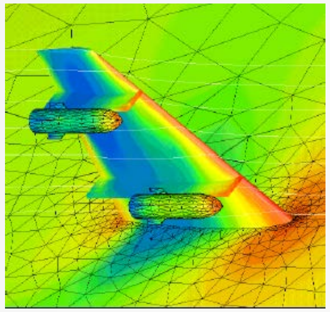
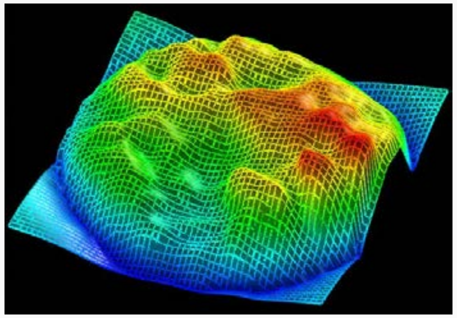
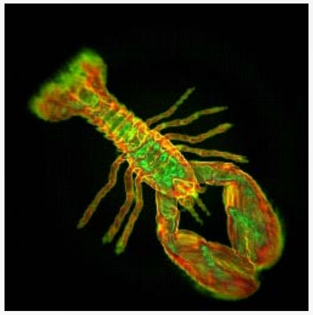
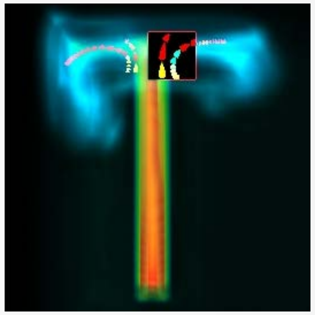





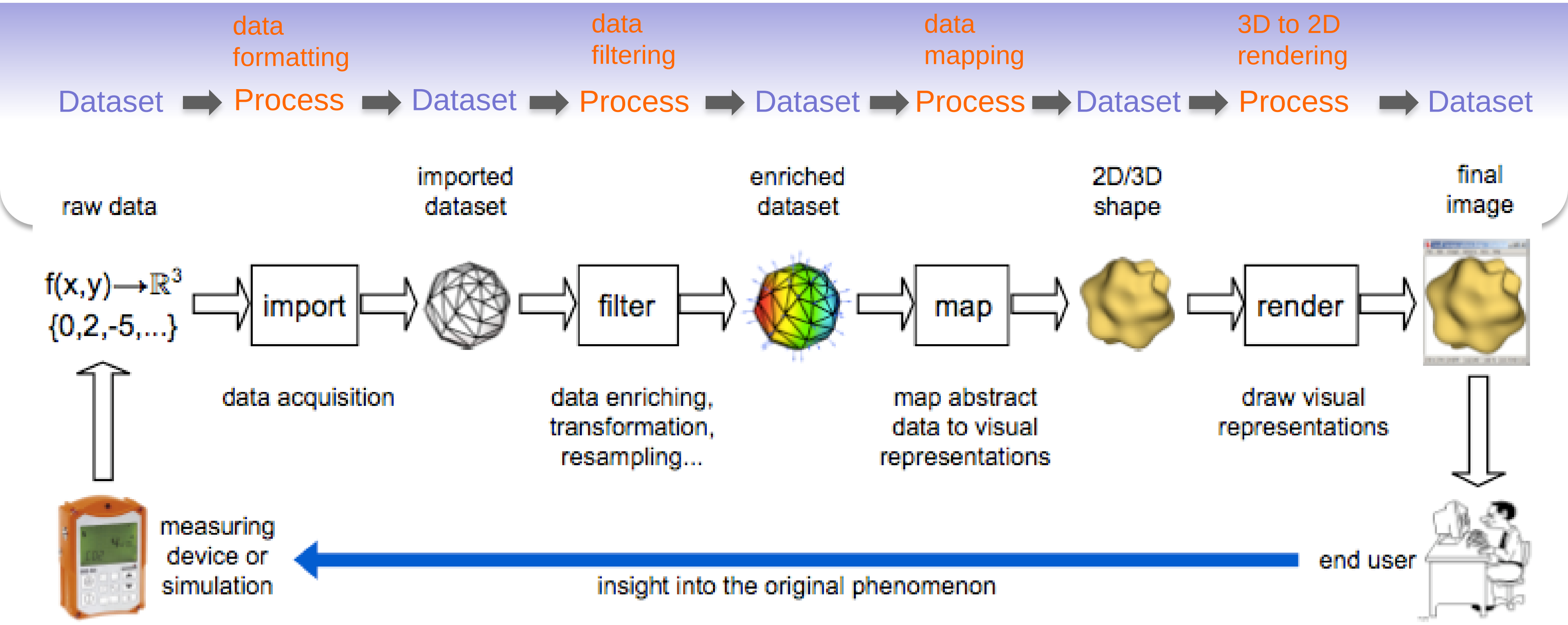
Vector Data

Cmpt 767 Visualization

Steven Bergner
sbergner@sfu.ca

[based on slides by A. C. Telea]

The Visualization Pipeline - Recall



[A.C Telea: Data Visualization, Principles and Practice, 2nd edition, CRC Press, 2014]

Vector algorithms (Telea, Ch. 6)

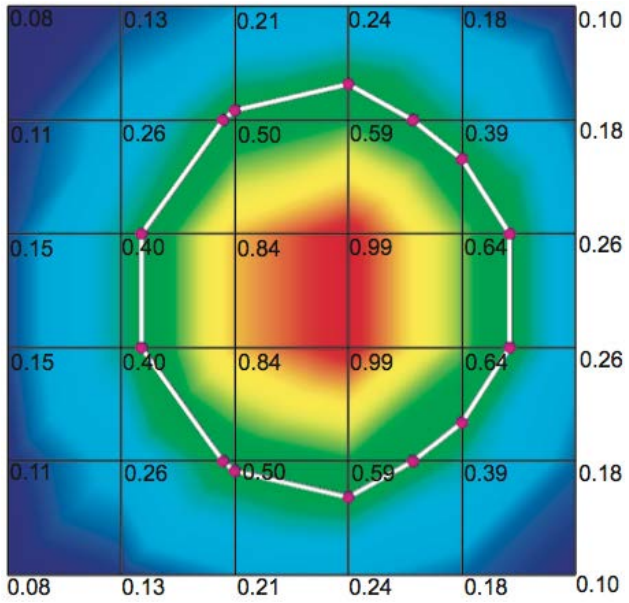
1. **Scalar derived quantities**
   - divergence, curl, vorticity
2. **0-dimensional shapes**
   - hedgehogs and glyphs
   - color coding
3. **1-dimensional and 2-dimensional shapes**
   - displacement plots
   - stream objects
4. **Image-based algorithms**
   - image-based flow visualization in 2D, curved surfaces, and 3D

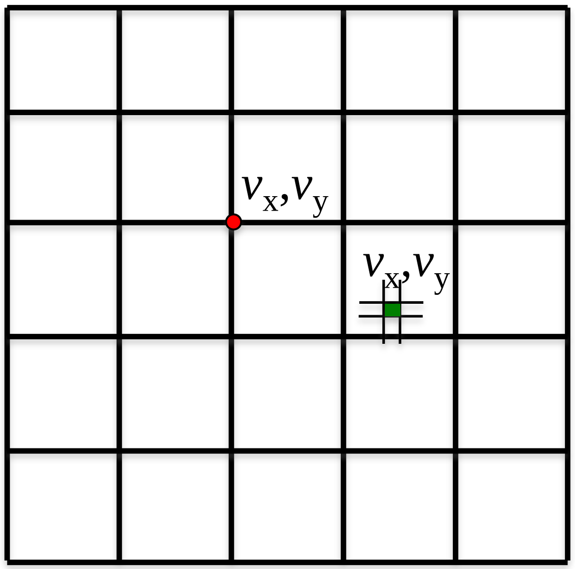
Basic problem

Input data

•vector field	$v : D \rightarrow \mathbf{R}^n$
•domain D	2D planar surfaces, 2D surfaces embedded in 3D, 3D volumes
•variables	n=2 (fields tangent to 2D surfaces) or n=3 (volumetric fields)

Challenge: comparison with scalar visualization





Scalar visualization

- challenge is to map D to 2D screen
- after that, we have 1 pixel per scalar value

Vector visualization

- challenge is to map D to 2D screen
- after that, we have 1 pixel for 2 or 3 scalar values!

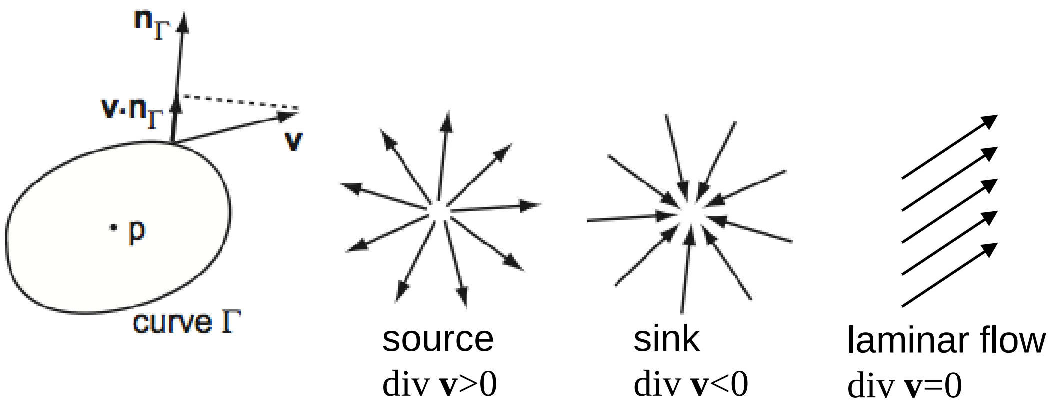
First solution: Reuse scalar visualization

- compute derived scalar quantities from vector fields
- use known scalar visualization methods for these

1.Divergence

- think of vector field as encoding a fluid flow
- intuition: amount of mass (air, water) created, or absorbed, at a point in D
- given a field $\mathbf{v} : \mathbf{R}^3 \to \mathbf{R}^3$, $\text{div } \mathbf{v} : \mathbf{R}^3 \to \mathbf{R}$ is

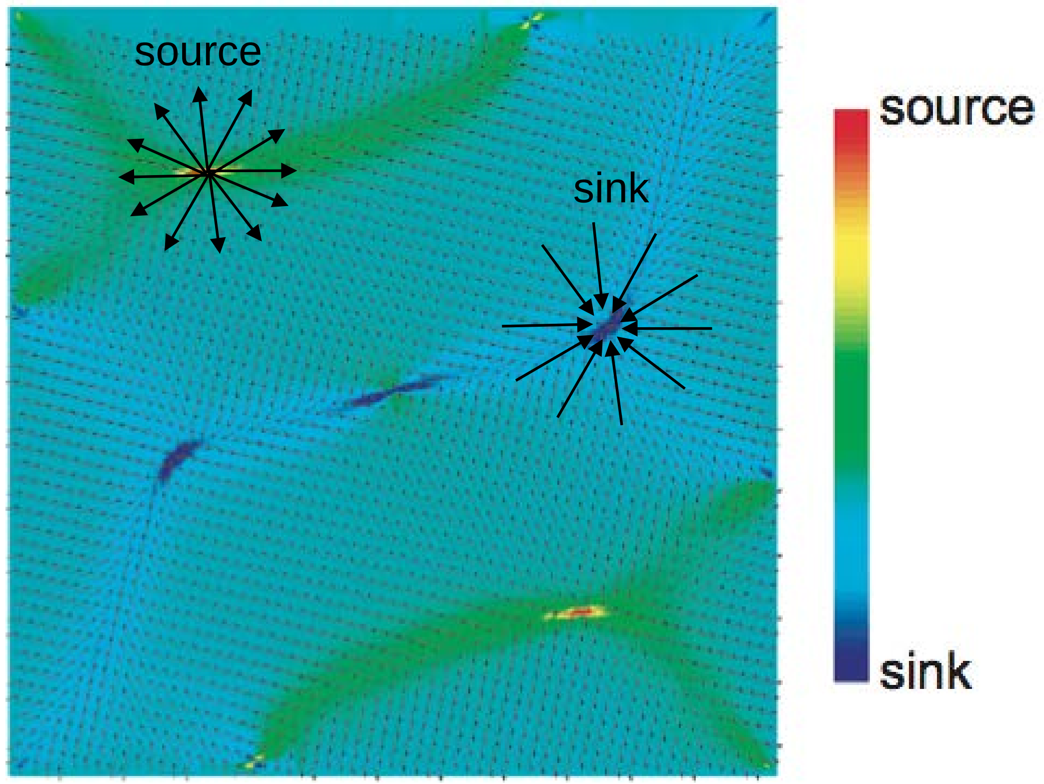
$$\text{div } \mathbf{v} = \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z} \quad \text{equivalent to} \quad \text{div } \mathbf{v} = \lim_{\Gamma \to 0} \frac{1}{|\Gamma|} \int_\Gamma (\mathbf{v} \cdot \mathbf{n}_\Gamma) ds$$



div **v** is sometimes denoted as $\nabla \cdot v$

Divergence

•compute using definition with partial derivatives
•visualize using e.g. color mapping



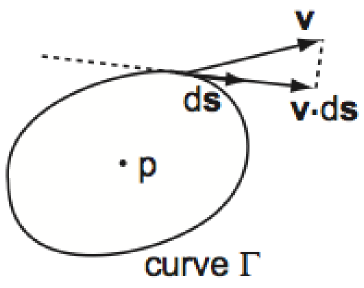
•gives a good impression of where the flow ‘enters’ and ‘exits’ some domain

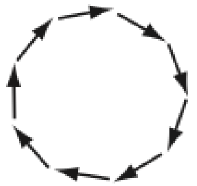
Curl

2. Curl (also called rotor)

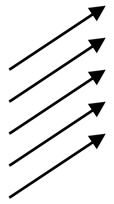
•consider again a vector field as encoding a fluid flow
•intuition: how quickly the flow ‘rotates’ around each point?
•given a field $\mathbf{v} : \mathbf{R}^3 \to \mathbf{R}^3$, $\text{rot } \mathbf{v} : \mathbf{R}^3 \to \mathbf{R}^3$ is

$$\text{rot } \mathbf{v} = \left(\frac{\partial v_z}{\partial y} - \frac{\partial v_y}{\partial z}, \frac{\partial v_x}{\partial z} - \frac{\partial v_z}{\partial x}, \frac{\partial v_y}{\partial x} - \frac{\partial v_x}{\partial y} \right)$$ equivalent to $$\text{rot } \mathbf{v} = \lim_{\Gamma \to 0} \frac{1}{|\Gamma|} \int_\Gamma \mathbf{v} \cdot \mathbf{ds}$$

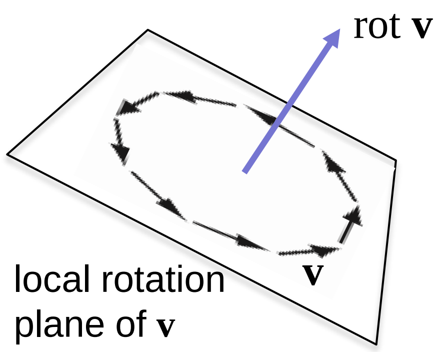




rotational flow
$\|\text{rot } \mathbf{v}\| > 0$



laminar flow
$\text{rot } \mathbf{v} = \mathbf{0}$



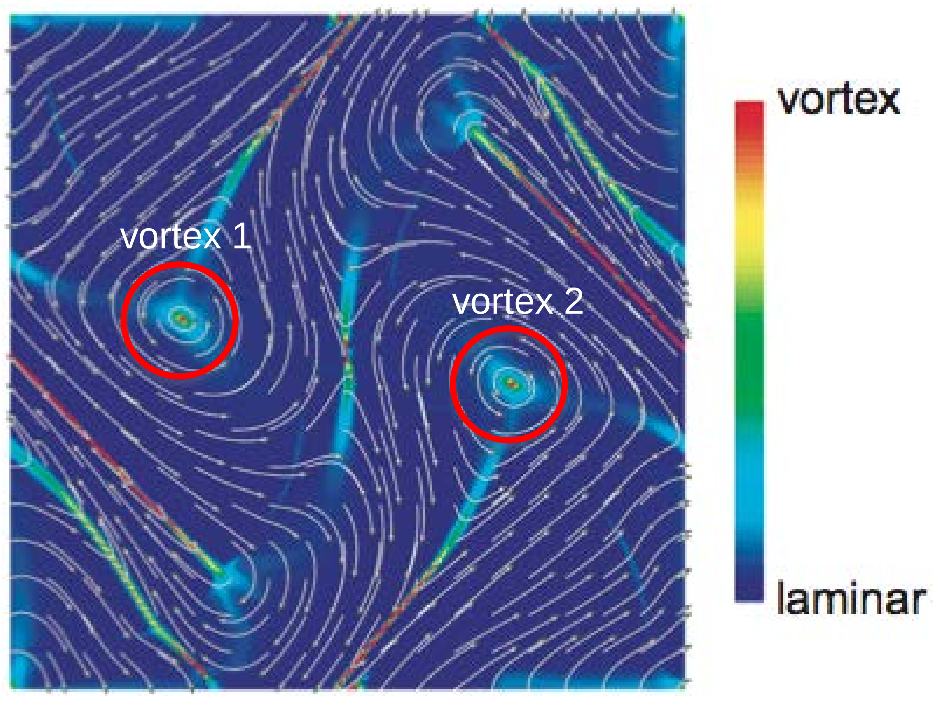
local rotation plane of $\mathbf{v}$

•rot $\mathbf{v}$ is locally perpendicular to plane of rotation of $\mathbf{v}$
•its magnitude: ‘tightness’ of rotation – also called vorticity

rot $\mathbf{v}$ is sometimes denoted as $\nabla \times v$

Curl

- compute using definition with partial derivatives
- visualize magnitude $\|\mathrm{rot}\ \mathbf{v}\|$ using e.g. color mapping

