Vector Field Visualization

The need to visualize vector fields in two or three dimensions is common in many scientific disciplines, as well as it is necessary in some industrial branches. Examples of vector fields include velocities of wind and ocean currents, results of fluid dynamics solutions, magnetic fields, blood flow or components of stress and strain in materials. Existing techniques for vector field visualization differ in how well they represent such attributes of the vector field as magnitude, direction and critical points.

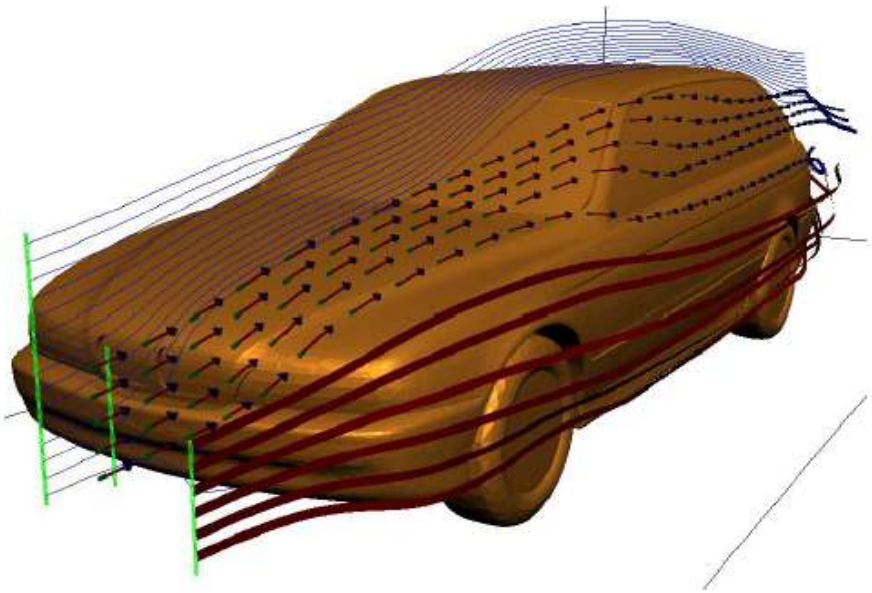
> **Vector**:
>
> A mathematical object having the properties of magnitude and direction, usually pictorially represented by a line segment with an arrowhead. In 3D modeling a vector is often given a geometric interpretation, for example, position or displacement. Vector algebra and calculus are effective tools for creating and manipulating certain kinds of models.

> **Vector field**:
>
> A region of space under the influence of some vector quantity, such as magnetic field strength, in which each point can be described by a vector.

An 3D vector data set each point is represented by a vector that stands for a direction and magnitude. The vector field is given by a $n-tupel(f_1,\ldots,f_n)$ with $f_k=f_k(x_1,\ldots,x_n), n\geqslant 2\wedge 1\leqslant k\leqslant n$ and specific transformation properties. Typically the dimension is $n=k=2$ or $n=k=3$.
The main application of vector field visualization is flow visualization with the following attributes:

- Motion of fluids (gas, liquids)
- Geometric boundary conditions
- Velocity (flow) field $v(x,t)$
- Pressure p
- Temperature T
- Vorticity $\nabla\times v$ (rotation operator)
- Density ρ
- Conservation of mass, energy, and momentum
- Navier-Stokes equations
- CFD (Computational Fluid Dynamics)

CFD data based flow visualization with lines, icons and glyphs. By extending the lines to 2D bands, one can see if the band rotates around the own axis.



For all kinds of flow visualization, but especially in the range of automobile wind tunnel testing a good classification is needed:

- Dimension (2D or 3D) e.g. the surface is 2D but lies in 3D space.
- Time-dependency: stationary (steady) vs. instationary (unsteady).
- Grid type: e.g. cartesian or tetrahedral grids have different ways to trace particles through the grid.
- Compressible vs. incompressible fluids

In most cases numerical methods are required for flow visualization.

1 Vector Calculus

- Review of basics of vector calculus
- Deals with vector fields and various kinds of derivatives
- Flat (Cartesian) manifolds only
- Cartesian coordinates only
- 3D only

At first just in order to refresh some basics again, a few needful definitions:

> **Gradient**:
>
> The gradient of a (differentiable) function $f = f(x) = f(x_1, \ldots, x_0)$ of n variables is the vector of partial derivatives $grad(f) = \nabla f = (\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n})$ It can be pictured as a vector pointing in the direction of fastest increase of f, whose length $|\nabla f|$ is the rate of increase of f in this direction. Thus, at any given point ∇f is normal to the level surface $f(x_1, \ldots, x_n)$ = constant which goes through that point.

> **Jacobi matrix**:
>
> In general a function $f = f(x)$ of one variable is differentiable at x with derivative $f'(x)$ if $f(x+h) = f(x) + f'(x) \cdot h + R(h)$ where $\lim_{|h| \to 0} R(h)/|h| = 0$.
> This definition can be generalized to the case of m functions of n variables. Then

x and h are $n\times 1$ matrices, f and R are $m\times 1$ matrices, and one defines for example $|h|^2=h_1^2+\ldots+h_n^2$.

$f'(x)$ then becomes an $m\times n$ matrix, called the Jacobi matrix whose elements are the partial derivatives:

$$f'(x)=\begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix}$$

Divergence:

Given a vector field $v(x,y,z)=v_1(x,y,z)i+v_2(x,y,z)j+v_3(x,y,z)k$. Each of its 3 components has derivatives with respect to each of the 3 variables, which gives a total of 9 first derivatives. The divergence of v is a combination of these of particular importance, it is a scalar function, defined as the dot product of the vector operator $\left(\frac{\partial}{\partial x},\frac{\partial}{\partial y},\frac{\partial}{\partial z}\right)$. Thus we have $div\ v=\frac{\partial v_1}{\partial x}+\frac{\partial v_2}{\partial y}+\frac{\partial v_3}{\partial z}$.

This vector operator is often written as ∇ , and called "del". The divergence is written as either $(div\ v)$ or $(\nabla\cdot v)$.

- **Scalar function** $\rho(x,t)$
- **Gradient** $\nabla\rho(x,t)=\begin{pmatrix} \frac{\partial}{\partial x}\rho(x,t) \\ \frac{\partial}{\partial y}\rho(x,t) \\ \frac{\partial}{\partial z}\rho(x,t) \end{pmatrix}=\begin{pmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial x} \\ \frac{\partial}{\partial x} \end{pmatrix}\rho(x,t)$ points into direction of maximum change of $\rho(x,t)$.

  The gradient takes a scalar field and gives us a vector field.
- **Laplace** $\Delta\rho(x,t)=\nabla\cdot\nabla\rho(x,t)$

  $=\frac{\partial^2}{\partial x^2}\rho(x,t)+\frac{\partial^2}{\partial y^2}\rho(x,t)+\frac{\partial^2}{\partial z^2}\rho(x,t)$ where

  $\nabla\cdot\nabla$ = scalar product of two vectors is a scalar value

  $\rho(x,t)$ = scalar function

  The Laplace function takes a vector and gives back a scalar value.
- **Vector function** $v(x,t)$
- **Jacobi matrix** ("Gradient Tensor")

$$J=\nabla v(x,t)=\begin{pmatrix} grad\ V_x \\ grad\ V_y \\ grad\ V_z \end{pmatrix}=\begin{pmatrix} \frac{\partial}{\partial x}V_x & \frac{\partial}{\partial y}V_x & \frac{\partial}{\partial z}V_x \\ \frac{\partial}{\partial x}V_y & \frac{\partial}{\partial y}V_y & \frac{\partial}{\partial z}V_y \\ \frac{\partial}{\partial x}V_z & \frac{\partial}{\partial y}V_z & \frac{\partial}{\partial z}V_z \end{pmatrix}$$

The Jacobi matrix tells you everything of the first order derivatives (gradient) of the vector field, and therefore how various components of the vector field change if you go in a certain direction. The derivative of velocity is acceleration (the first diagonal in the Jacobi matrix).

- **Divergence**: $div\ v(x,t) = \nabla \cdot v(x,t) = \frac{\partial}{\partial x} V_x(x,t) + \frac{\partial}{\partial y} V_y(x,t) + \frac{\partial}{\partial z} V_z(x,t)$ where

  ∇ = Nabla is an operator vector with 3 components.

  $\frac{\partial}{\partial x} V_x(x,t)$ = result is something scalar.

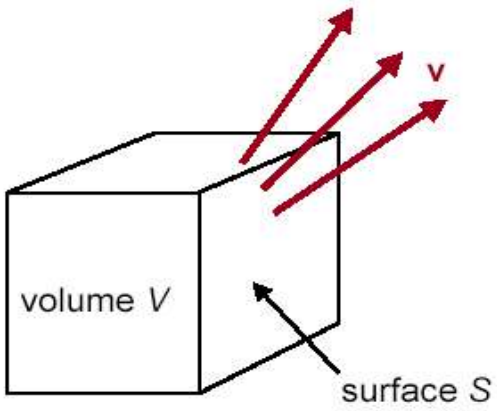
  The Divergence takes a vector field and gives back a scalar field. In physical terms, the divergence of a vector field is the extent to which the vector field flow behaves like a source or a sink at a given point.

The **Divergence theorem** in vector calculus is more commonly known as **Gauss theorem**. It is a result that links the divergence of a vector field to the value of surface integrals of the flow defined by the field. Let V be a region in space with boundary ∂V . Then the volume integral of the divergence $\nabla \cdot v$ of v over V and the surface integral of F over the boundary ∂V of V are related by:

$$\int_V (\nabla \cdot v)\, dV = \int_{\partial V} v \cdot dA$$

The divergence theorem is a mathematical statement of the physical fact, that in the absence of the creation or destruction of matter, the density within a region of space can change only by having it flow into or away from the region through its boundary.
A special case of the divergence theorem follows by specializing to the plane. Letting S be a region in the plane with boundary ∂S the equation then collapses to :

$$\int_V (\nabla \cdot v)\, dV = \oint_S v \cdot ds$$

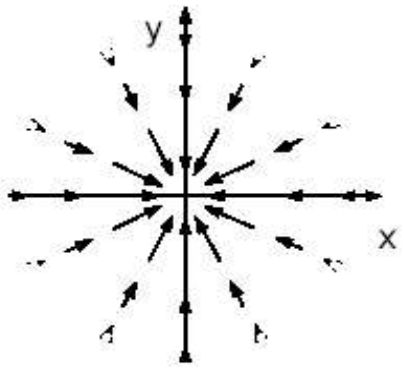


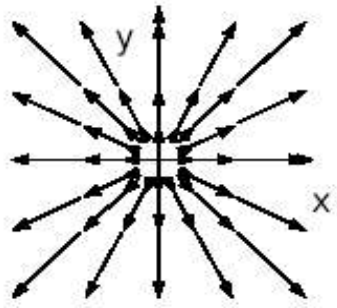
The Gauss theorem tells us about the behavior of the vector field on the surface. We integrate over the outer bounds instead over the whole volume to get what is flowing into or out of it.

Properties of divergence:

- $div\ v$ is a scalar field (v is a vector field)
- $div\ v(x_0) > 0 : v$ has a source in x_0
- $div\ v(x_0) < 0 : v$ has a sink in x_0
- $div\ v(x_0) = 0 : v$ is source-free in x_0
- Describes flow into/out of a region

Divergence with sink in x_0 *and divergence with source in* x_0 .





- The **Continuity equation** expresses a conservation law by equating a net flux over a surface with a loss or gain of material within the surface. Continuity equations often can be expressed in either integral or differential form.
    - Flow of mass into a volume V with surface S : $-\oint_S \rho v \cdot dA$ where

      ρ = volume intensity (integrating over intensity yields the mass)
    - Change of mass inside the volume: $\frac{\partial}{\partial t}\int_V \rho \, dV = \int_V \frac{\partial \rho}{\partial t} dV$

      If there is mass generated inside the volume or mass has been lost, it must have flown into or out of the volume.
    - Conservation of mass: $\int_V \frac{\partial \rho}{\partial t} dV = -\oint_S \rho v \cdot dA$ where

    $\oint_S \ldots$ = closed integral over all densities in the volume

    Application of Gauss theorem to the continuity equation (must be met for any volume element):

    $$\int_V \frac{\partial \rho}{\partial t} dV + \oint_S \rho v \cdot dA = \int_S \left(\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho v) \right) dV$$

    yields the continuity equation in differential form:

    $\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho v) = 0$ where

    ρ = velocity

    v = scalar value

    ρv = volume current j

    The change intensity is described by the sources or the sinks of the volume and they are described by the divergence.

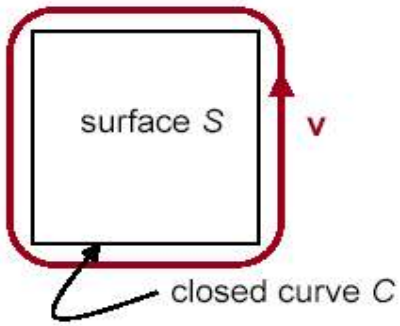
- **Curl**:

$$curl\, v(x,t) = \nabla \times v(x,t) = \begin{pmatrix} \frac{\partial}{\partial y} V_z(x,t) - \frac{\partial}{\partial z} V_y(x,t) \\ \frac{\partial}{\partial z} V_x(x,t) - \frac{\partial}{\partial x} V_z(x,t) \\ \frac{\partial}{\partial x} V_z(x,t) - \frac{\partial}{\partial y} V_x(x,t) \end{pmatrix}$$

The curl operator takes a vector field and gives back a vector field.

- **Stokes theorem**: The Navier-Stokes equation is the fundamental partial differential equation that describes the flow of incompressible fluids. It relates line integrals of vector fields to surface integrals of vector fields.

