CS 519: Scientific Visualization

Vector Field Visualization: Flowlines Texture-Based Techniques

Eric Shaffer

Some slides adapted Alexandru Telea, *Data Visualization Principles and Practice*

Numerical Integration: Euler

- Generating a streamline can be done by numerically integrating a differential equation
- In his textbook *Institutionum Calculi Integralis*, Leonhard Euler proposed:

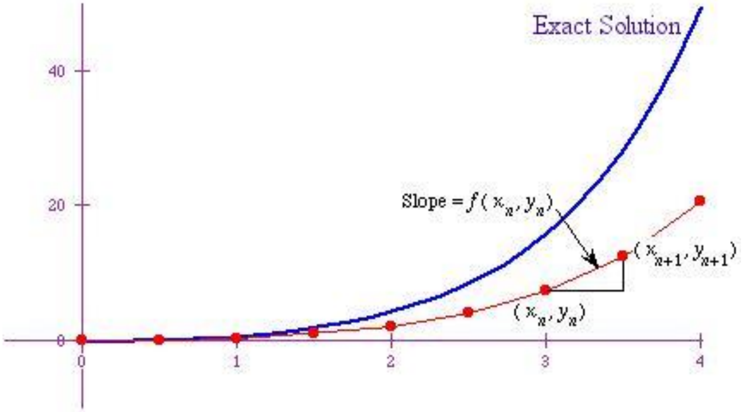
$$\mathbf{p}_{k+1} = \mathbf{p}_k + h\,\mathrm{v}(p_k)$$

- In this formulation, **p** is a point in d-dimensional space
- **v** is a vector-valued function that gives the velocity at a point
  - remember velocity = first derivative of a function specifying position
- *h* is a step-size
  - may be determined be the grid for sampled data

Numerical Integration: Euler

$$\mathbf{p}_{k+1} = \mathbf{p}_k + h\,\mathrm{v}(p_k)$$

Generates points along a solution curve to the differential equation



But there's error….

The error in each step moves you to different solution curve

Step-size controls the error

Error is on the order of *O(h)*

Runge Kutta Methods

- Used to solve Ordinary Differential Equations
- 4th-order Runge Kutta (RK4) preferred
- Second Order Runge-Kutta (Heun's Method)

$$\mathbf{p}_{\mathrm{n+1}} = \mathbf{p}_{\mathrm{n}} + \frac{1}{2}\mathbf{k}_1 + \frac{1}{2}\mathbf{k}_2$$

$$\mathbf{k}_1 = h\,\mathrm{v}\left(\mathbf{p}_{\mathrm{n}}\right)$$

$$\mathbf{k}_2 = h\,\mathrm{v}\left(\mathbf{p}_{\mathrm{n}} + \mathbf{k}_1\right)$$

- Error is $O(h^2)$

Fourth Order Runge-Kutta

Error O(h^4)· The general form of the equations:

$$\mathbf{p}_{n+1} = \mathbf{p}_n + \frac{h}{6}\left[\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4\right]$$

