Scientific Visualization Module 4
Vector Visualization
(adapted - some slides modified or omitted)

prof. dr. Alexandru (Alex) Telea

Department of Mathematics and Computer Science
University of Groningen, the Netherlands
The Visualization Pipeline - Recall

Dataset → Process → Dataset → Process → Dataset → Process → Dataset

Dataset

Process

Dataset

Process

Dataset

Process

Dataset

Process

Dataset

Process

Dataset

Process

Dataset

Process

Dataset

Process

Dataset

Process

Dataset

Process

Dataset

Process

Dataset

3D to 2D rendering

data formatting

data filtering

data mapping

raw data

imported dataset

enriched dataset

2D/3D shape

final image

f(x,y) → R^3

{0, 2, -5, ...}

import

filter

map

render

data acquisition

data enriching, transformation, resampling...

map abstract data to visual representations

draw visual representations

insight into the original phenomenon

measuring device or simulation

end user

www.cs.rug.nl/svcg
Vector algorithms (Chapter 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 \( v : D \rightarrow \mathbb{R}^n \)
- 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 $\mathbf{v} : \mathbb{R}^3 \rightarrow \mathbb{R}^3$, $\text{div } \mathbf{v} : \mathbb{R}^3 \rightarrow \mathbb{R}$ is

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

\[
\text{div } \mathbf{v} = \lim_{\Gamma \rightarrow 0} \frac{1}{|\Gamma|} \int_{\Gamma} (\mathbf{v} \cdot \mathbf{n}_\Gamma) ds
\]

$\text{div } \mathbf{v}$ is sometimes denoted as $\nabla \cdot \mathbf{v}$
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
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 \( \mathbf{v} : \mathbb{R}^3 \rightarrow \mathbb{R}^3 \), \( \text{rot} \ \mathbf{v} : \mathbb{R}^3 \rightarrow \mathbb{R}^3 \) 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

\[
\text{rot} \ \mathbf{v} = \lim_{\Gamma \to 0} \frac{1}{|\Gamma|} \int_{\Gamma} \mathbf{v} \cdot d\mathbf{s}
\]

- \( \text{rot} \ \mathbf{v} \) is locally perpendicular to plane of rotation of \( \mathbf{v} \)
- its magnitude: ‘tightness’ of rotation – also called vorticity

\( \text{rot} \ \mathbf{v} \) is sometimes denoted as \( \nabla \times \mathbf{v} \)
Curl

- compute using definition with partial derivatives
- visualize magnitude $|\text{rot } \mathbf{v}|$ 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
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 $(x, x + kv(x))$
  - optionally color map $||v||$ onto it

MHD simulation
256$^2$ grid

128$^2$ glyph grid

64$^2$ glyph grid
Vector glyphs

Observations
• trade-offs
  • more samples: more data points depicted, but more potential clutter
  • fewer samples: fewer 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

**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

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

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

128x85x42 volume field
456960 data points

100K subsamples
10K subsamples
3D vector glyphs

128x85x42 volume field
456960 data points

Alpha blending
- extremely simple and powerful tool
- reduce *perceived* occlusion
  - low-speed zones: highly transparent
  - high-speed zones: opaque and highly coherent

100K subsamples $\alpha=0.1$

100K subsamples $\alpha=0.1$
no color mapping
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

Observations
- glyphs near-tangent to our surface
Vector color coding

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

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 $S_{\text{displ}} = \{x + v(x)\Delta t, \forall x \in S\}$

- 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 by vector field component perpendicular to $S$
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 $\Delta t$
• self-intersections can occur
• we must choose an initial surface to warp (‘seeding problem’)

www.cs.rug.nl/svcg
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 \( p_0 \in D \)
  • the streamline \( S \) seeded at \( p_0 \) is given by

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

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

Practical construction

- numerically integrate

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

- discretizing time yields

\[ \int_{t=0}^{T} v(p) dt = \sum_{i=0}^{\tau/\Delta t} v(p_i) \Delta t \quad \text{where } p_i = p_{i-1} + v_{i-1} \Delta t \quad \text{(simple Euler integration)} \]

- Euler integration explained
  - we consider \( v \) constant between two sample points \( p_i \) and \( p_{i+1} \)
  - we compute \( v(p) \) by linear interpolation within the cell containing \( p \)
  - variant: use \( v(p)/\|v(p)\| \) instead of \( v(p) \) in integral
  - \( S \) will be a polyline, \( S = \{p_i\} \)
  - stop when \( \tau = T \) or \( v(p) = 0 \) or \( p \notin D \)
Streamlines

How do streamlines compare with vector glyphs?

• hint: do we have more or fewer 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 time
• however, efficient and robust solutions of these optimizations are generally hard

Details: See book, p. 184-185
Stream tubes

Like stream objects, but 3D

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

In 2D they are a nicer option than hedgehog/glyph plots
Stream tubes

Variations

• modulate tube thickness by
  • data
  • 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

**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

• 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 blurred (convolving) noise along the streamlines of $v$

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

gray value at pixel $p$
$N = $ noise texture
Image-based vector field visualization

**Line integral convolution**
- highly coherent images along streamlines (why? because of 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 \( I : D \times \mathbb{R}_+ \rightarrow \mathbb{R}_+ \) (e.g. gray value)
• advect \( I \) in time over \( D \)

\[
I(x + v(x, t) \Delta t, t + \Delta t) = I(x, t)
\]
• …and also inject some noise at each point of \( D \)

\[
I(x + v(x, t) \Delta t, t + \Delta t) = (1 - \alpha)I(x, t) + \alpha N(x + v(x, t) \Delta t, t + \Delta t)
\]

advedted term injected noise term

balance between advection and noise injection
Image-based animated flow visualization

Animation
  • now, make $N(x,t)$ a
    • periodic signal in time
    • but spatially random signal

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

this is the purely spatial random noise like in LIC:

Think of
  • $N$ as the phase of the noise
  • $f$ as the time-period of the noise
Image-based flow visualization (IBFV)

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

Variants on 3D curved surfaces and 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

Details: See book, p. 203-204
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: Tensor visualization