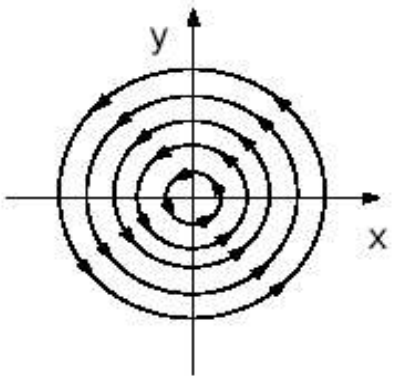
$$\int_S \nabla \times v \cdot dA = \oint_C v \cdot ds$$



Looking at the curl of a vector field, we have to integrate over its surface. By drawing a closed path around the surface, we can integrate over this line instead, to simplify the equation.

Properties of curl:

- Describes vortex characteristics in the flow
- From non-curl free flow $\nabla \times v \neq 0$ we can follow that $\int_S \nabla \times v \cdot dA \neq 0$ and $\oint_C v \cdot ds \neq 0$
- **Vorticity**: $\nabla \times v$
  If we compute the curl of a vector field with velocity = 0 we will get its vorticity.



Curl of a vector field.

- **Decomposition of a vector field** (**Helmholtz theorem**): If we know the divergence and the curl of a vector field then we know everything there is to know about the field. In fact, this is the case. There is a mathematical theorem which sums this up. It is called Helmholtz theorem. The equation tells us, that we can take any vector field and split it up into two essential parts:

- Divergence-free (transversal) part
- Curl-free (longitudinal) part

Decomposition:

$v(x)=v_c(x)+v_d(x)$ where

$\nabla\times v_c(x)=0$ = curl-free part. If eq. Is true, there are no closed field lines.

$\nabla\cdot v_d(x)=0$ = divergence-free part. If eq. Is true, there are no sinks or sources.

Other way round: v is uniquely determined by divergence-free and curl free parts (if v vanishes at boundary / infinity).

Now, we want to represent any given vector field as laplacian vector field. Suppose we have a representation of v by the Poisson equation:

$v(x)=-\Delta z(x)$

the curl-free part written as a gradient:

$v_c(x)=-\nabla u(x)$ where

$u(x)=\nabla\cdot z(x)$ = (scalar) potential u

and the divergence-free part as a curl:

$v_d(x)=\nabla\times w(x)$ where

$w(x)=\nabla\times z(x)$ = vector potential w

2 Characteristic Lines

The visualization of flow data has become one of the research topics in scientific visualization. Several techniques for visualizing the flow of fluids and gases have been developed. In this chapter we introduce some important approaches of characteristic lines in a vector field *[Theisel-1998-UFF]*:

- Stream lines: tangential to the vector field.
- Path lines: trajectories of massless particles in the flow.
- Streak lines: trace of dye that is released into the flow at a fixed position.
- Time lines (time surfaces): propagation of a line (surface) of massless elements in time.

Stream lines:

Stream lines are the tangent curves of the vector field V. For every time and every location there is one and only one stream line through it (except for critical points). The stream line is a solution to the initial value problem of an ordinary differential equation:

$L(0)=x_0$, $\frac{dL(u)}{du}=v(L(u),t)$ where

L = curve of the stream line

$L(0)$ = initial value with seed point x_0

$\frac{dL(u)}{du}$ = tension direction of the curve is the first derivative (tangential)

$v(L(u),t)$ = velocity at this position at fixed time t

The stream line for one point is the curve $L(u)$ with the parameter u *, if we want the stream lines for the whole domain, we have to integrate the ordinary differential equation.*

Path lines:

Path lines are obtained by setting out a particle and tracing its path in the unsteady flow. In other words, they are trajectories of massless particles in a time-dependent flow. Therefore, path lines are projections of the tangent curves of V into a plane $t=const.$

For every location and every time there is one and only one path line through it (except for critical points). Path line is a solution to the initial value problem of an ordinary differential equation:

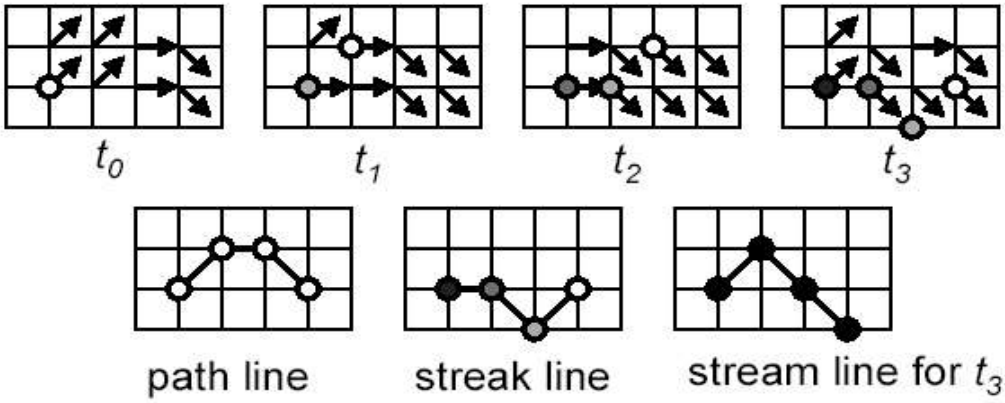
$L(0)=x_0$, $\frac{dL(u)}{du}=v(L(u),u)$ where

L = curve of the stream line

$L(0)$ = initial value with seed point x_0

$\frac{dL(u)}{du}$ = tension direction of the curve is the first derivative (tangential)

$v(L(u),u)$ = if we move along the path, we have to take the new velocity at that point and time.



Comparison of path lines, streak lines and stream lines.

Path lines are time dependent and follow one particle through time and space. Streak lines connect all particles at one time step that started at the same seed point (from jungest to oldest particle). Stream lines integrate from point to point at one time step to yield a tangent curve.

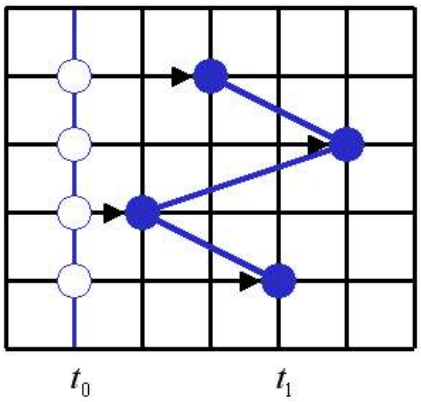
Path lines, streak lines and stream lines are identical for steady flows.

Streak lines:
A streak line is the connection of all particles set out at one seed point at different time steps. It can be imagined the trace of dye, that is released into the flow at a fixed point. Dealing with streak lines gives the following two problems:

- A streak line through a certain point u at time step t_1 is not uniquely defined. Another choice of t_0 might lead to another streak line through u at t_1 .
- Computing a streak line through u at t_1 , we have to compute the tangent curve l_1 of V . This is only possible by integrating l_1 numerically – a procedure we wanted to avoid.

Time lines:
Time lines are obtained by setting out particles located on a straight line at a fixed time and tracing them in the unsteady flow. The result is the propagation of a line (surface) of massless elements in time.

A time line connects particles that were released simultaneously.



t_0 $(x_1, y_1)\ldots(x_1, y_5)$ t_1

$(x_1, y_1)\rightarrow(x_4, y_1)$ $(x_1, y_2)\rightarrow(x_2, y_2)$ $(x_1, y_3)\rightarrow(x_5, y_3)$

$(x_1, y_4)\rightarrow(x_3, y_4)$

Summary of characteristic lines:

- Stream lines: generated by tracing the direction of flow at one single time step.
- Path lines: generated by tracing the path of a single particle in continuous time.
- Streak lines: generated by continuously injecting particles from a fixed location.
- Time lines: generated by tracing a line of particles which are all released at the same time.
- For a vector field that is constant in time it is essential that path lines, streak lines and stream lines are equal.

Difference between Eulerian and Lagrangian point of view:

- **Lagrangian**:
  The Lagrangian point of view can be imagined as to sit on a particle and move along with it (good for stream lines). We have the following properties:
  - Individual particles
  - Can be identified
  - Attached are position, velocity, and other properties
  - Explicit position
  - Standard approach for particle tracing

- **Eulerian**:
  For the Eulerian point of view one stays at a certain grid point and watches what is passing by. For other algorithms then stream lines, this method is better and suits for graphics hardware, where you can describe points in space as texture. We have the following properties:
  - No individual particles
  - Properties given on a grid
  - Position is implicit

3 Arrows and Glyphs

Same as in 2D visualization, for 3D vector field visualization we also need graphical primitives that can express important features. That's where arrows and glyphs come into account again. Arrows are icons for vectors, and the most obvious way to visualize a velocity field in computer graphics is an arrow plot (sometimes called ’hedgehog’). Base points are given in one or more planes, or throughout the field, and arrows indicate both velocity direction and speed at these points. In this way, a complete view of a vector field can be given. Arrows can give reasonable results in 2D, but there are numerous problems

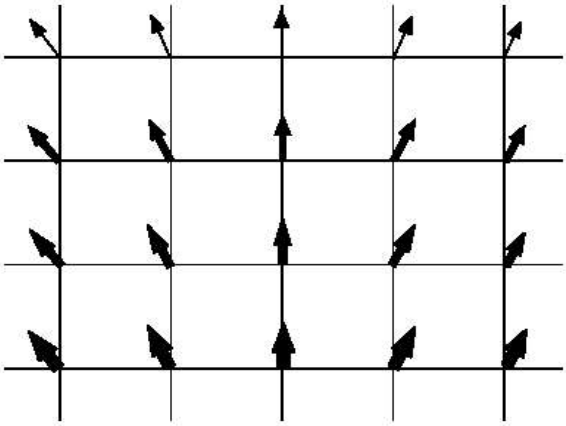
in the 3D case: directional ambiguity, cluttering and a poor spatial effect. As a result, in general arrow plots are not suitable for 3D vector field visualization. Precisely because it is very complicated to get and represent global information of a 3D vector field, we are primary interested to visualize local features of the vector field, like:

- Vector itself
- Vorticity
- Extern data: temperature, pressure, etc.

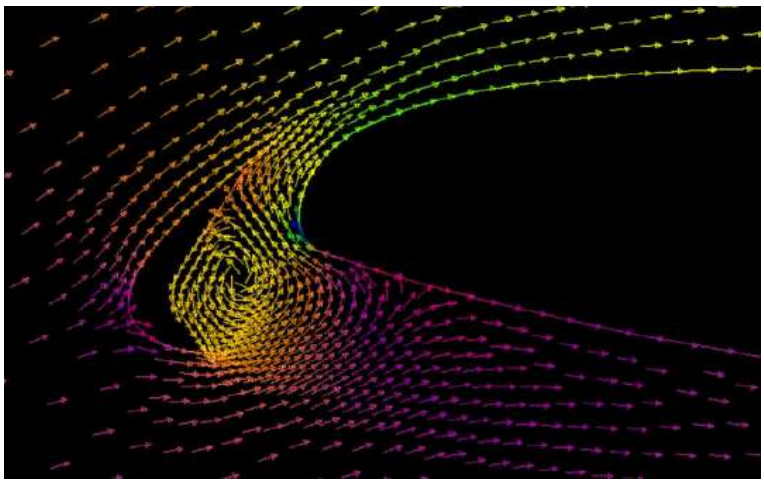
And important elements of a vector:

- Direction
- Magnitude
- Not: components of a vector

Today's approaches tend to use arrow plots or glyphs. Particularly for particle tracing which is the standard mechanism in vector field visualization. With simple arrows, we can visualize direction and orientation of a vector field, and by varying the length, thickness or color of the vector, this can stand for its magnitude.



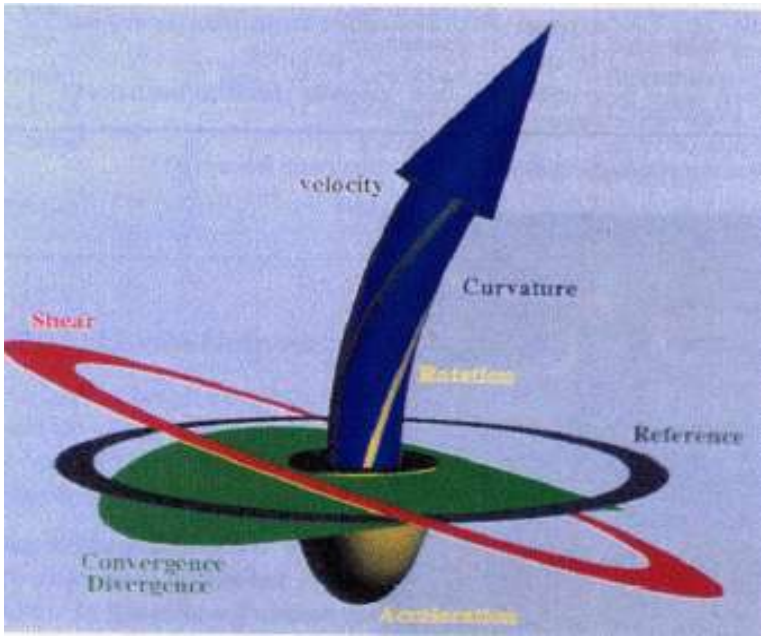
Visualization of a vector field by simple arrows.



A so called hedgehog image.

With glyphs we can visualize more features of the vector field (flow field), but one has to learn how to read the symbol.

A glyph with 7 dimensions.



Advantages of glyphs and arrows:

- Simple to generate (go to position of grid and look up velocity).
- 3D effects.

