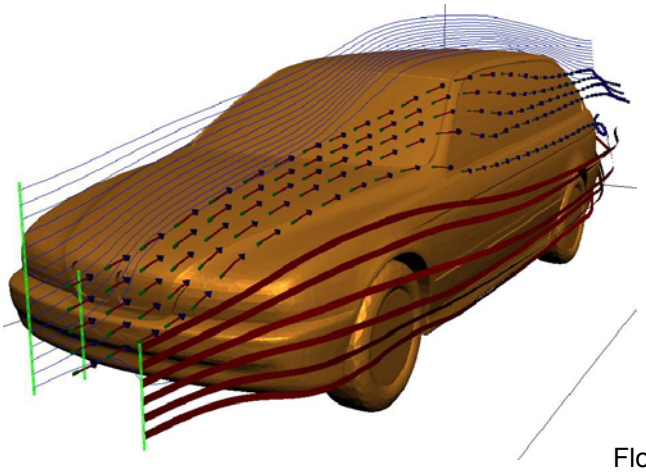
7. Vector Field Visualization

- Vector data set
- Represent direction and magnitude
- Given by a n-tupel $(f_1,...,f_n)$ with $f_k=f_k(x_1,...,x_n)$, $n \geq 2$ and $1 \leq k \leq n$
- Specific transformation properties
- Typically $n = k = 2$ or $n = k = 3$

7. Vector Field Visualization

- Main application of vector field visualization is flow visualization
  - Motion of fluids (gas, liquids)
  - Geometric boundary condititions
  - Velocity (flow) field $\boldsymbol{v}(\boldsymbol{x},t)$
  - Pressure p
  - Temperature T
  - Vorticity $\nabla \times \boldsymbol{v}$
  - Density ρ
  - Conservation of mass, energy, and momentum
  - Navier-Stokes equations
  - CFD (Computational Fluid Dynamics)

7. Vector Field Visualization



Flow visualization
based on CFD data

7. Vector Field Visualization

- Flow visualization – classification
  - Dimension (2D or 3D)
  - Time-dependency: stationary (steady) vs. instationary (unsteady)
  - Grid type
  - Compressible vs. incompressible fluids
- In most cases numerical methods required for flow visualization

7.1. 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

7.1. Vector Calculus

- Scalar function $\rho(\boldsymbol{x},t)$
- Gradient $$\nabla\rho(\boldsymbol{x},t) = \begin{pmatrix} \frac{\partial}{\partial x}\rho(\boldsymbol{x},t) \\ \frac{\partial}{\partial y}\rho(\boldsymbol{x},t) \\ \frac{\partial}{\partial z}\rho(\boldsymbol{x},t) \end{pmatrix} = \begin{pmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} \end{pmatrix} \rho(\boldsymbol{x},t)$$
- Gradient points into direction of maximum change of $\rho(\boldsymbol{x},t)$
- Laplace $$\Delta\rho(\boldsymbol{x},t) = \nabla \bullet \nabla\rho(\boldsymbol{x},t)$$ $$= \frac{\partial^2}{\partial x^2}\rho(\boldsymbol{x},t) + \frac{\partial^2}{\partial y^2}\rho(\boldsymbol{x},t) + \frac{\partial^2}{\partial z^2}\rho(\boldsymbol{x},t)$$

7.1. Vector Calculus

- Vector function ***v(x,t)***
- Jacobi matrix ("Gradient tensor")

$$\mathbf{J} = \nabla \boldsymbol{v}(\boldsymbol{x},t) = \begin{pmatrix} \frac{\partial}{\partial x}\boldsymbol{v}_x & \frac{\partial}{\partial y}\boldsymbol{v}_x & \frac{\partial}{\partial z}\boldsymbol{v}_x \\ \frac{\partial}{\partial x}\boldsymbol{v}_y & \frac{\partial}{\partial y}\boldsymbol{v}_y & \frac{\partial}{\partial z}\boldsymbol{v}_y \\ \frac{\partial}{\partial x}\boldsymbol{v}_z & \frac{\partial}{\partial y}\boldsymbol{v}_z & \frac{\partial}{\partial z}\boldsymbol{v}_z \end{pmatrix}$$

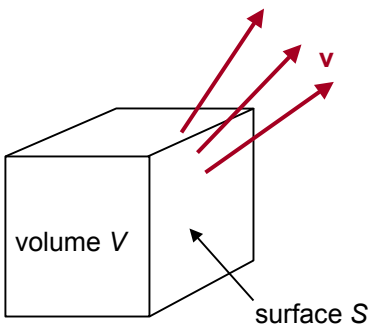
- Divergence

$$\text{div } \boldsymbol{v}(\boldsymbol{x},t) = \nabla \cdot \boldsymbol{v}(\boldsymbol{x},t) = \frac{\partial}{\partial x}\boldsymbol{v}_x(\boldsymbol{x},t) + \frac{\partial}{\partial y}\boldsymbol{v}_y(\boldsymbol{x},t) + \frac{\partial}{\partial z}\boldsymbol{v}_z(\boldsymbol{x},t)$$

7.1. Vector Calculus

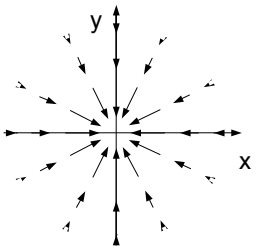
- Gauss theorem (divergence theorem)

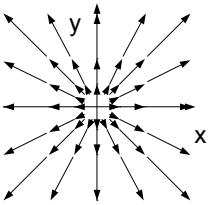
$$\int_V \nabla \cdot \mathbf{v} dV = \oint_S \mathbf{v} \cdot d\mathbf{A}$$



7.1. Vector Calculus

- Properties of divergence:
  - div **v** is a scalar
  - div $\mathbf{v}(\mathbf{x}_0) > 0$: **v** has a source in $\mathbf{x}_0$
  - div $\mathbf{v}(\mathbf{x}_0) < 0$: **v** has a sink in $\mathbf{x}_0$
  - div $\mathbf{v}(\mathbf{x}_0) = 0$: **v** is source-free in $\mathbf{x}_0$
  - Describes flow into/out of a region





7.1. Vector Calculus

- Continuity equation
  - Flow of mass into a volume *V* with surface *S*

$$-\oint_S \rho \mathbf{v} \cdot d\mathbf{A}$$

  - Change of mass inside the volume

$$\frac{\partial}{\partial t}\int_V \rho dV = \int_V \frac{\partial \rho}{\partial t} dV$$

  - Conservation of mass

$$\int_V \frac{\partial \rho}{\partial t} dV = -\oint_S \rho \mathbf{v} \cdot d\mathbf{A}$$

7.1. Vector Calculus

- Continuity equation *(cont.)*
  - Application of Gauss theorem

$$\int_V \frac{\partial \rho}{\partial t} dV + \oint_S \rho \mathbf{v} \cdot d\mathbf{A} = \int_V \left(\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) \right) dV = 0$$

  - Above equation must be met for any volume element
  - Yields

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

continuity equation
in differential form

current $\mathbf{j} = \rho \mathbf{v}$

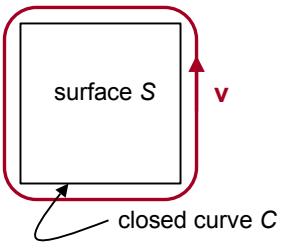
7.1. Vector Calculus

- Curl

$$\text{curl}\, \boldsymbol{v}(\boldsymbol{x},t) = \nabla \times \boldsymbol{v}(\boldsymbol{x},t) = \begin{pmatrix} \frac{\partial}{\partial y}\boldsymbol{v}_z(\boldsymbol{x},t) - \frac{\partial}{\partial z}\boldsymbol{v}_y(\boldsymbol{x},t) \\ \frac{\partial}{\partial z}\boldsymbol{v}_x(\boldsymbol{x},t) - \frac{\partial}{\partial x}\boldsymbol{v}_z(\boldsymbol{x},t) \\ \frac{\partial}{\partial x}\boldsymbol{v}_y(\boldsymbol{x},t) - \frac{\partial}{\partial y}\boldsymbol{v}_x(\boldsymbol{x},t) \end{pmatrix}$$

- Stokes theorem

$$\int_S \nabla \times \mathbf{v} \cdot d\mathbf{A} = \oint_C \mathbf{v} \cdot d\mathbf{s}$$