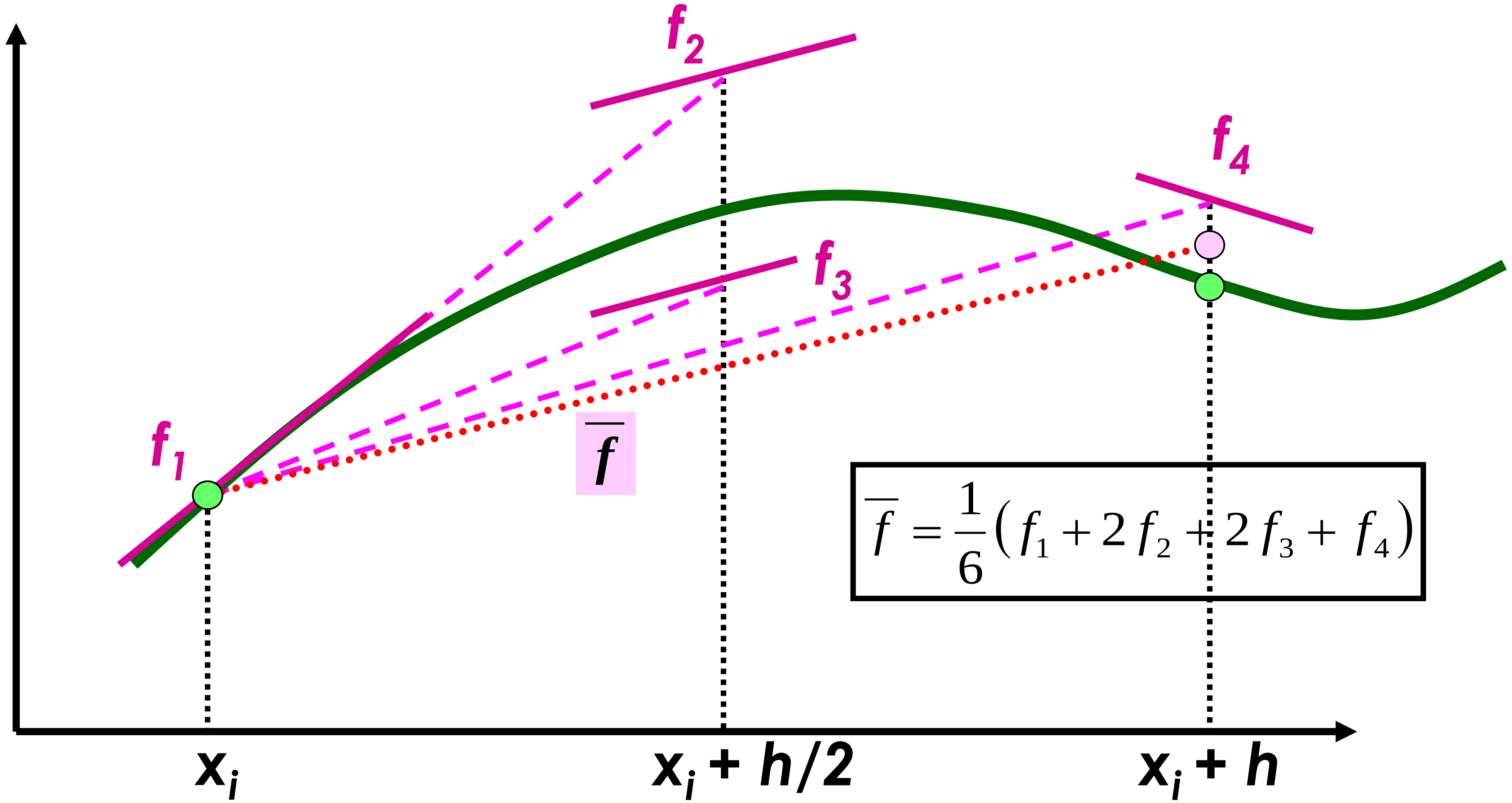
$$\mathbf{k}_1 = \mathbf{v}(\mathbf{p}_n)$$