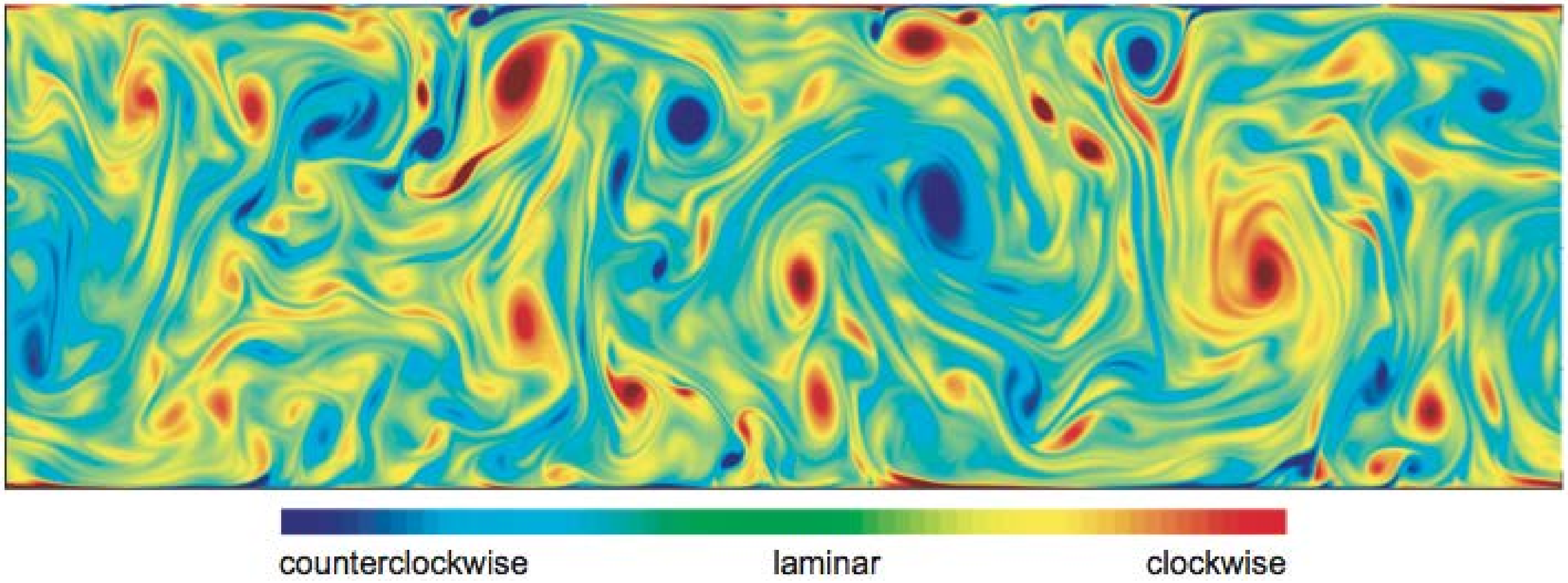


- very useful in practice to find vortices = regions of high vorticity
- these are highly important in flow simulations (aerodynamics, hydrodynamics)

Curl

Example of vorticity

- 2D fluid flow
- simulated by solving Navier-Stokes equations
- visualized using vorticity



Observations

- vortices appear at different scales
- see the ‘pairing’ of vortices spinning in opposite directions
- what happens with the flow close to the boundary? Why

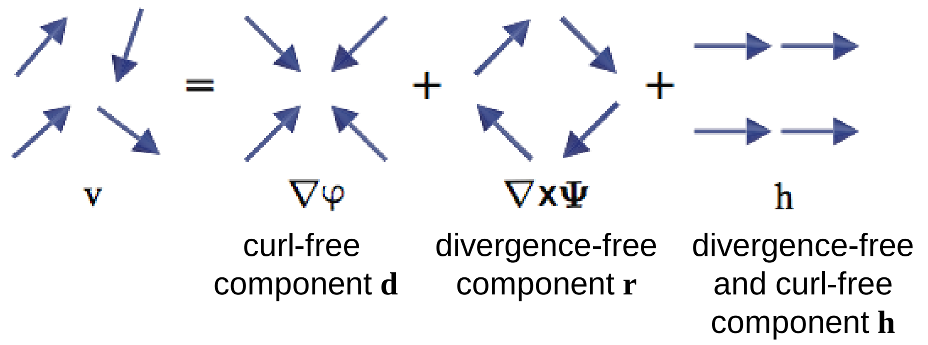
A Simple Fluid Solver based on the FFT, J. Stam, J. of Graphics Tools 6(2), 2001, 43-52

Vector field decomposition

Helmholtz-Hodge theorem

- any vector field **v** can be uniquely decomposed into three components

$$\mathbf{v} = \mathbf{d} + \mathbf{r} + \mathbf{h}$$


=
+
+
v
∇φ
curl-free component d
∇xΨ
divergence-free component r
h
divergence-free and curl-free component h

$$\begin{aligned} \nabla \times \mathbf{d} &= 0 \\ \nabla \cdot \mathbf{r} &= 0 \\ \Delta \mathbf{h} &= 0 \end{aligned}$$

d, **r**, **h** are computed from two intermediate **potential fields** φ, Ψ

$$\begin{aligned} \mathbf{d} &= \nabla\varphi \\ \mathbf{r} &= \nabla \times \Psi \\ \mathbf{h} &= \mathbf{v} - \mathbf{d} - \mathbf{r} \end{aligned}$$

curl-free since $\nabla \times (\nabla\varphi) = 0$

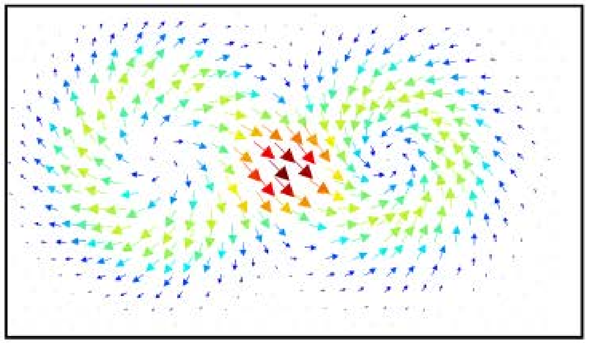
divergence-free since $\nabla \cdot (\nabla \times \Psi) = 0$

For full details, see the paper below

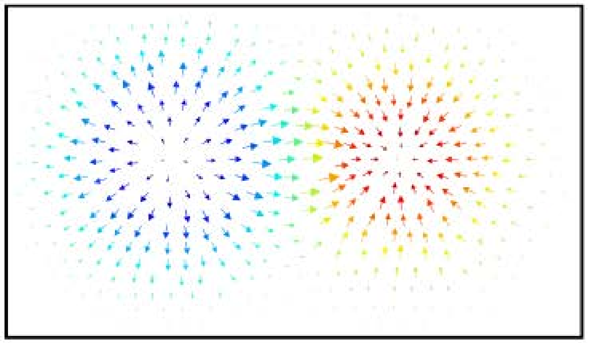
F. Petronetto, A. Paiva, M. Lage, G. Tavares, H. Lopes, H. Lewiner, Meshless Helmholtz-Hodge decomposition, *IEEE TVCG*, 2008

Vector field decomposition

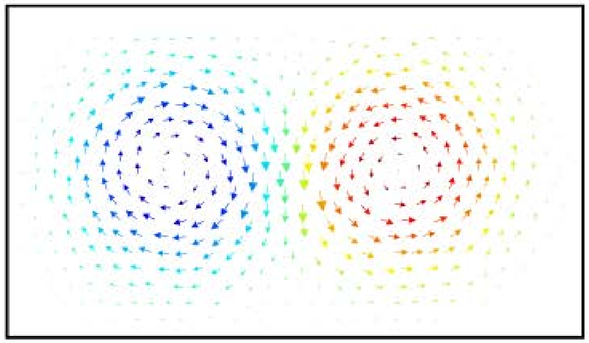
color: vector field magnitude $\|\mathbf{v}\|$

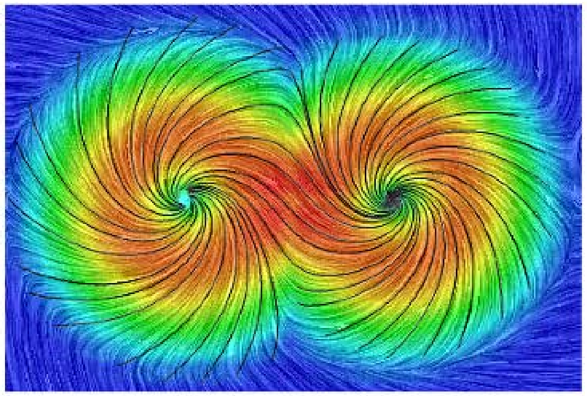


color: divergence $\mathrm{div}\,\mathbf{d}$

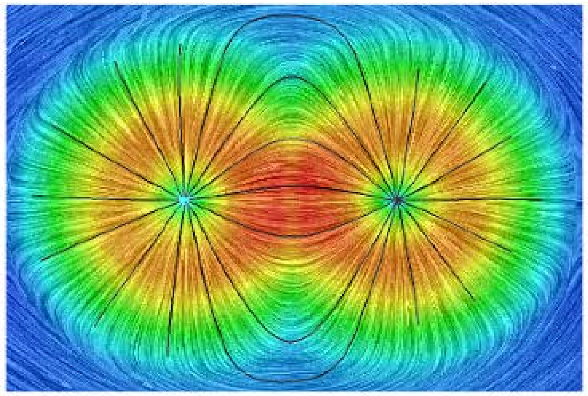


color: vorticity $\|\mathrm{rot}\,\mathbf{r}\|$

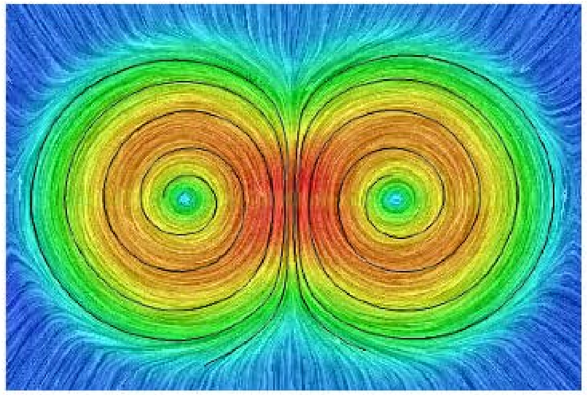




color: vector field magnitude $\|\mathbf{v}\|$



color: magnitude $\|\mathbf{d}\|$



color: magnitude $\|\mathbf{r}\|$

input field $\mathbf{v}$ = curl-free component $\mathbf{d}$ + divergence-free component $\mathbf{r}$

Vector glyphs

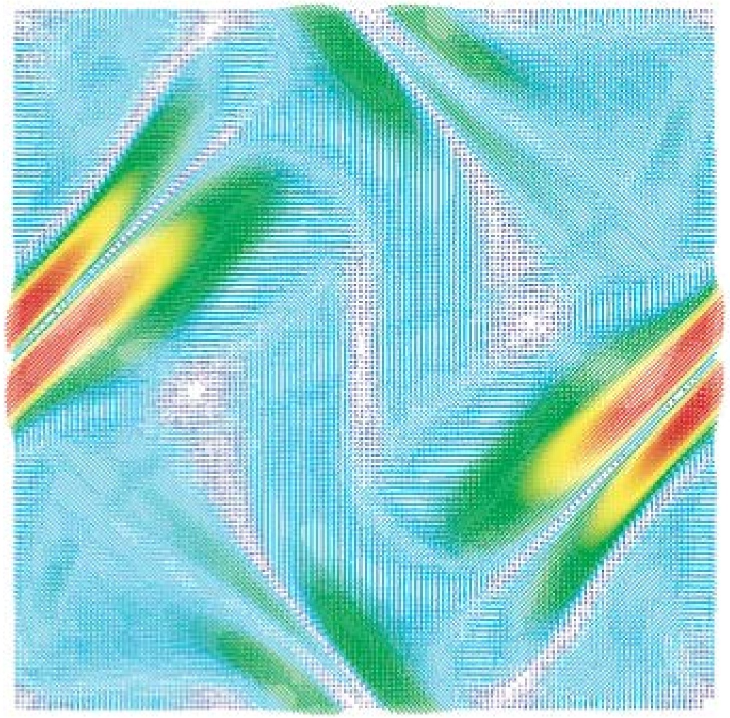
Icons, or signs, for visualizing vector fields

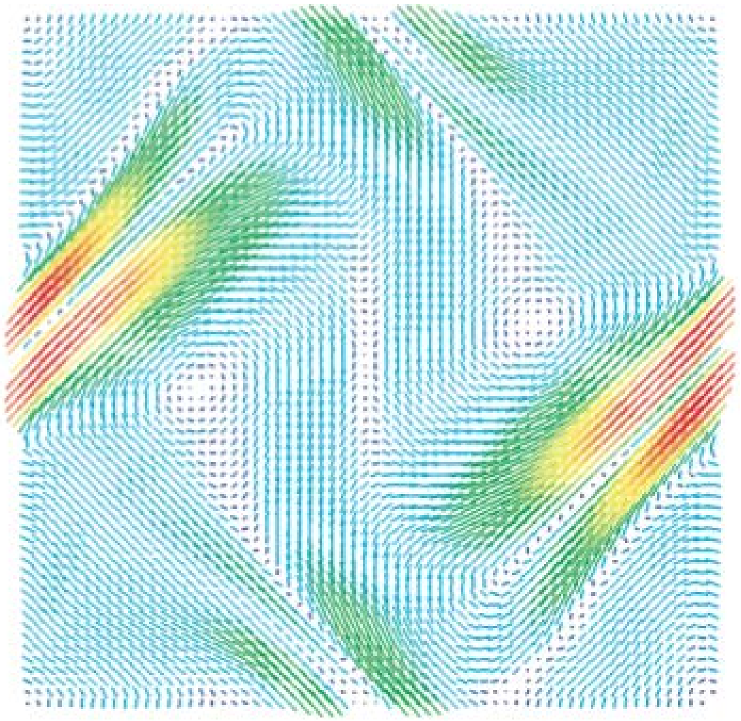
•placed by (sub)sampling the dataset domain
•attributes (scale, color, orientation) map vector data at sample points

Simplest glyph: Line segment (hedgehog plots)