Disadvantages:

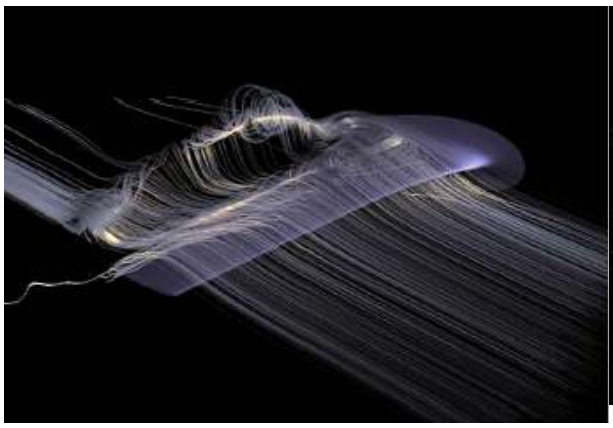
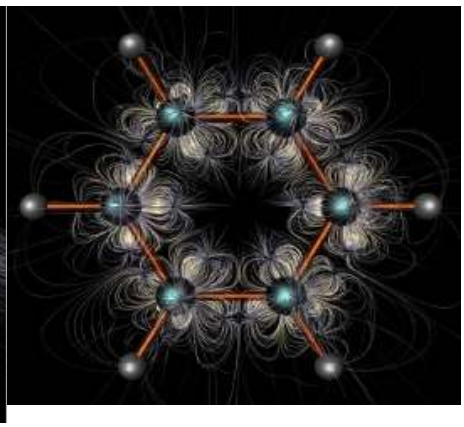
- Heavy load in the graphics subsystem.
- Inherent occlusion effects.
- Poor result if magnitude of velocity changes rapidly (Use arrows of constant length and color code magnitude).

4 Mapping Methods Based on Particle Tracing

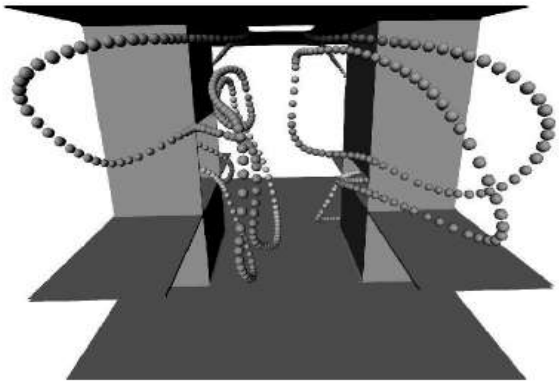
The use of particles is well-known in computer graphics. Recently several variations of the concept were developed for visualization purposes. Moving particles give a good indication of the direction and magnitude of the velocity. So the basic idea is to trace particles and visualize them effectively with characteristic lines. One great advantage of particle tracing is, that global methods exist, which will be discussed later in this chapter. Besides characteristic lines of course there exist other mapping approaches like surfaces, individual particles or sometimes even animations.

- **Path lines**: As described before path lines are trajectories of individual particles released in the flow.

Examples for path lines.

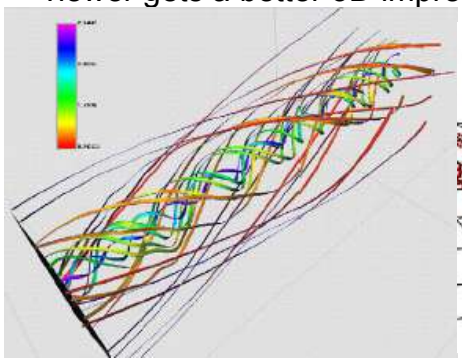
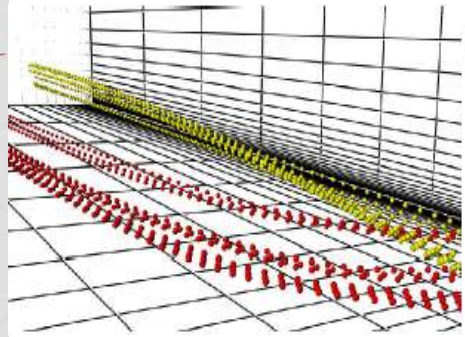
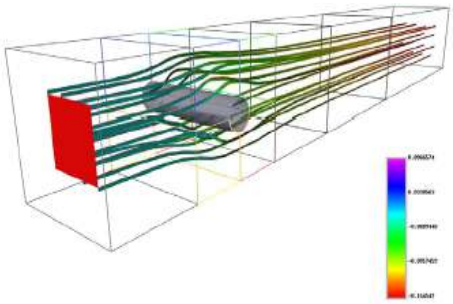



- **Stream balls**: Here we can encode an additional scalar value by radius. If the balls are far apart, velocity is high, if they are close to each other, velocity is small.



In this picture, the stream balls represent the air flow in a ventilated room.

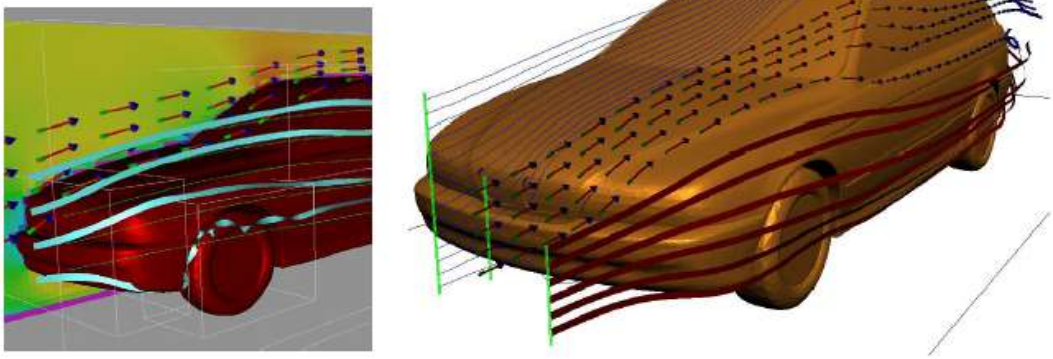
- **Streak lines**: Illuminating the surface of the bands leads to shading effects, so that the viewer gets a better 3D impression.

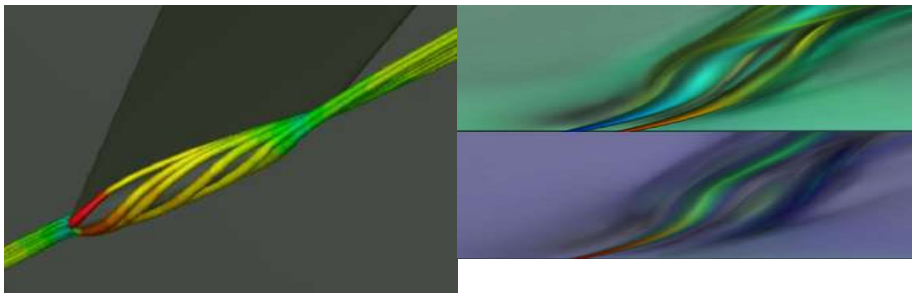
Flow visualization with streak lines and stream bands.

- **Stream ribbons**: The concept of a streamline can be readily extended to a surface. If the stream lines of two close-by traced particles are connected by a mesh of polygons, a stream ribbon results (note that we have to keep the distance constant). Alternatively, a ribbon can reflect local rotation (vorticity) of the flow by using the vorticity vector to orient the ribbon at each step.

With stream ribbons vorticity can be displayed effectively.



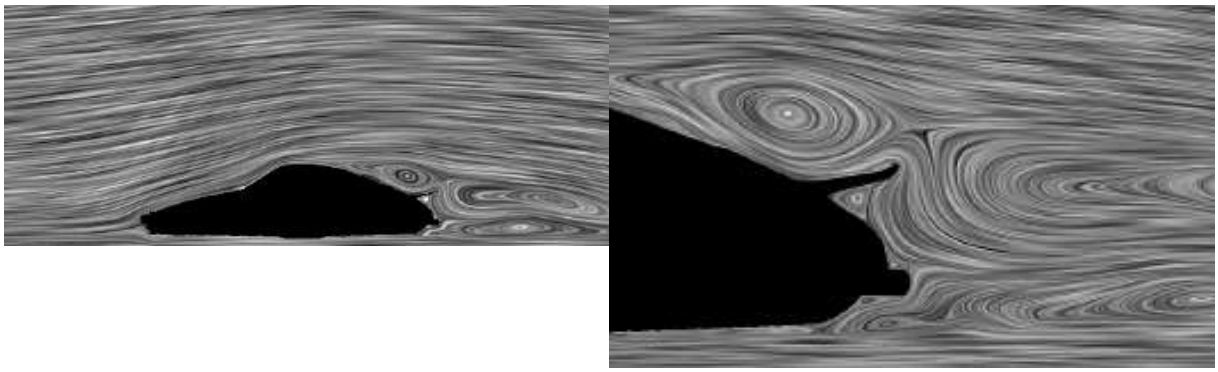
- **Stream tubes**: Stream tubes are nothing else than the extension of stream lines to a 3D tube. We specify a contour, e.g. a triangle or circle, and trace it through the flow.



Examples of different stream tubes.

- **Motion of individual particles**: Particle motion can be effectively visualized using animation. This is perhaps the most realistic visualization of flow. Velocity of particles is shown directly, provided that consecutive particle positions are given at constant time steps, and display update time is constant.

- **LIC** (Line Integral Convolution): LIC is a very important global visualization method for 2D and 3D and will be discussed in the following section.

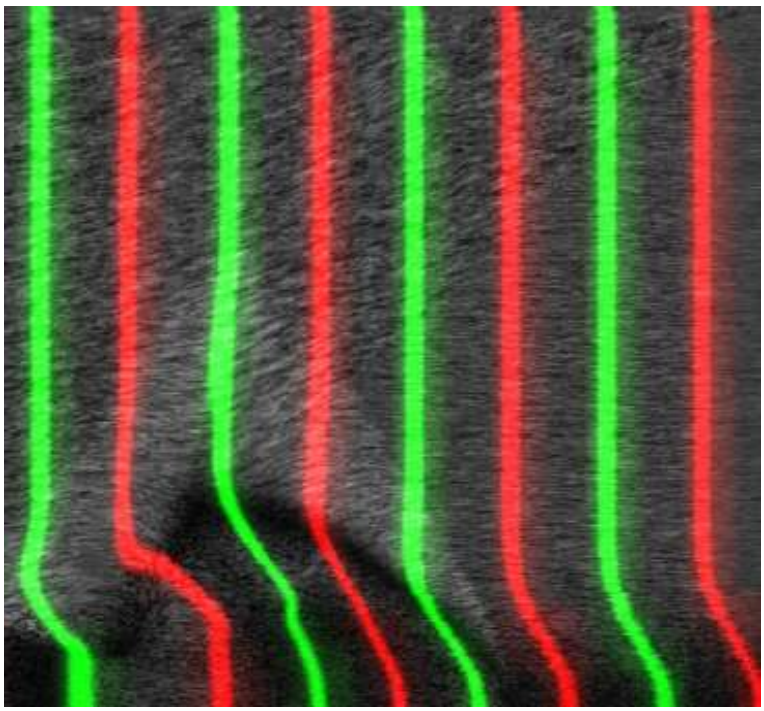


LIC generates texture by noise and gives a global representation of the flow.

- **Texture advection**: Can be used for animation.

> **Advection**:
>
> The term advection is generally understood as transport with the current and therefore a concentration peak should move through the system, like a sheet on the water.

Visualization of the interaction of a planar shock with a longitudinal vortex by noise and dye advection. The intensity of the gray-scale image is weighted by the magnitude of the velocity to enhance the important structures in the flow.



5 Numerical Integration of Ordinary Differential Equations

Numerical solution of ordinary differential equations (ODE) is the most important technique in continuous time dynamics. Since most ordinary differential equations are not soluble analytically, numerical integration is the only way to obtain information about the trajectory. Many different methods have been proposed and used in an attempt to solve accurately various types of ordinary differential equations. There are a handful of methods known and used universally that we will discuss in this chapter, i.e. Runge-Kutta, Euler, Midpoint. All these methods discretize the differential system to produce a difference equation or map. The methods obtain different maps from the same differential equation, but they have the same aim; that the dynamics of the map should correspond closely to the dynamics of the differential equation.
The typical problem of particle tracing (for path lines) can be defined as follows:

$L(0)=x_0$, $\frac{dL(u)}{du}=v(L(u),u)$ where

L = the path
u = velocity

Here we have the initial value problem, well, first we have to decide what kind of numerical solver to choose, than the conditions on the solution of the differential equation are all specified at the start of the trajectory – they are initial conditions.
As we rewrite the ODE in generic form, this leads to initial value problem for:

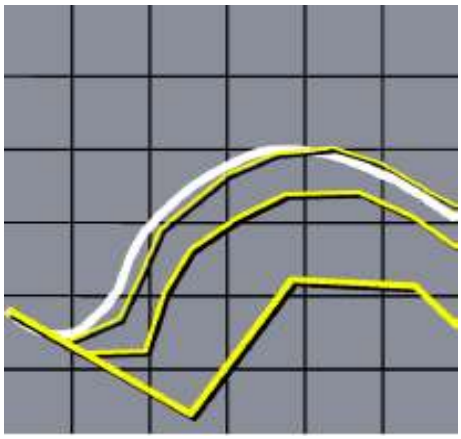
$\dot{x}(t)=f(x,t)$ where
$\dot{x}$ = (x,y,z) given by derivative of time
$f(x,t)$ = function depends on position and time

The most simple (naive) approach is the **Euler method**:

$x(t+\Delta t)=x(t)+\Delta t\, f(x,t)$ where

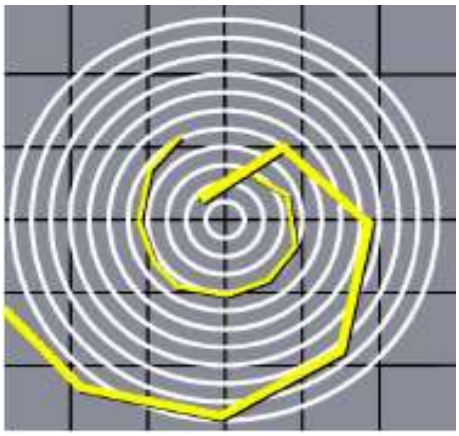
$x(t+\Delta t)$ = position at next time step
$x(t)$ = position at time t
$f(x,t)$ = velocity function

Based on Taylor expansion:
$x(t+\Delta t)=x(t)+\Delta t\,\dot{x}(t)+O(\Delta t^2)$ where
$\Delta t\,\dot{x}(t)$ = derivative of x in sector t = $f(x,t)$



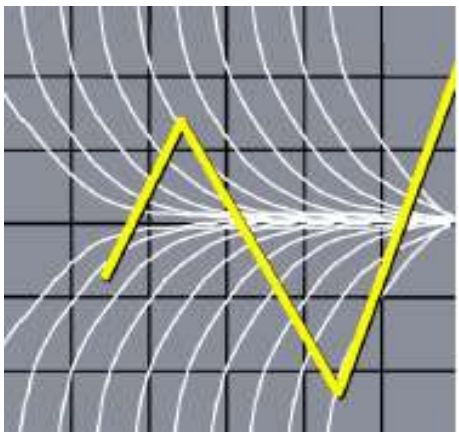
Euler method, showing three different stepsizes.