7.1. Vector Calculus

- Properties of curl:
  - Describes vortex characteristics in the flow
  - From non-curl free flow

$$\nabla \times \mathbf{v} \neq \boldsymbol{0}$$

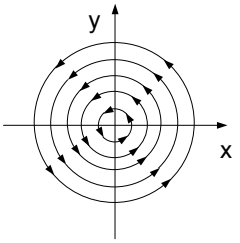
can follow that

$$\int_S \nabla \times \mathbf{v} \cdot d\mathbf{A} \neq 0$$

and

$$\oint_C \mathbf{v} \cdot d\mathbf{s} \neq 0$$

  - Vorticity $\nabla \times \mathbf{v}$



7.1. Vector Calculus

- Decomposition of a vector field (Helmholtz theorem)
  - Divergence-free (transversal) part
  - Curl-free (longitudinal) part
  - Decomposition:

$$\mathbf{v}(\mathbf{x}) = \mathbf{v}_c(\mathbf{x}) + \mathbf{v}_d(\mathbf{x})$$

with $\nabla \times \mathbf{v}_c(\mathbf{x}) = 0$

and $\nabla \bullet \mathbf{v}_d(\mathbf{x}) = 0$

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

7.1. Vector Calculus

- Decomposition of a vector field *(cont.)*
  - Suppose we have a representation of **v** by the Poisson equation

$$\mathbf{v}(\mathbf{x}) = -\Delta\mathbf{z}(\mathbf{x})$$

  - Curl-free part written as a gradient:

$$\mathbf{v}_c(\mathbf{x}) = -\nabla u(\mathbf{x})$$

with $u(\mathbf{x}) = \nabla \bullet \mathbf{z}(\mathbf{x})$ (scalar) potential u

  - Divergence-free part written as a curl

$$\mathbf{v}_d(\mathbf{x}) = \nabla \times \mathbf{w}(\mathbf{x})$$

with $\mathbf{w}(\mathbf{x}) = \nabla \times \mathbf{z}(\mathbf{x})$ vector potential **w**

7.2. Characteristic Lines

- Types of characteristic lines in a vector field:
  - 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

7.2. Characteristic Lines

- Stream lines
  - Tangential to the vector field
  - Vector field at an arbitrary, yet fixed time *t*
  - Stream line is a solution to the initial value problem of an ordinary differential equation:

$$\boldsymbol{L}(0) = \boldsymbol{x}_0 \quad , \quad \frac{d\boldsymbol{L}(u)}{du} = \boldsymbol{v}(\boldsymbol{L}(u), t)$$

initial value
(seed point $\mathbf{x}_0$)

ordinary differential equation

  - Stream line is curve $\mathbf{L}(u)$ with the parameter u

7.2. Characteristic Lines

- Path lines
  - Trajectories of massless particles in the flow
  - Vector field can be time-dependent (unsteady)
  - Path line is a solution to the initial value problem of an ordinary differential equation:

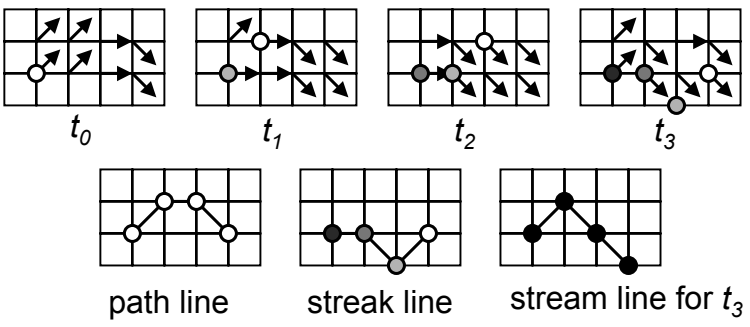
$$\boldsymbol{L}(0) = \boldsymbol{x}_0 \quad , \quad \frac{d\boldsymbol{L}(u)}{du} = \boldsymbol{v}(\boldsymbol{L}(u), u)$$

7.2. Characteristic Lines

- Streak lines
  - Trace of dye that is released into the flow at a fixed position
  - Connect all particles that passed through a certain position
- Time lines (time surfaces)
  - Propagation of a line (surface) of massless elements in time
  - Idea: “consists” of many point-like particles that are traced
  - Connect particles that were released simultaneously

7.2. Characteristic Lines

- Comparison of path lines, streak lines, and stream lines



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

7.2. Characteristic Lines

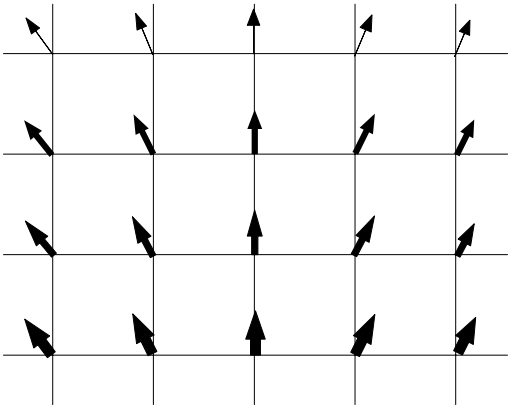
- Difference between Eulerian and Lagrangian point of view
- Lagrangian:
  - Individual particles
  - Can be identified
  - Attached are position, velocity, and other properties
  - Explicit position
  - Standard approach for particle tracing
- Eulerian:
  - No individual particles
  - Properties given on a grid
  - Position is implicit

7.3. Arrows and Glyphs

- Visualize **local** features of the vector field:
  - Vector itself
  - Vorticity
  - Extern data: temperature, pressure, etc.
- Important elements of a vector:
  - Direction
  - Magnitude
  - Not: components of a vector
- Approaches:
  - Arrow plots
  - Glyphs

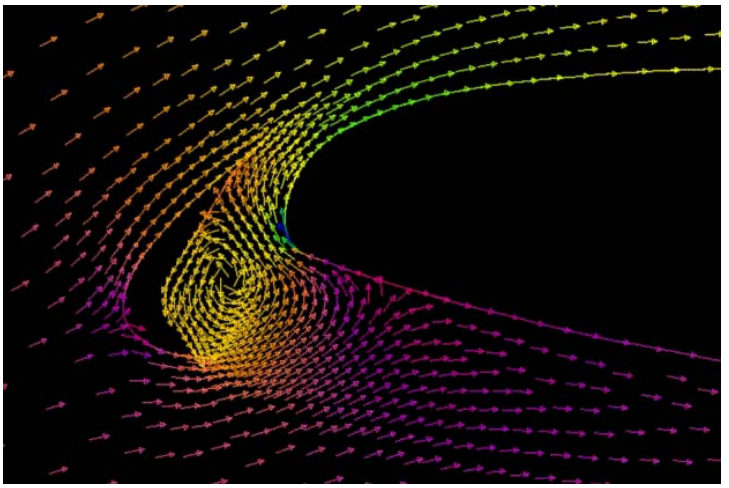
7.3. Arrows and Glyphs

- Arrows visualize
  - Direction of vector field
  - Orientation
  - Magnitude:
    - Length of arrows
    - Color coding



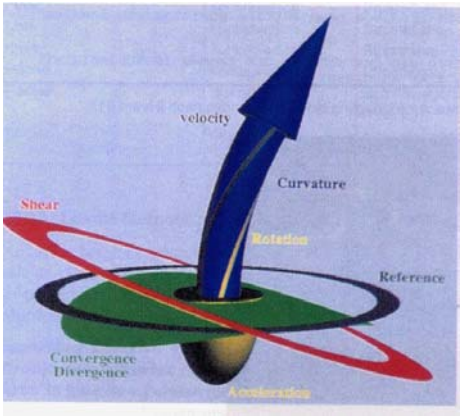
7.3. Arrows and Glyphs

- Arrows



7.3. Arrows and Glyphs

- Glyphs
  - Can visualize more features of the vector field (flow field)



7.3. Arrows and Glyphs

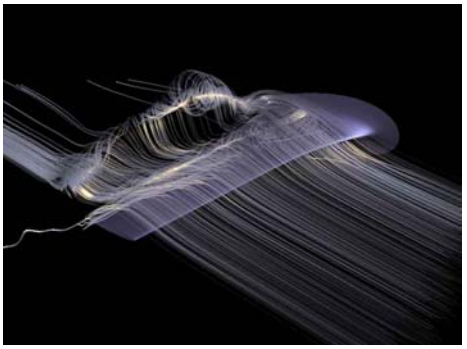
- Advantages and disadvantages of glyphs and arrows:
  + Simple
  + 3D effects
  - Heavy load in the graphics subsystem
  - Inherent occlusion effects
  - Poor results if magnitude of velocity changes rapidly
    (Use arrows of constant length and color code magnitude)

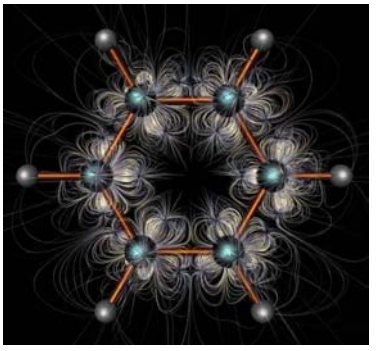
7.4. Mapping Methods Based on Particle Tracing

- Basic idea: trace particles
- Characteristic lines
- **Global** methods
- Mapping approaches:
  - Lines
  - Surfaces
  - Individual particles
  - Sometimes animated