•for every sample point $x \in D$

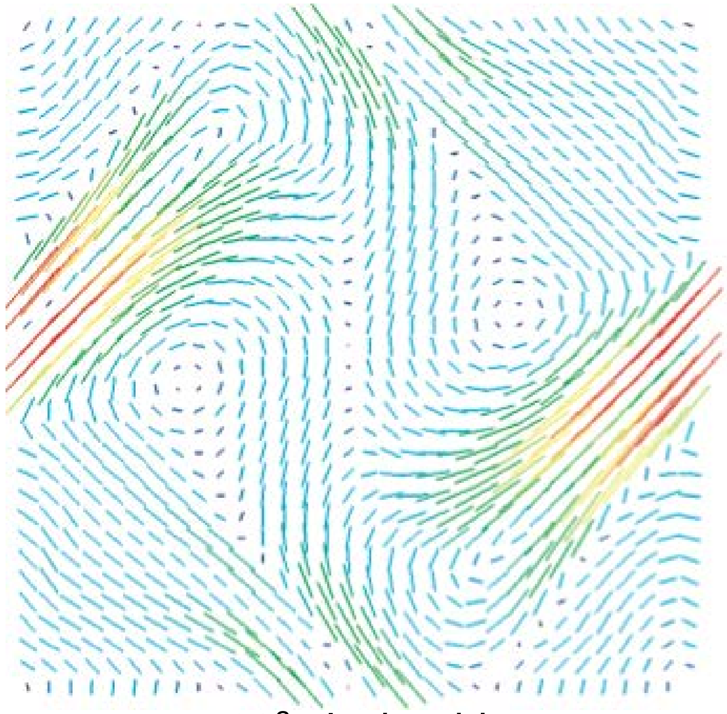
- draw line $(x, x + k\mathbf{v}(x))$
- optionally color map $||\mathbf{v}||$ onto it

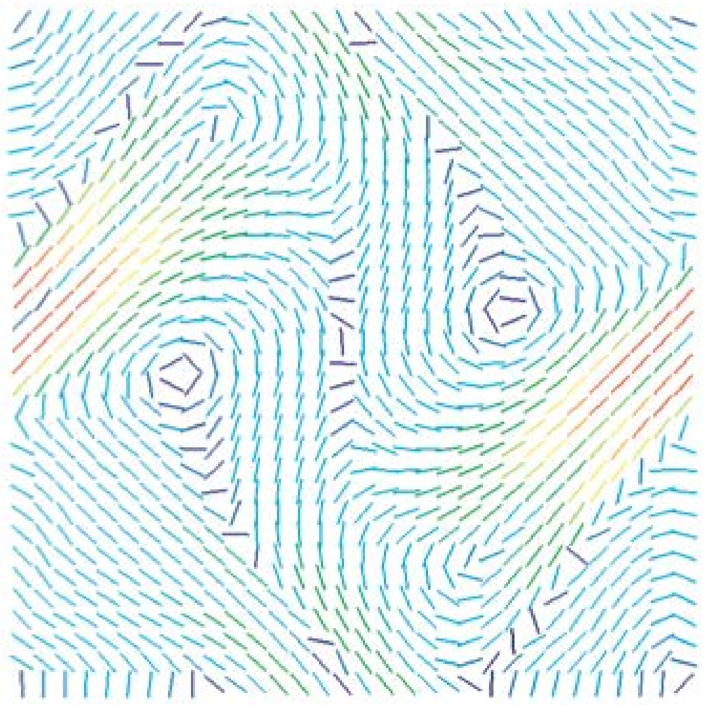

128^2 glyph grid


64^2 glyph grid

MHD simulation
256^2 grid

Vector glyphs


32^2 glyph grid


32^2 glyph grid, no line scaling

MHD simulation
256^2 grid

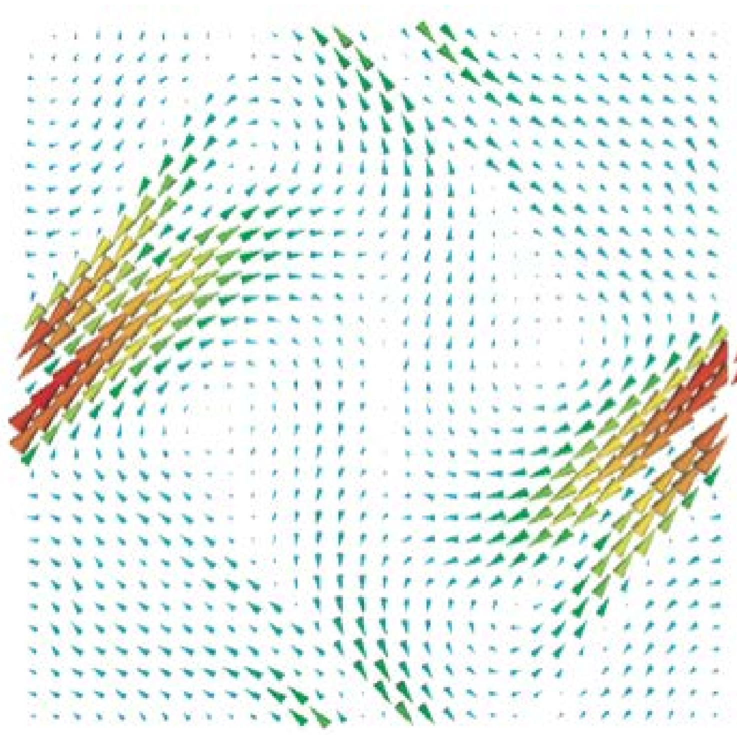
Observations

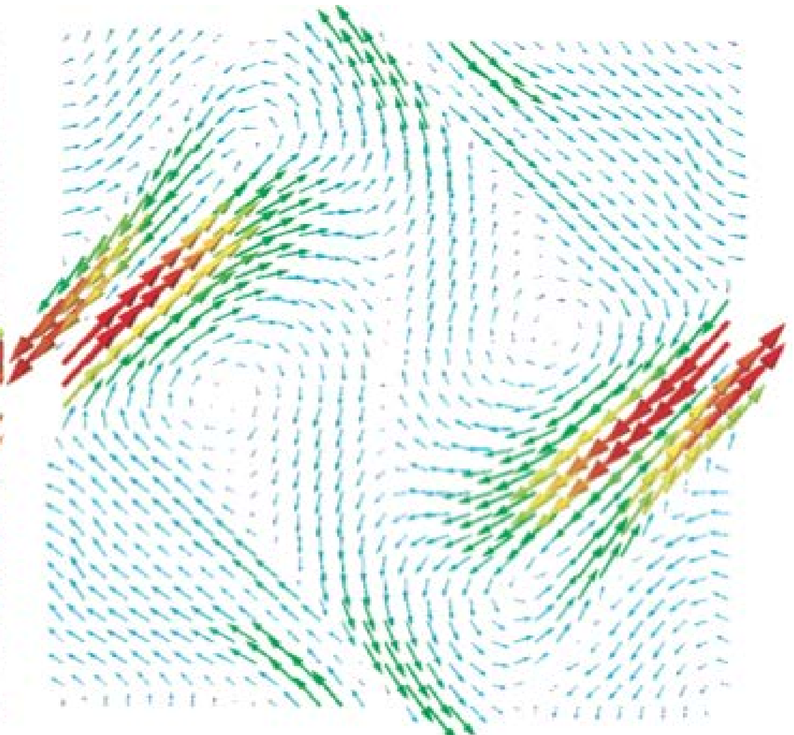
•trade-offs

- more samples: more data points depicted, but more potential clutter
- less samples: less data points depicted, but higher clarity
- more line scaling: easier to see high-speed areas, but more clutter
- less line scaling: less clutter, but harder to perceive directions

Can you observe other pro's and con's of line glyphs?

Vector glyphs


3D cone glyphs


3D arrow glyphs

MHD simulation
256^2 grid

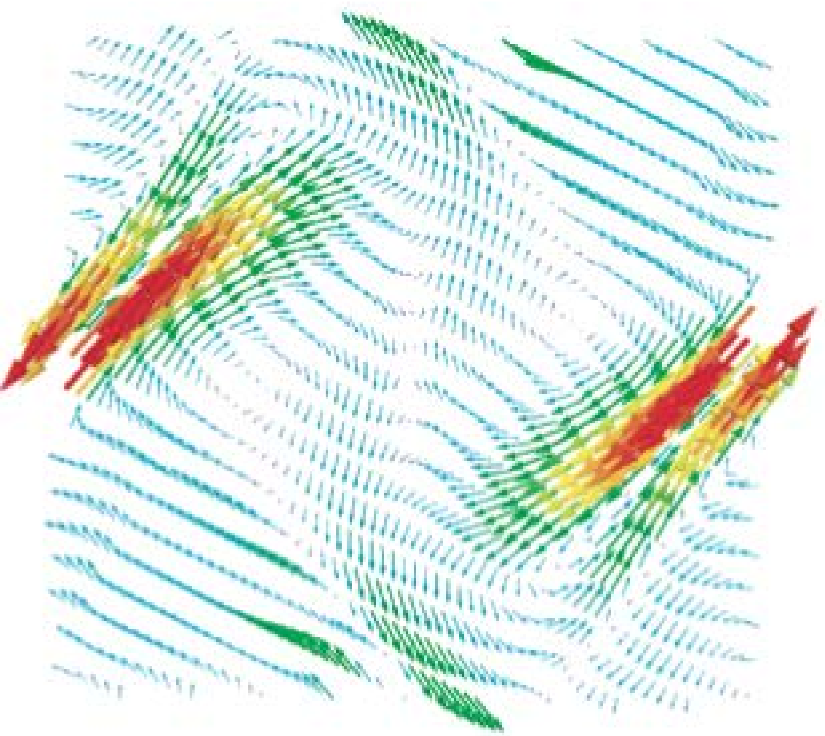
Variants

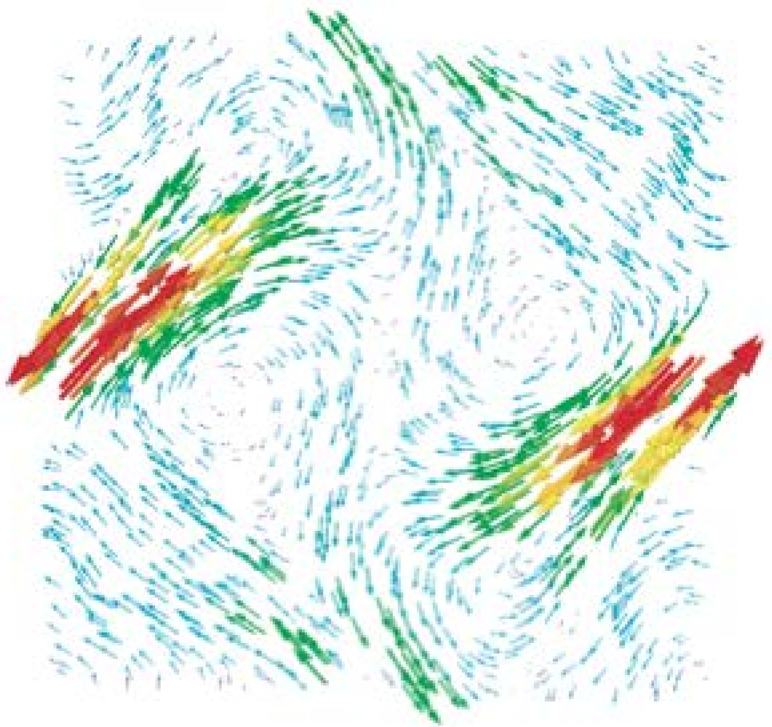
•cones, arrows, …

- show *orientation* better than lines
- but take more space to render
- shading: good visual cue to separate (overlapping) glyphs

Can you observe other pro's and con's of cone or arrow glyphs?

Vector glyphs


samples on a rotated grid


random samples, quasi-uniform density

How to choose sample points

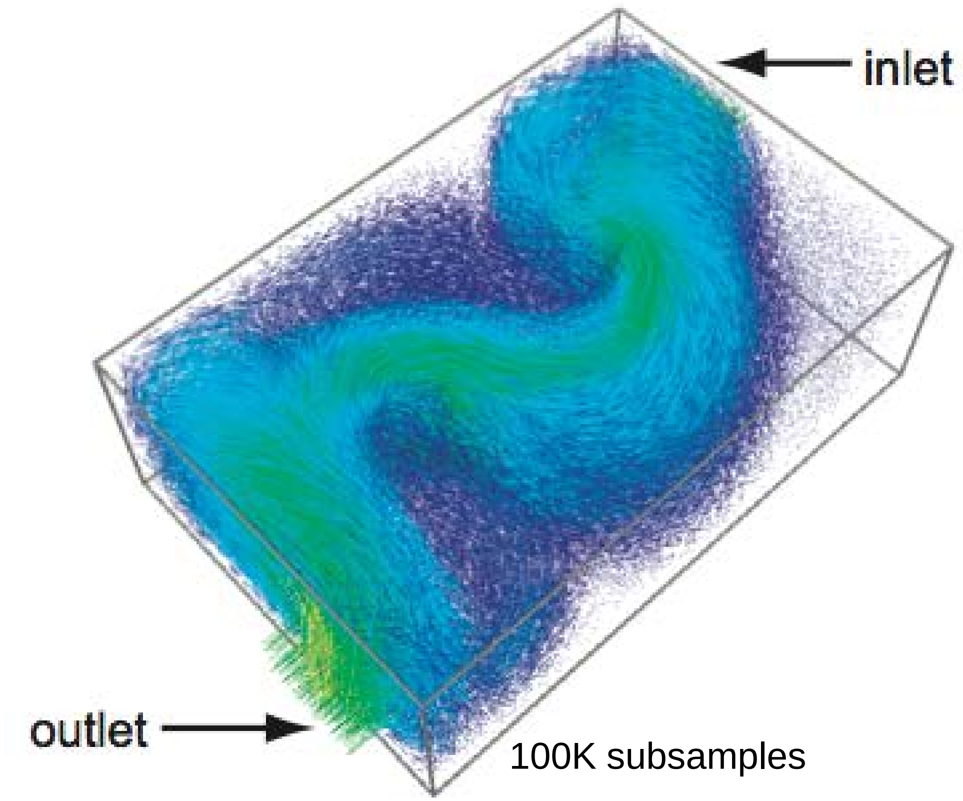
•avoid uniform grids! (why? See sampling theory, 'beating artifacts')

•random sampling: generally OK

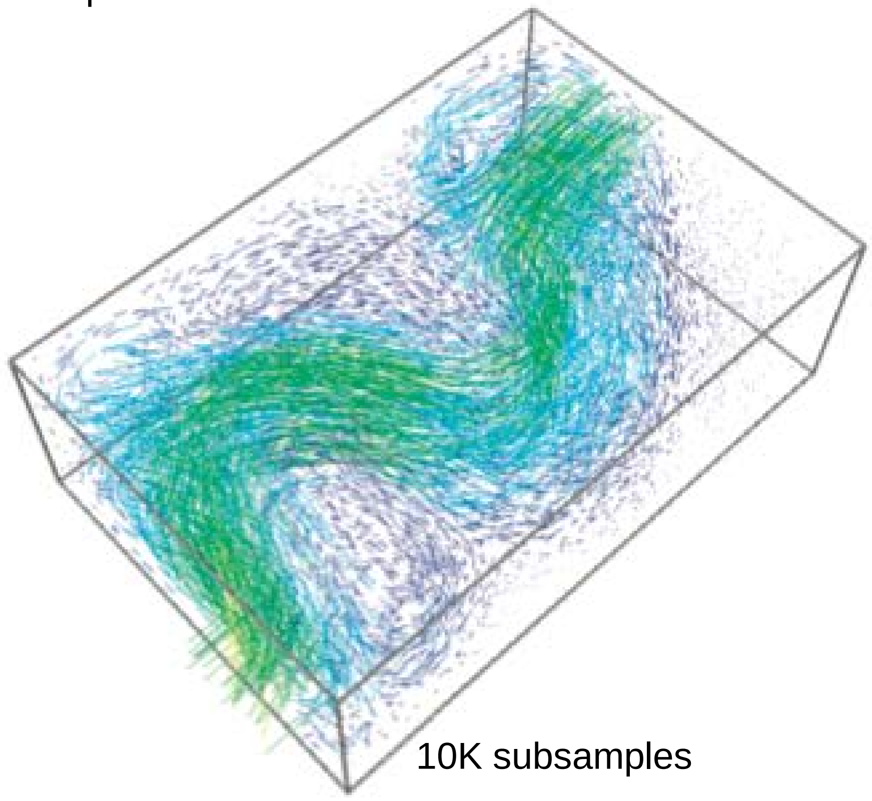
What false impressions does the left plot convey w.r.t. the right plot?