$$\mathbf{k}_2 = \mathbf{v}\left(\mathbf{p}_n + \frac{h}{2}\mathbf{k}_1\right)$$

$$\mathbf{k}_3 = \mathbf{v}\left(\mathbf{p}_n + \frac{h}{2}\mathbf{k}_2\right)$$

$$\mathbf{k}_4 = \mathbf{v}(\mathbf{p}_n + h\mathbf{k}_3)$$

Fourth-Order Runge-Kutta

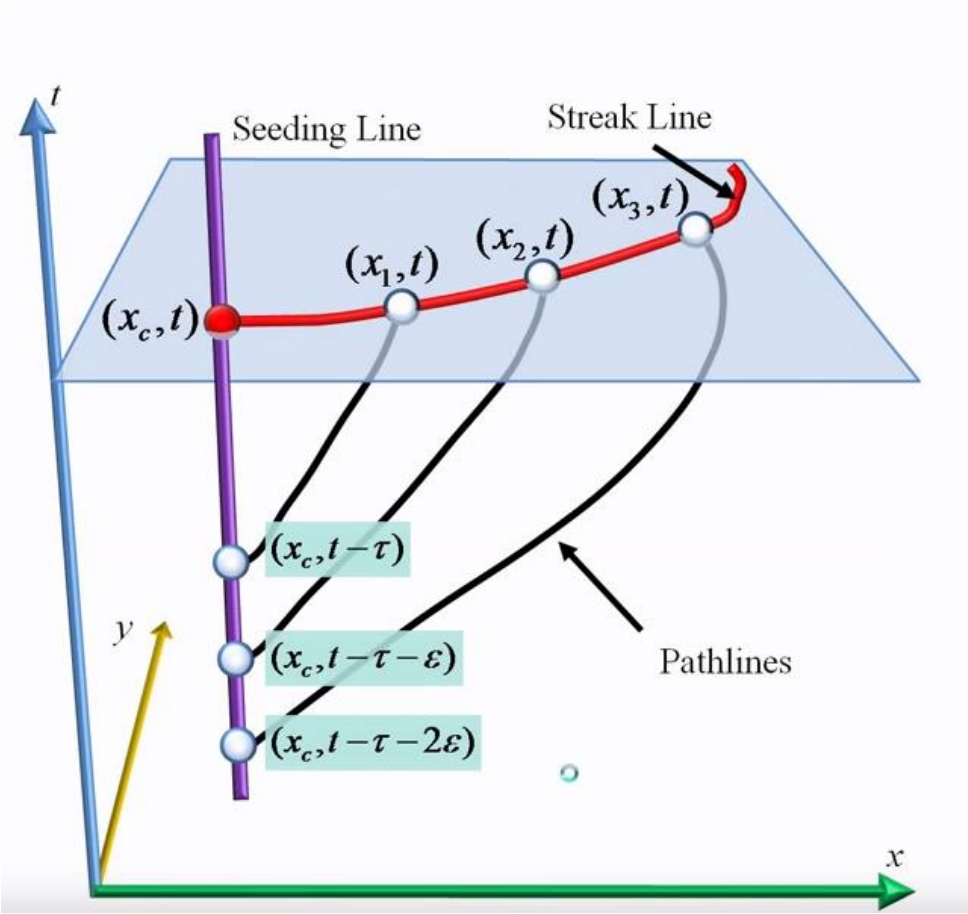


Streamlines versus Streaklines

- For steady-state vector fields, they are the same
- For unsteady flows (i.e. vector field changes over time):
  - Streamline: curve instantaneously tangent to the vector field
  - Pathline: the set of points a particle travels through as the field evolves
  - Streakline: imagine continuously injecting particles at a fixed point. Streakline is formed by the positions of the particles

  - Next slide:
    Streamlines are dashed
    Pathline is red
    Streakline is blue

Pathlines and Streaklines



Texture-based Methods

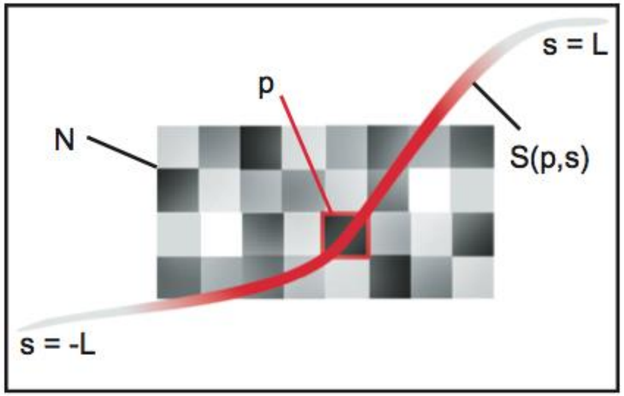
So far

- mostly discrete visualizations (glyphs, streamlines, stream ribbons)

