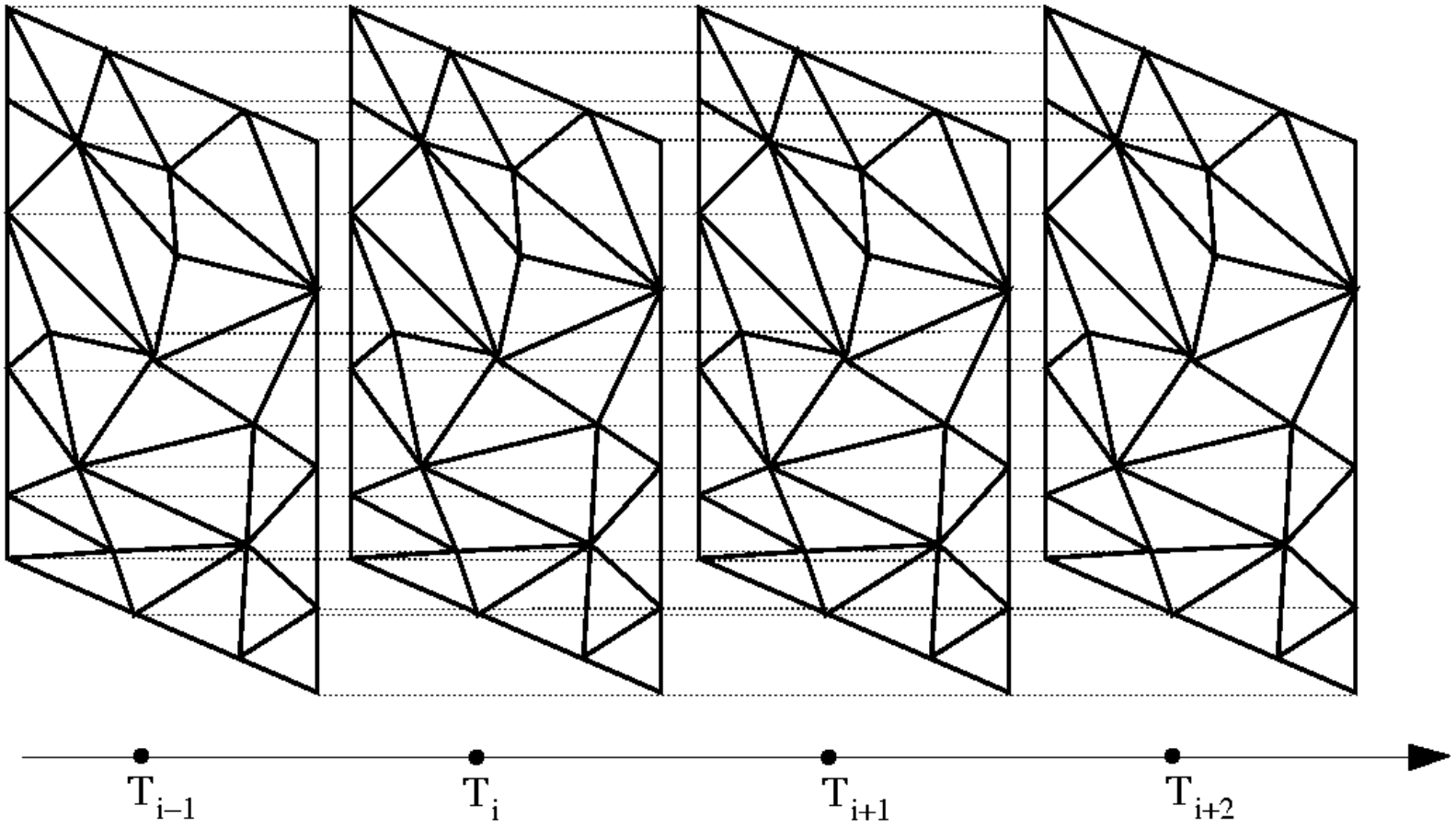
Vector Algorithms (continued)

Tracking closed streamlines over time

In order to understand how and when closed streamlines occur, we can take a look at a vector field that changes over time. At various instances in time, a 2-D vector field is given. Through linear interpolation between consecutive time steps we can compute vectors at every instance in time and any location within the 2-D space defined by the domain of the vector field.

Vector Algorithms (continued)

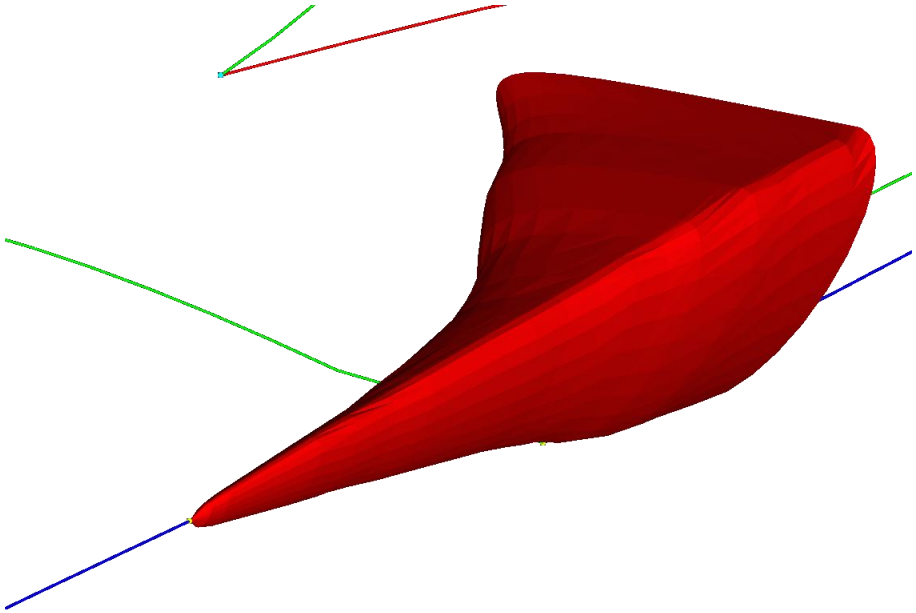
2-D grid structure including time



Vector Algorithms (continued)

Closed streamlines over time

We can now determine the location of the closed streamlines within each of the time steps and then connect the resulting curves with triangles:



Vector Algorithms (continued)

Line integral convolution

The basic idea of *line integral convolution (LIC)* is to deform a texture according to the given vector field in order to get a visualization.