The Euler approach is a first order method. Problem with this simple method is that you need a very small step size if you want to achieve good results, which increases computation time. Though, most particle tracing modules have implemented only Euler method. Further problems are the inaccuracy and unstableness. Inaccuracy becomes apparent, when you try to track a circle (which is not linear), you will end up with a spiral. This means that you will leave the circle, no matter how small the time steps are. Unstableness for example appears, when the derivative of the function looks quite the same as the function itself.
Example:
$f=-kx$
$x=e^{-kt}$ divergence for $\Delta t>\frac{2}{k}$



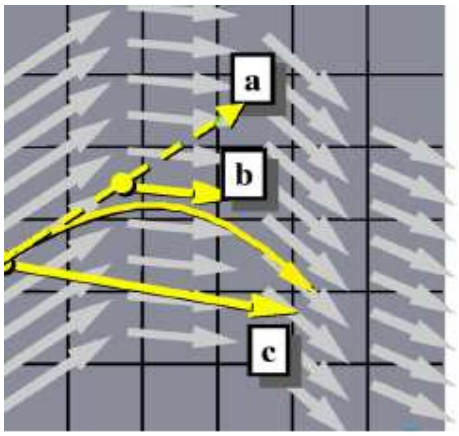
This picture shows the inaccuracy problem of the Euler method.

The Euler approach is unstable when functions have similar derivations.



Better and more accurate approaches are methods of higher order. The **midpoint method** is a second order method and therefore more exact:

1. Euler step brings us
   $\Delta x = \Delta t\, f(x,t)$
2. Evaluation of f at midpoint
   $$f_{mid} = f\left(x + \frac{\Delta x}{2}, t + \frac{\Delta t}{2}\right)$$
3. Complete step with value at midpoint
   $x(t+\Delta t) = \Delta t\, f_{mid}$



More accurate results with midpoint approach.

Runge-Kutta schemes are one-step or self-starting methods; they give x_{n+1} in terms of x_n only, and thus they produce a one-dimensional map if they are dealing with a single differential equation. For the Runge-Kutta of 4th order we interpolate velocity at 4 positions by trilinear interpolation, this means more effort for computation but gives good results.

$k_1 = \Delta t\, f(x,t)$	Function at start point.
$k_2 = \Delta t\, f\left(x + \frac{k_1}{2}, t + \frac{\Delta t}{2}\right)$	Function a little away from start point.
$k_3 = \Delta t\, f\left(x + \frac{k_2}{2}, t + \frac{\Delta t}{2}\right)$	Function near mid point.
$k_4 = \Delta t\, f(x + k_3, t + \Delta t)$	Function at final point.

$x(t+\Delta t) = x + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + O(\Delta t^5)$ Where

$x(t+\Delta t)$ = next position and Δt fixed step size.

$x+\frac{k_1}{6}+\ldots$ = actual position plus various contributions.

Adaptive step size control is a very usefull acceleration technique for Runge-Kutta schemes, the approach can be done as follows:

- Change step size according to the error.
- Increase / decrease step size depending on whether the actual local error is low / high
- Higher integration speed in “simple” regions.
- Good error control.

Scientific approaches are:

- Step size doubling.
- Embedded Runge-Kutta schemes.

Step size doubling:

- Estimating the error based on two way (one big step vs. two small steps):

  $$\begin{matrix} x_1 = RK4(x, 2\Delta t) \\ x_2 = RK4(RK4(x, \Delta t), \Delta t) \end{matrix} \Rightarrow x(t+2\Delta t) = \begin{matrix} x_1 + (2\Delta t)^5 \phi + O(\Delta t^6) \\ x_2 + 2(\Delta t)^5 \phi + O(\Delta t^6) \end{matrix}$$

  so if we end up with the same position for x_1 and x_2, we can use $2\Delta t$ as step size.
- Indication of truncation error: $\Delta = \|x_2 - x_1\|$ (length difference)
- If error Δ is larger than a given tolerance level τ, the step is redone with a new step size Δt.
- Otherwise the step is taken and $\Delta t'$ is estimated for the next integration step:

  $\Delta t' = \Delta t \cdot \rho \cdot \sqrt[5]{\frac{\tau}{\Delta}}$, with safety factor $\rho < 1$

  this can take some time, because it is repeated until the step size fits.

The **embedded Runge-Kutta** was originally invented by Fehlberg. The algorithm compares between results taken from two steps with schemes of different order $\Delta = \|x - x'\|$. A typical example: x is computed with fifth order, x' with fourth order RK. The main idea is to reuse function evaluations from higher order scheme to speed up the evaluation of lower order scheme.
Example with fourth order RK embedded in fifth order RK: Fourth order RK needs only the (six) function evaluations from fifth order RK and no further evaluations!
Practical choice for new step size for 4th / 5th order embedded RK:

$$\Delta t' = \begin{cases} \Delta t \cdot \rho \cdot \sqrt[5]{\frac{\tau}{\Delta}} & , if\ \tau \geqslant \Delta \\ \Delta t \cdot \rho \cdot \sqrt[4]{\frac{\tau}{\Delta}} & , if\ \tau < \Delta \end{cases}$$

The choice for safety factor is for example $\rho = 0.9$

So far we have only spoken about **explicit methods**, which means that for all formulas, the needed information to compute the position of the new time step only involves information of old time steps. But the stability problem can be solved by **implicit methods**.

- **Implicit Euler method**:

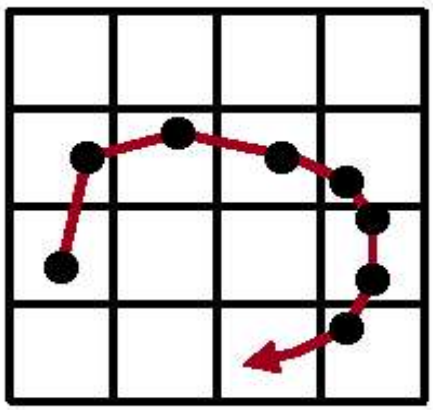
  $x(t+\Delta t) - x(t) = \Delta t\, f(x(t+\Delta t), t+\Delta t)$ where

  $x(t+\Delta t)$ = depends on position we do not know jet, so we transpose this problem to the next time step.

- "Reversing" the explicit Euler integration step.
- Taylor expansion around $t+\Delta t$ instead of t .
- Solving the system of non-linear equations to determine $x(t+\Delta t)$.
- Using implicit methods allows larger time steps.

6 Particle Tracing on Grids

Particle tracing is a widely used method to analyze and interpret results of flow simulations. In addition, it is a preliminary step for more advanced techniques like line integral convolution. If a vector field is given on a grid, we have to solve $L(0)=x_0$, $\frac{dL(t)}{dt}=v(L(t),t)$, to get a discretized path line of the particle. As we learned before, we have to numerically integrate this ordinary differential equation, which is easily done with the Euler formula.



Discretized path line of a particle trace.

The most simple case of finding a path line is on cartesian grids. We can incremental integrate step by step using Euler, Midpoint or Runge-Kutta. The basic algorithm looks like:

```
Basic particle tracing algorithm:
Select start point (seed point)
Find cell that contains start point                          //
point location
While (particle in domain) do
        Interpolate vector field at current point   // interpolation
        Integrate to new position                            //
        integration
        Find new cell                                // point location
        Draw line segment between latest
          particle positions
Endwhile
```

Note that vector field interpolation is done by bilinear / trilinear interpolating each component x,y,z separately.

Point location (cell search) on Cartesian grids:

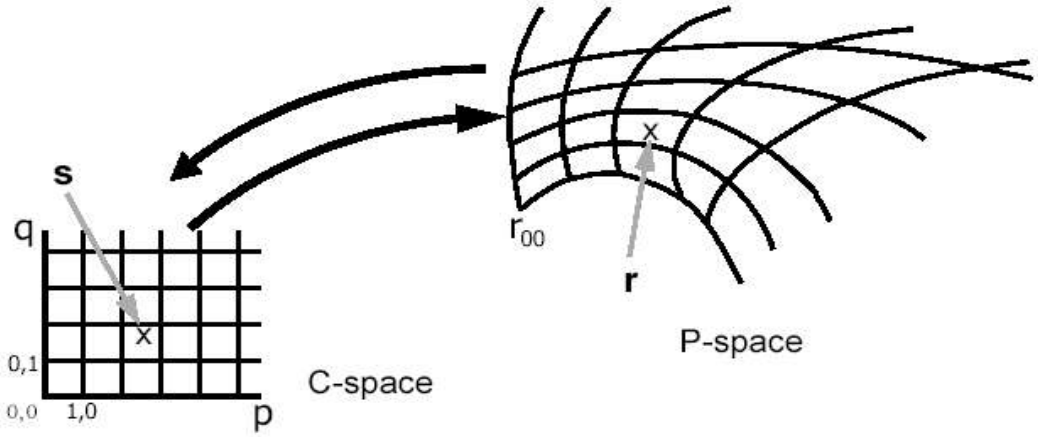
- Indices of cell directly from position (x,y,z)
- For example: $i_x=\frac{(x-x_0)}{\Delta x}$

- Simple and fast

Interpolation on Cartesian grids:

- Bilinear (in 2D) or trilinear (in 3D) interpolation
- Required to compute the vector field (= velocity) inside a cell
- Component-wise interpolation
- Based on offsets (= local coordinates within cell)

Since the Cartesian grid is not the only kind of grid used in visualization, we need particle tracing for all other kinds of grids. For example curvilinear grids are very common too, so the question is, how are curvilinear grids handled? A simple answer is, if we have a problem we don't know how to solve, we try to reduce it to a problem we already know how to solve. So for curvilinear grids we have C-space (computational space) vs. P-space (physical space) and we need to find a non linear mapping from $(P \rightarrow C)$.



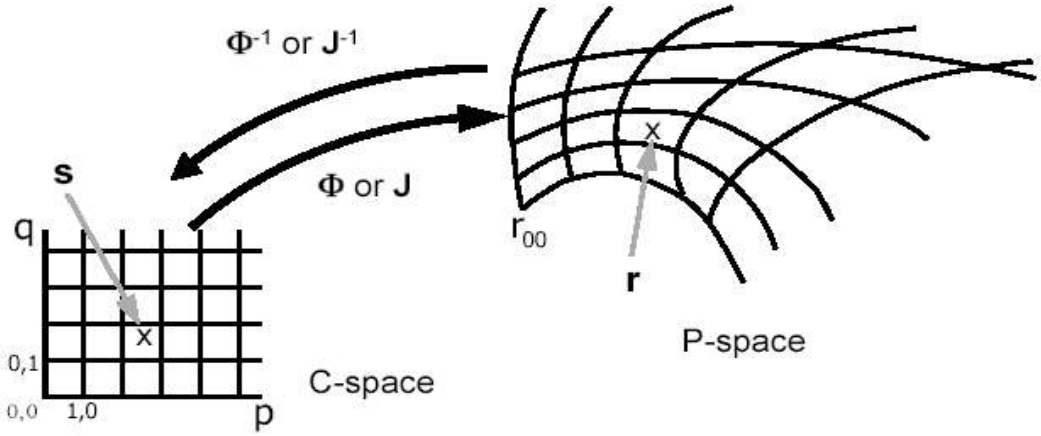
Mapping from P-space to C-space.

Particle tracing can either be done in C-space or P-space. Since tracing in C-space is much easier, we trace in C-space and transform the result back into P-space like shown in the picture below.

Transformation of

- Points by Φ
- Vectors by J

All cells have a different function Φ and a different Jacobian J , since all cells are differently deformed.



Transformation between C-space and P-space by Φ and J .