Goal

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

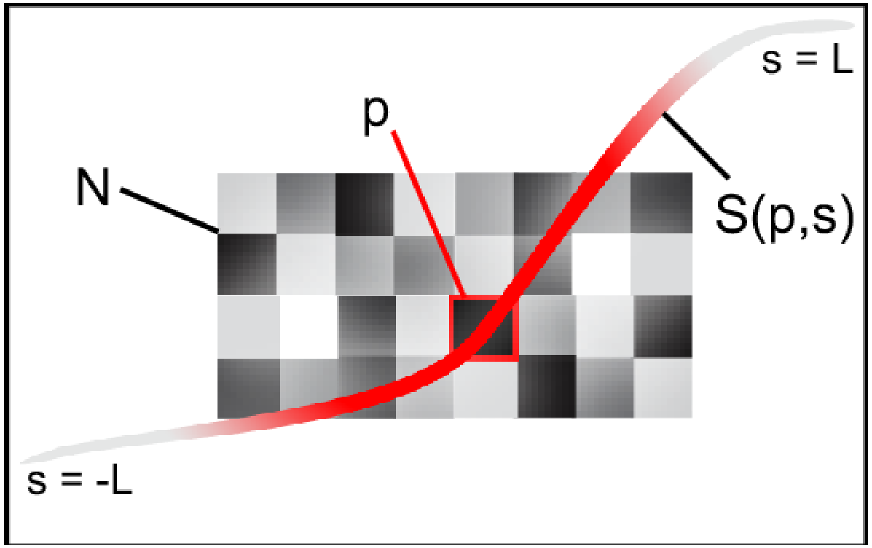
Principle



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

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* noise along the streamlines of $\mathbf{v}$