This approach heavily depends on the choice of texture as we will see later. Hence, a bad choice of texture may occlude important features of the vector field.

Before go into the details of line integral convolution, we should begin with a simpler case, the DDA convolution.

Vector Algorithms (continued)

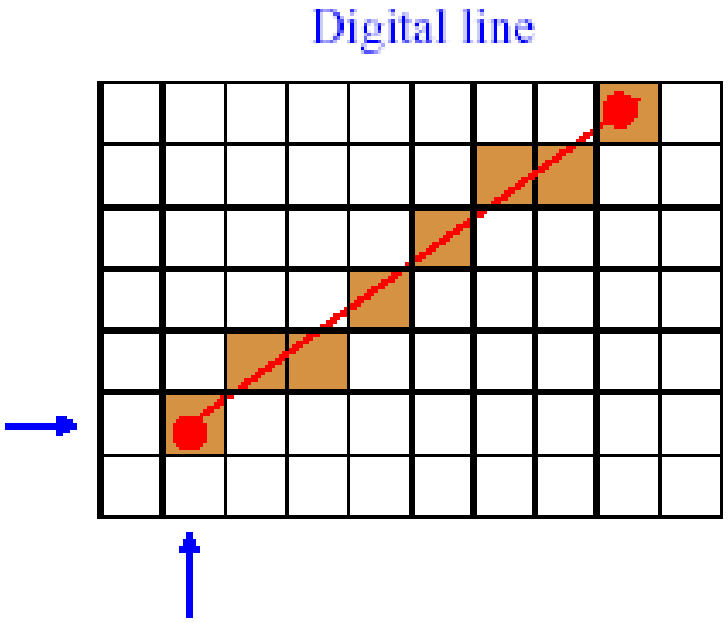
DDA convolution

This approach is a generalization of traditional line drawing techniques and the spatial convolution algorithms given by Cabral and Leedom (line integral convolution) or Van Wijk and Perlin (spot noise).

Vector Algorithms (continued)

DDA = Digital Differential Analyzer

Idea: Determine the pixel's y coordinate by evaluating the change in the y position of the ideal line when x increases by 1



$$y = \underbrace{\frac{y2-y1}{x2-x1}}_{=m} x + b$$

- Suppose the k-th pixel alor the line has been drawn an is (x_k, y_k)
- What is the y value of the next pixel to be drawn?

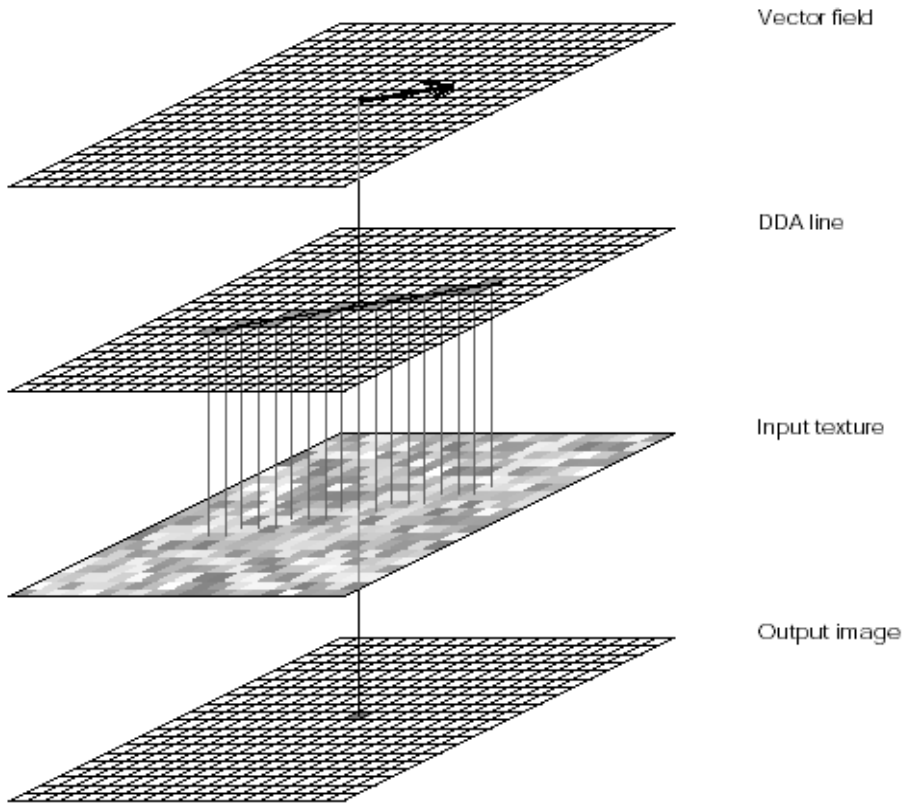
- $y_{k+1} = y_k + m$

Vector Algorithms (continued)

The convolution algorithm then consists of three steps:

- Each vector in the field is used to define a long, narrow, DDA generated filter kernel that is tangential to the vector and going in the positive and negative vector direction some fixed distance, the *kernel length* L.
- A texture is then mapped one-to-one onto the vector field.
- The input texture pixels under the filter kernel are summed, normalized by the length of the filter kernel, $2L$, and placed in an output pixel image for the vector position.