3D vector glyphs

128x85x42 volume field
456960 data points



100K subsamples

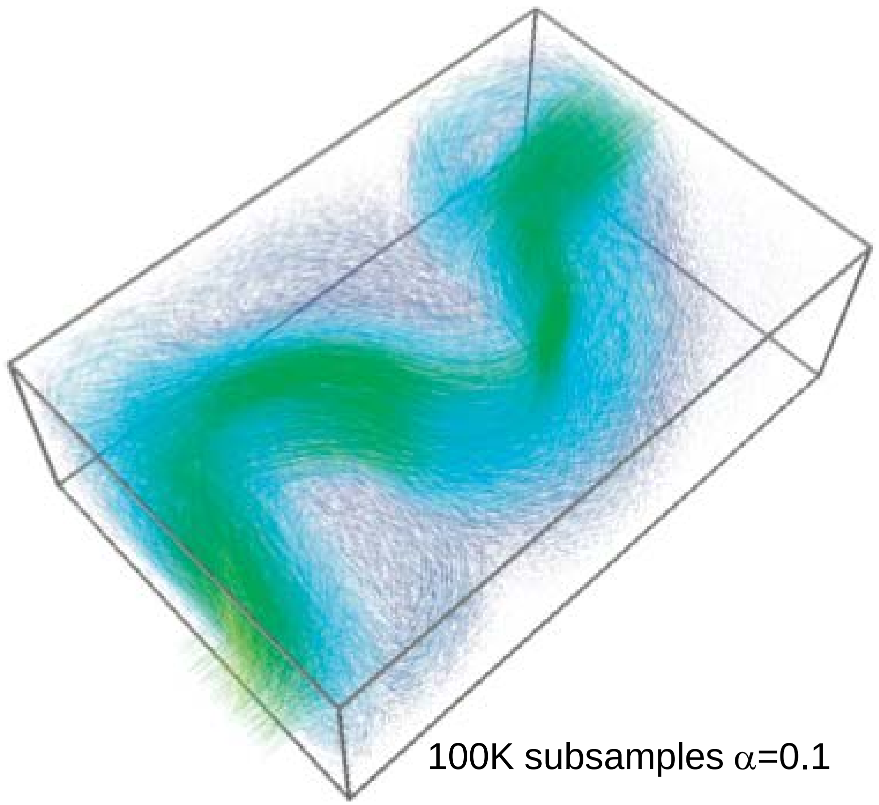


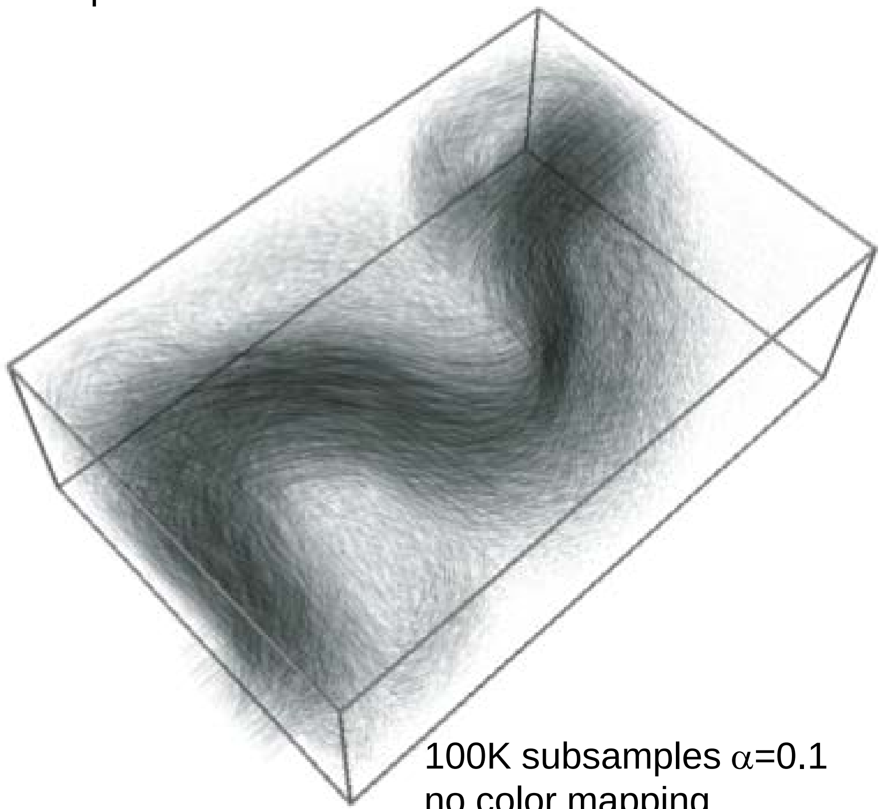
10K subsamples

- same idea/technique as 2D vector glyphs
- 3D additional problems
    - more data, same screen space
    - occlusion
    - perspective foreshortening
    - viewpoint selection

3D vector glyphs

128x85x42 volume field
456960 data points

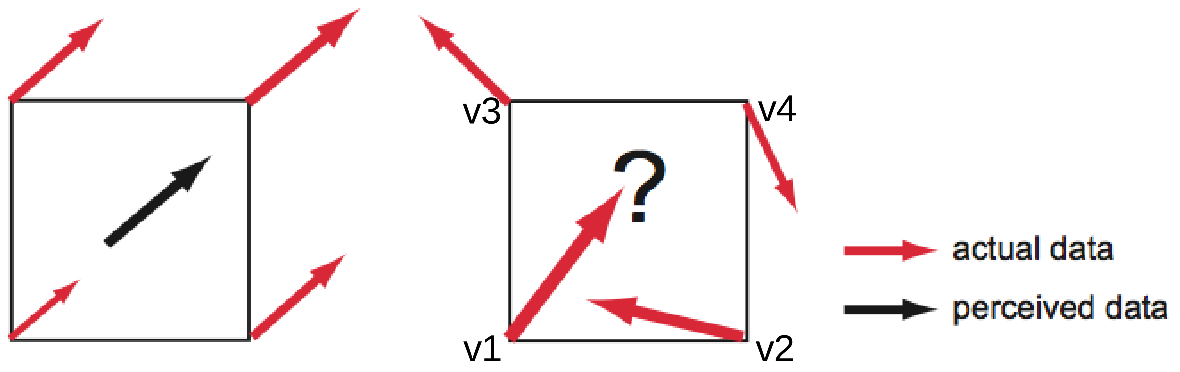

100K subsamples α=0.1

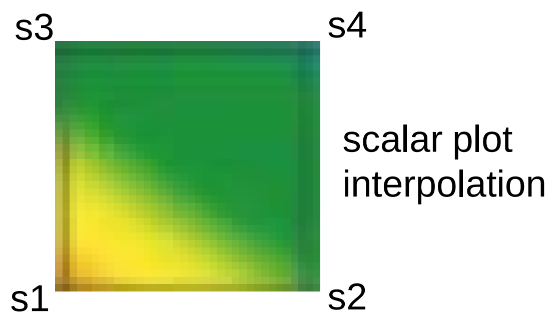

100K subsamples α=0.1
no color mapping

Alpha blending

- extremely simple and powerful tool
- reduce *perceived* occlusion
    - low-speed zones: highly transparent
    - high-speed zones: opaque and highly coherent (why?)

Glyph problem revisited





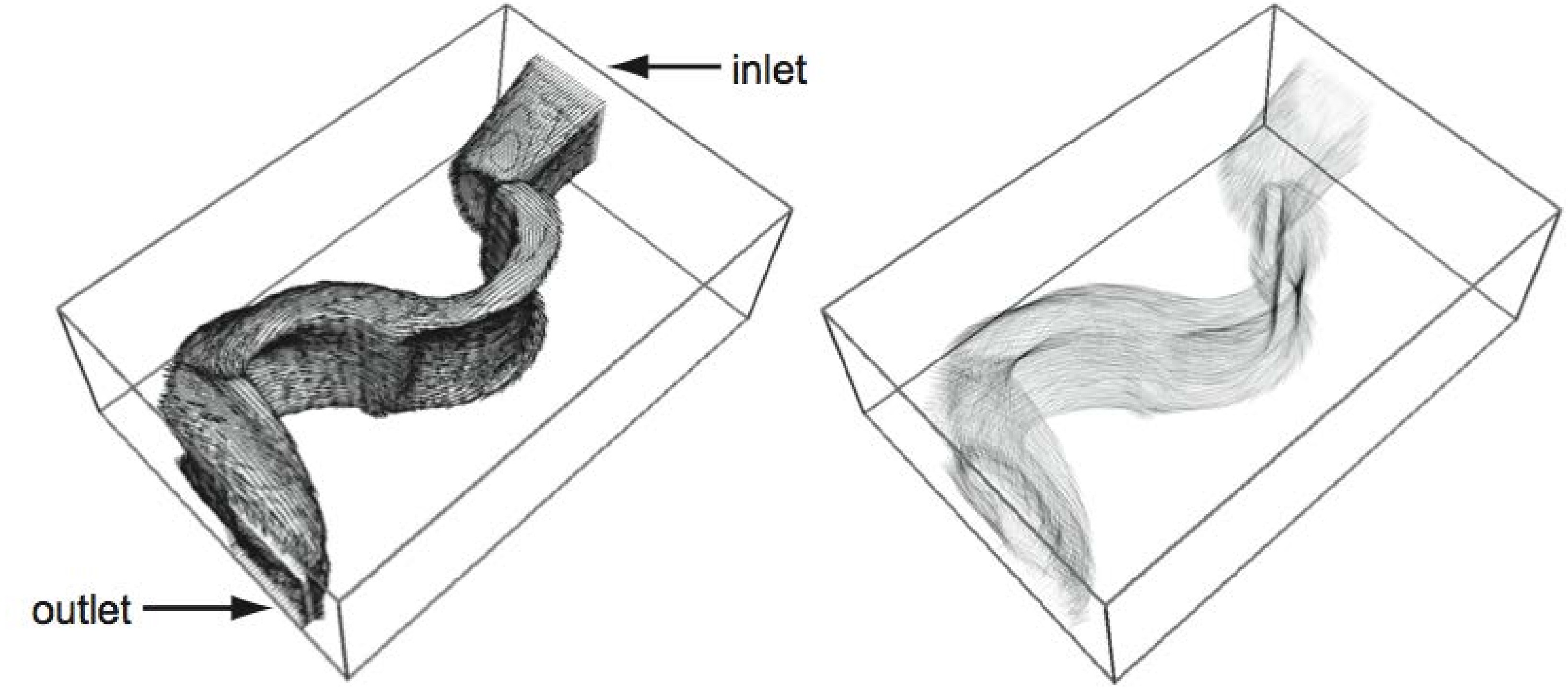
Recall the 'inverse mapping' proposal

•we render something…
•…so we can visually map it to some data/phenomenon

Glyph problems

- •**no interpolation** in glyph space (unlike for scalar plots with color mapping!)
- •a glyph takes more space than a pixel
- •we (humans) aren't good at visually interpolating arrows…
- •scalar plots are **dense**; glyph plots are **sparse**
  - this is why glyph positioning (sampling) is extra important

Vector glyphs on 3D surfaces



Trade-off between vector glyphs in 2D planes and in full 3D

•find interesting surface

- e.g. **isosurface** of flow velocity

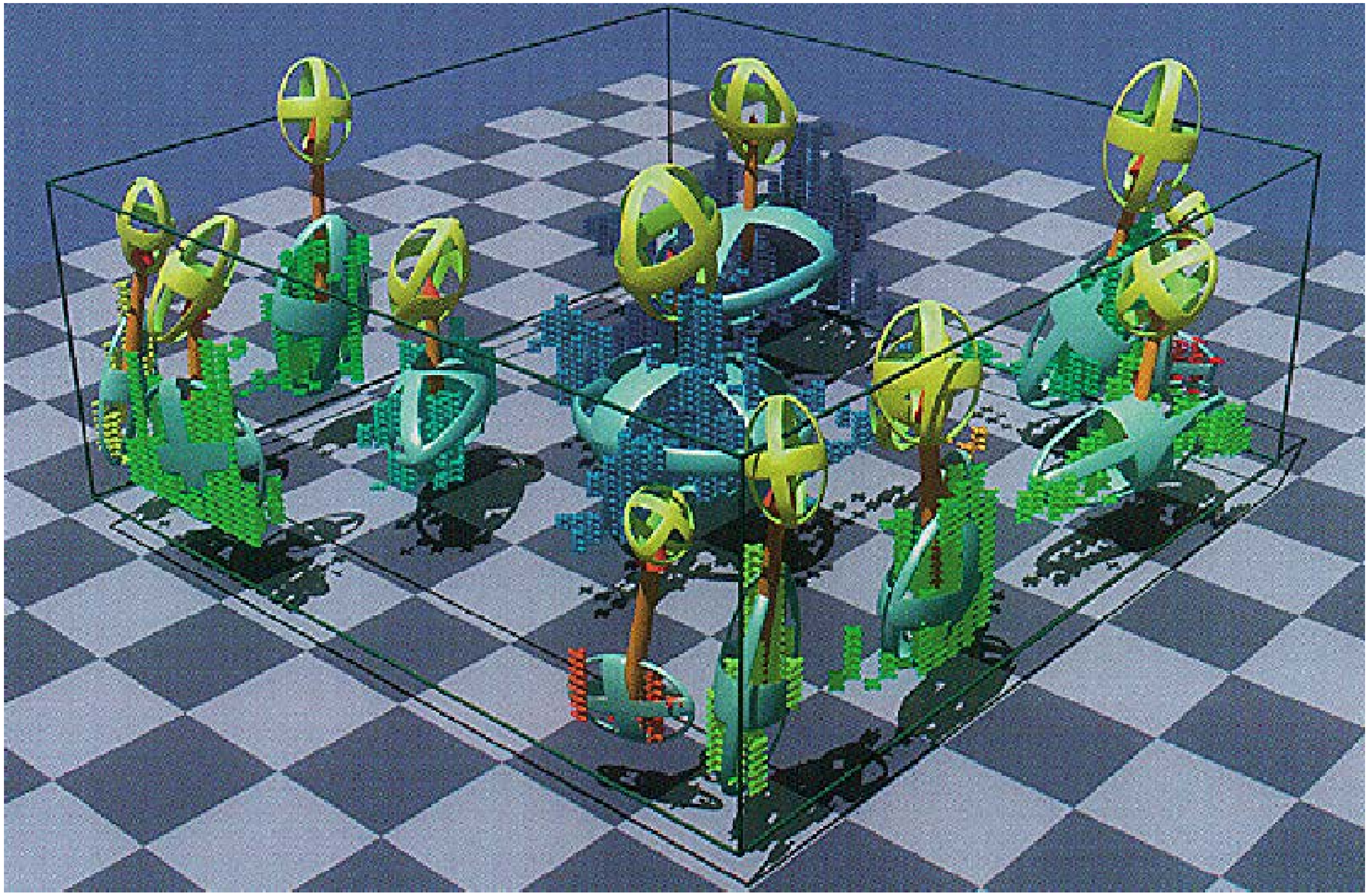
•plot 3D vector glyphs on it

•in our example, we don't use color-mapping of velocity (why?)

Observations

•glyphs near-tangent to our surface (why?)