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

$$k(s) = e^{-s^2}$$

N : noise texture

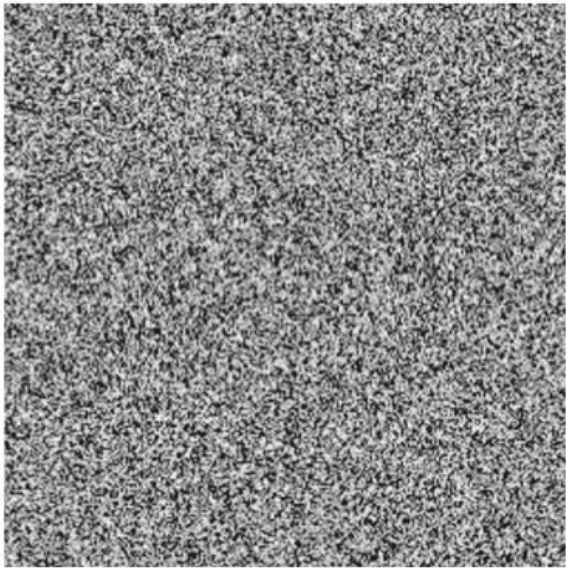
$S(p,s)$: streamline of seed point P

$k(s)$: weighting or blurring function

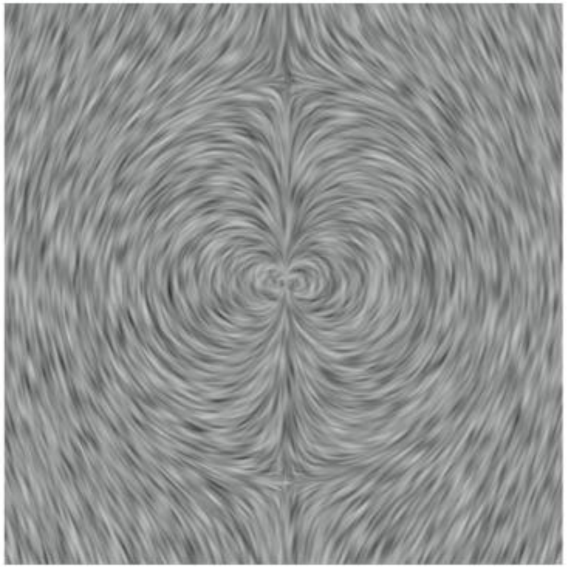
L : width of blurring function

LIC: Line Integral Convolution

Texture-based Methods



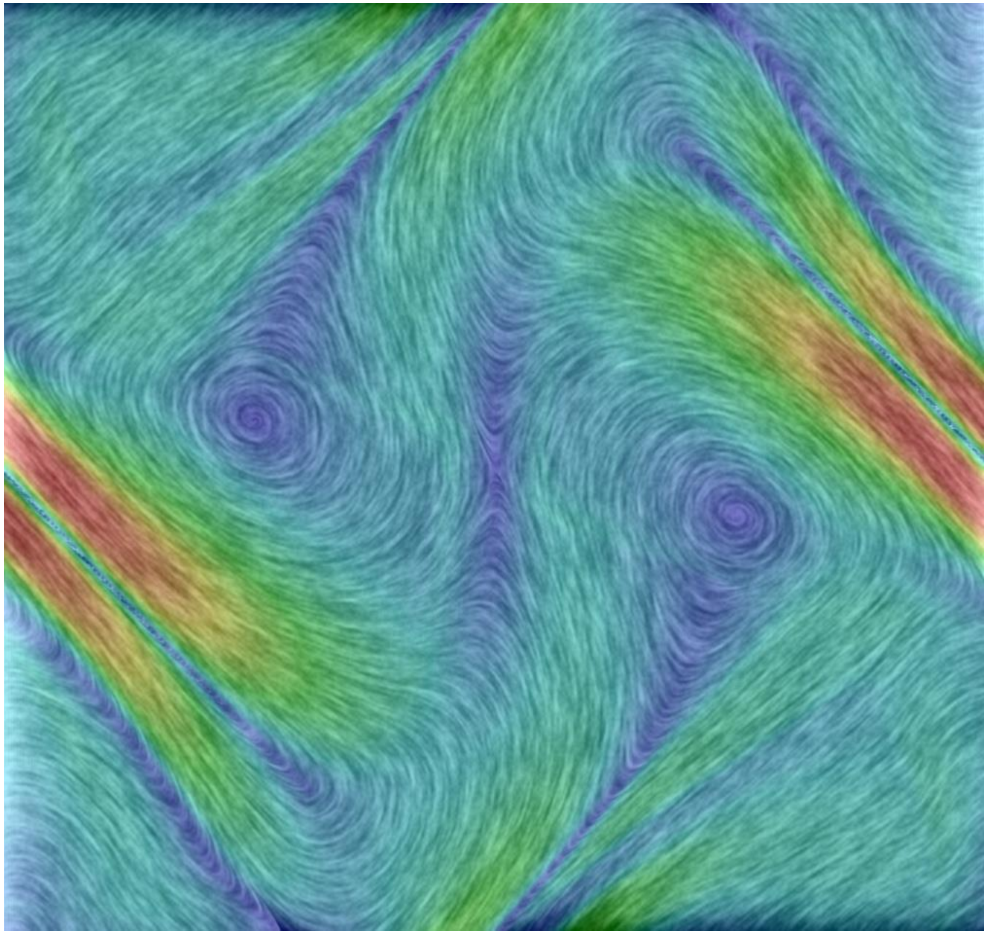
noise texture



line integral convolution (LIC)

Line integral convolution

- highly coherent images along streamlines (why?
- highly contrasting images across streamlines (why?



Vector magnitude: Color

Vector direction: Graininess

LIC Animation

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

LIC Animation

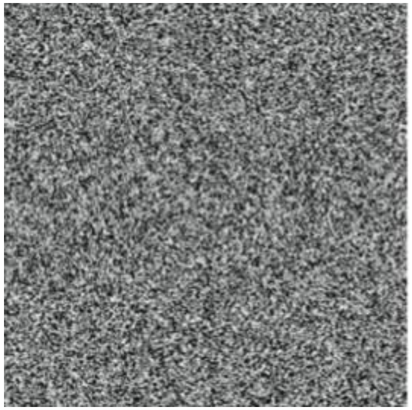
Animation

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

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

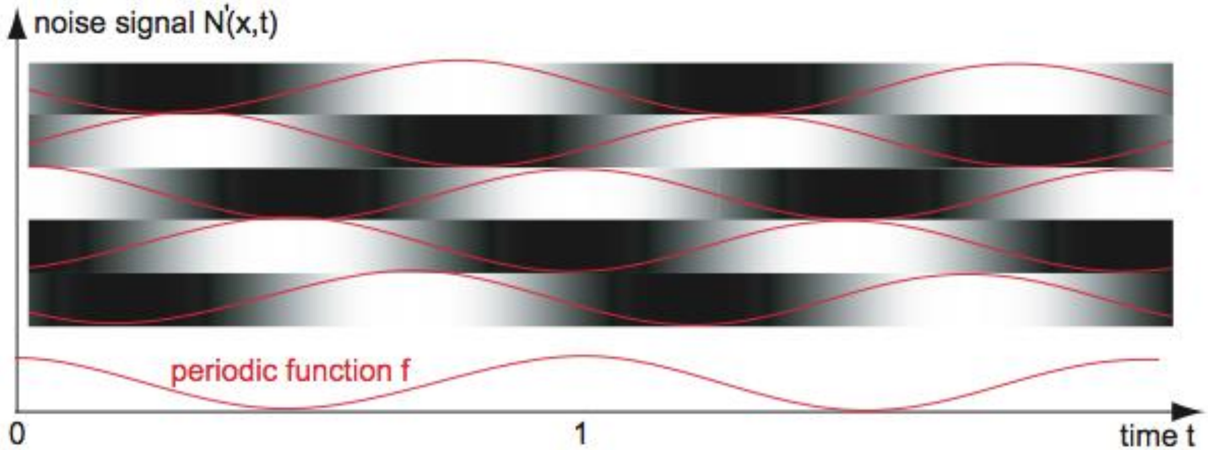
this is the purely spatial random noise like in LIC:

$N(x)$



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

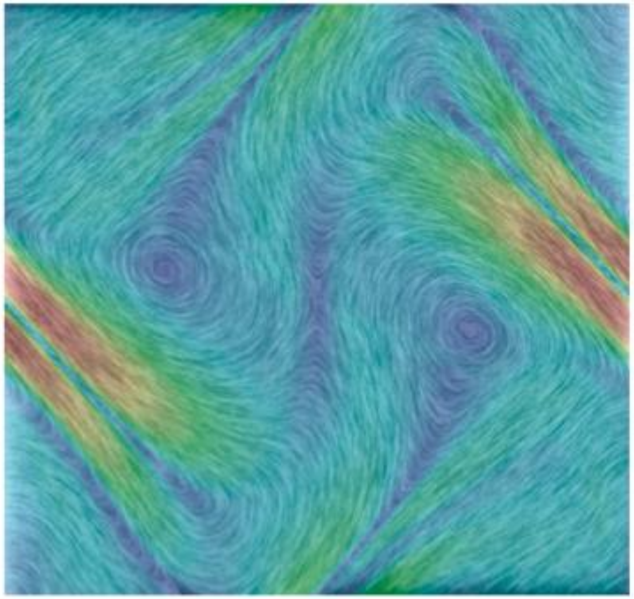
is a time-periodic function with period 1



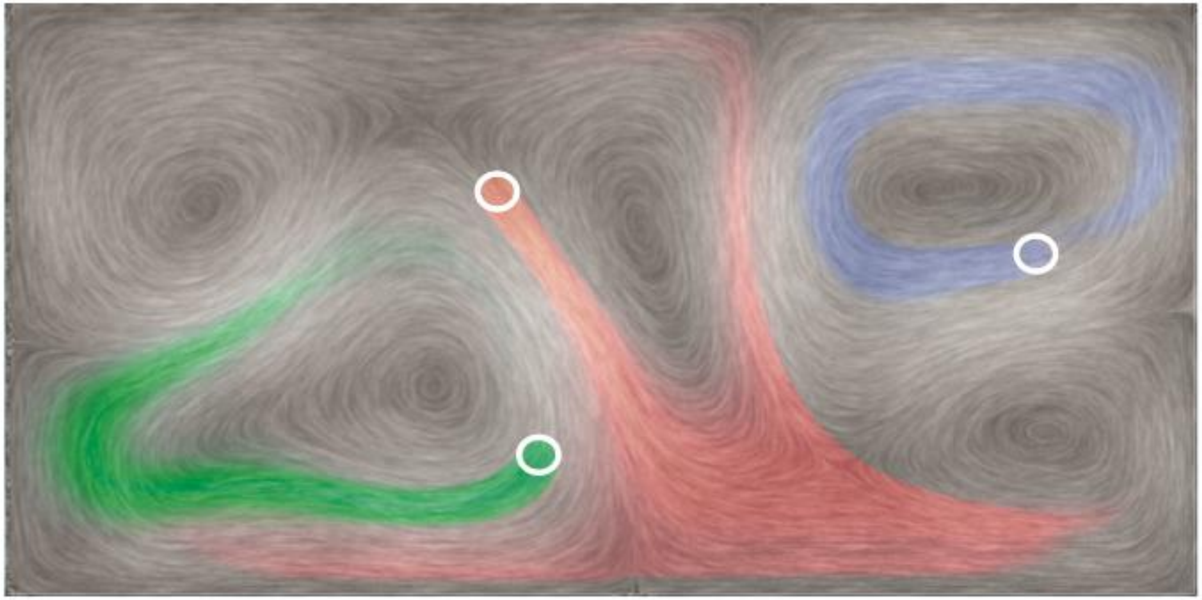
Think of

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

Image-Based Flow Visualization (IBFV)