7.4. Mapping Methods Based on Particle Tracing

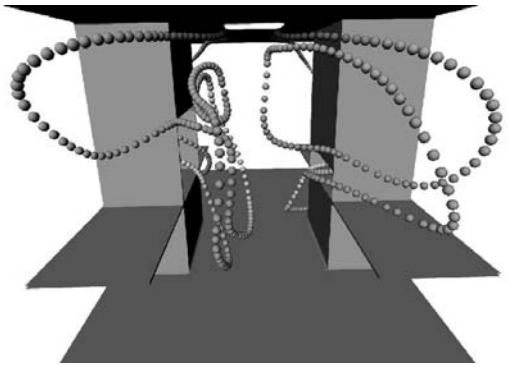
- Path lines





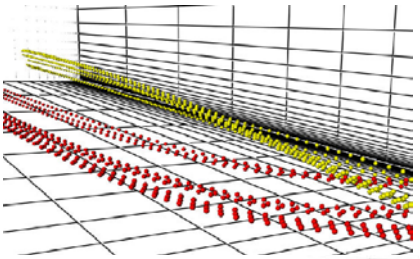
7.4. Mapping Methods Based on Particle Tracing

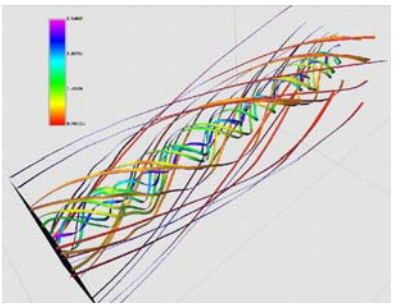
- Stream balls
  - Encode additional scalar value by radius

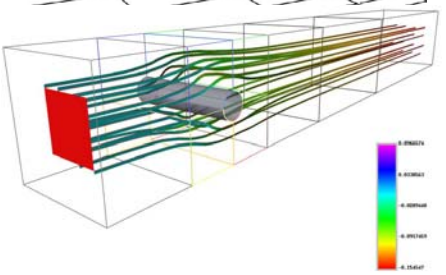


7.4. Mapping Methods Based on Particle Tracing

- Streak lines

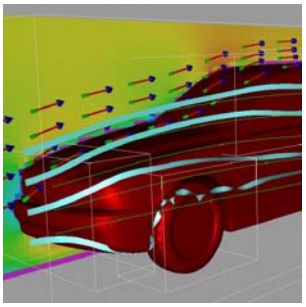


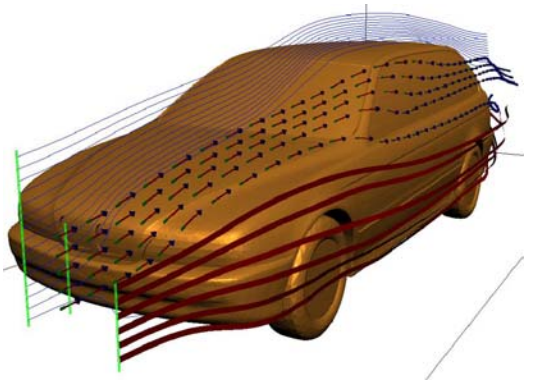




7.4. Mapping Methods Based on Particle Tracing

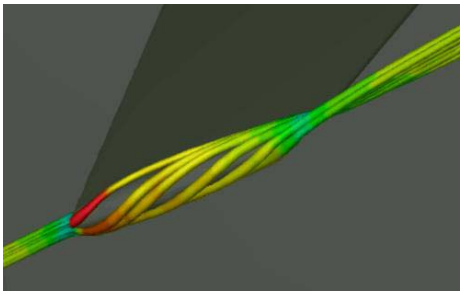
- Stream ribbons
  - Trace two close-by particles
  - Keep distance constant

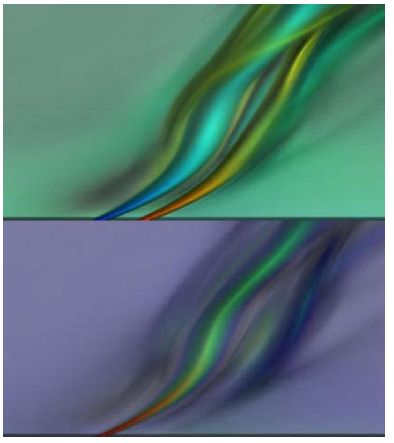




7.4. Mapping Methods Based on Particle Tracing

- Stream tubes
  - Specify contour, e.g. triangle or circle, and trace it through the flow



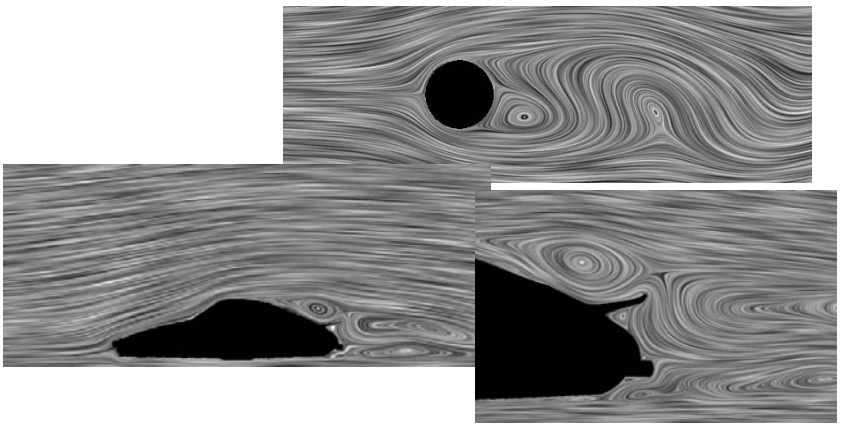


7.4. Mapping Methods Based on Particle Tracing

- Motion of individual particles

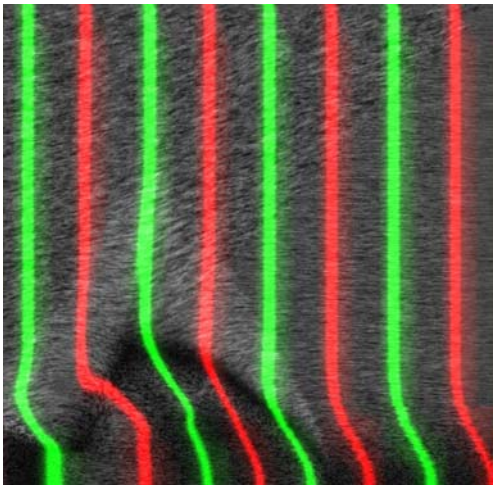
7.4. Mapping Methods Based on Particle Tracing

- LIC (Line Integral Convolution)



7.4. Mapping Methods Based on Particle Tracing

- Texture advection
  - Animation



7.5. Numerical Integration of Ordinary Differential Equations

- Typical example of particle tracing problem (path line):

$$\boldsymbol{L}(0) = \boldsymbol{x}_0 \quad , \quad \frac{d\boldsymbol{L}(u)}{du} = \boldsymbol{v}(\boldsymbol{L}(u), u)$$

- Initial value problem for ordinary differential equations (ODE)
- What kind of numerical solver?

7.5. Numerical Integration of Ordinary Differential Equations

- Rewrite ODE in generic form
- Initial value problem for:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}, t)$$

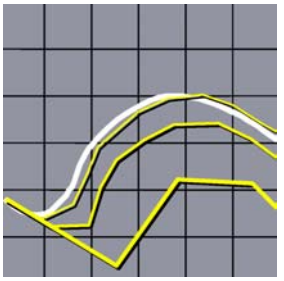
- Most simple (naive) approach: Euler method

$$\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + \Delta t\, \mathbf{f}(\mathbf{x}, t)$$

- Based on Taylor expansion

$$\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + \Delta t\, \dot{\mathbf{x}}(t) + O(\Delta t^2)$$

- First order method
- Higher accuracy with smaller step size

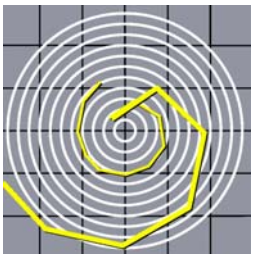


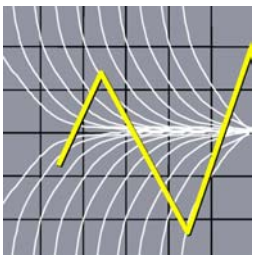
7.5. Numerical Integration of Ordinary Differential Equations

- Problem of Euler method
  - Inaccurate
  - Unstable
- Example:

$f = -kx$

$x = e^{-kt}$ divergence for $\Delta t > 2/k$





7.5. Numerical Integration of Ordinary Differential Equations

- Midpoint method

  1. Euler step

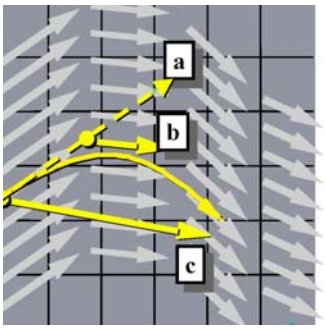
  $$\Delta\mathbf{x} = \Delta t\, \mathbf{f}(\mathbf{x}, t)$$

  2. Evaluation of **f** at midpoint

  $$\mathbf{f}_{\text{mid}} = \mathbf{f}\left(\mathbf{x} + \frac{\Delta\mathbf{x}}{2}, t + \frac{\Delta t}{2}\right)$$