Pushing vector glyphs to the limit

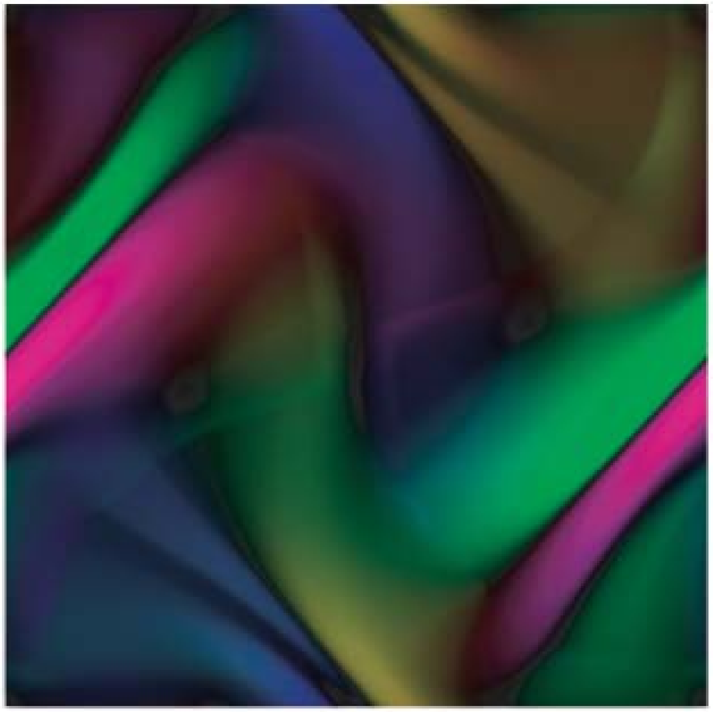


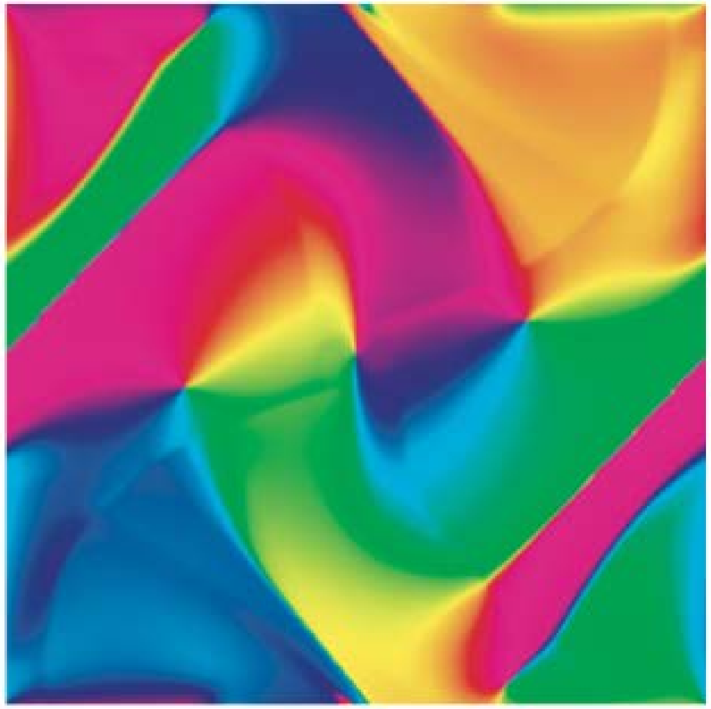
Average velocity (arrow) and velocity distribution (ellipsoids) for fluid regions with high reaction speed (voxel selection)

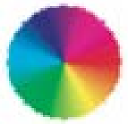
•3*3+3*3+3+1 values per glyph

•nice try, but glyphs are very large → few sample points

Vector color coding


magnitude=luminance


constant luminance (direction coding only)


direction color wheel

Reduce vector data to scalar data (using HSV color model)

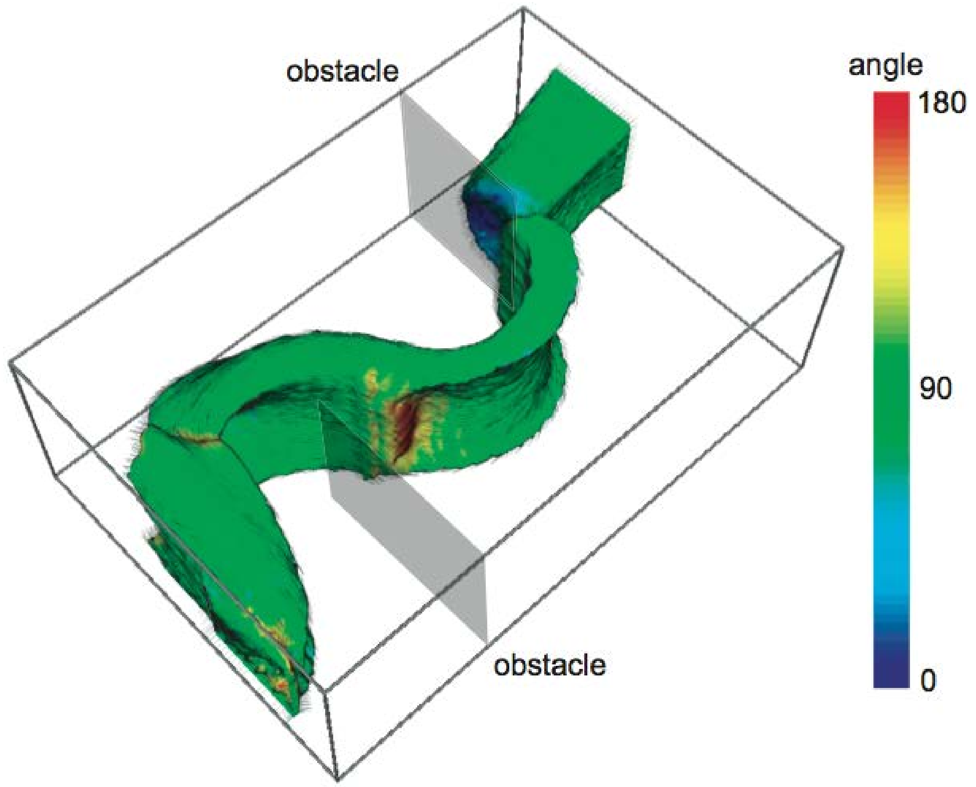
•direction = hue

•magnitude = luminance (optional)

•no occlusion/interpolation problems…

•…but images are highly abstract (recall: we don't naturally *see* directions)

Vector color coding



color = angle between vector field and normal of some given surface

See if vectors are tangent to some given surface

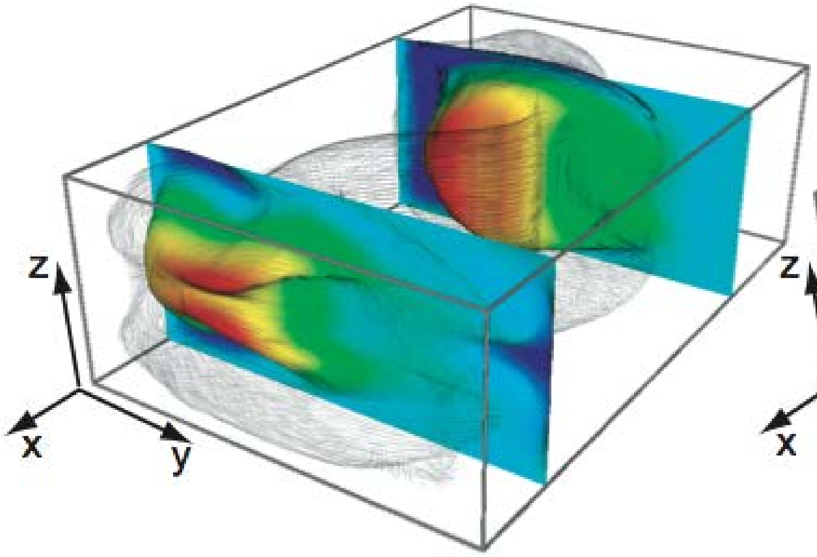
- color-code angle between vector and surface normal
- easily spot
    - tangent regions (flow stays on surface, green)
    - inflow regions (flow enters surface, red)
    - outflow regions (flow exits surface, blue)

Displacement plots (also called warp plots)

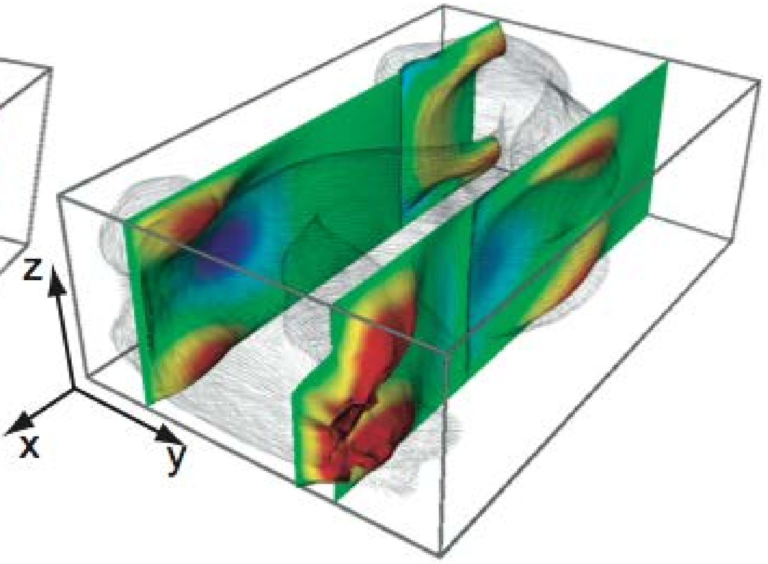
Show motion of a 'probe' surface in the field

•define probe surface $S \subseteq D$

•create displaced surface $S_{displ} = \{x + \mathbf{v}(x)\Delta t,\ \forall x \in S\}$



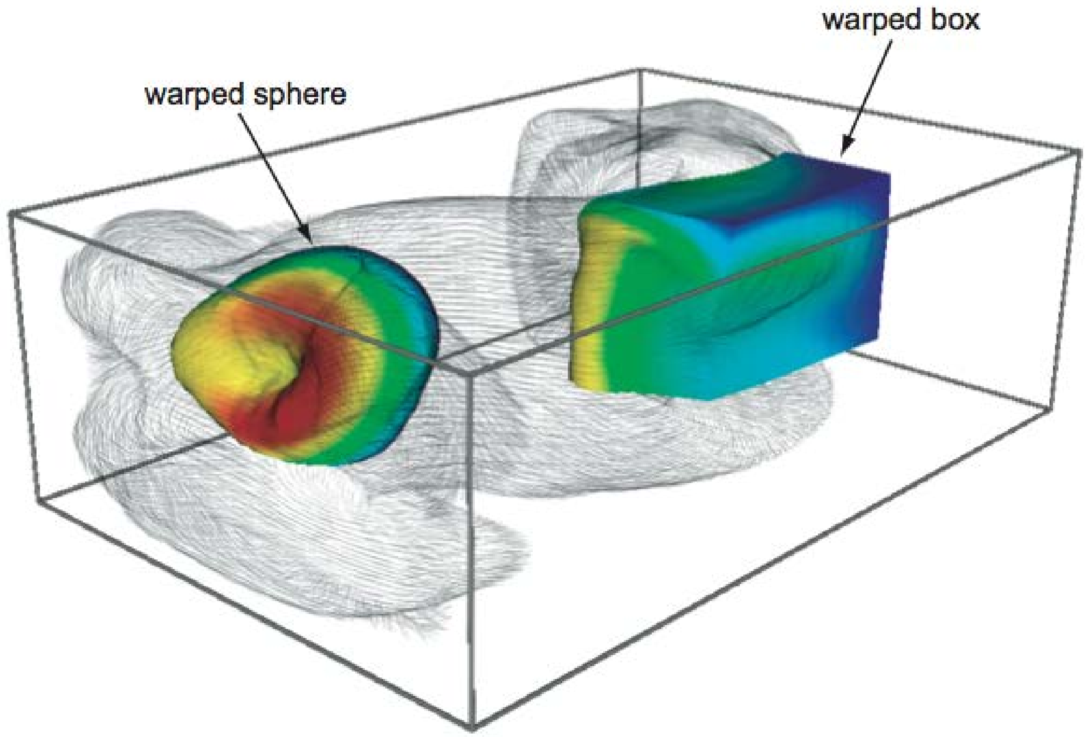
two displacement surfaces
orthogonal to x axis



two displacement surfaces
orthogonal to y axis

- analogy: think of a flexible sheet bent into the wind
- color can map additional scalar
- robust extension: $S_{displ} = \{x + (\mathbf{v}(x)\mathbf{n}(x))\mathbf{n}(x)\Delta t,\ \forall x \in S\}$
  - removes tangential displacements

Displacement plots



we can displace any kind of surface

Added value

•see what a *specific* shape becomes like when warped in the vector field

Limitations

•cannot use too high displacement factors Δt
•self-intersections can occur
•we must choose an initial surface to warp (‘seeding problem’)

Stream objects

Main idea

•think of the vector field $\mathbf{v} : D$ as a flow field
•choose some ‘seed’ points $s \in D$
•move the seed points s in $\mathbf{v}$
•show the trajectories

Stream lines

•assume that $\mathbf{v}$ is not changing in time (stationary field)
•for each seed $p_o \in D$

- the streamline S seeded at p_o is given by

$$S = \{p(\tau), \tau \in [0, T]\}, p(\tau) = \int_{t=0}^{T} \mathbf{v}(p)dt, \quad \text{where } p(0) = p_0$$

integrate p_o in vector field $\mathbf{v}$ for time T

•if v is time dependent $\mathbf{v}=\mathbf{v}(t)$, streamlines are called particle traces

Stream objects

Practical construction

•numerically integrate

$$S = \{p(\tau), \tau \in [0,T]\}, p(\tau) = \int_{t=0}^{\tau} \mathbf{v}(p)dt, \quad \text{where } p(0) = p_0$$

•discretizing time yields

$$\int_{t=0}^{\tau} \mathbf{v}(p)dt = \sum_{i=0}^{\tau/\Delta t} \mathbf{v}(p_i)\Delta t \quad \text{where } p_i = p_{i-1} + \mathbf{v}_{i-1}\Delta t$$

(simple Euler integration)

•recall our discussion on interpolation and basis functions

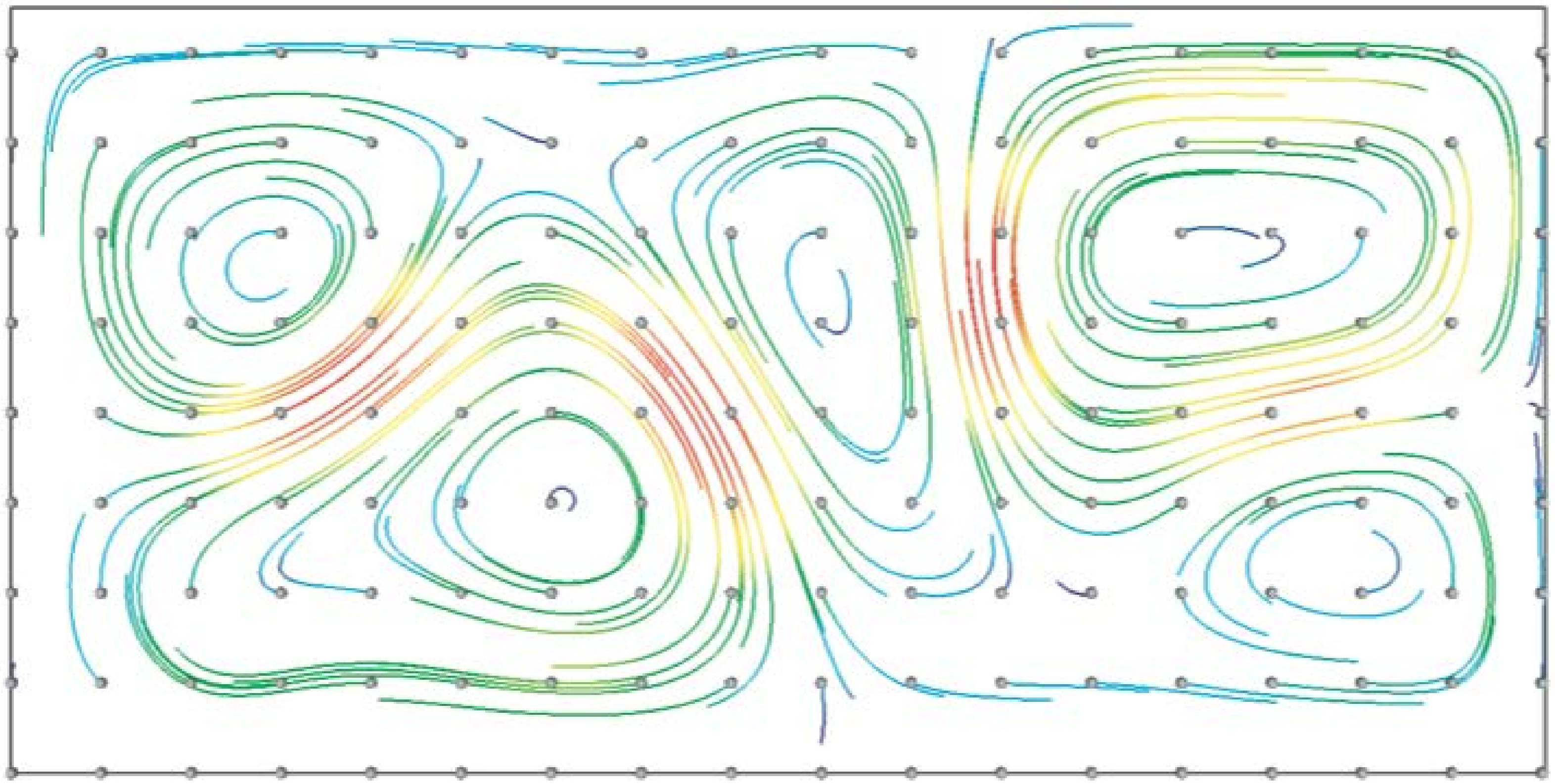
•Euler integration explained

- we consider $\mathbf{v}$ constant between two sample points p_i and $p_{\mathrm{i}+1}$
- we compute $\mathbf{v}(p)$ by linear interpolation within the cell containing p
- variant: use $\mathbf{v}(p)/\|\mathbf{v}(p)\|$ instead of $\mathbf{v}(p)$ in integral (why better?)
- S will be a polyline, $S = \{p_\mathrm{i}\}$

•stop when $\tau=T$ or $\mathbf{v}(p)=0$ or $p \notin D$

- what does $\tau=T$ mean when we use $\mathbf{v}(p)/\|\mathbf{v}(p)\|$?