Vector Algorithms (continued)



The mapping of a vector onto a DDA line and input pixel field generating a single output pixel.

Vector Algorithms (continued)

Simple circular vector field with white noise (texture)

Computational fluid dynamics code with white noise

Vector Algorithms (continued)

Symmetry

This algorithm is very sensitive to symmetry of the DDA algorithm and filter. If the algorithm weights the forward direction more than the backward direction, the circular field in the previous figure (left) would appear to spiral inward implying a vortical behavior that is not present in the field (i.e. representing the center singularity as a spiral singularity).

Vector Algorithms (continued)

Disadvantages

This algorithm assumes that the local vector field can be approximated by a straight line.

For complex structures smaller than the length of the DDA line, the local radius of curvature is small and is not well approximated by a straight line.

In a sense, DDA convolution renders the vector field unevenly, treating linear portions of the vector field more accurately than small scale vortices (high curvature areas).

Vector Algorithms (continued)

Line Integral Convolution (LIC)

The LIC algorithm is a derivative of the DDA technique that, instead of using a vector, uses a local streamline to generate the filter. The local behavior of the vector field can be approximated by computing a local stream line that starts at the center of pixel (x, y) and moves out in the positive and negative directions.

Vector Algorithms (continued)

The forward coordinate advection

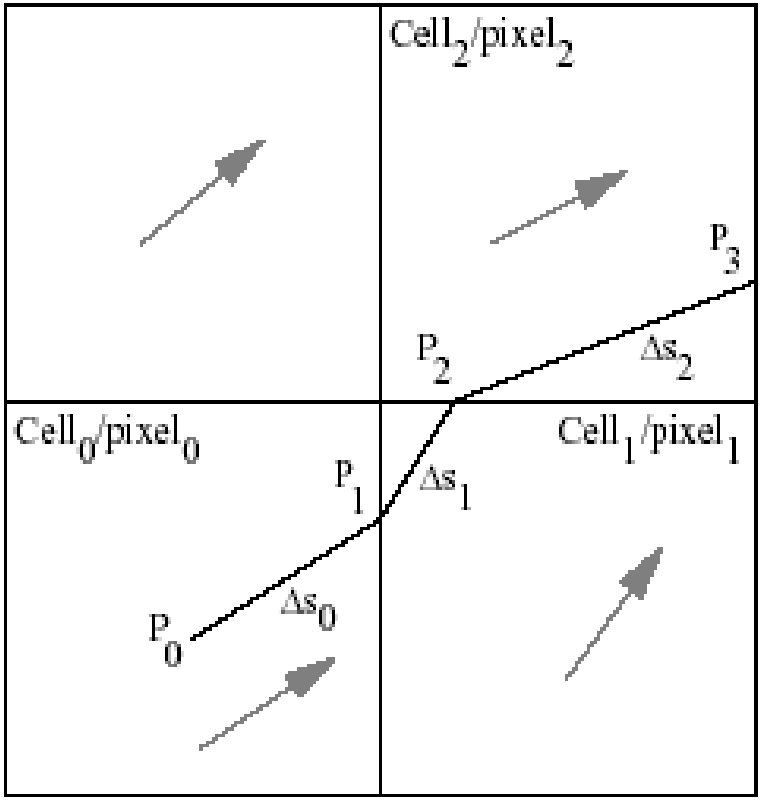
$$P_0 = (x + 0.5, y + 0.5)$$

$$P_l = P_{l-1} + \frac{V(\lfloor P_{l-1} \rfloor)}{\| V(\lfloor P_{l-1} \rfloor) \|} \Delta s_{l-1} \qquad (1)$$

$V(\lfloor P \rfloor)$ = the vector from the input vector field at lattice point $(\lfloor P_x \rfloor, \lfloor P_y \rfloor)$

$$s_e = \begin{cases} \infty \text{ if } V \| e \\ 0 \text{ if } \dfrac{\lfloor P_c \rfloor - P_c}{V_c} < 0 \\ \dfrac{\lfloor P_c \rfloor - P_c}{V_c} \text{ otherwise} \end{cases} \quad \text{for } (e, c) \in \begin{Bmatrix} (top, y) \\ (bottom, y) \\ (left, x) \\ (right, x) \end{Bmatrix} \qquad (2)$$

$$\Delta s_l = \min (s_{top}, s_{bottom}, s_{left}, s_{right})$$



Vector Algorithms (continued)

As with the DDA algorithm, it is important to maintain symmetry about a cell. Hence, the local stream line is also advected backwards by the negative of the vector field as shown in this equation:

$$P'_0 = P_0$$

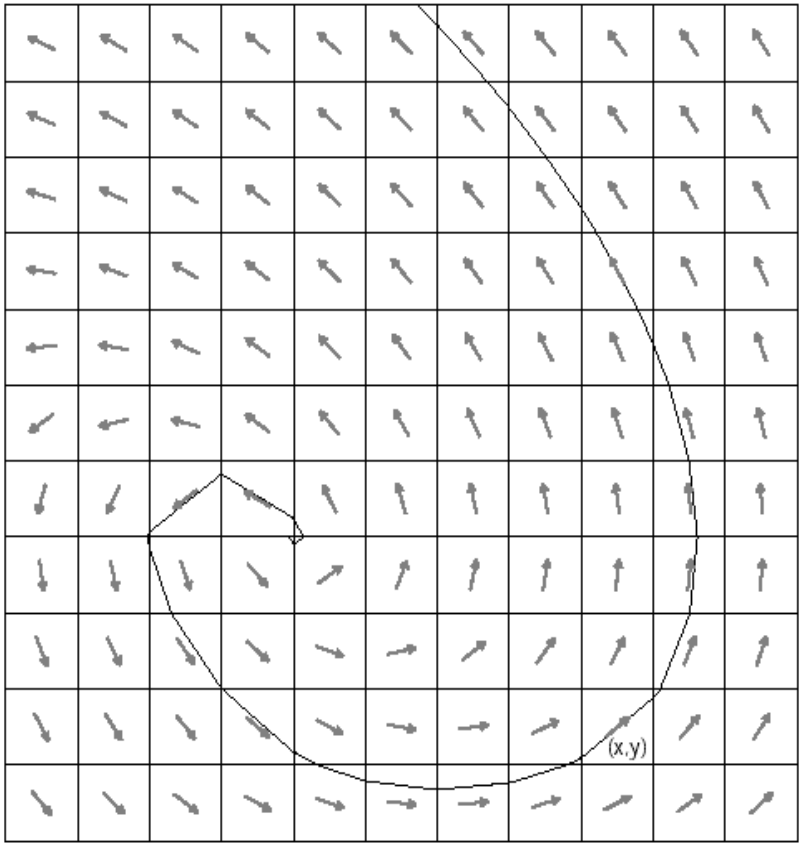
$$P'_i = P'_{i-1} - \frac{V(\lfloor P'_{i-1} \rfloor)}{\|V(\lfloor P'_{i-1} \rfloor)\|} \Delta s'_{i-1}$$

Primed variables represent the negative direction counterparts to the positive direction variables and are not repeated in subsequent definitions. As above $\Delta s'_i$, is always positive.

Vector Algorithms (continued)

Illustration of local stream line calculation

Continuous sections of the local stream line – i.e. the straight line segments in the figure on the right – can be thought of as parameterized space curves in s and the input texture pixel mapped to a cell can be treated as a continuous scalar function of x and y. It is then possible to integrate over this scalar field along each parameterized space curve.



Vector Algorithms (continued)

Line Integrals of the First Kind (LIFK)

Such integrals can be summed in a piecewise C^1 fashion and are known as *line integrals of the first kind* (LIFK):

$$h_i = \int_{s_t}^{s_t + \Delta s_t} k(w)dw$$

where

$$s_0 = 0$$

$$s_i = s_{i-1} + \Delta s_{i-1}$$

Vector Algorithms (continued)

Result of Applying LIFK

This results in a variation of the DDA approach that locally follows the vector field and captures small radius of curvature features.

For each continuous segment, i, an exact integral of a convolution kernel $k(w)$ is computed and used as a weight in the LIC as shown in the equation on next slide.

Vector Algorithms (continued)

LIC for an output pixel *F'(x, y)*

The entire LIC for output pixel $F'(x, y)$ is given by the following equation

$$F'(x,y) = \frac{\sum_{i=0}^{l} F(\lfloor P_i \rfloor) h_i + \sum_{i=0}^{r} F(\lfloor P_i' \rfloor) h_i'}{\sum_{i=0}^{l} h_i + \sum_{i=0}^{r} h_i'}$$

where

$F(\lfloor P_i \rfloor)$ is the input pixel corresponding to the vector at position $(\lfloor P_x \rfloor, \lfloor P_y \rfloor)$

$l = i$ such that $s_i \le L \le s_{i+1}$

Vector Algorithms (continued)

Local Streamline L

The length of the local stream line, $2L$, is given in unit pixels. Depending on the input pixel field, F, if L is too large, all the resulting LICs will return values very close together for all coordinates (x, y).

On the other hand, if L is too small then an insufficient amount of filtering occurs. Since the value of L dramatically affects the performance of the algorithm, the smallest effective value is desired.

Vector Algorithms (continued)

The effect of varying L

Photograph of flowers processed using LIC with L equal to 0, 5, 10 and 20 (left to right, top to bottom).

Vector Algorithms (continued)

Truncation of the streamline / termination of algorithm

Singularities in the vector field occur when vectors in two adjacent local stream line cells geometrically “point” at a shared cell edge. This results in Δs_i values equal to zero leaving fraction in the previous equation undefined. This situation can easily be detected and the advection algorithm terminated.

If the vector field goes to zero at any point, the LIC algorithm is terminated as in the case of a field singularity. Both of these cases generate truncated stream lines.

Vector Algorithms (continued)

Periodic motion filters

The LIC algorithm visualizes local vector field tangents, but not their direction. The local vector field direction can be rendered via animation of successive LIC imaged vector fields using varying phase shifted periodic filter kernels.

Vector Algorithms (continued)

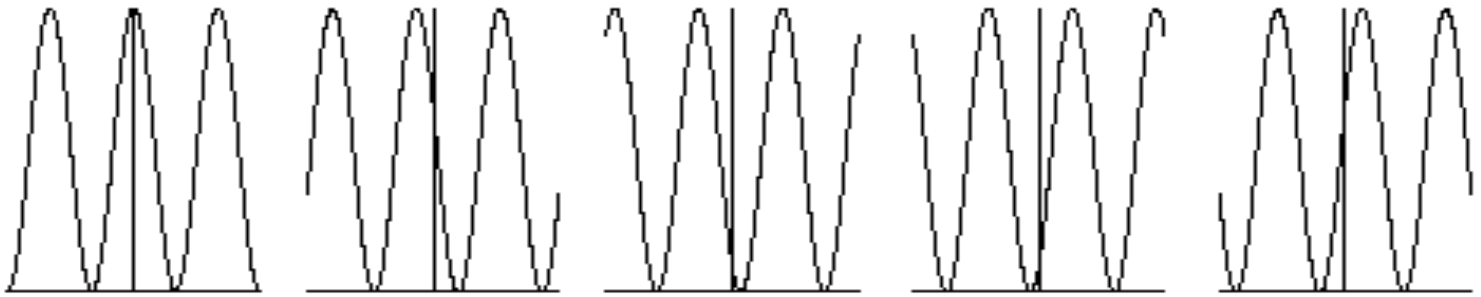
Required Properties of the filter

- The success of this technique depends on the shape of the filter.
- If the filter is periodic, by changing the phase of such filters as a function of time, apparent motion in the direction of the vector field is created.
- It is possible, and desirable, to create periodic low-pass filters to blur the underlying texture in the direction of the vector field.