  3. Complete step with value at midpoint

  $$\mathbf{x}(t + \Delta t) = \Delta t\, \mathbf{f}_{\text{mid}}$$



- Method of second order

7.5. Numerical Integration of Ordinary Differential Equations

- Runge-Kutta of fourth order

$$\mathbf{k}_1 = \Delta t\, \mathbf{f}(\mathbf{x}, t)$$

$$\mathbf{k}_2 = \Delta t\, \mathbf{f}\left(\mathbf{x} + \frac{\mathbf{k}_1}{2}, t + \frac{\Delta t}{2}\right)$$

$$\mathbf{k}_3 = \Delta t\, \mathbf{f}\left(\mathbf{x} + \frac{\mathbf{k}_2}{2}, t + \frac{\Delta t}{2}\right)$$

$$\mathbf{k}_4 = \Delta t\, \mathbf{f}\left(\mathbf{x} + \mathbf{k}_3, t + \Delta t\right)$$

$$\mathbf{x}(t + \Delta t) = \mathbf{x} + \frac{\mathbf{k}_1}{6} + \frac{\mathbf{k}_2}{3} + \frac{\mathbf{k}_3}{3} + \frac{\mathbf{k}_4}{6} + O\left(\Delta t^5\right)$$

7.5. Numerical Integration of Ordinary Differential Equations

- Adaptive stepsize control
  - Change step size according to the error
  - Decrease/increase step size depending on whether the actual local error is high/low
  - Higher integration speed in “simple” regions
  - Good error control
- Approaches:
  - Stepsize doubling
  - Embedded Runge-Kutta schemes

- Further reading:
  - WH Press, SA Teukolsky, WT Vetterling, BP Flannery: *Numerical Recipes*

7.5. Numerical Integration of Ordinary Differential Equations

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

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

  - Indication of truncation error: $\Delta = \|x_2 - x_1\|$

  - 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:

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

7.5. Numerical Integration of Ordinary Differential Equations

- Embedded Runge-Kutta
  - Originally invented by Fehlberg
  - Comparison between results from two steps taken with schemes of different order

$$\Delta = \left\| x - x^* \right\|$$

  - Typical example: x is computed with fifth order, x^* with fourth order RK
  - Main idea: 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!

7.5. Numerical Integration of Ordinary Differential Equations

- Embedded Runge-Kutta (*cont.*)
  - Practical choice for new stepsize for 4th / 5th embedded RK

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

  - Choice for safety factor, e.g., $\rho = 0.9$

7.5. Numerical Integration of Ordinary Differential Equations

- So far only explicit methods
- Stability problem can be solved by implicit methods

- Implicit Euler method

$$\mathbf{x}(t+\Delta t)-\mathbf{x}(t)=\Delta t\, f\big(\mathbf{x}(t+\Delta t),t+\Delta t\big)$$

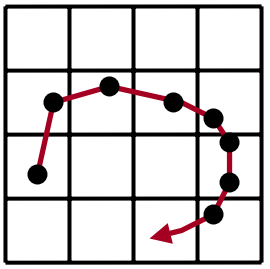
- „Reversing“ the explicit Euler integration step
- Taylor expansion around $t + \Delta t$ instead of t
- Solving the system of non-linear equations to determine $\mathbf{x}(t + \Delta t)$
- Using implicit methods allows larger time steps

7.6. Particle Tracing on Grids

- Vector field given on a grid
- Solve

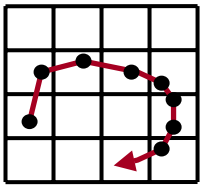
$$\mathbf{L}(0)=\mathbf{x}_0 \quad , \quad \frac{d\mathbf{L}(t)}{dt}=\mathbf{v}\big(\mathbf{L}(t),t\big)$$

  for the path line
- Incremental integration
- Discretized path of the particle



7.6. Particle Tracing on Grids

- Most simple case: Cartesian grid for the path line



- Basic algorithm:

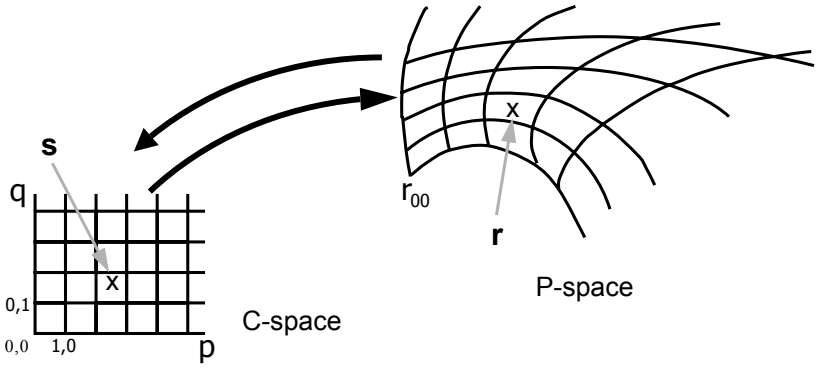
```
Select start point (seed point)
Find cell that contains start point                    point location
While (particle in domain) do
    Interpolate vector field at
      current position                                 interpolation
    Integrate to new position                          integration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

7.6. Particle Tracing on Grids

- Point location (cell search) on Cartesian grids:
  - Indices of cell directly from position (x, y, z)
  - For example: $i_x = (x - x_0) / \Delta x$
  - 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)

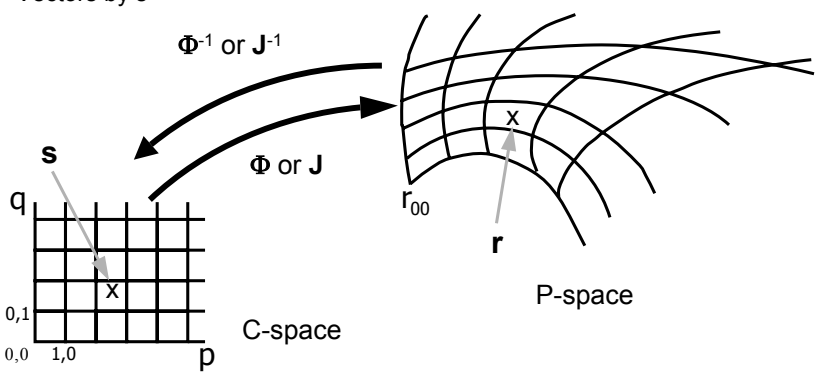
7.6. Particle Tracing on Grids

- How are curvilinear grids handled?
- C-space (computational space) vs. P-space (physical space)



7.6. Particle Tracing on Grids

- Particle tracing can either be done in C-space or P-space
- Transformation of
  - Points by Φ
  - Vectors by **J**



7.6. Particle Tracing on Grids

- Transformation of points:
  - From C-space to P-space: $\mathbf{r} = \Phi(\mathbf{s})$
  - From P-space to C-space: $\mathbf{s} = \Phi^{-1}(\mathbf{r})$
- Transformation of vectors:
  - From C-space to P-space: $\mathbf{v} = \mathbf{J} \cdot \mathbf{u}$
  - From P-space to C-space: $\mathbf{u} = \mathbf{J}^{-1} \cdot \mathbf{v}$
  - $\mathbf{J}$ is Jacobi matrix:

$$\mathbf{J} = \begin{pmatrix} \frac{\partial \Phi_x}{\partial p} & \frac{\partial \Phi_x}{\partial q} \\ \frac{\partial \Phi_y}{\partial p} & \frac{\partial \Phi_y}{\partial q} \end{pmatrix}$$

(2D case)

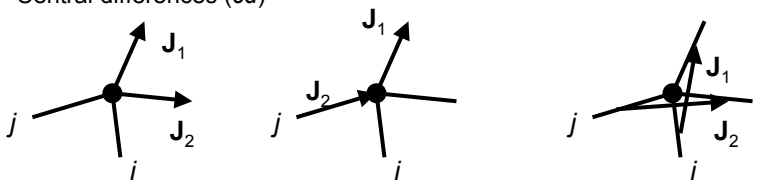
7.6. Particle Tracing on Grids

- Particle tracing in C-space
- Algorithm