Transformation of points:

- From C-space to P-space: $r = \Phi(s)$
- From P-space to C-space: $s = \Phi^{-1}(r)$

Transformation of vectors:

- From C-space to P-space: $v = J \cdot u$
- From P-space to C-space: $u = J^{-1} \cdot v$
- J is Jacobi matrix:

$$J = \begin{pmatrix} \frac{\partial \Phi_x}{\partial p} & \frac{\partial \Phi_x}{\partial q} \\ \frac{\partial \Phi_y}{\partial p} & \frac{\partial \Phi_y}{\partial q} \end{pmatrix} \quad \text{(2D case)}$$

The Jacobian tells us, if we move a little bit in C-space, how we move in P-space. Δp and Δq in C-space gives us Δx and Δy in P-space.

```
Particle tracing algorithm in C-space:
Select start point  r  in P-space (seed point)
Find P-space cell                                   // point location
Transform start point to C-space  s
While (particle in domain) do
        Transform  v→u  at vertices  (P→C)          // once we
        have velocities
                                                    // we can
        interpolate
        Interpolate  u  in C-space                  // interpolation,
        use velocities
                                                    // from all 4
        cell vertices
        Integrate to new position  s  in C-space    // integration
        If outside current cell then
                Clipping
                Find new cell                       // point location
        Endif
        Transform to P-space  s→r     (P→C)
        Draw line segment between latest
          particle positions
Endwhile
```

Transformation of points from C-space to P-space $r = \Phi(s)$:

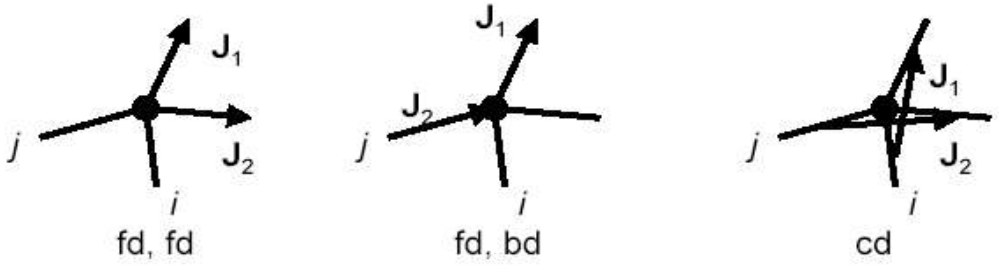
- Bilinear (in 2D) or trilinear (in 3D) interpolation between coordinates of the cell's vertices

Transformation of vectors from P-space to C-space $u = J^{-1} \cdot v$:

- Needs inverse of the Jacobian

Numerical computation of elements of the Jacobi matrix:

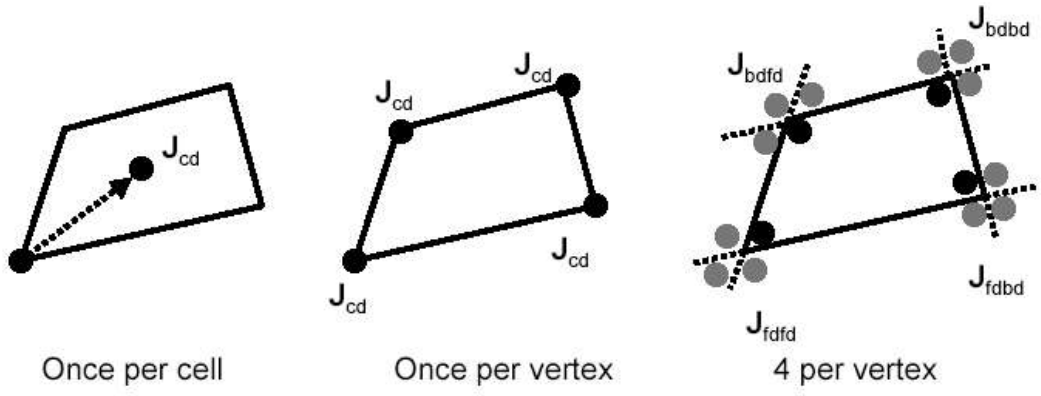
- Backward differences (bd)
- Forward differences (fd)
- Central differences (cd)



Example for forward, backward and central differences.

Choices for attaching Jacobi matrix on curvilinear grids:

- Once per cell (central differences): fast, inaccurate
- Once per vertex (central differences): slower, higher accuracy
- 4 (in 2D) or 8 (in 3D) per vertex (backward and forward differences): very time-consuming, completely compatible with bilinear / trilinear interpolation, accurate



Different possibilities to attach Jacobian to curvilinear grids.

```
Particle tracing algorithm in P-Space:
Select start point   r in P-space (seed point)
Find P-space cell and local coords                          //
point location
While (particle in domain) do
        Interpolate  v  in P-space                  // interpolation
        Integrate to new position  r  in P-space    // integration
        Find new position                           // point location
        Draw line segment between latest
          particle positions
Endwhile
```

Main problem: Point location / local coordinates in cell

- Given is point r in P-space
- What is corresponding position s in C-space?
- Coordinates s are needed for interpolating v

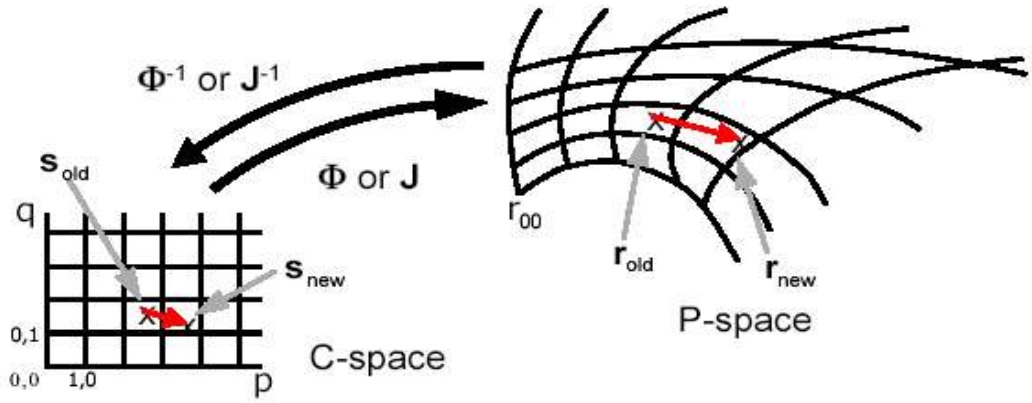
Solution: **Stencil-walk** algorithm *[Bunnig '89]*

- Iterative technique
- Needs Jacobi matrices

Stencil-walk on curvilinear grids

- Given: $r_{old}, r_{new}, s_{old}, \Phi(s_{old}) = r_{old}$
- Determine s_{new}
- Transformation of distances $\Delta s = s_{new} - s_{old}$ and $\Delta r = r_{new} - r_{old}$
- Taylor expansion: $\Delta r = r_{new} - r_{old} = \Phi(s_{new}) - \Phi(s_{old}) = J_{\Phi}(s_{old}) \cdot \Delta s + \ldots$
- Therefore $\Delta s \approx [J_{\Phi}(s_{old})]^{-1} \Delta r$

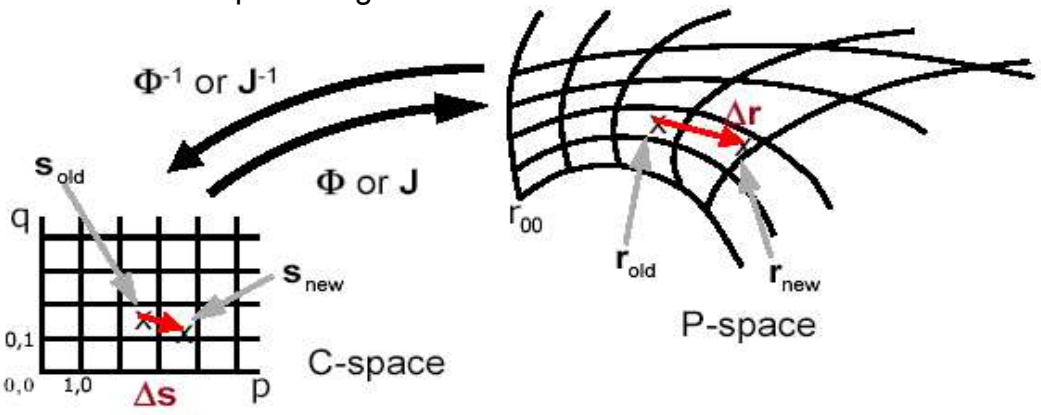
Stencil walk on curvilinear grid.



p q

Stencil Walk is an iterative technique that can operate directly on the curvilinear grid. It appears that Stencil Walk is very accurate, but also rather slow. The basic idea of stencil-walk is to:

- Pick a point in P-space. We do not know the corresponding point in C-space.
- Guess start point in C-space.
- Compute difference between corresponding position and target position in P-space.
- Improve position in C-space.
- Iterate so that position gets more and more accurate.



Stencil-walk algorithm:
Given: target position r_{target} in P-space required accuracy ε in C-space
Guess start point s in C-space
Do
Transformation to P-space $r=\Phi(s)$
Difference in P-space $\Delta r=r_{target}-r$
Transformation to C-space $\Delta s=J^{-1}\Delta r$
If ($|\Delta s|<\varepsilon$)then exit
$s=s+\Delta s$
If (s outside current cell) then
Set s = midpoint of corresponding neighboring cell
Endif
Repeat

High convergence speed of stencil walk

- Typically 3-5 iteration steps in a cell

Interpolation of velocity in P-space approach:

- Bilinear or trilinear within cell, based on local coordinates s

- Alternative method: inverse distance weighting → no stencil walk needed

Important properties of C-space integration:
☑Simple incremental cell search
☑Simple interpolation
– Complicated transformation of velocities / vectors

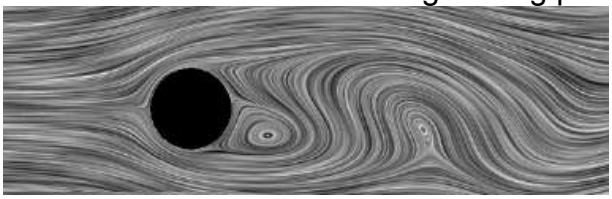
Important properties of P-space integration:
☑No transformation of velocities / vectors
– Complicated point location (stencil walk) for bilinear or trilinear interpolation

Remarks on P–space and C–space algorithms
- C–space algorithm with one Jacobian per cell is fastest
- C–space algorithms with one Jacobian per cell or per node might give incorrect results
- Accuracy of C–space algorithm depends strongly on the deformation of the grid
- The best C–space algorithm can be as good as P–space algorithms
- In general, however, P-space algorithms are more accurate and even faster than C-space algorithms
    - Only if stencil walk implementation is not considered
- Choice of method depends on:
    - Required accuracy
    - Data set
    - Interpolation method

7 Line Integral Convolution

The **Line Integral Convolution** method (**LIC**) *[Cabral-1993-IVF]* is a texture synthesis technique that can be used to visualize vector field data like dense flow fields, by imaging its integral curves. Taking a vector field and a random texture, usually a stationary white noise image, as input, the algorithm uses a low pass filter to perform one-dimensional convolution on the noise image. The convolution kernel follows the paths of streamlines originating from each pixel in both positive and negative directions. As a result, the output intensity values of the LIC pixels along each streamline are strongly correlated so the global features of the flow field can be easily visualized.

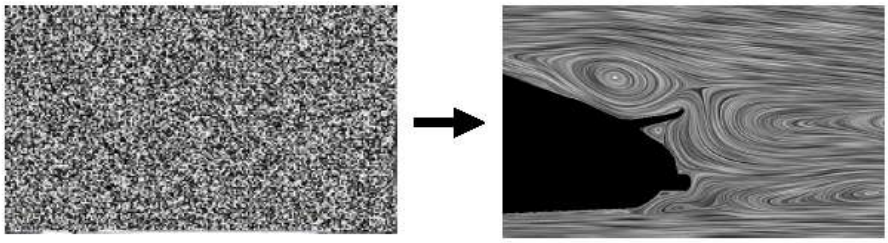
Look of 2D LIC images:
- Intensity distribution along path lines shows high correlation.
- No correlation between neighboring path lines.



The 2D LIC gives information about global features of the flow.

In a nutshell, the basic idea of Line Integral Convolution (LIC) is:
- Global visualization technique
- Start with random texture
- Smear out along stream lines

A Stationary white noise texture as input results in the final smeared LIC texture.