Vector Algorithms (continued)

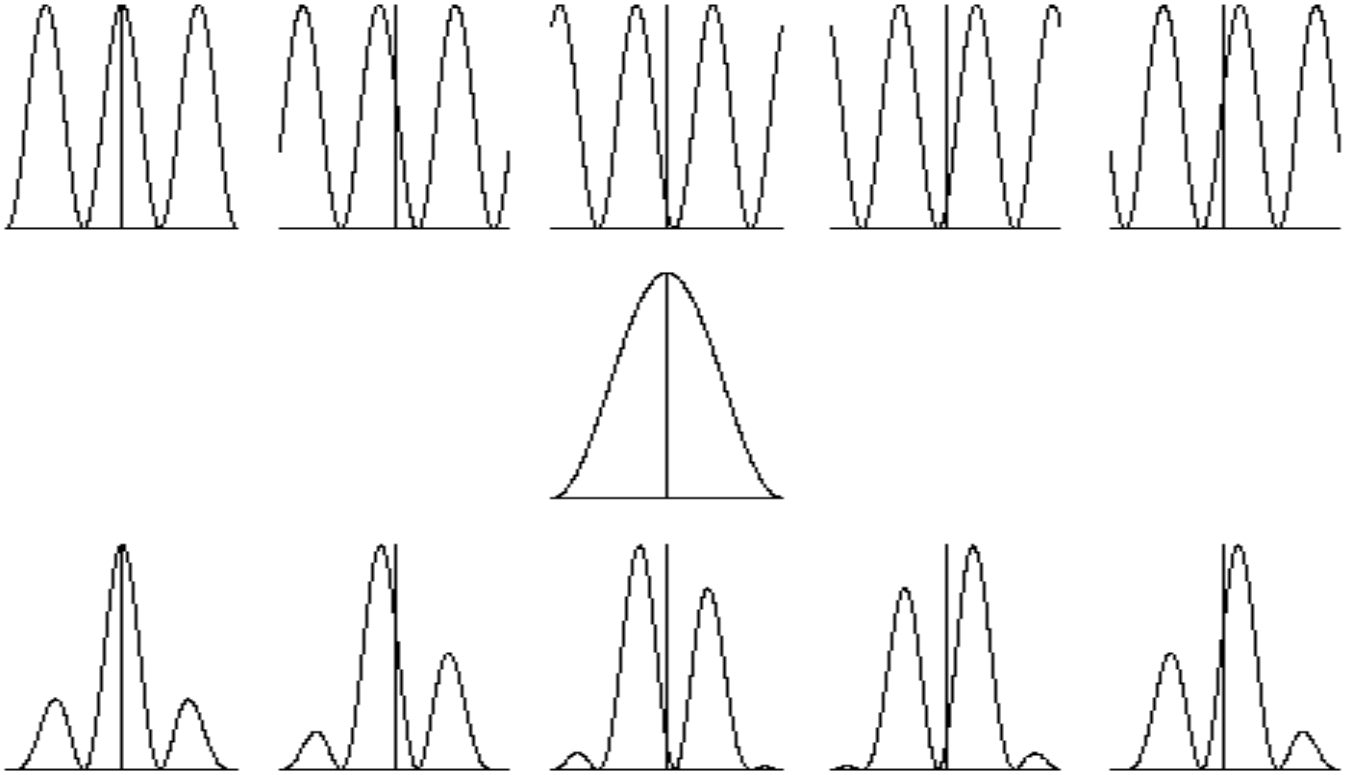
Hanning Filter

A *Hanning filter*, $\frac{1}{2}(1 + \cos(w+b))$, has this property. It has low band-pass filter characteristics, it is periodic by definition and has a simple analytic form. This function will be referred to as the *ripple* filter function.



Vector Algorithms (continued)

Ripple filter function



Phase shifted Hanning ripple functions(top), a Hanning windowing function(middle), and Hanning ripple functions multiplied by the Hanning window function(bottom).

Vector Algorithms (continued)

General form of the filter

The general form of this function (Hanning ripple function • Hanning window function) is shown in the following equation:

$$k(w) = \frac{1+\cos(cw)}{2} \cdot \frac{1+\cos(dw+\beta)}{2}$$

$$= \frac{1}{4}(1+\cos(cw)+\cos(dw+\beta)+\cos(cw)\cos(dw+\beta))$$

c : dilation constant of the Hanning window function

d : dilation constant of the Hanning ripple function

β : phase shift of the ripple function given in radian

Vector Algorithms (continued)

Periodicity of the Ripple function

Choosing the periodicity of the ripple function represents making a design trade-off between maintaining a nearly constant frequency response as a function of phase shift and the quality of the apparent motion.