Stream objects



streamlines: seeds from regular grid; use un-normalized **v** for integration; color by ||**v**||

Why is this better than vector glyphs?

- hint: do we have more or less intersections than for hedgehog plots? Why?
- hint: is the image more continuous? Why?

Good stream objects design

Coverage

- each dataset point should be close to a stream object
- why?
    - because we need to easily do the inverse mapping at any dataset point

Uniformity

- stream object density should be quasi-uniform
- why?
    - because we want to avoid high-clutter areas *and* no-information areas

Continuity

- long stream objects preferable to short ones
- why?
    - because we can easier follow few, long, objects than many short ones

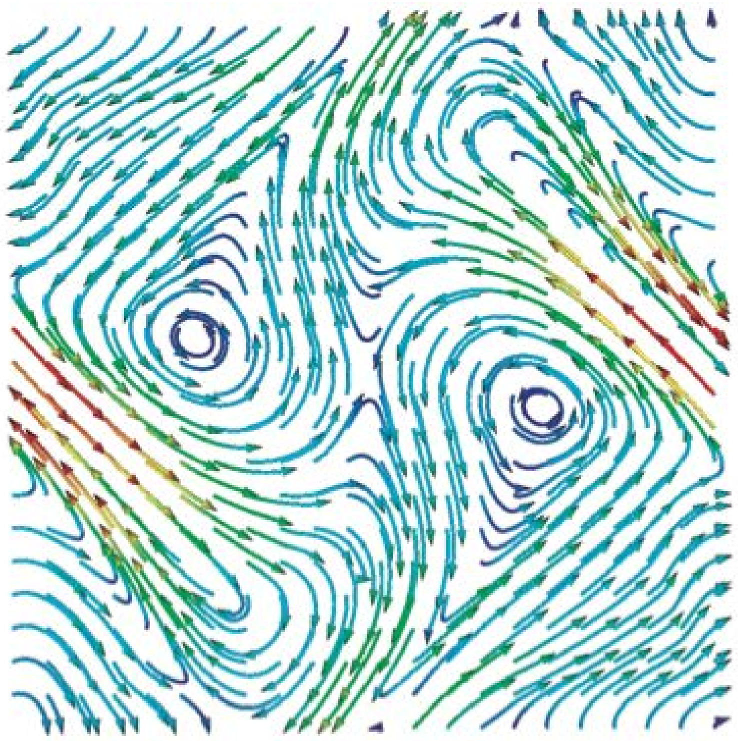
Note:

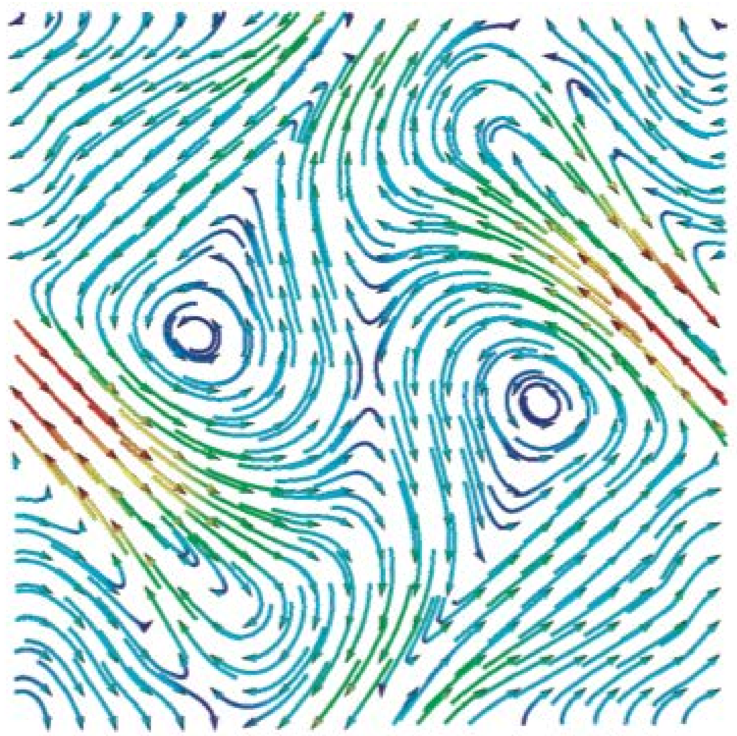
- all above can be seen as an *optimization process* on the seeds and integration tim
- however, efficient and robust solutions of this optimizations are generally hard

Stream tubes

Like stream objects, but 3D

- compute 1D stream objects (e.g. streamlines)
- sweep (circular) cross-section along these
- visualize result with shading


stream tubes, forward integration

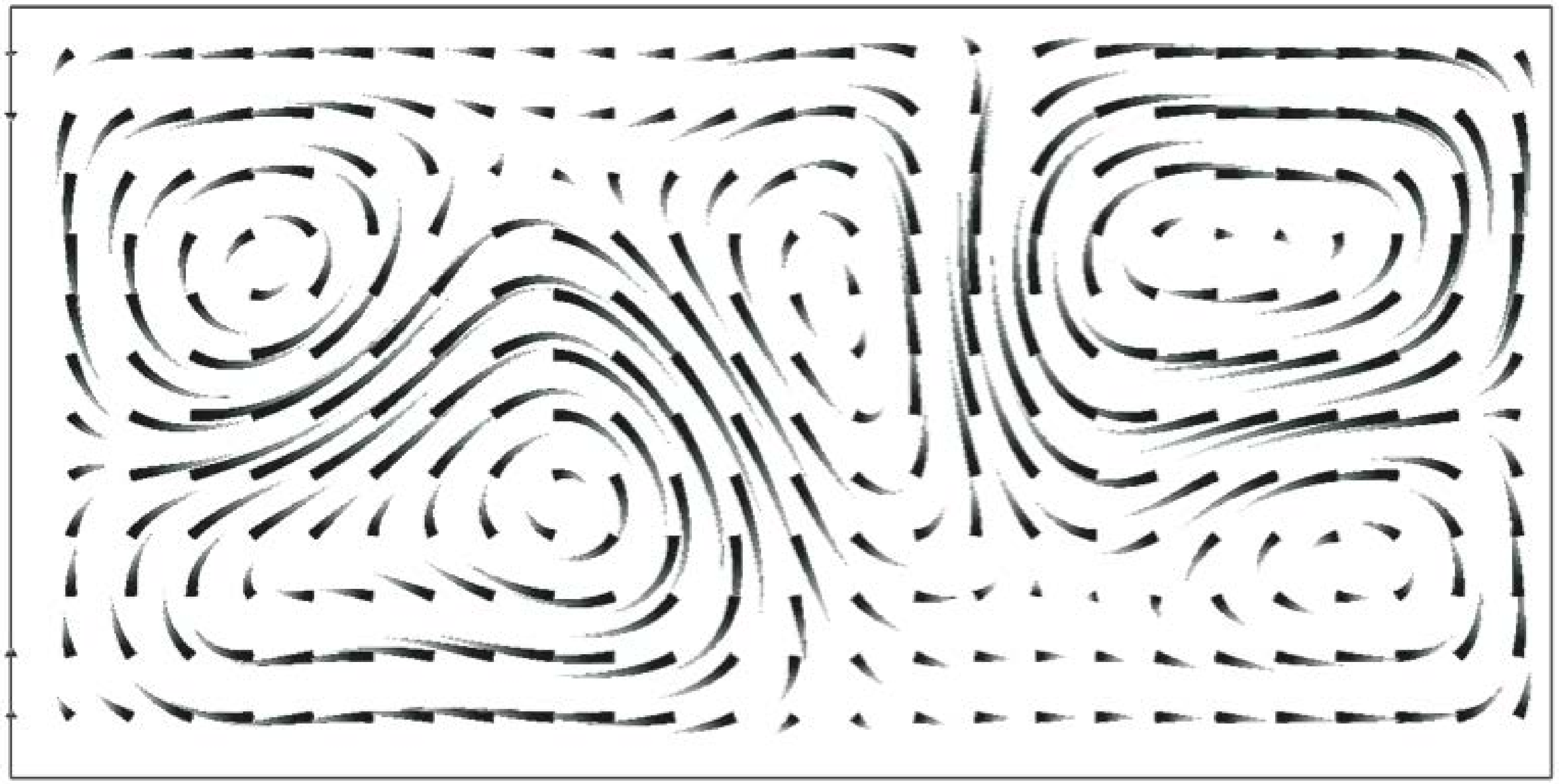

stream tubes, backward integration

- in 2D they are a nicer option than hedgehog/glyph plots

Stream tubes

Variations

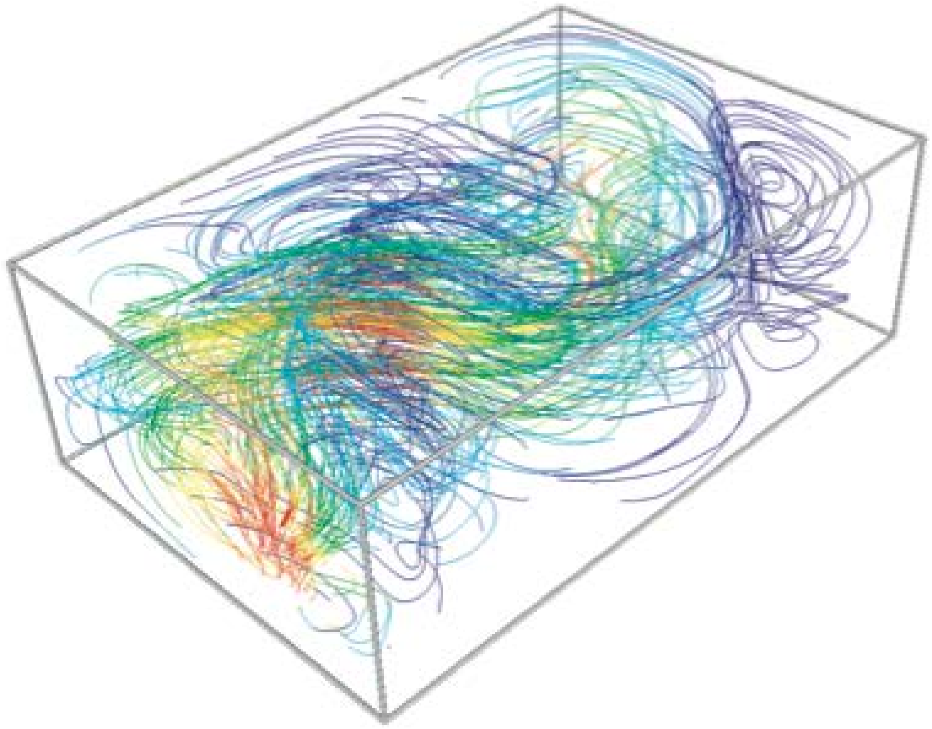
- modulate tube thickness by
    - data (we'll see this later in Module 5 – hyperstreamlines)
    - integration time – we obtain nice tapered arrows