IBFV, velocity color-coded



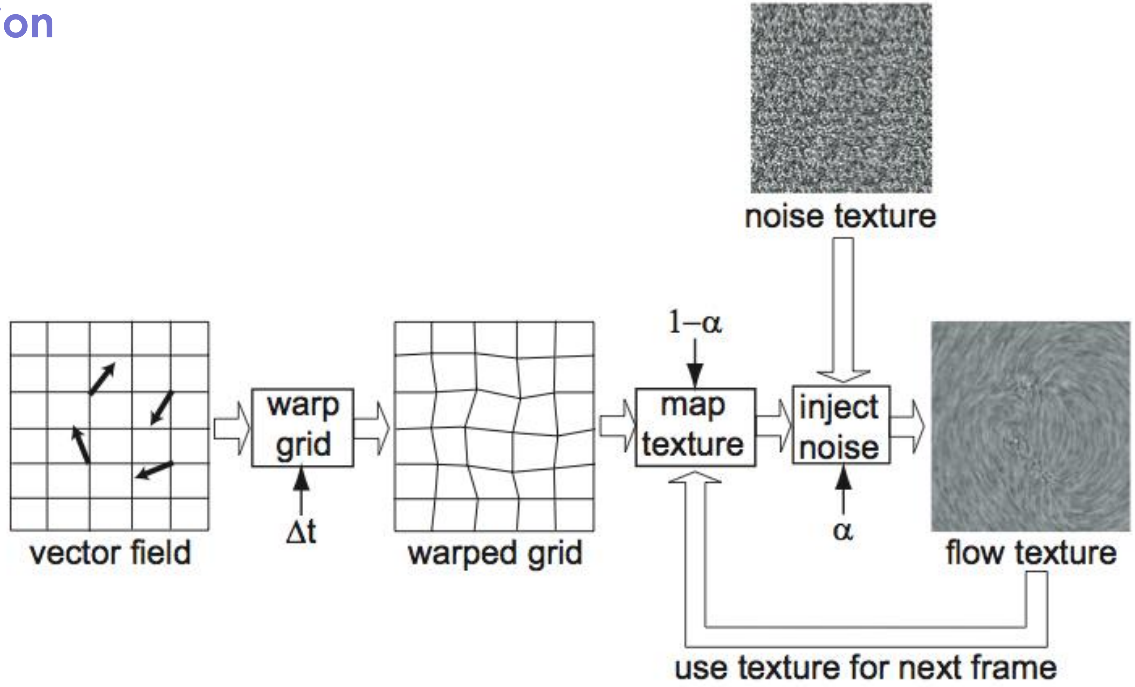
IBFV, with user-placed colored ink seeds
and luminance-coded velocity magnitude

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)