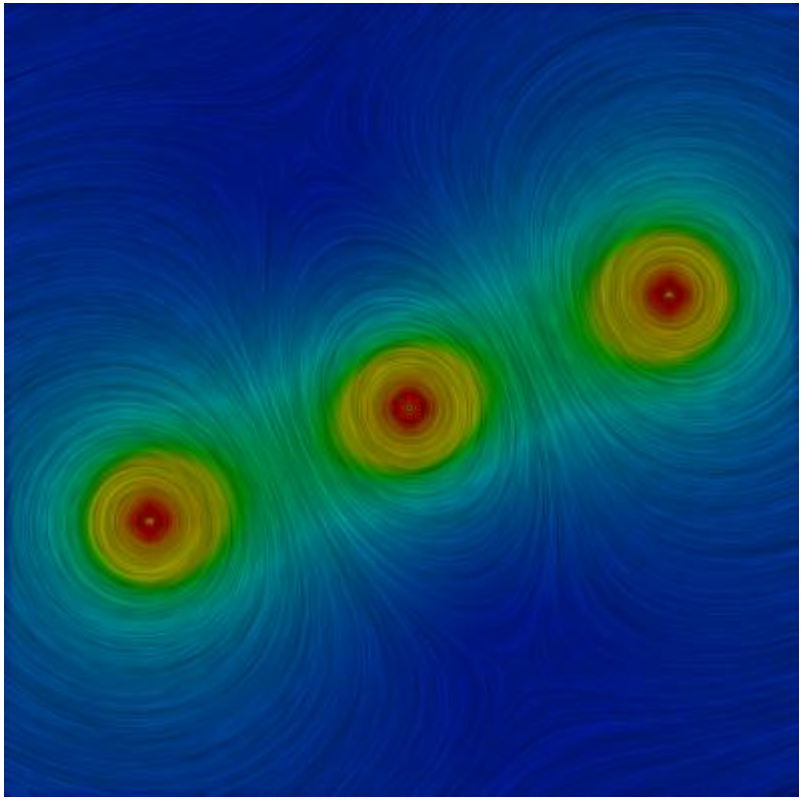
Vector Algorithms (continued)

Frequency of the filter

- A *low frequency* ripple function results in a windowed filter whose frequency response noticeably changes as a function of phase. This appears as a periodic blurring and sharpening of the image as the phase changes.
- *Higher frequency* ripple functions produce windowed filters with a nearly constant frequency response. However, the feature size picked up by the ripple filter is smaller and the result is less apparent motion.
- If the ripple frequency exceeds the Nyquist limit of the pixel spacing the apparent motion disappears.

Vector Algorithms (continued)

Example



Vector Algorithms (continued)

Normalization

A normalization to the convolution integral is performed in the equation below to ensure that the apparent brightness and contrast of the resultant image is well behaved as a function of kernel shape, phase and length.

$$F'(x,y) = \frac{\sum_{i=0}^{l} F(\lfloor P_i \rfloor) h_i + \sum_{i=0}^{r} F(\lfloor P_i' \rfloor) h_i'}{\sum_{i=0}^{l} h_i + \sum_{i=0}^{r} h_i'}$$

Vector Algorithms (continued)

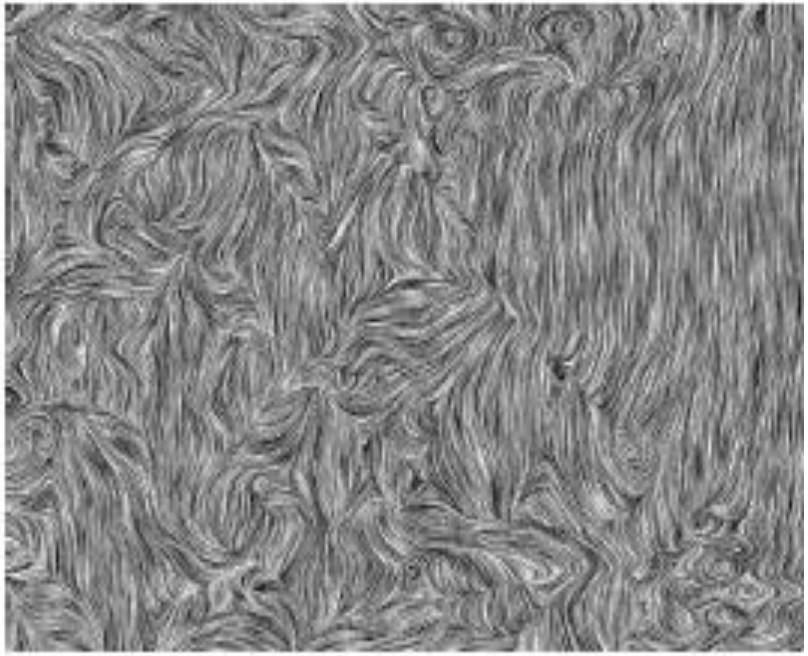
Variable and Constant kernel normalization

Because the actual length of the LIC may vary from pixel to pixel, the denominator cannot be pre-computed.

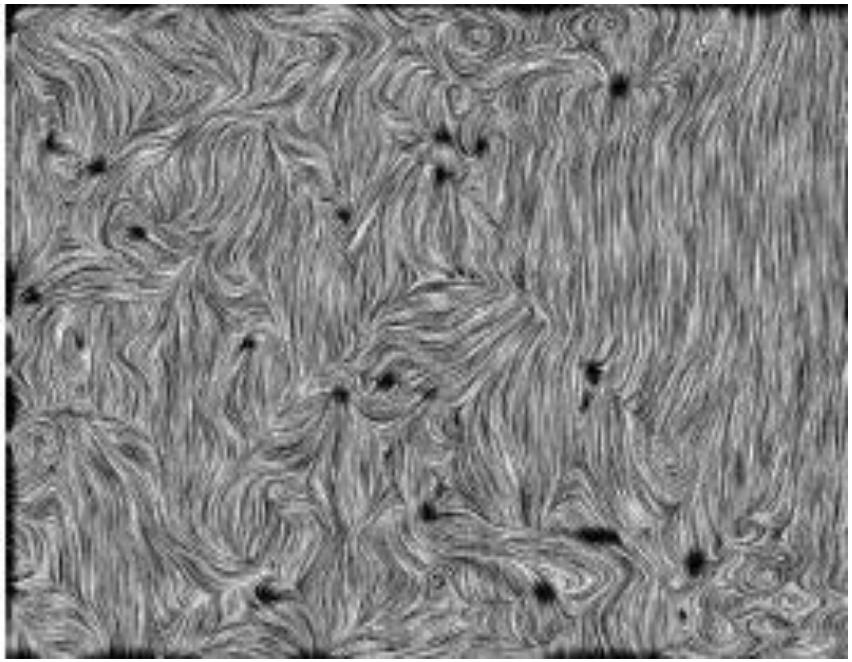
However, an interesting effect is observed if a fixed normalization is used. Truncated stream lines are attenuated which highlights singularities.