```
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)
    Interpolate u in C-space                        interpolation
    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    (C → P)
    Draw line segment between latest
      particle positions
Endwhile
```

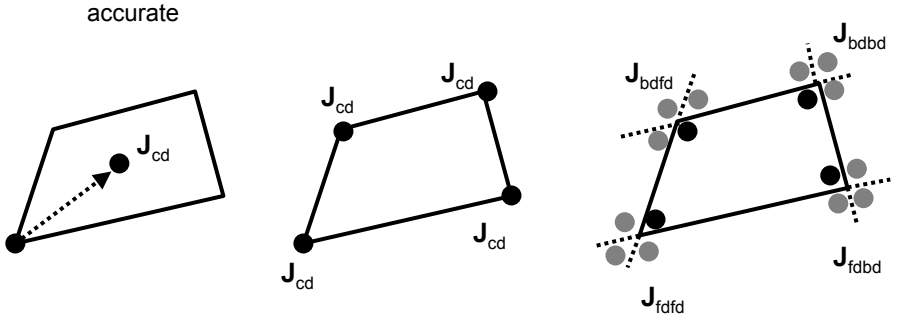
7.6. Particle Tracing on Grids

- Transformation of points from C-space to P-space: $\mathbf{r} = \Phi(\mathbf{s})$:
  - Bilinear (in 2D) or trilinear (in 3D) interpolation between coordinates of the cell's vertices
- Transformation of vectors from P-space to C-space: $\mathbf{u} = \mathbf{J}^{-1} \cdot \mathbf{v}$
  - Needs inverse of the Jacobian
- Numerical computation of elements of the Jacobi matrix:
  - Backward differences (*bd*)
  - Forward differences (*fd*)
  - Central differences (*cd*)



7.6. Particle Tracing on Grids

- 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 bi / trilinear interpolation, accurate



7.6. Particle Tracing on Grids

- Particle tracing in P-Space
- Algorithm

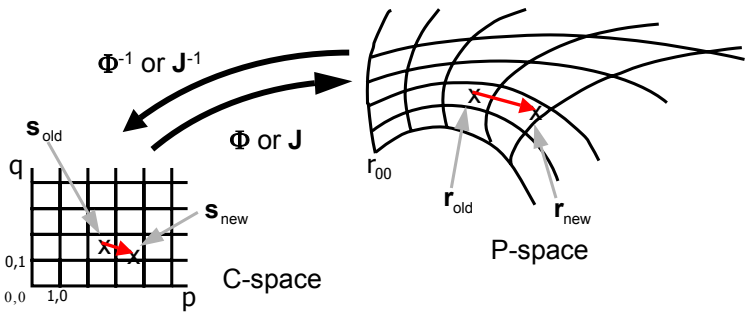
```
Select start point r in P-space (seed point)
Find P-space cell and local coords                  point location
While (particle in domain) do
    Interpolate v in P-space                        interpolation
    Integrate to new position r in P-space          integration
    Find new position   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

7.6. Particle Tracing on Grids

- 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

7.6. Particle Tracing on Grids

- Stencil-walk on curvilinear grids
  - Given: $\mathbf{r}_{old}$, $\mathbf{r}_{new}$, $\mathbf{s}_{old}$, $\Phi(\mathbf{s}_{old}) = \mathbf{r}_{old}$
  - Determine $\mathbf{s}_{new}$

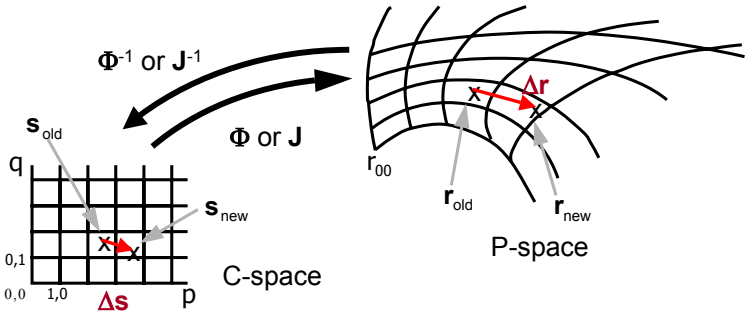


7.6. Particle Tracing on Grids

- Stencil-walk on curvilinear grids (*cont.*)
  - Transformation of distances $\Delta\mathbf{s} = \mathbf{s}_{new} - \mathbf{s}_{old}$ and $\Delta\mathbf{r} = \mathbf{r}_{new} - \mathbf{r}_{old}$
  - Taylor expansion:

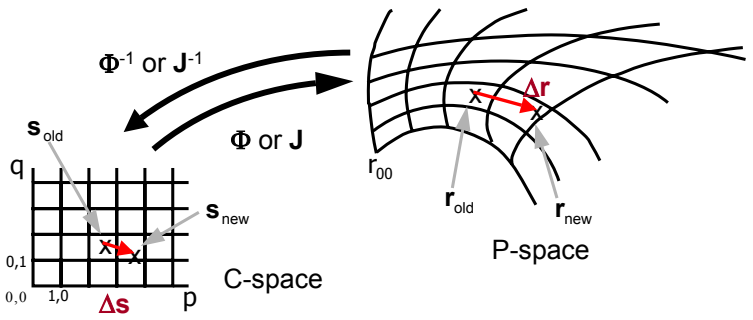
$$\Delta\mathbf{r} = \mathbf{r}_{new} - \mathbf{r}_{old} = \Phi(\mathbf{s}_{new}) - \Phi(\mathbf{s}_{old}) = \mathbf{J}_{\Phi}(\mathbf{s}_{old}) \cdot \Delta\mathbf{s} + \ldots$$

  - Therefore $\Delta\mathbf{s} \approx [\mathbf{J}_{\Phi}(\mathbf{s}_{old})]^{-1}\Delta\mathbf{r}$



7.6. Particle Tracing on Grids

- Idea of stencil-walk:
  - Guess start point in C-space
  - Compute difference between corresponding position and target position in P-space
  - Improve position in C-space
  - Iterate