Algorithm for 2D LIC

- Let $t \to \Phi_0(t)$ be the path line containing the point (x_0, y_0)
- $T(x, y)$ is the randomly generated input texture
- Compute the pixel intensity as:

  $I(x_0, y_0) = \int_{-L}^{L} k(t) \cdot T(\phi_0(t))\, dt$ convolution with kernel where

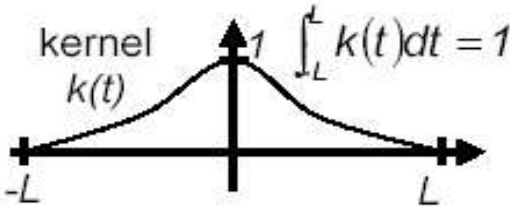
  $k(t)$ = weighted kernel

  ϕ_0 = texture look up

  t = parameter along the stream line

Kernel:

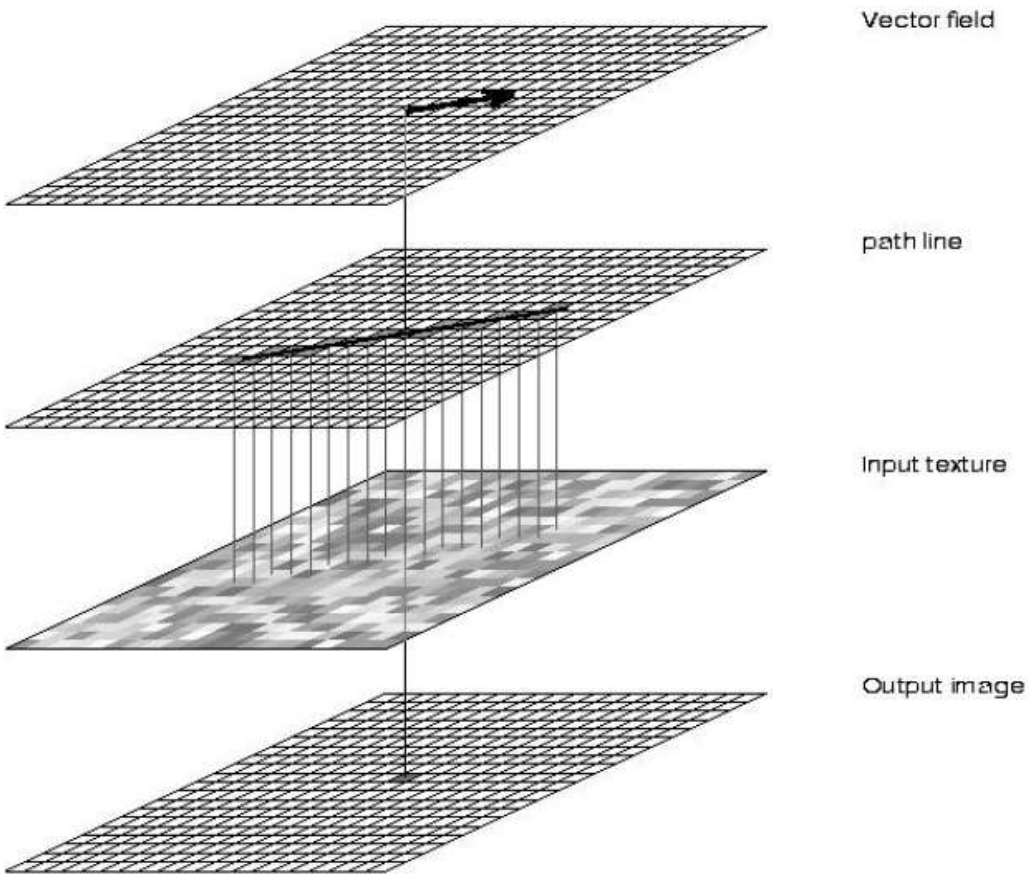
- Finite support $[-L, L]$
- Normalized
- Often simple box filter
- Often symmetric (isotropic)

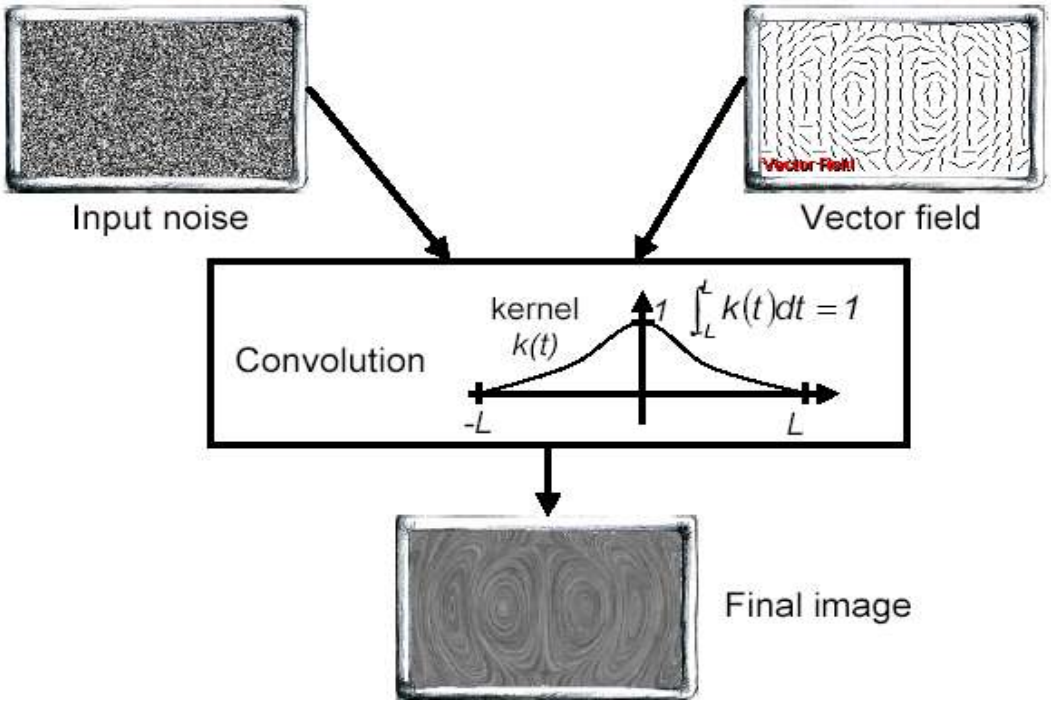


Symmetric convolution kernel.

$$L \qquad \int_{-L}^{L} k(t)\, dt = 1$$

Convolve a random texture along the stream lines. For the pixel, move up into the vector field and compute the stream line.





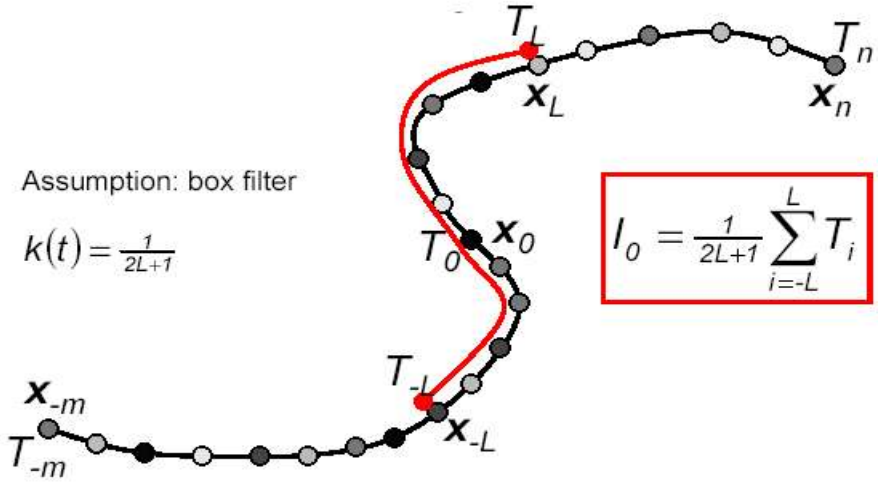
This picture illustrates the proceeding of the algorithm.

Problems with standard LIC:

- New stream line is computed at each pixel
- Convolution (integral) is computed at each pixel
- Slow

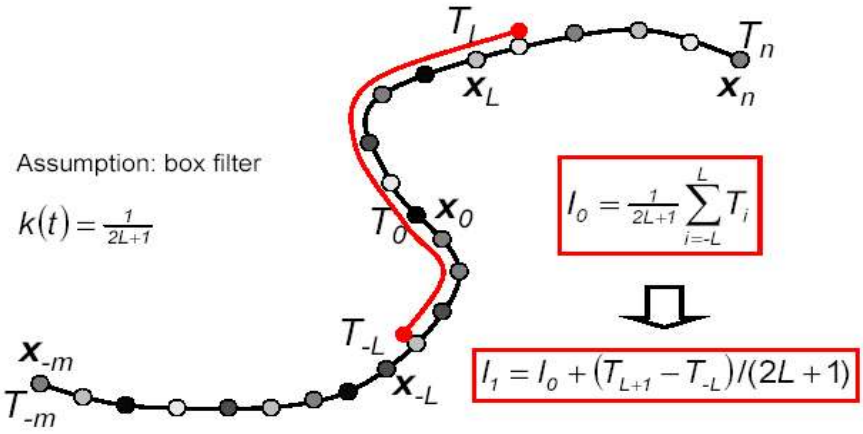
Idea of Fast LIC:

- Compute very long stream lines
- Reuse these stream lines for many different pixels
- Incremental computation of the convolution integral
- Discretization of convolution integral
- Summation: we go L steps forward and L steps backward, sum them all up and devide by $2L+1$.



Summation of $2L+1$ steps.

x_0



For the next position we only have to substract the last part and add the new part in front. This makes computation very fast.

Fast LIC incremental integration for constant kernel:

- Stream line $x_{-m},\ldots,x_0,\ldots,x_n$ with $m,n\geq L$
- Given texture values $T_{-m},\ldots,T_0,\ldots,T_n$
- What are results of convolution: $I_{-m+L},\ldots,I_0,\ldots,I_{n-L}$
- For box filter (constant kernel): $I_0=\frac{1}{2L+1}\sum_{i=-L}^{L}T_i$
- Incremental integration: $I_{j+1}-I_j=\frac{1}{2L+1}\sum_{i=-L}^{L}(T_{i+j+1}-T_{-L+j})$

Fast LIC incremental integration for polynomial kernels:

- Assumption: polynomial kernel (monom representation)

$$k(i)=\sum_{p=0}^{d}\alpha_p\cdot I^p$$

- Value

$$I_j=\sum_{p=0}^{d}\alpha_p\cdot I_j^p \quad \text{with} \quad I_j^p=\sum_{i=-L}^{L}T_{i+j}\cdot I^p$$

- Incremental update for I_i^p

$$I_{j+1}^p-I_j^p=\sum_{i=-L}^{L}(T_{j+i+1}-T_{j+i})\cdot I^p$$

$$=\sum_{i=-L}^{L}T_{j+i}\cdot((i-1)^p-i^p)+\overbrace{T_{j+L+1}\cdot L^p-T\,j-L\cdot(-L-1)^p}^{\Lambda_{j+1}^p}$$

$$=\sum_{q=0}^{p-1}\binom{p}{q}(-1)^{p-q}I^q-j+\Lambda_{j+1}^p$$

Data structure for output: Luminance/Alpha image:

- Luminance = gray-scale output
- Alpha = number of streamline passing through that pixel

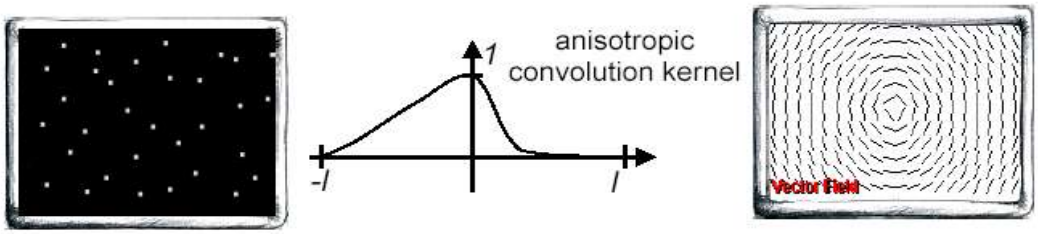
```
For each pixel p in output image
        If (Alpha( p ) < #min) Then
                Initialize streamline computation with x0 =
                center of p
                Compute convolution I(x0)
                Add result to pixel p
                For m=1 to Limit M
                        Incremental convolution for I(xm) and
                        I(x-m)
                        Add results to pixels containing xm
                        and x-m
                End for
        End if
End for
Normalize all pixels according to Alpha
```

One characteristic of fast LIC is that during computation of stream lines, we sometimes get several paths that pass through the same pixel, so we have to devide the intensity for this pixel by the number of paths passing through. The fast LIC is significantly faster than normal LIC, but still achieves no interactive computation time.

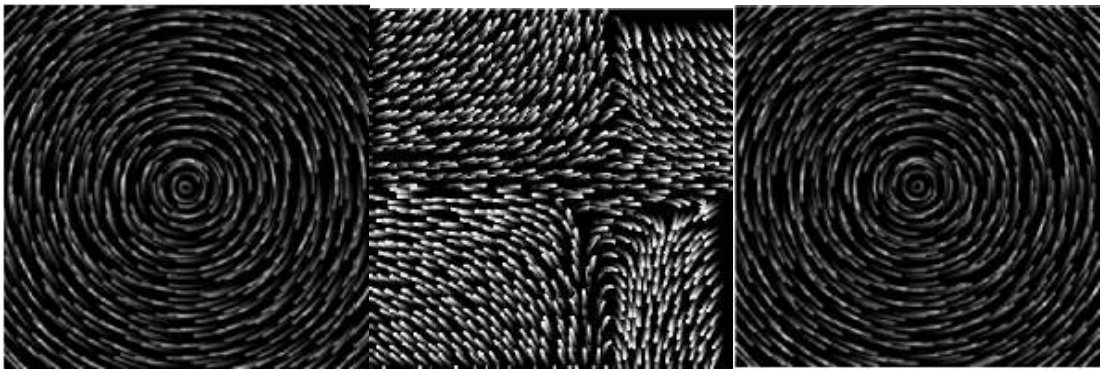
Oriented LIC (**OLIC**):

Oriented Line Integral Convolution (OLIC) illustrates flow fields by convolving a sparse texture with an anisotropic convolution kernel. The kernel is aligned to the underlying flow of the vector field. OLIC does not only show the direction of the flow but also its orientation.

Acceleration: integrate individual drops and compose them to final image.



Oriented LIC works with an anisotropic convolution kernel, that has a different weighting behavior for upstream and downstream movement.



OLIC visualizes direction and orientation of the flow. The length of the little icons depends of the length of the kernel.

Note that LIC in general doesn't draw streamlines, but pixels that are correlated to streamlines.

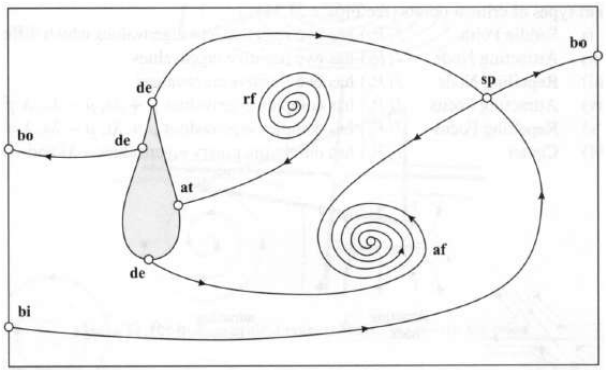
Summary:
- Dense representation of flow fields
- Convolution along stream lines → correlation along stream lines
- For 2D and 3D flows
- Stationary flows
- Extensions:
    - Unsteady flows
    - Animation
    - Texture advection

8 Vector Field Topology

Visualization of topological information of a vector field can provide useful information on the structure of the field. **Vector field topology** *[Helman-1989-RDV]* has a completely different concept as we've seen so far. The idea is to draw only stream lines that connect “important” points and consequently show only a topological skeleton. Important points in the vector field are critical points, they have the following attributes:
- Points where the vector field vanishes: $v=0$
- Points where the vector magnitude goes to zero and the vector direction is undefined
- Sources, sinks, …

The critical points are connected to divide the flow into regions with similar properties. Once you have found all critical points, compute stream lines until they hit the end of the domain, an object or another critical point. Stream lines don't cross, they only meet at settle points. By looking at the Eigenvalues of the Jacobian, you can expect the behavior around the critical point.