stream tubes – radius *and* opacity decrease with integration time

Stream lines in 3D

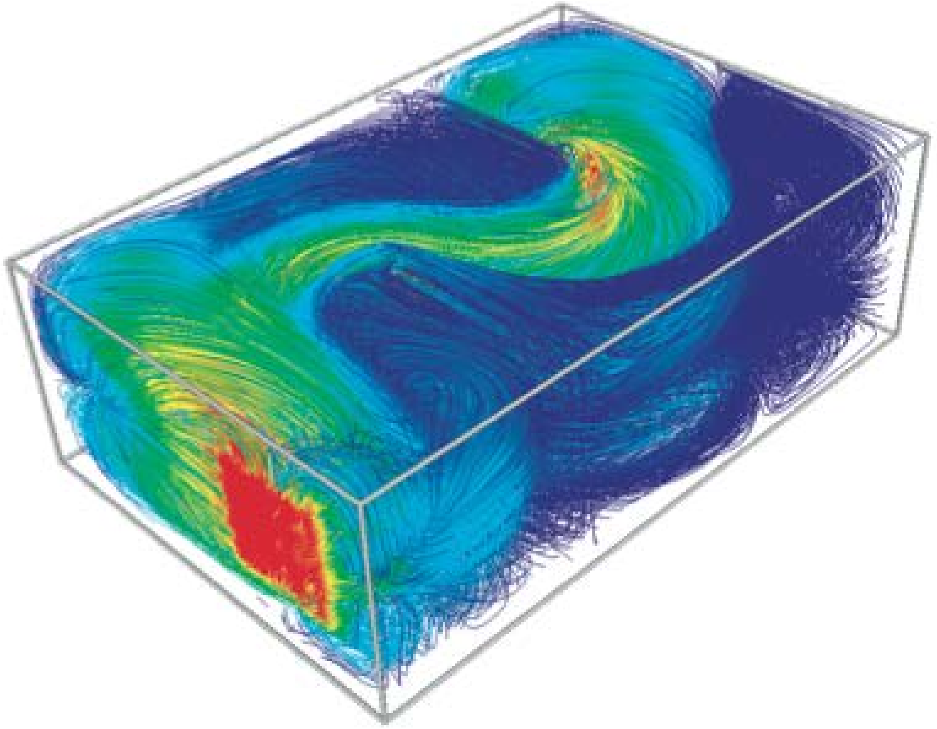
Tough problem

• more lines, so increased occlusion/clutter



undersampling 10x10x10, opacity=1

- not too much occlusion
- but little insight in the flow field



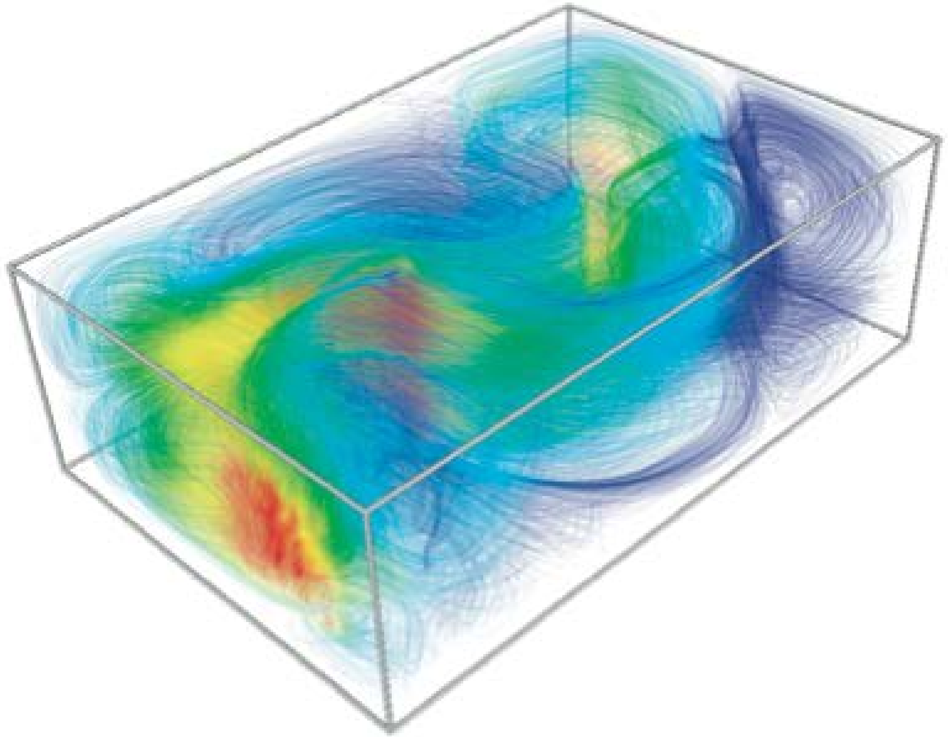
undersampling 3x3x3, opacity=1

- more local insight (better coverage)
- but too much occlusion

Stream lines in 3D

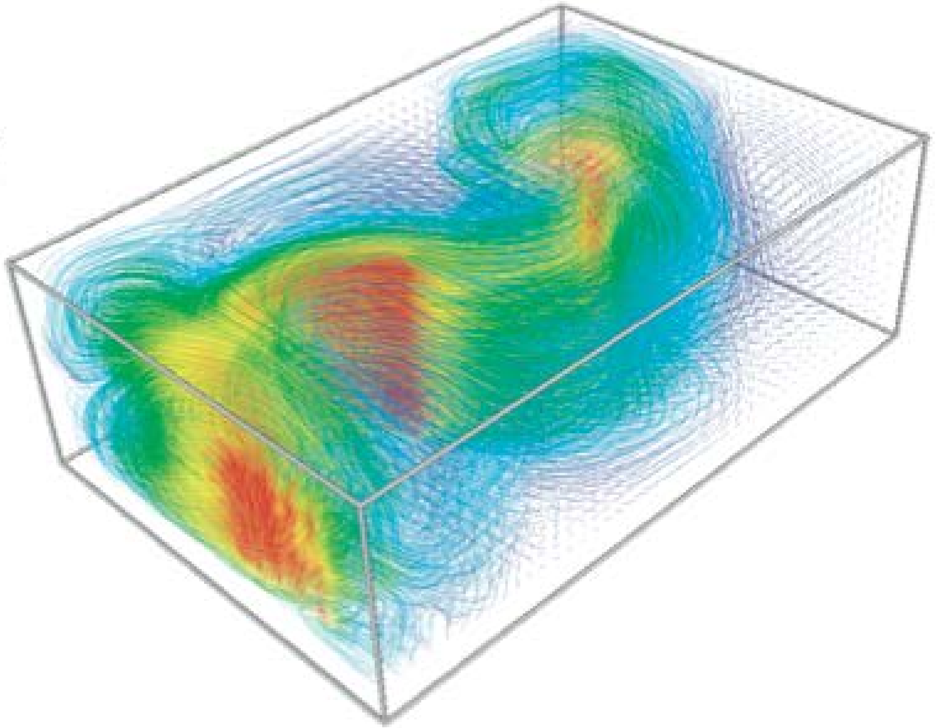
Variations

• play with opacity, seeding density, integration time



undersampling 3x3x3, opacity=0.1

- less occlusion (see through)
- good coverage

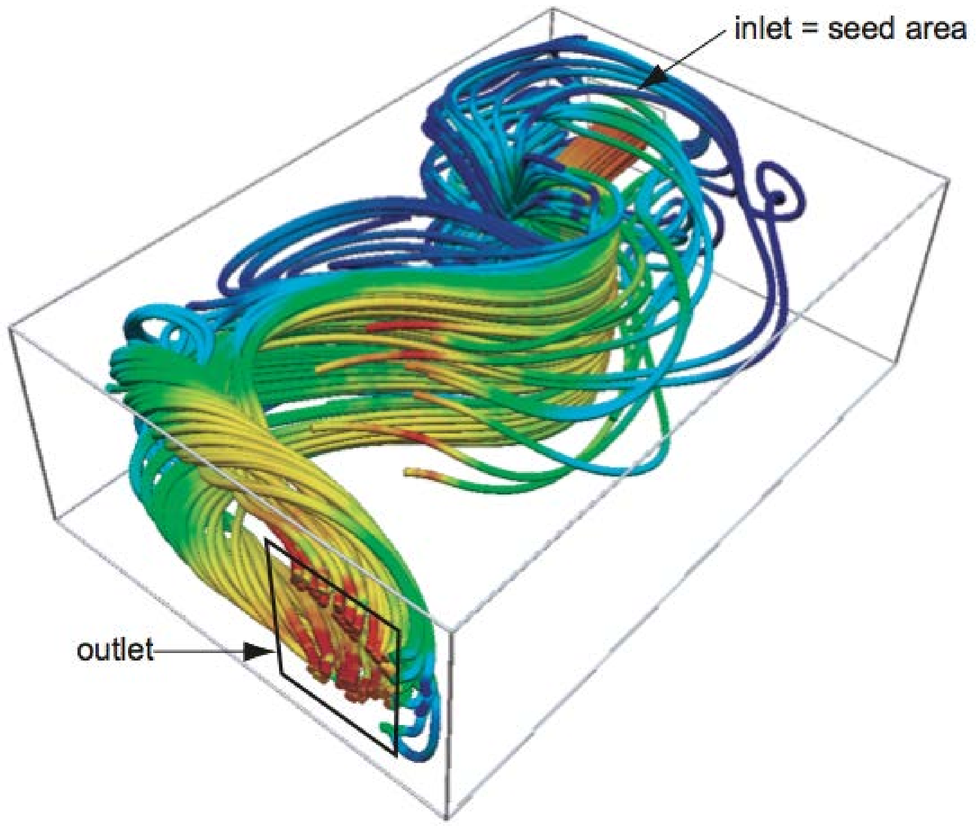


undersampling 3x3x3, shorter time

- more local insight (better coverage)
- even less occlusion
- but less continuity

Stream tubes in 3D

- even higher occlusion problem than for 3D streamlines
- must reduce number of seeds

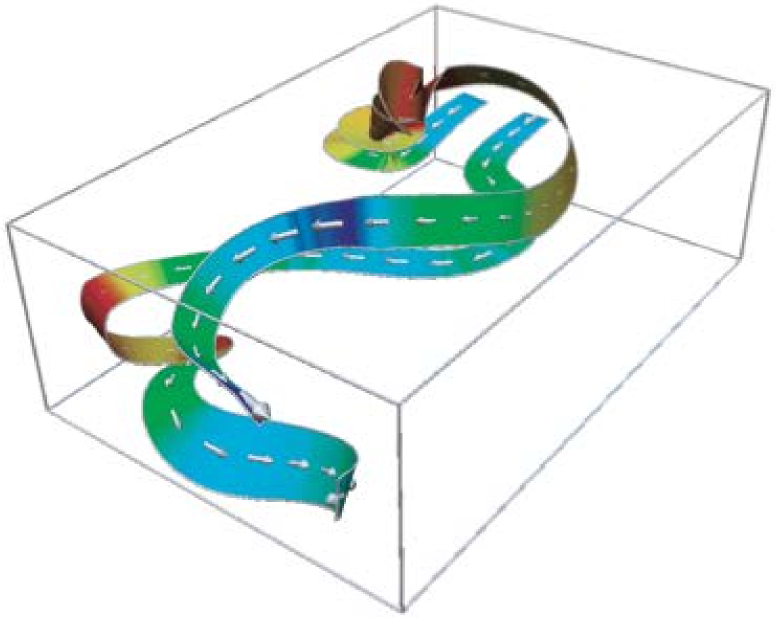


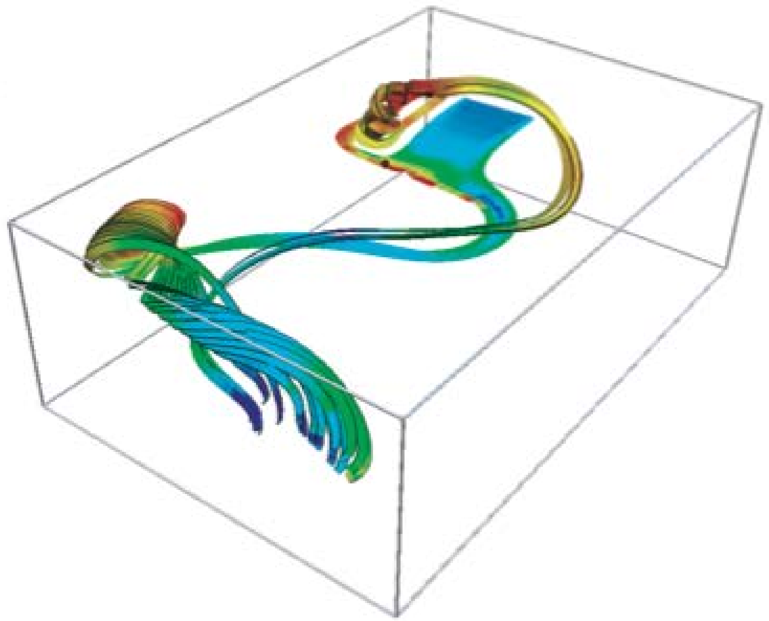
stream tubes traced from inlet to outlet

- show where incoming flow arrives at
- color by flow velocity
- shade for extra occlusion cues

Stream ribbons

- visualize how the vector field ‘twists’ around itself as it advances in space
- visualizes the so-called *helicity* of a vector field


stream ribbons: two thick ribbons


stream ribbons: 20 thin ribbons

Algorithm

- define pairs of close seeds (p_a, p_b)
- trace streamlines S_a, S_b from (p_a, p_b)
- construct strip surface connecting closest points on S_a, S_b

Image-based vector field visualization

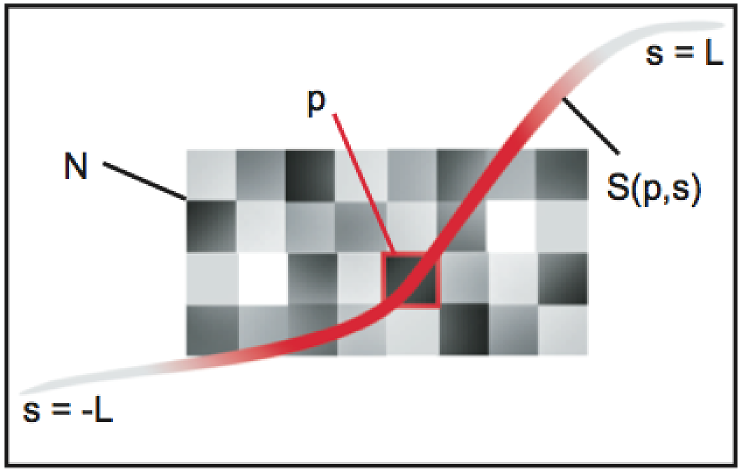
So far

- we had discrete visualizations (glyphs, streamlines, stream ribbons, warp plots)

Now

- we want a dense, pixel-filling, continuous, vector field visualization

Principle

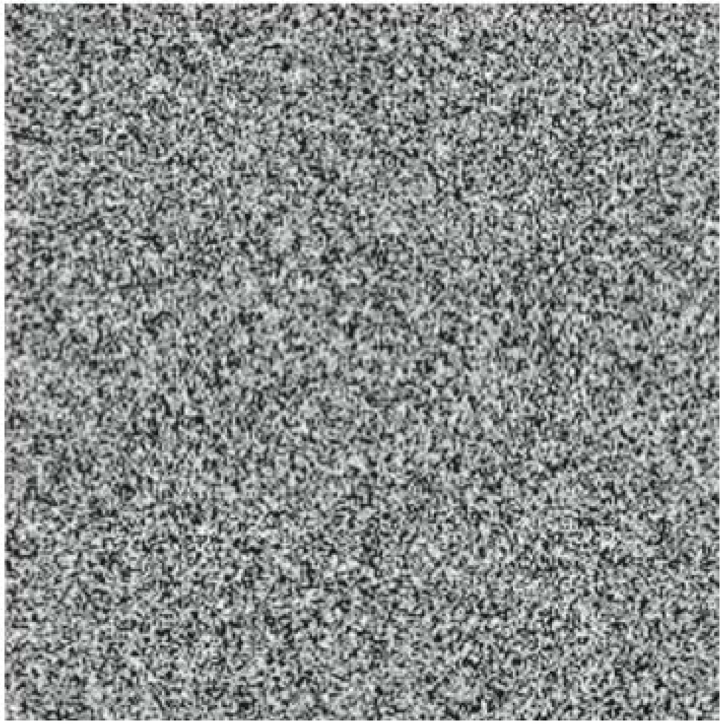


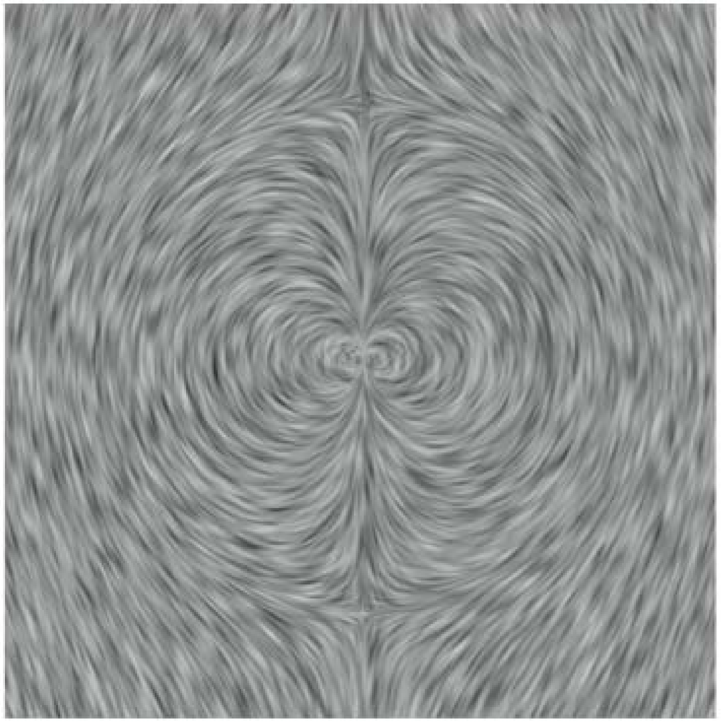
$$T(p) = \frac{\int_{-L}^{L} N(S(p,s))k(s)ds}{\int_{-L}^{L} k(s)ds}$$

gray value at pixel p
N = noise texture

- take each pixel p of the screen image
- trace a streamline from p upstream and downstream (as usual)
- blend all streamlines, pixel-wise
  - multiplied by a random-grayscale value at p
  - with opacity decreasing (exponentially) on distance-along-streamline from p
- identical to *blurring* (convolving) noise along the streamlines of $\mathbf{v}$

Image-based vector field visualization


noise texture


line integral convolution (LIC)

Line integral convolution

- highly coherent images along streamlines (why? because of $\mathbf{v}$-oriented blurring)
- highly contrasting images across streamlines (why? because of random noise)
- easy to interpret images

Image-based animated flow visualization

Main idea

•extend LIC with animation

•dynamics help seeing *orientation* and *speed* (not shown by LIC)

Algorithm

•consider a time-and-space dependent property $I : D \times \mathbf{R}_+ \rightarrow \mathbf{R}_+$ (e.g. gray value)

•advect I in time over D

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

•…and also inject some noise at each point of D

$$I(x+\mathbf{v}(x,t)\Delta t, t+\Delta t) = (1-\alpha)I(x,t) + \alpha N(x+\mathbf{v}(x,t)\Delta t, t+\Delta t)$$

advected term

injected noise term

balance between advection and noise injection

Image-based animated flow visualization

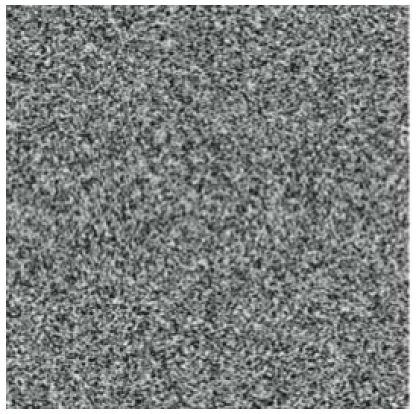
Animation

- now, make $N(\mathrm{x},\mathrm{t})$ a
  - *periodic* signal in time
  - but spatially *random* signal

$$N'(x,t) = f((t + N(x)) \bmod 1)$$

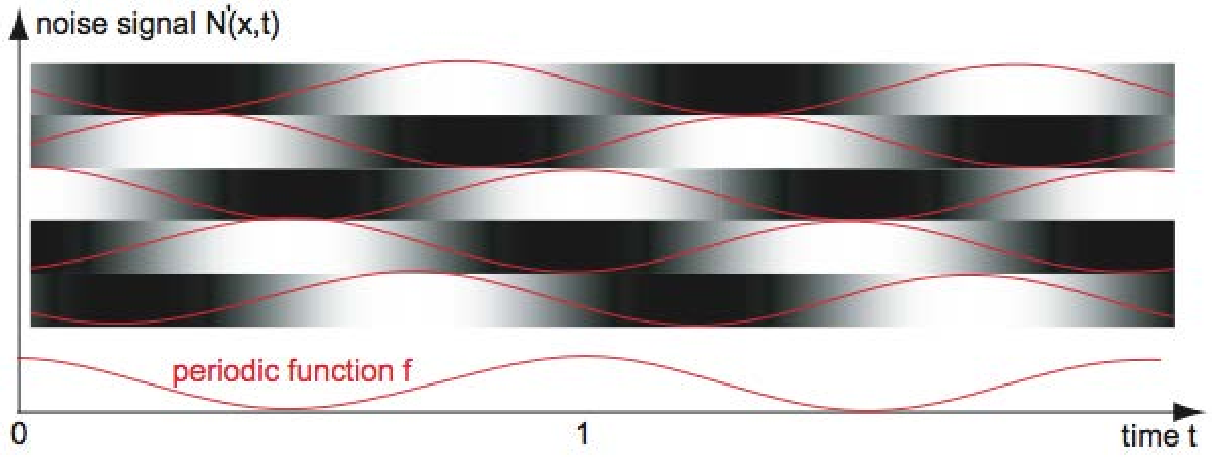
this is the purely spatial random noise like in LIC:

$N(\mathrm{x})$



$f : \mathbb{R}_+ \to [0,1]$

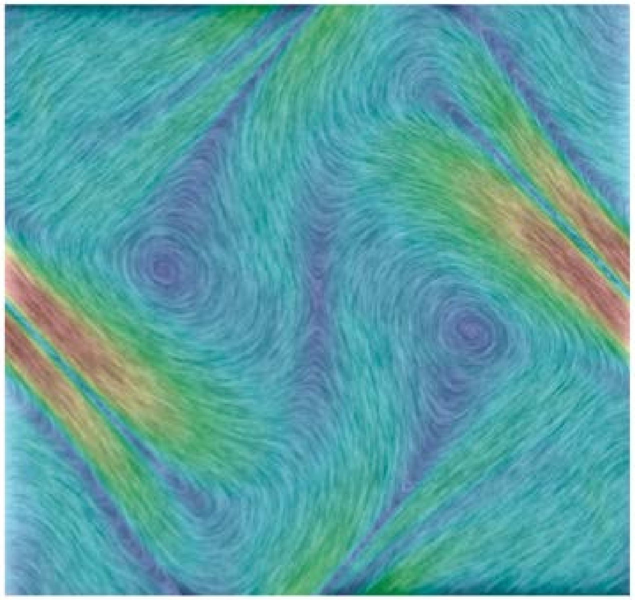
is a time-periodic function with period 1

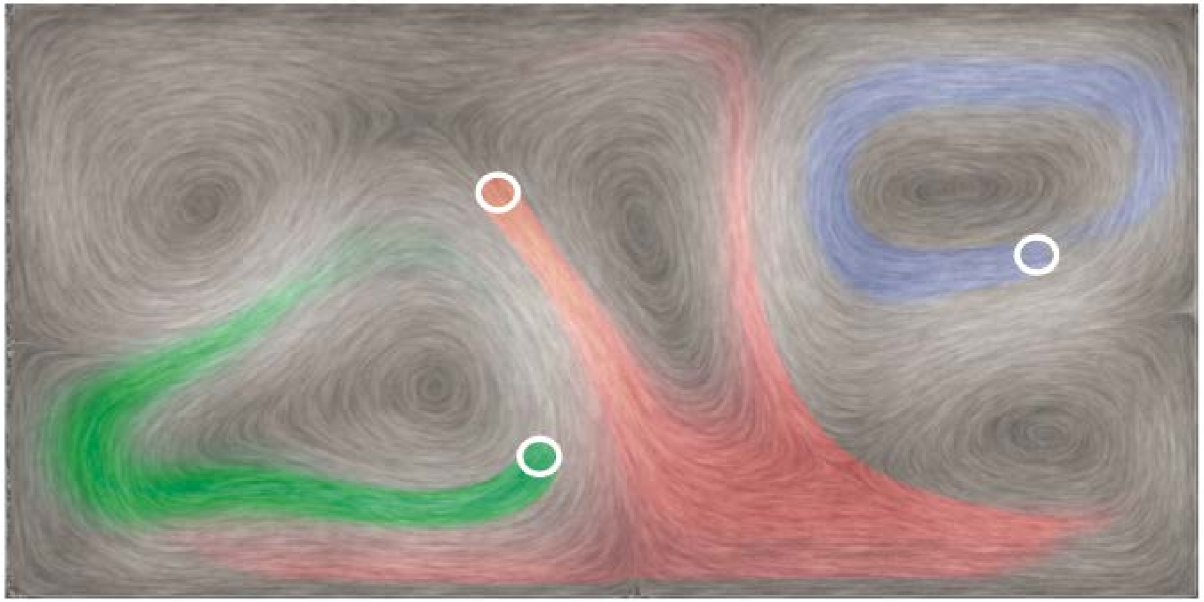


Think of

- N as the phase of the noise
- f as the time-period of the noise

Image-based flow visualization (IBFV)


IBFV, velocity color-coded


IBFV, with user-placed colored ink seeds
and luminance-coded velocity magnitude

Implementation

•sounds complex, but it's really easy☺ (200 LOC C with OpenGL, see Listing 6.2)

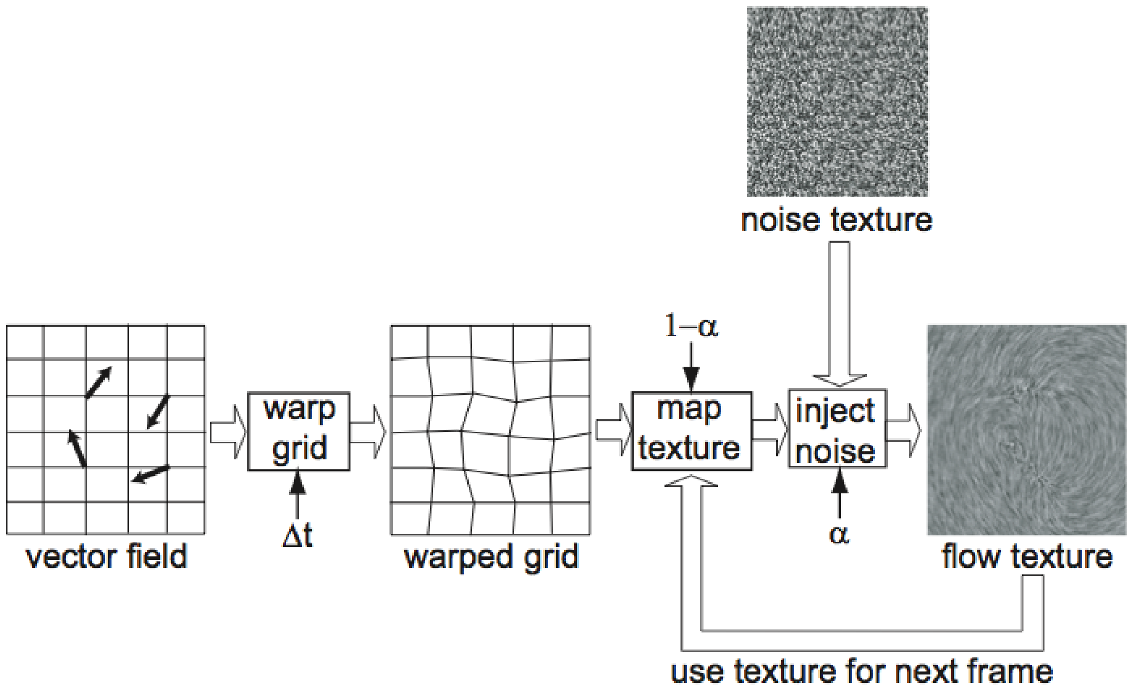
- see next slide for details

•real-time (hundreds of frames per second) even for modest graphics cards

•naturally handles time-dependent vector fields

Image-based flow visualization (IBFV)

Implementation

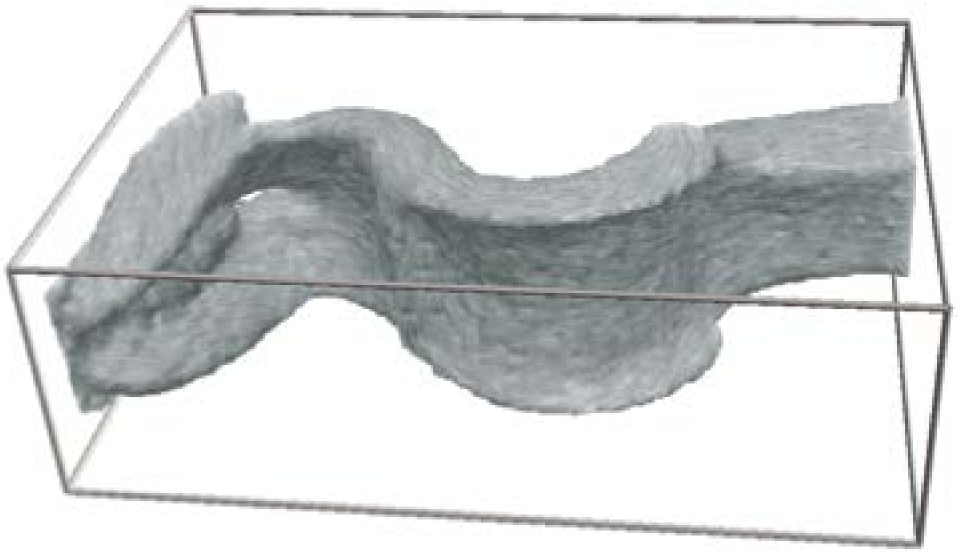


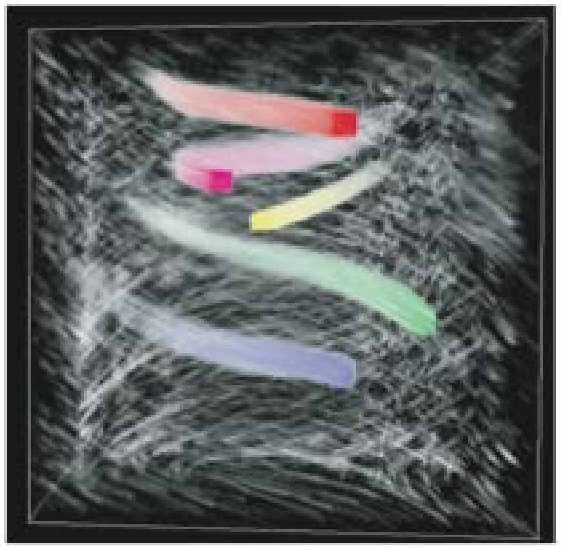
•define grid on 2D flow domain D
•warp grid D along $\mathbf{v}$ into D_{warp}
•forever

- read current frame buffer into I
- draw D_{warp} textured with I (advection) with opacity $1\text{-}\alpha$
- blend noise texture N' atop of I (injection) with opacity α

Image-based flow visualization (IBFV)

Variants on 3D curved surfaces and 3D volumes


IBFV on curved surfaces


IBFV in 3D volumes

Curved surfaces

•basically same as in planar 2D, just some implementation details different

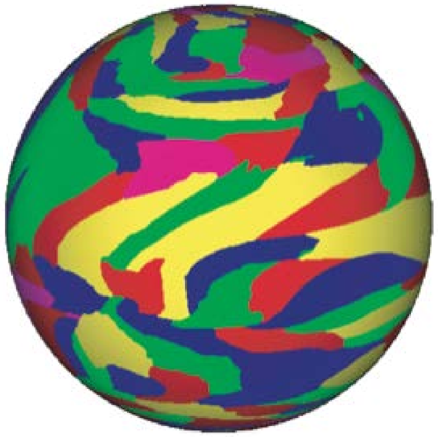
3D volumes

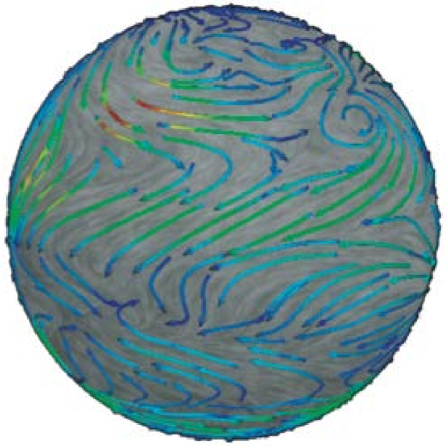
•must do something to ‘see through’ the volume
•use an ‘opacity noise’ (similarly injected as grayvalue noise)
•effect: similar to snowflakes drifting in wind on a black background

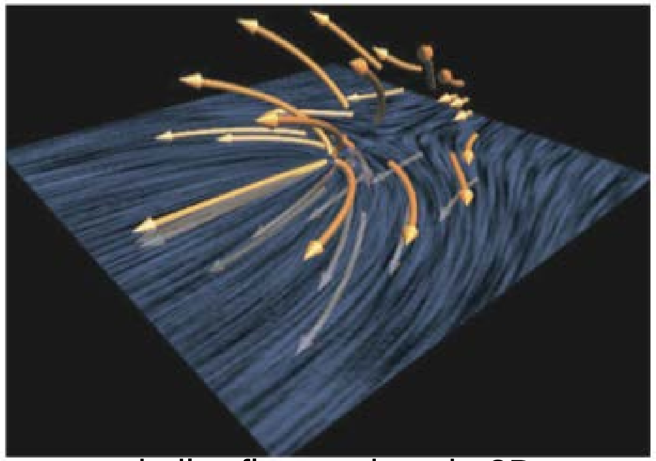
Advanced vector field visualization

Decomposition

- find areas in dataset domain D having similar-direction vectors v
- visualize these areas as compact regions
    - thus, easily identify same-flow areas


similar-flow regions
on Earth surface


one streamline per
similar-flow region

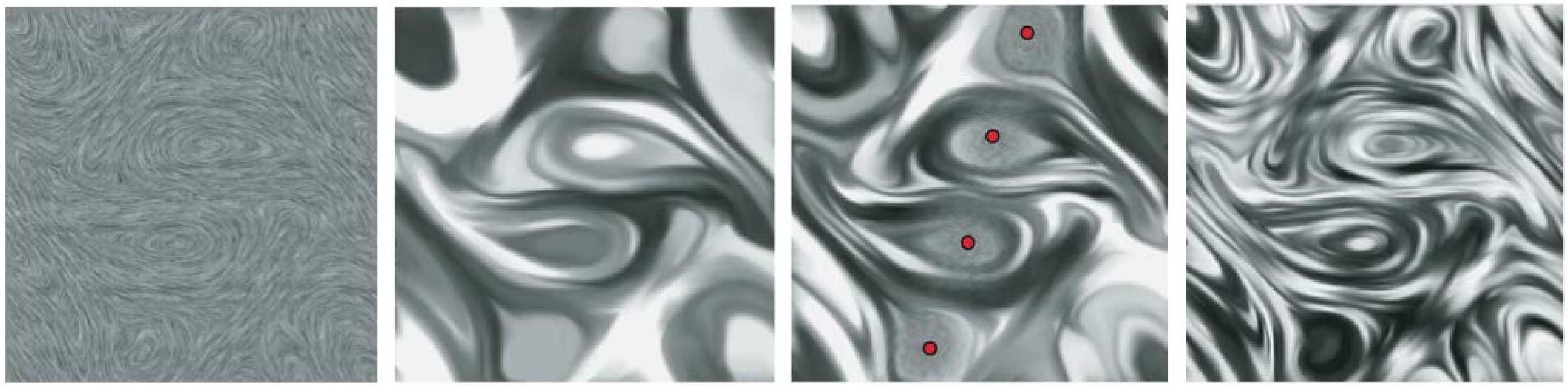

similar-flow regions in 3D
(laminar flow bouncing against a ball)

Algorithms

- cluster dataset points bottom-up based on vector field direction similarity
- same idea as for image segmentation, but using vector rather than color data

Advanced vector field visualization

Multiscale IBFV



- apply IBFV, but use vector-field-aligned noise patterns on multiple scales
    - build such patterns upfront by vector field decomposition (see prev. slide)

Results

- like IBFV, but user can choose scale (coarseness) of patterns
- shows animated flow in a *simplified* way

Details: See book, Sec. 6.7.2

Summary

Vector field visualization (book Chapter 6)

- fundamentally harder than scalar visualization
  - interpolation problem
  - 3D occlusion problem
  - seed placement problems

- methods
  - reduce vectors to scalars (divergence, gradient, vorticity, direction coding)
  - vector glyphs
  - displacement plots
  - stream objects (streamlines, stream ribbons)
  - image-based methods (LIC, IBFV)

Next module: Tensor visualization