Vector Algorithms (continued)

Comparison between different normalizations



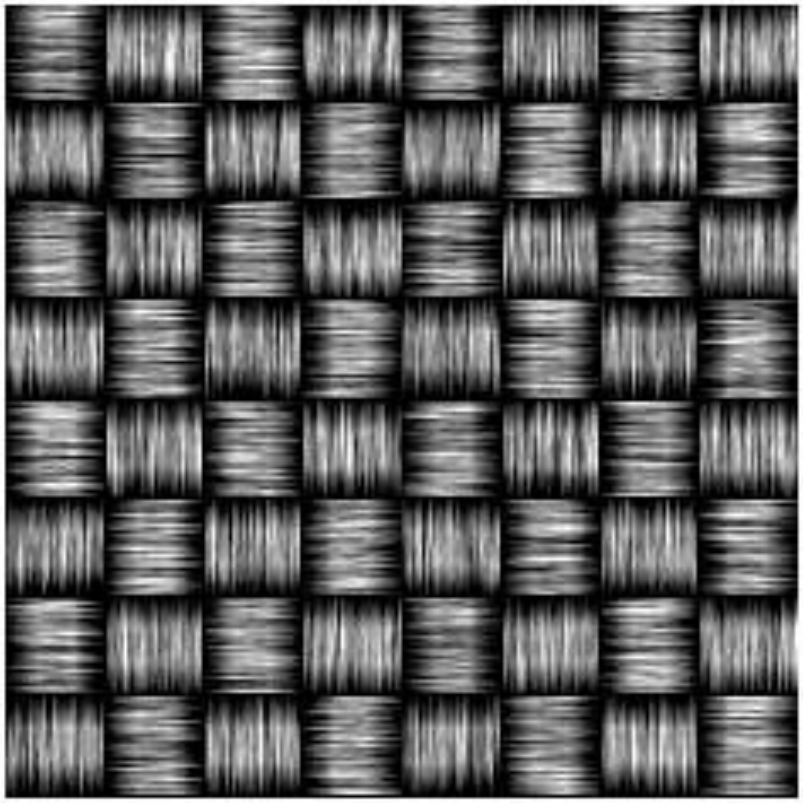
Variable kernel normalization



Constant kernel normalization

Vector Algorithms (continued)

White noise convolved with checkerboard vector field



fixed normalization

gradient shaded normalization

Vector Algorithms (continued)

Three-dimensional LIC

The LIC algorithm easily generalizes to higher dimensions.

In the 3D case, cell edges are replaced with cell faces. Both the input vector field and input texture must be three-dimensional.

The output of the three-dimensional LIC algorithm is a 3D image or scalar field.

Vector Algorithms (continued)

3D rendering

This is a 3D rendering of an electrostatic field with two point charges placed a fixed distance apart from one another.

Vector Algorithms (continued)

Implementation of the LIC algorithm

The LIC algorithm is designed as a function, which maps an input vector field and texture to a filtered version of the input texture. The dimension of the output texture is that of the vector field. Careful attention must be paid to the size of the input texture relative to that of the vector field.

Vector Algorithms (continued)

Size of the input texture

- If the texture is too large it is cropped to the vector field dimensions.
- If the input texture is smaller than the vector field the implementation of the algorithm wraps the texture using a toroidal topology. That is, the right and left edges wrap as do the top and bottom edges.
- If too small a texture is used, the periodicity induced by the texture tiling will be visible.