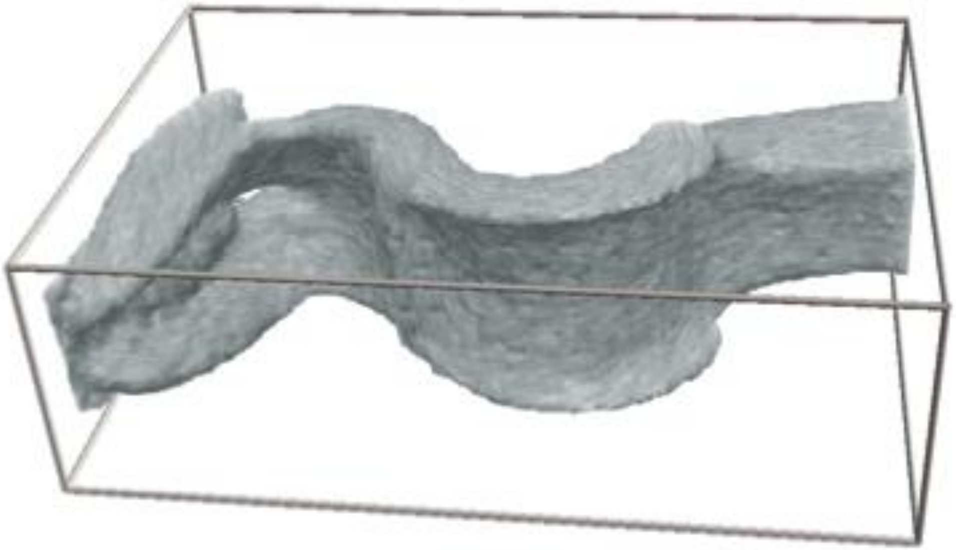
Implementation



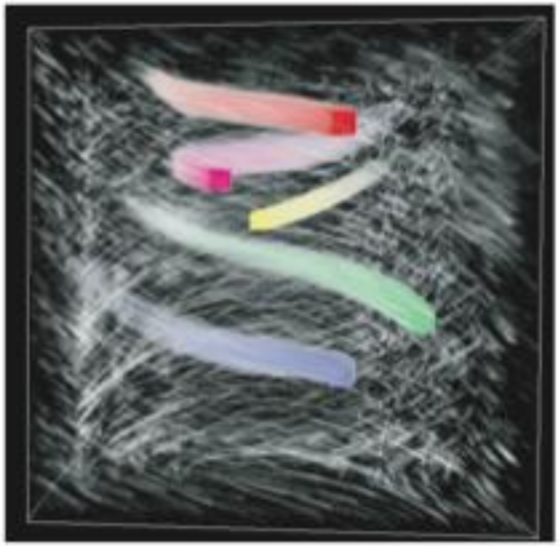
- define grid on 2D flow domain D
- warp grid D along $\mathbf{v}$ into D_{warp}
- forever
  - read current frame buffer into I
  - draw D_{warp} textured with I (advection) with opacity $1\text{-}\alpha$
  - blend noise texture N' atop of I (injection) with opacity α

Image-Based Flow Visualization (IBFV)

Variants on 3D curved surfaces and 3D volumes



IBFV on curved surfaces



IBFV in 3D volumes

Curved surfaces

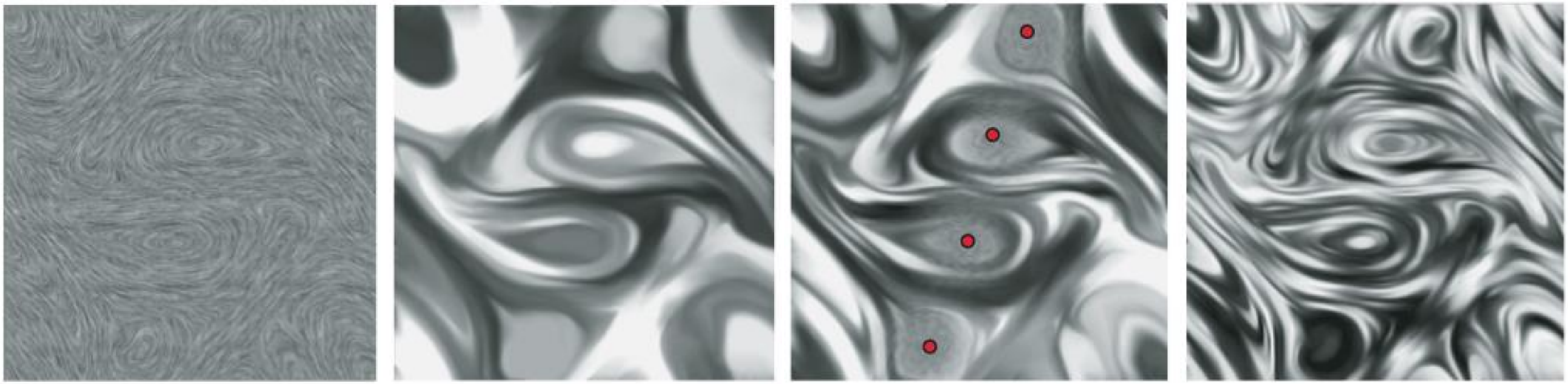
- basically same as in planar 2D, just some implementation details different

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

Details: See book, p. 203-204

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