This picture shows an airfoil with all critical points.



- Taylor expansion for the velocity field around a critical point r_c :
  $v(r)=v(r_c)+\nabla v\cdot(r-r_c)+O(r-r_c)^2\approx J\cdot(r-r_c)$

- Divide Jacobian into symmetric and anti-symmetric parts:
  $$J=J_s+J_a=\frac{(J+J^T)+(J-J^T)}{2}$$
  $$J_s=\frac{(J+J^T)}{2}$$
  $$J_a=\frac{(J-J^T)}{2}$$

- The **symmetric part** can be solved to give real eigenvalues R and real eigenvectors:
  $J_s r_s=Rr_s$, $R=R_1,R_2,R_3$
- Eigenvectors r_s are an orthonormal set of vectors
- Describes change of size along eigenvectors
- Describes flow into or out of region around critical point

- **Anti-symmetric part:**
- $$J_a\cdot d=\frac{1}{2}(J-J^T)\cdot d$$

$$=\frac{1}{2}\begin{pmatrix} 0 & \frac{\partial v_x}{\partial y}-\frac{\partial v_y}{\partial x} & \frac{\partial v_x}{\partial z}-\frac{\partial v_z}{\partial x} \\ \frac{\partial v_y}{\partial x}-\frac{\partial v_x}{\partial y} & 0 & \frac{\partial v_y}{\partial z}-\frac{\partial v_z}{\partial y} \\ \frac{\partial v_z}{\partial x}-\frac{\partial v_x}{\partial z} & \frac{\partial v_z}{\partial y}-\frac{\partial v_y}{\partial z} & 0 \end{pmatrix}\cdot d$$

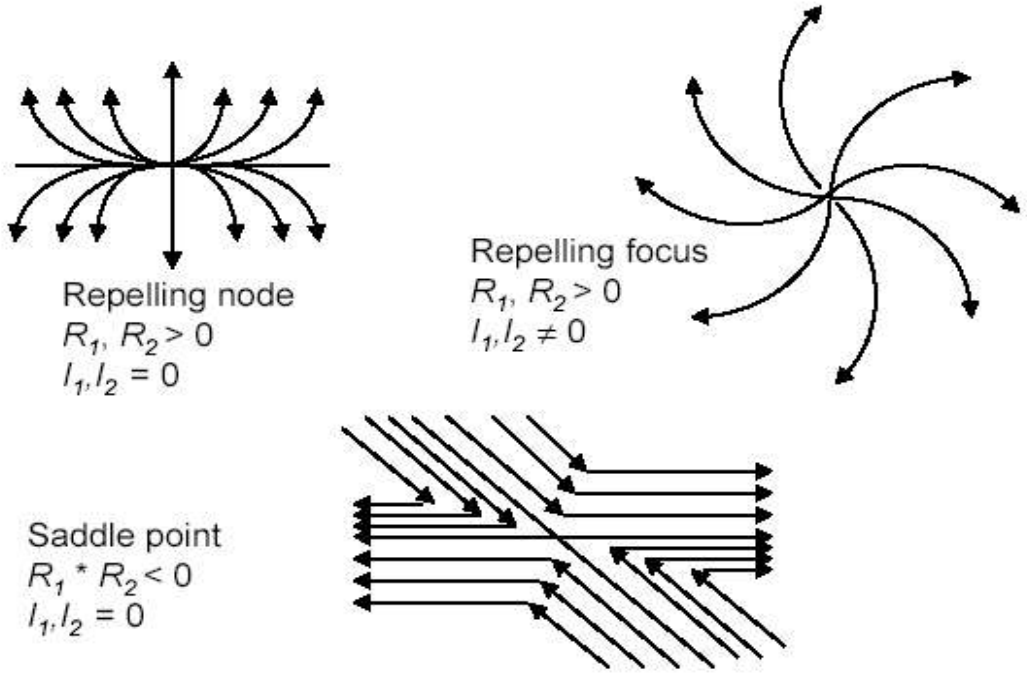
$$=\frac{1}{2}(\nabla\times v)\times d$$

where derivative of $\frac{\partial v_y}{\partial x}-\frac{\partial v_x}{\partial y}$ is nothing else than the curl of the vector field.

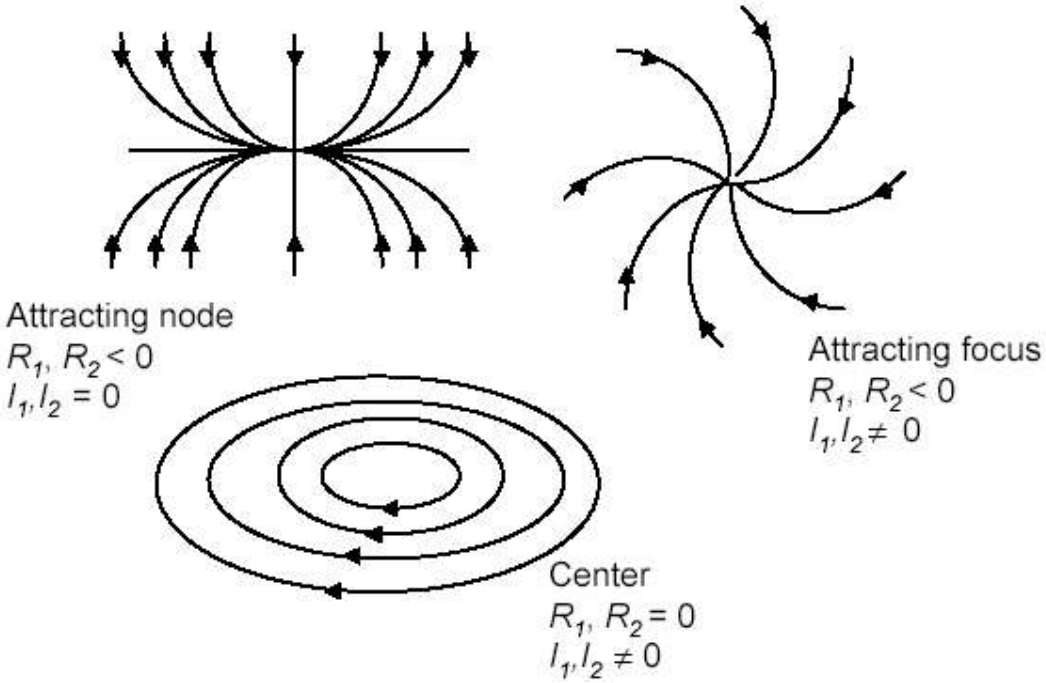
- Describes rotation of difference vector $d=(r-r_c)$
- The anti-symmetric part can be solved to give imaginary eigenvalues I :
  $J_a r_a=I r_a$, $I=I_1,I_2,I_3$

• 2D structure: eigenvalues are (R1, R2) and (I1,I2)

Classification criteria for critical points. $R1$ *and* $R2$ *denote the real part of the*

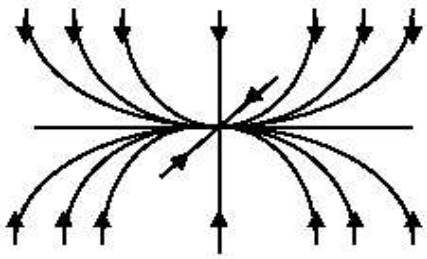


eigenvalues of the Jacobian, $I1$ and $I2$ the imaginary part.

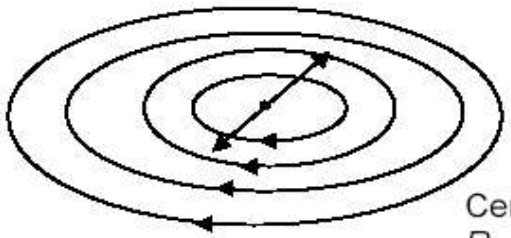


2D structure: eigenvalues are $(R1, R2)$ and $(I1, I2)$. If you have J and determine the eigenvectors, you have a node which repels or attracts stream lines → source or sink.

Some examples for the 3D case.



Attracting node
$R_1, R_2, R_3 < 0$
$I_1, I_2, I_3 = 0$



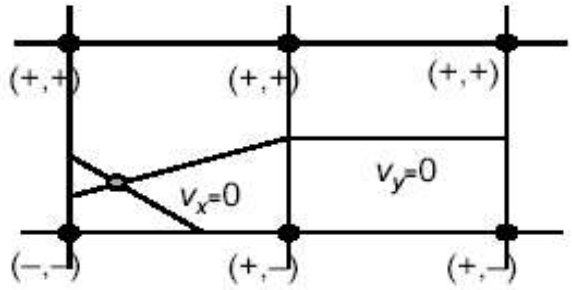
Center
$R_1, R_2 = 0, R_3 > 0$
$I_1, I_2 \neq 0, , I_3 = 0$

Mapping to graphical primitives: streamlines
- Find critical points
- Start streamlines close to critical points
- Initial direction along the eigenvectors

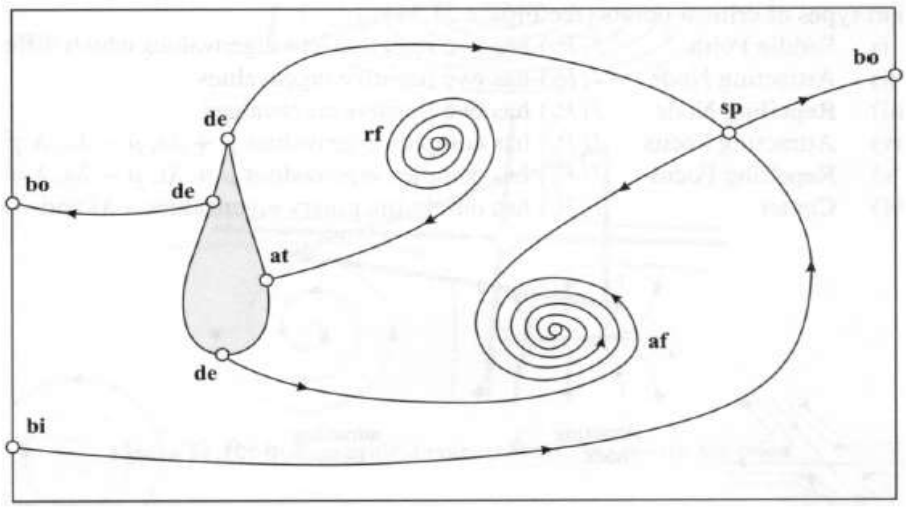
Trace particle line with numerical integration until you hit:
- Other “real” critical points
- Interior boundaries of an object: attachment or detachment points
- Boundaries of the computational domain

How to find critical points is very similar to Marching Cubes:
- Cell search (for cells which contain critical points):
    - Mark vertices by (+,+), (–, –), (+, –) or (–,+), depending on the signs of v_x and v_y
    - Determine cells that have vertices where the sign changes in both components → these are the cells that contain critical points
- How to find critical points within a (quad) cell ?
    - Find the critical points by interpolation
    - Determine the intersection of the isolines (c=0) of the two components,
    - Two bilinear equations to be solved
    - Critical points are the solutions within the cell boundaries
- How to find critical points within simplex?
    - Based on barycentric interpolation
    - Solve analytically
- Alternative method:
    - Iterative approach based on 2D / 3D nested intervals
    - Recursive subdivision into 4 / 8 subregions if critical point is contained in cell

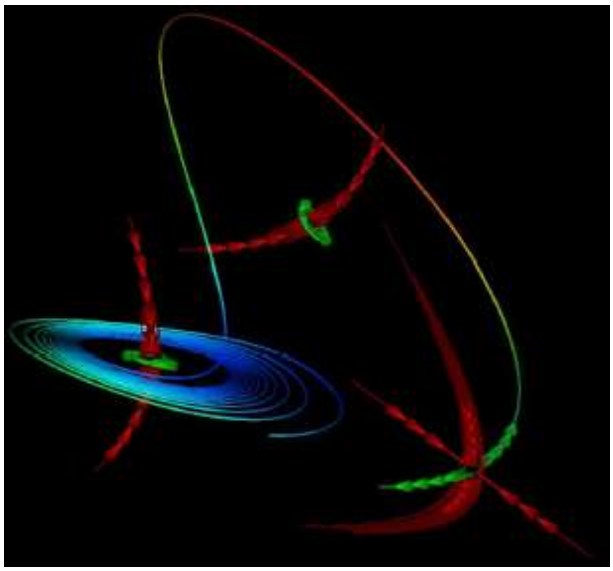
$(-, y)$ x *component may be 0 somewhere.*



$(x,-)$ y *component may be 0 somewhere.*

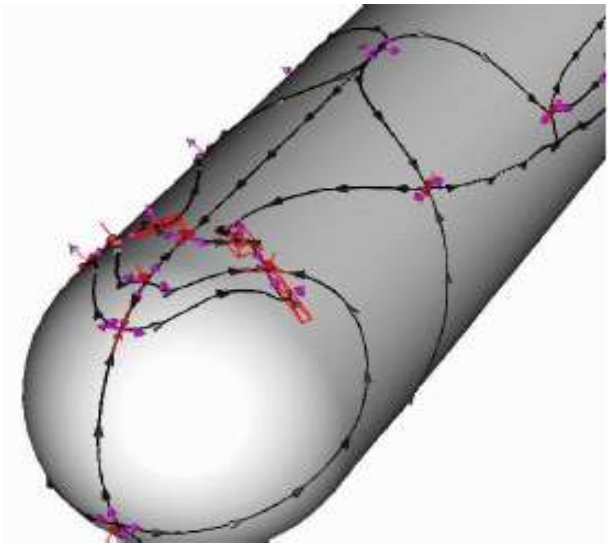


Example of a topological graph of 2D flow field.