7.6. Particle Tracing on Grids

- 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 = Φ(s)
    Difference in P-space Δr = r_target − r
    Transformation to C-space Δs = J^-1 Δr
    If (|Δs| < ε) then exit
    s = s + Δs
    If (s outside current cell) then
      Set s = midpoint of corresponding neighboring cell
    Endif
Repeat
```

7.6. Particle Tracing on Grids

- High convergence speed of stencil walk
  - Typically 3-5 iteration steps in a cell
- Interpolation of velocity in P-space approach:
  - Bi / trilinear within cell, based on local coordinates **s**
  - Alternative method: inverse distance weighting → no stencil walk needed

7.6. Particle Tracing on Grids

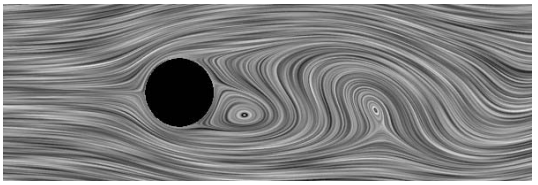
- Important properties of C-space integration:
  + Simple incremental cell search
  + Simple interpolation
  - Complicated transformation of velocities / vectors
- Important properties of P-space integration:
  + No transformation of velocities / vectors
  - Complicated point location (stencil walk) for bi / trilinear interpolation

7.6. Particle Tracing on Grids

- 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

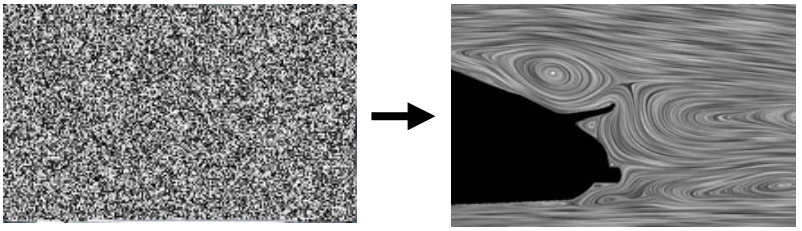
7.7. Line Integral Convolution

- Line Integral Convolution (LIC)
  - Visualize dense flow fields by imaging its integral curves
  - Cover domain with a random texture (so called ‚input texture', usually stationary white noise)
  - Blur (convolve) the input texture along the path lines using a specified filter kernel
- Look of 2D LIC images
  - Intensity distribution along path lines shows high correlation
  - No correlation between neighboring path lines



7.7. Line Integral Convolution

- Idea of Line Integral Convolution (LIC)
  - Global visualization technique
  - Start with random texture
  - Smear out along stream lines



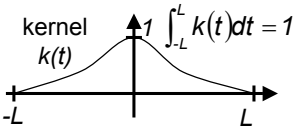
7.7. Line Integral Convolution

- Algorithm for 2D LIC
  - Let $t \rightarrow \Phi_0(t)$ 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:

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

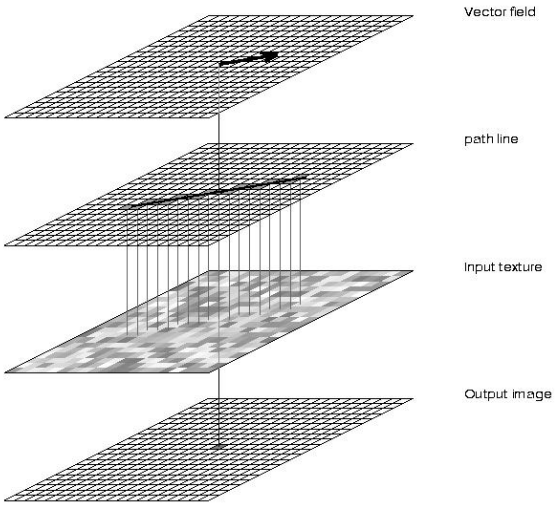
convolution with kernel

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

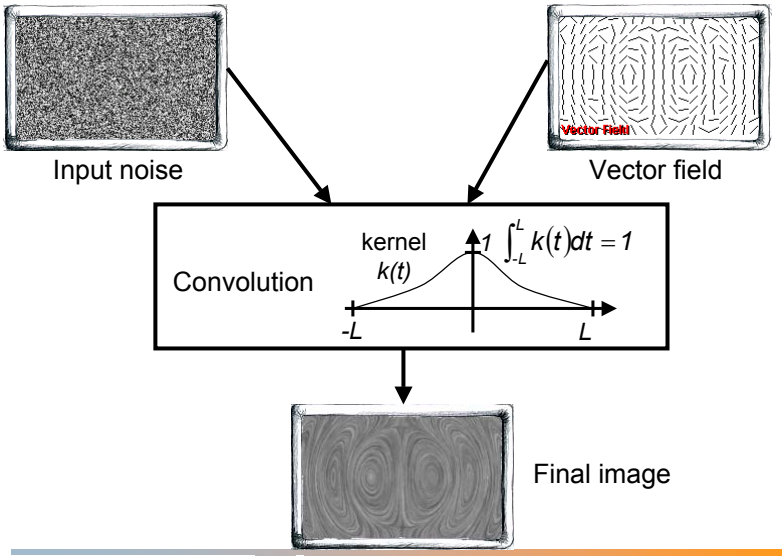


7.7. Line Integral Convolution

- Algorithm for 2D LIC
  - Convolve a random texture along the stream lines



7.7. Line Integral Convolution

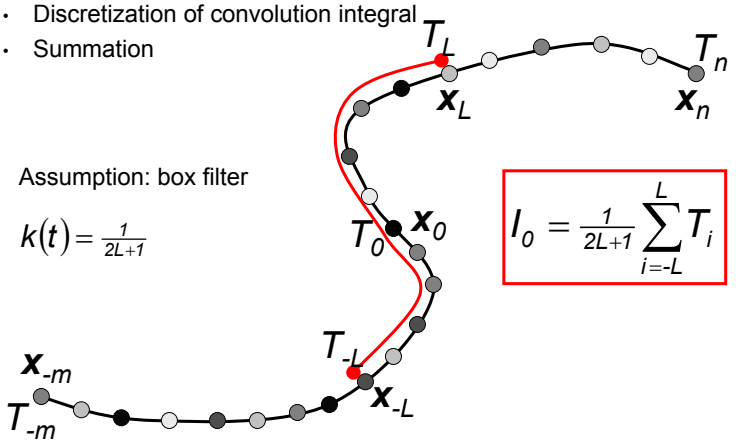


7.7. Line Integral Convolution

- Fast LIC
- Problems with LIC
  - New stream line is computed at each pixel
  - Convolution (integral) is computed at each pixel
  - Slow
- Idea:
  - Compute very long stream lines
  - Reuse these stream lines for many different pixels
  - Incremental computation of the convolution integral

7.7. Line Integral Convolution

- Fast LIC: incremental integration
  - Discretization of convolution integral
  - Summation



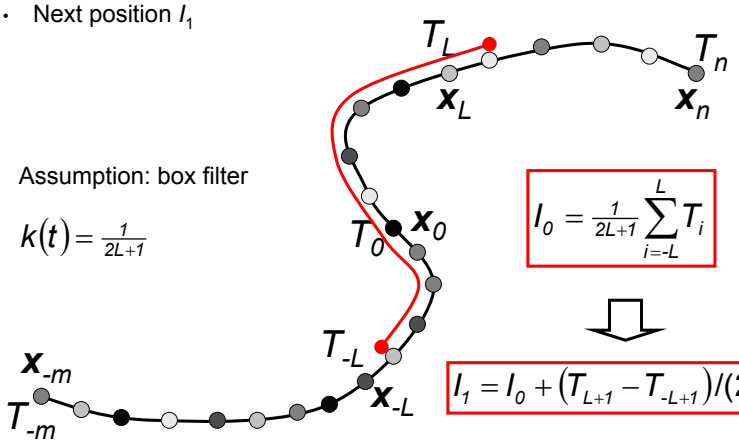
Assumption: box filter

$$k(t) = \frac{1}{2L+1}$$

$$I_0 = \frac{1}{2L+1} \sum_{i=-L}^{L} T_i$$

7.7. Line Integral Convolution

- Fast LIC: incremental integration
  - Next position I_1



Assumption: box filter

$$k(t) = \frac{1}{2L+1}$$

$$I_0 = \frac{1}{2L+1}\sum_{i=-L}^{L} T_i$$

$$I_1 = I_0 + (T_{L+1} - T_{-L+1})/(2L+1)$$

7.7. Line Integral Convolution

- Fast LIC: incremental integration for constant kernel
  - Stream line $\boldsymbol{x}_{-m},\ldots, \boldsymbol{x}_0,\ldots, \boldsymbol{x}_n$ 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:

$$I_{j+1} - I_j = \frac{1}{2L+1}\sum_{i=-L}^{L}\left(T_{i+j+1} - T_{i+j}\right) = \frac{1}{2L+1}\left(T_{L+j+1} - T_{-L+j}\right)$$

7.7. Line Integral Convolution

- Fast LIC: incremental integration for polynomial kernels
  - Assumption: polynomial kernel (monom representation)

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

  - Value

$$I_j = \sum_{p=0}^{d} \alpha_p \cdot I_j^p$$

with $$I_j^p = \sum_{i=-L}^{L} T_{j+i} \cdot i^p$$

7.7. Line Integral Convolution

- Fast LIC: incremental integration for polynomial kernels
  - Incremental update for I_j^p

$$\begin{aligned} I_{j+1}^p - I_j^p &= \sum_{i=-L}^{L} \left(T_{j+1+i} - T_{j+i}\right) \cdot i^p \\ &= \sum_{i=-L}^{L} T_{j+i} \cdot \left((i-1)^p - i^p\right) \overbrace{+ T_{j+L+1} \cdot L^p - T_{j-L} \cdot (-L-1)^p}^{\Lambda_{j+1}^p} \\ &= \sum_{q=0}^{p-1} \binom{p}{q} (-1)^{p-q} I_j^q + \Lambda_{j+1}^p \end{aligned}$$

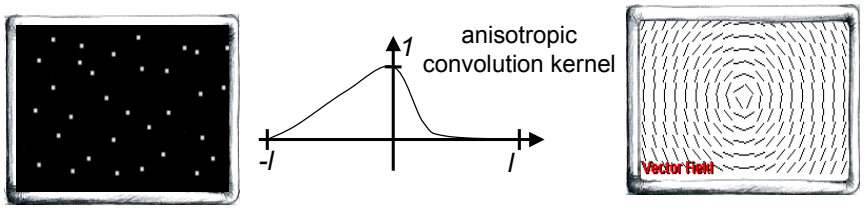
7.7. Line Integral Convolution

- Fast LIC: Algorithm
  - 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 x_0 = center of p
        Compute convolution I(x_0)
        Add result to pixel p
        For m = 1 to Limit M
            Incremental convolution for I(x_m) and I(x_-m)
            Add results to pixels containing x_m and x_-m
        End for
    End if
End for
Normalize all pixels according to Alpha
```

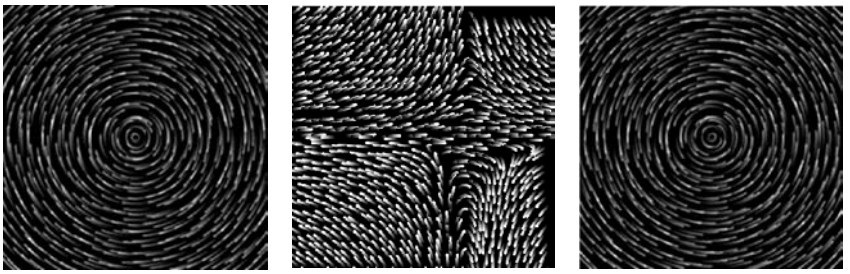
7.7. Line Integral Convolution

- Oriented LIC (OLIC):
  - Visualizes orientation (in addition to direction)
  - Sparse texture
  - Anisotropic convolution kernel
  - Acceleration: integrate individual drops and compose them to final image



7.7. Line Integral Convolution

- Oriented LIC (OLIC)

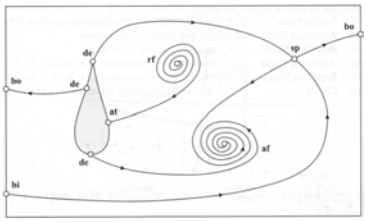


7.7. Line Integral Convolution

- 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

7.8. Vector Field Topology

- Idea:
  Do not draw “all” streamlines, but only the “important” streamlines
- Show only *topological skeletons*
- Important points in the vector field: critical points
- Critical points:
  - 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



7.8. Vector Field Topology

- Taylor expansion for the velocity field around a critical point $\mathbf{r}_c$:

$$\mathbf{v}(\mathbf{r}) = \mathbf{v}(\mathbf{r}_c) + \nabla\mathbf{v}\cdot(\mathbf{r}-\mathbf{r}_c) + O(\mathbf{r}-\mathbf{r}_c)^2$$
$$\approx \mathbf{J}\cdot(\mathbf{r}-\mathbf{r}_c)$$