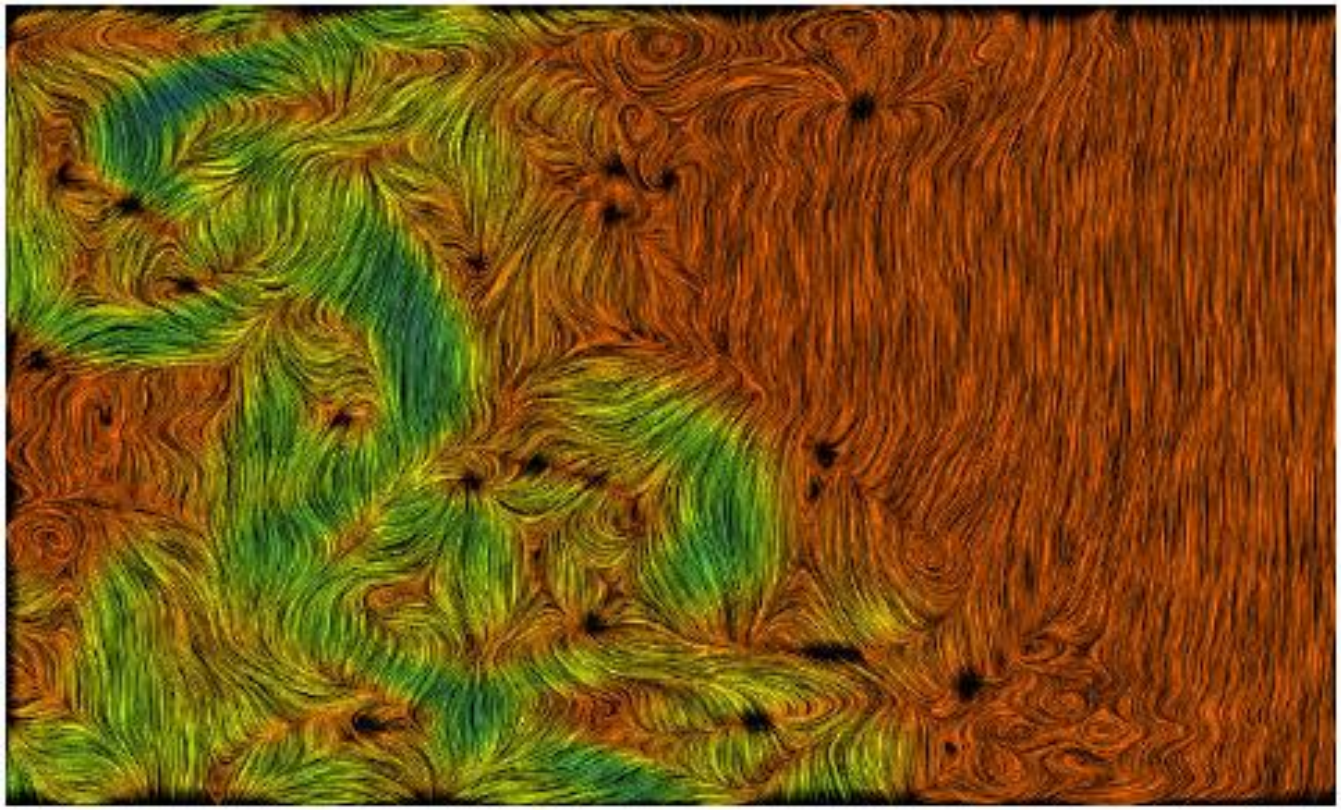
Vector Algorithms (continued)

Post processing

The output of the LIC algorithm can be operated on in a variety of ways. In this section several standard techniques are used in combination with LIC to produce novel results.

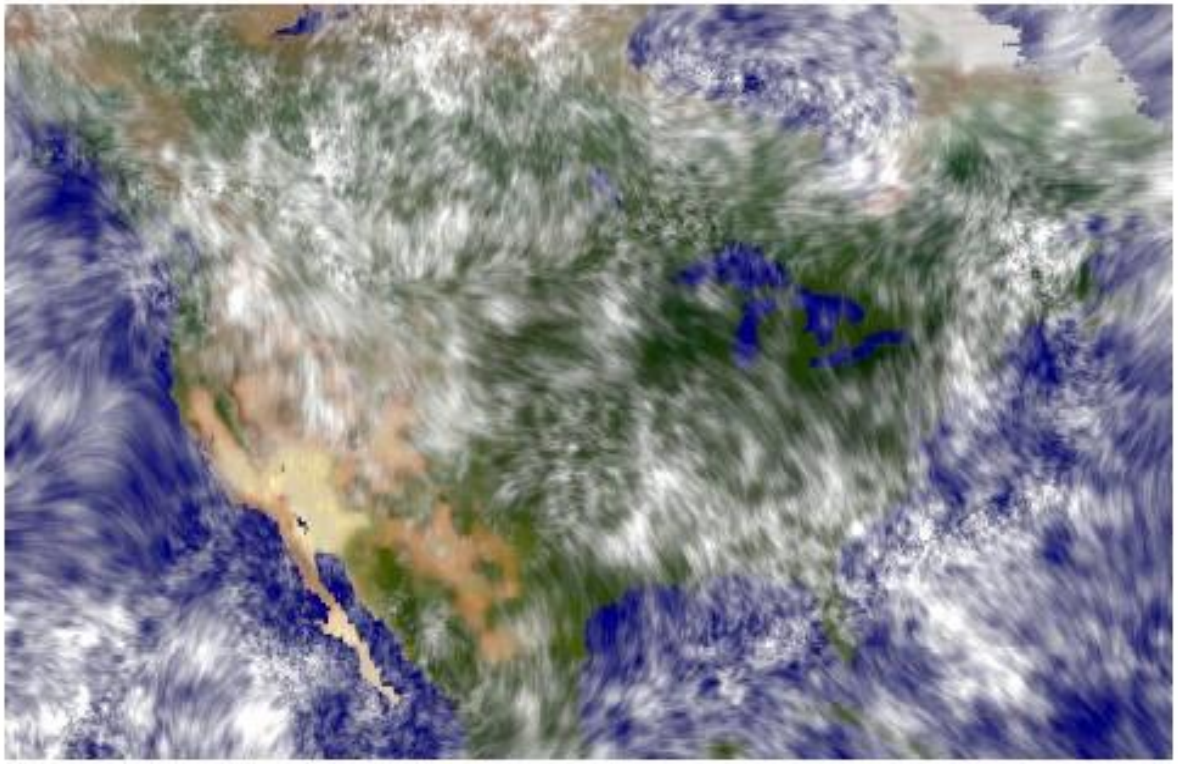
Vector Algorithms (continued)

The fixed normalization fluid dynamics field is multiplied by a color image of the magnitude of the vector field:



Vector Algorithms (continued)

A wind velocity visualization is created by compositing an image of North America under an image of the velocity field rendered using variable length LIC over noise:



Vector Algorithms (continued)

The LIC algorithm can be used to process an image using a vector field generated from the image itself, e.g. image gradient:

Vector Algorithms (continued)

The LIC algorithm can also be used to post process images to generate motion blur. The original photo on the left shows no motion blurring. The photo on the right uses variable length LIC to motion blur Boris Yeltsin's waving arm, simulating a slower shutter.