Further examples of topology-guided streamline positioning.

Military dataset of a torpedo.

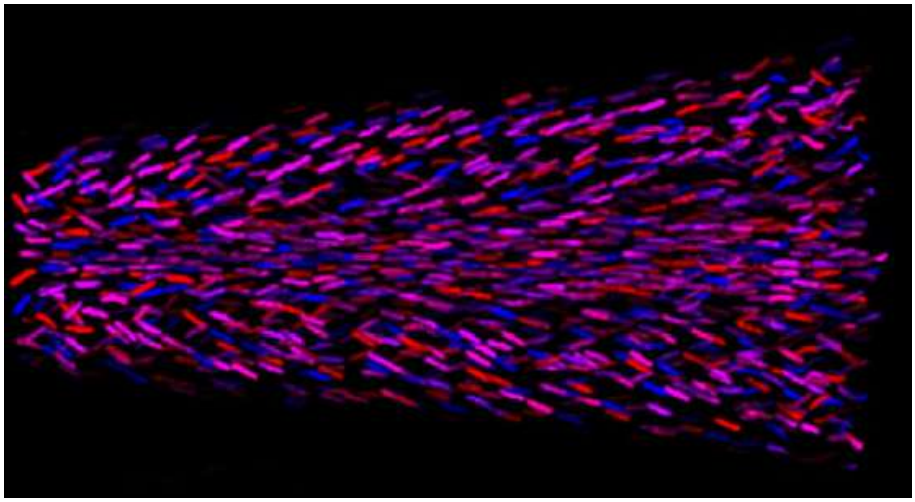


Summary:
- Draw only relevant streamlines (topological skeleton)
- Partition domain in regions with similar flow features
- Based on critical points
- Good for 2D stationary flows
- Instationary flows?
- 3D?

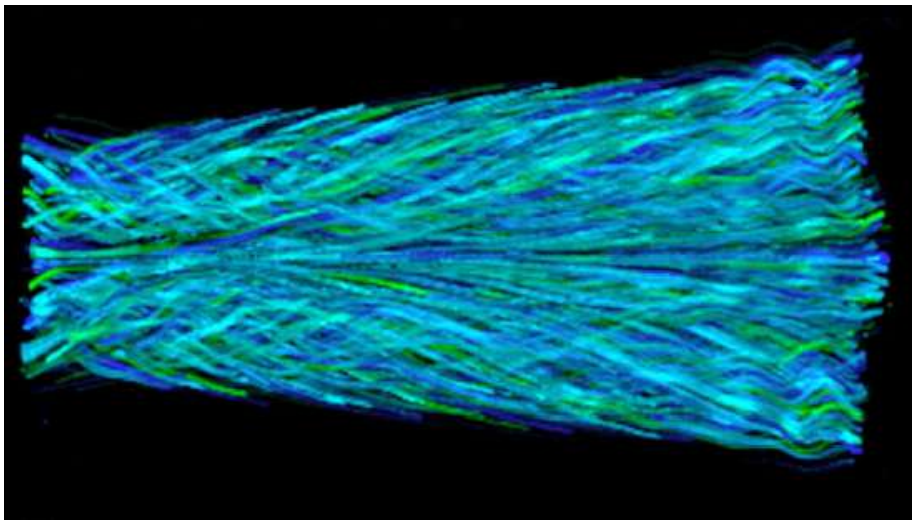
9 3D Vector Fields

Approaches to occlusion issue:
- Sparse representations
- Animation
- Color differences to distinguish separate objects
- Continuity

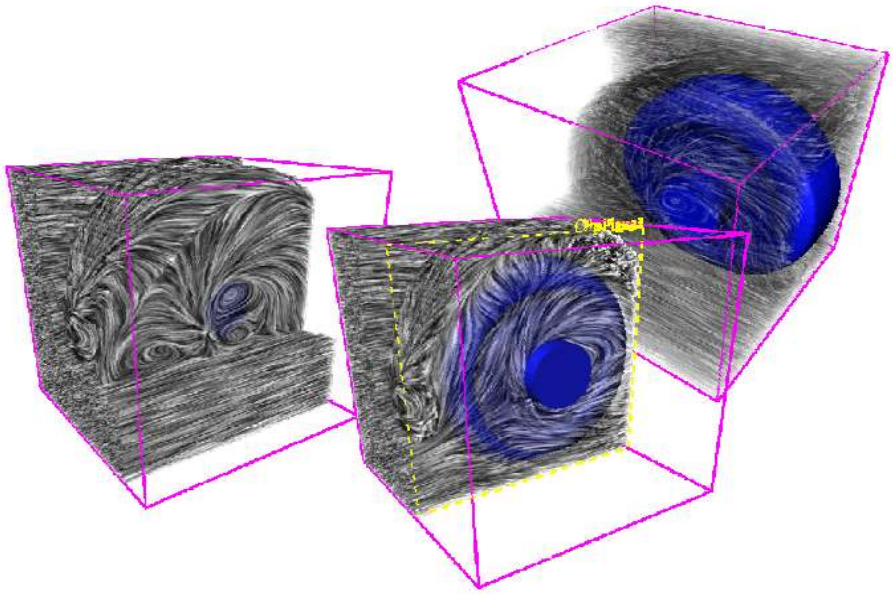
Reduction of visual data:
- Sparse representations
- Clipping
- Importance of semi-transparency



Missing continuity, hard to see the direction of flow.



Color differences to identify connected structures.

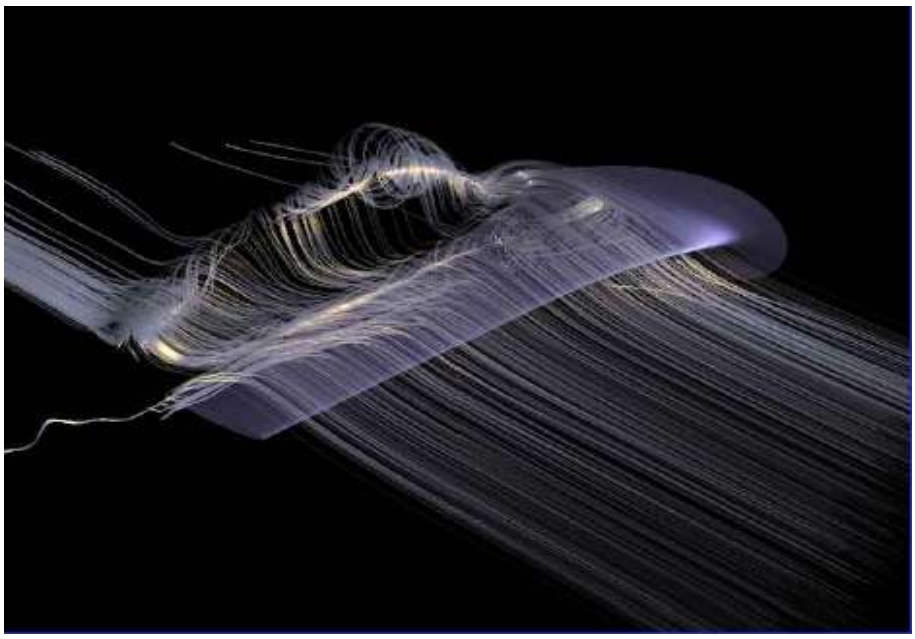


Reduction of visual data - 3D LIC. 3D flow around a wheel in a wheel housing, to see if the breaks get enough air cooling.

Improving spatial perception:

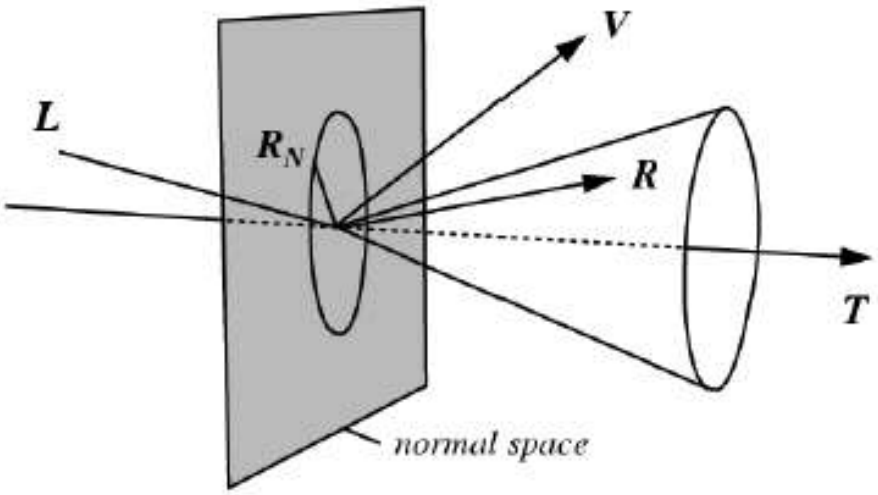
- Depth cues
    - Perspective
    - Occlusion
    - Motion parallax
    - Stereo disparity
    - Color (atmospheric, fogging)
- Halos
- Orientation of structures by shading (highlights)

Illumination of stream lines.



Illuminated streamlines *[Zoeckler-1996-ISL]*

- Model: streamline is made of thin cylinders
- Problem
    - No distinct normal vector on surface
    - Normal vector in plane perpendicular to tangent: normal space
    - Cone of reflection vectors

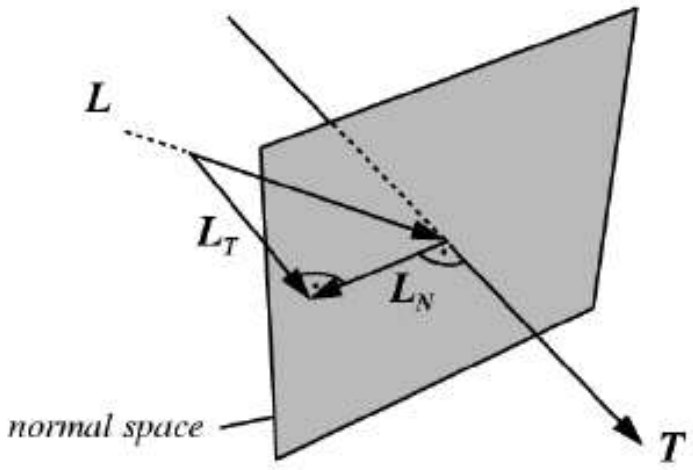


Normal space is perpendicular to the tangential part of the stream line.

- Light vector is split in tangential and normal parts

$$\begin{aligned} V\cdot R &= V\cdot(L_T - L_N) \\ &= V\cdot((L\cdot T)T - (L\cdot N)N) \\ &= (L\cdot T)(V\cdot T) - (L\cdot N)(V\cdot N) \\ &= (L\cdot T)(V\cdot T) - \sqrt{1-(L\cdot T)^2}\sqrt{1-(V\cdot T)^2} \\ &= f((L\cdot T),(V\cdot T)) \end{aligned}$$

- Idea: Represent $f()$ by 2D texture



Split the light vector in two parts:

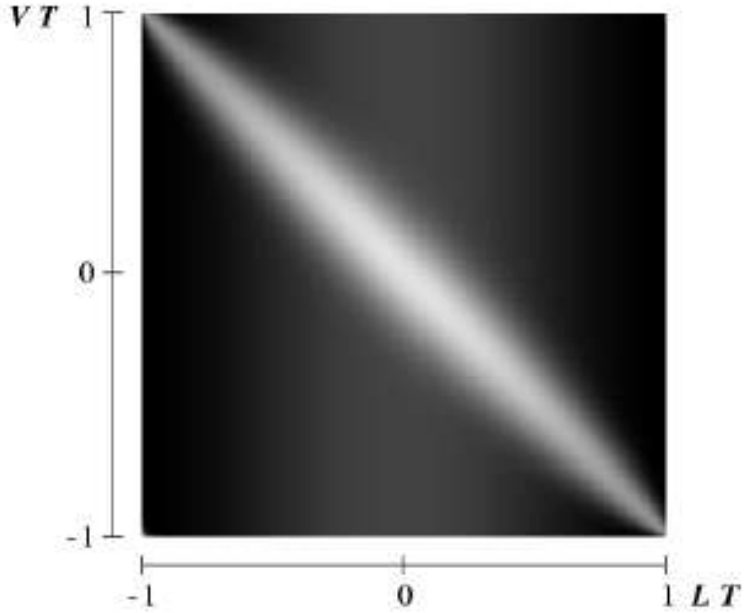
L_T *: Parallel to tangential.*

L_N *: Component of light vector in normal space with length =1.*

For illumination we can use the Phong model:

$I = I_{ambient} + I_{diffuse} + I_{specular} = k_a + k_d L \cdot N + k_s (V \cdot R)^n$

- Compute $L \cdot T$ and $V \cdot T$ per vertex (e.g. texture matrix or vertex program)
- Both $L \cdot N$ and $V \cdot R$ can be expressed in terms of $L \cdot T$ and $V \cdot T$
- Represent Phong model by 2D texture with texture coordinates $L \cdot T$ and $V \cdot T$



2D texture of the phong model.

Halos:

Flow visualization without and with halos.