- Divide Jacobian into symmetric and anti-symmetric parts

$$\mathbf{J} = \mathbf{J}_s + \mathbf{J}_a = ((\mathbf{J} + \mathbf{J}^T) + (\mathbf{J} - \mathbf{J}^T))/2$$

$$\mathbf{J}_s = (\mathbf{J} + \mathbf{J}^T)/2$$
$$\mathbf{J}_a = (\mathbf{J} - \mathbf{J}^T)/2$$

7.8. Vector Field Topology

- The symmetric part can be solved to give real eigenvalues R and real eigenvectors

$$\mathbf{J}_s\mathbf{r}_s = R\mathbf{r}_s \qquad R = R_1, R_2, R_3$$

  - Eigenvectors $\mathbf{r}_s$ are an orthonormal set of vectors
  - Describes change of size along eigenvectors
  - Describes flow into or out of region around critical point

7.8. Vector Field Topology

- Anti-symmetric part

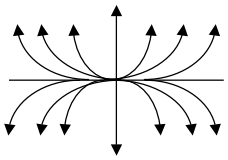
$$\mathbf{J}_a \cdot \mathbf{d} = \tfrac{1}{2}\left(\mathbf{J} - \mathbf{J}^T\right)\cdot\mathbf{d} =$$
$$\frac{1}{2}\begin{pmatrix} 0 & \frac{\partial \mathbf{v}_x}{\partial y} - \frac{\partial \mathbf{v}_y}{\partial x} & \frac{\partial \mathbf{v}_x}{\partial z} - \frac{\partial \mathbf{v}_z}{\partial x} \\ \frac{\partial \mathbf{v}_y}{\partial x} - \frac{\partial \mathbf{v}_x}{\partial y} & 0 & \frac{\partial \mathbf{v}_y}{\partial z} - \frac{\partial \mathbf{v}_z}{\partial y} \\ \frac{\partial \mathbf{v}_z}{\partial x} - \frac{\partial \mathbf{v}_x}{\partial z} & \frac{\partial \mathbf{v}_z}{\partial y} - \frac{\partial \mathbf{v}_y}{\partial z} & 0 \end{pmatrix}\cdot\mathbf{d} = \tfrac{1}{2}\left(\nabla\times\boldsymbol{v}\right)\times\mathbf{d}$$

  - Describes rotation of difference vector $\mathbf{d} = (\mathbf{r} - \mathbf{r}_c)$
  - The anti-symmetric part can be solved to give imaginary eigenvalues I

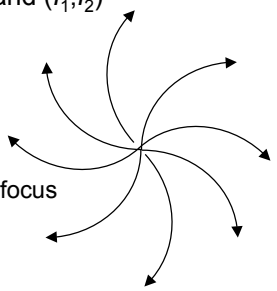
$$\mathbf{J}_a\mathbf{r}_a = I\mathbf{r}_a \qquad I = I_1, I_2, I_3$$

7.8. Vector Field Topology

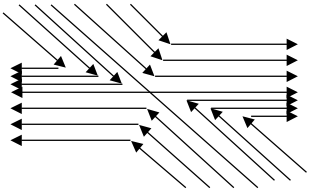
- 2D structure: eigenvalues are (R_1, R_2) and (I_1,I_2)



Repelling node
$R_1, R_2 > 0$
$I_1,I_2 = 0$



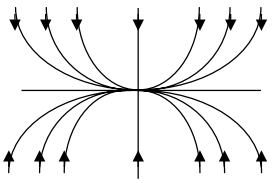
Repelling focus
$R_1, R_2 > 0$
$I_1,I_2 \neq 0$



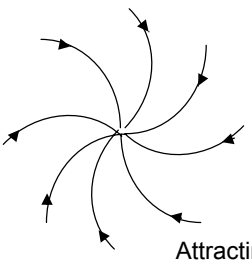
Saddle point
$R_1 * R_2 < 0$
$I_1,I_2 = 0$

7.8. Vector Field Topology

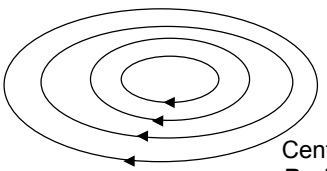
- 2D structure: eigenvalues are (R_1, R_2) and (I_1,I_2)



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



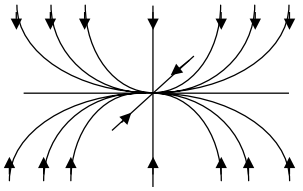
Attracting focus
$R_1, R_2 < 0$
$I_1,I_2 \neq 0$



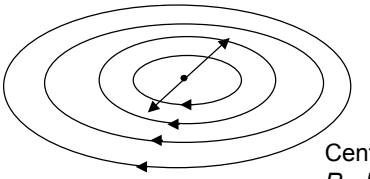
Center
$R_1, R_2 = 0$
$I_1,I_2 \neq 0$

7.8. Vector Field Topology

- Also in 3D
  - Some examples



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



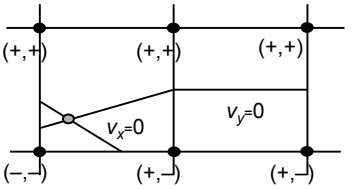
Center
$R_1, R_2 = 0, R_3 > 0$
$I_1, I_2 \neq 0, , I_3 = 0$

7.8. Vector Field Topology

- Mapping to graphical primitives: streamlines
  - Start streamlines close to critical points
  - Initial direction along the eigenvectors
- End particle tracing at
  - Other “real” critical points
  - Interior boundaries: attachment or detachment points
  - Boundaries of the computational domain

7.8. Vector Field Topology

- How to find critical points
  - 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

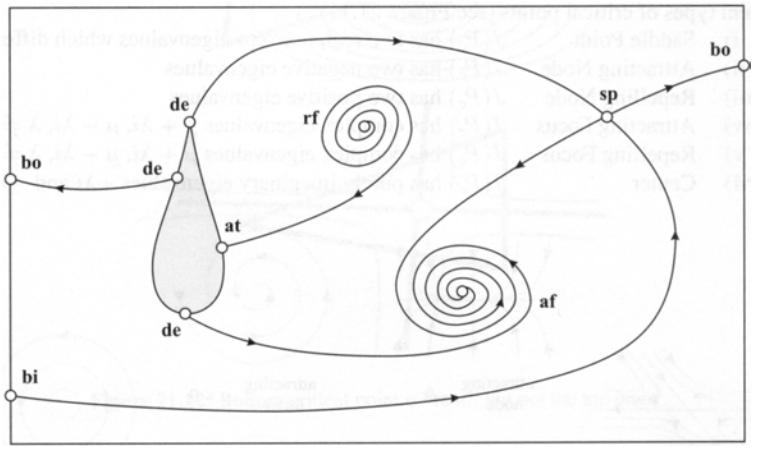


7.8. Vector Field Topology

- How to find critical points (cont.)
  - 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

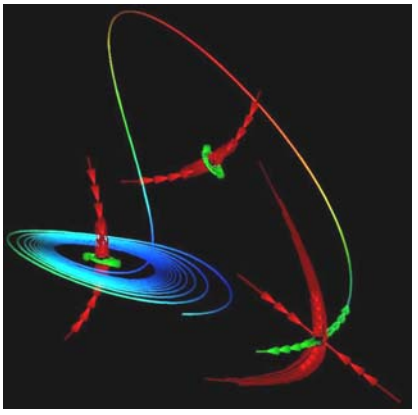
7.8. Vector Field Topology

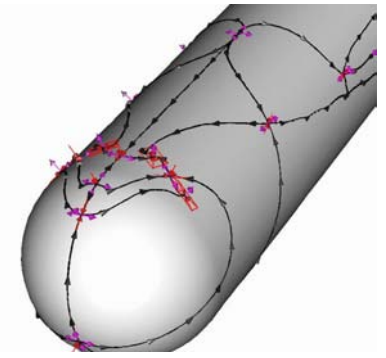
- Example of a topological graph of 2D flow field



7.8. Vector Field Topology

- Further examples of topology-guided streamline positioning





7.8. Vector Field Topology

- 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?

7.9. 3D Vector Fields

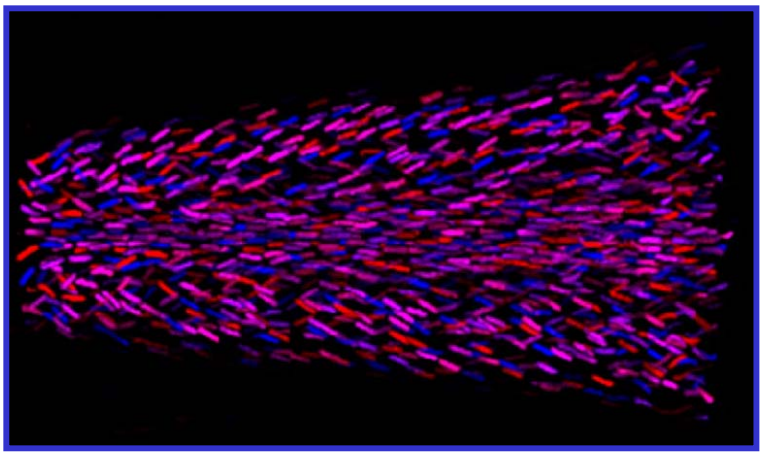
- Most algorithms can be applied to 2D and 3D vector fields
- Main problem in 3D: effective mapping to graphical primitives
- Main aspects:
  - Occlusion
  - Amount of (visual) data
  - Depth perception

7.9. 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

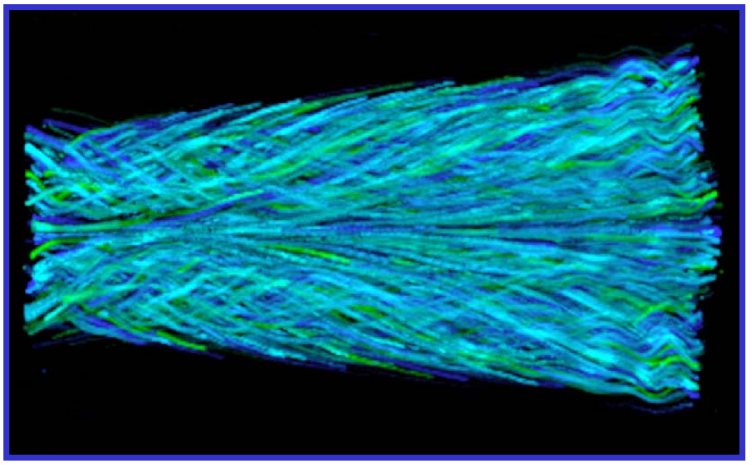
7.9. 3D Vector Fields

- Missing continuity



7.9. 3D Vector Fields

- Color differences to identify connected structures

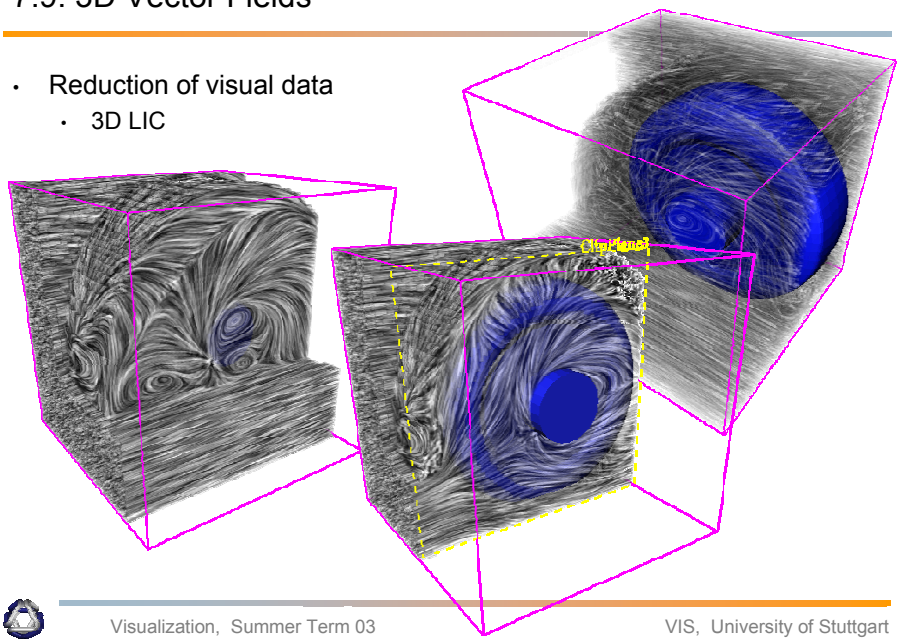


7.9. 3D Vector Fields

- Animation

7.9. 3D Vector Fields

- Reduction of visual data
  - 3D LIC

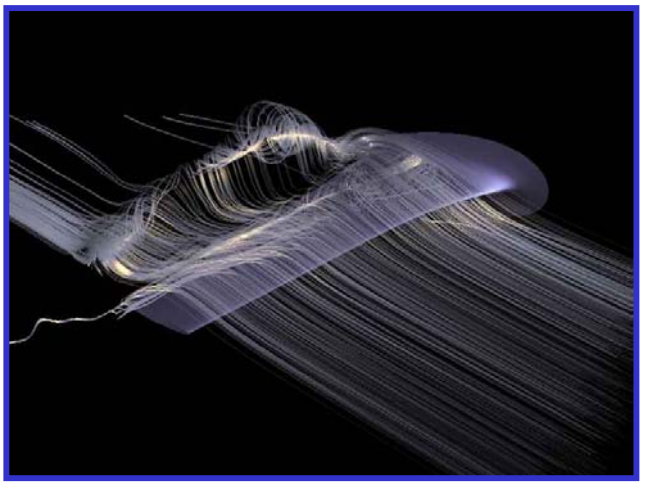


7.9. 3D Vector Fields

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

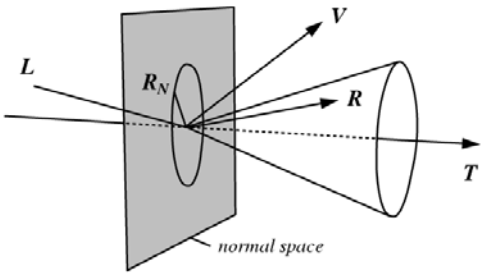
7.9. 3D Vector Fields

- Illumination



7.9. 3D Vector Fields

- Illuminated streamlines [Zöckler et al. 1996]
  - 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

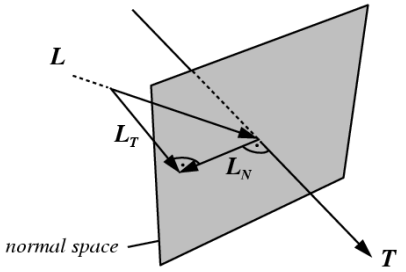


7.9. 3D Vector Fields

- Illuminated streamlines (*cont.*)
  - Light vector is split in tangential and normal parts

$$\begin{aligned}\mathbf{V}\cdot\mathbf{R} &= \mathbf{V}\cdot\left(\mathbf{L}_T - \mathbf{L}_N\right) = \mathbf{V}\cdot\left((\mathbf{L}\cdot\mathbf{T})\mathbf{T} - (\mathbf{L}\cdot\mathbf{N})\mathbf{N}\right)\\ &= (\mathbf{L}\cdot\mathbf{T})(\mathbf{V}\cdot\mathbf{T}) - (\mathbf{L}\cdot\mathbf{N})(\mathbf{V}\cdot\mathbf{N})\\ &= (\mathbf{L}\cdot\mathbf{T})(\mathbf{V}\cdot\mathbf{T}) - \sqrt{1-(\mathbf{L}\cdot\mathbf{T})^2}\sqrt{1-(\mathbf{V}\cdot\mathbf{T})^2}\\ &= f\left((\mathbf{L}\cdot\mathbf{T}),(\mathbf{V}\cdot\mathbf{T})\right)\end{aligned}$$

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

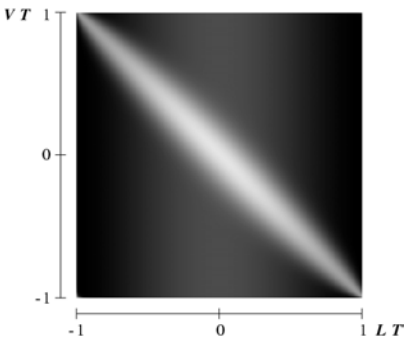


7.9. 3D Vector Fields

- Illuminated streamlines (*cont.*)
  - Phong model

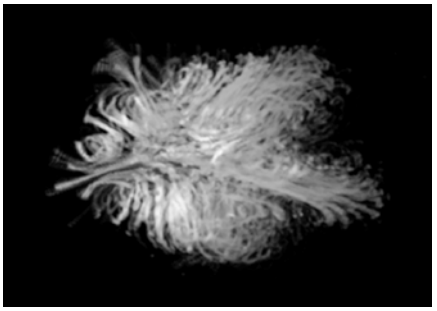
$$\begin{aligned}I &= I_{\text{ambient}} + I_{\text{diffuse}} + I_{\text{specular}}\\ &= k_a + k_d\mathbf{L}\cdot\mathbf{N} + k_s\left(\mathbf{V}\cdot\mathbf{R}\right)^n\end{aligned}$$

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

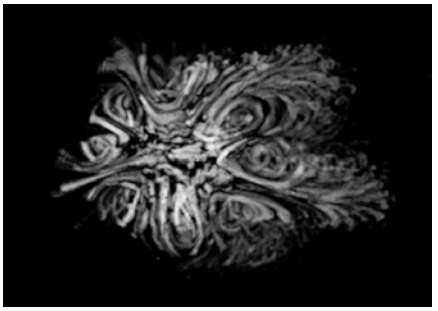


7.9. 3D Vector Fields

- Halos



Without halos



With halos

7.9. 3D Vector Fields

- Flow Visualization is a combination of all vector/scalar vis techniques