Visualization, DD2257

Prof. Dr. Tino Weinkauf

Vector Field Visualization

Image-Based Methods
(integration-based)

Spot Noise

method for creating noise textures

spot: small intensity impulse

convolution of a white noise texture with the spot

usable for data visualization

varying parameters of the spot

Jarke van Wijk, Siggraph 1991

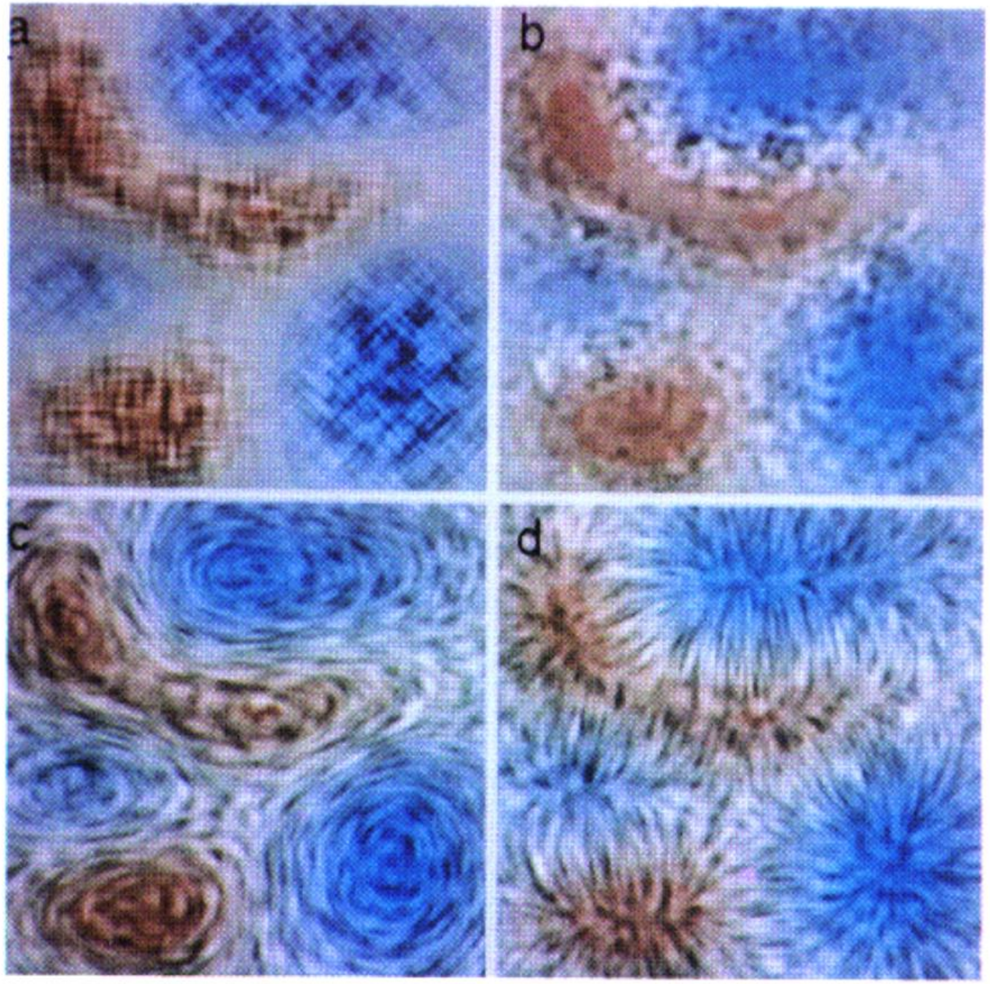


Fig. 9 Color denotes a scalar field. Texture is used to visualize attributes and interpretations of this field: (a) value, (b) gradients, (c) flow, (d) velocity potential.

- **Spot Noise**
- Different textures can be created using different spot shapes

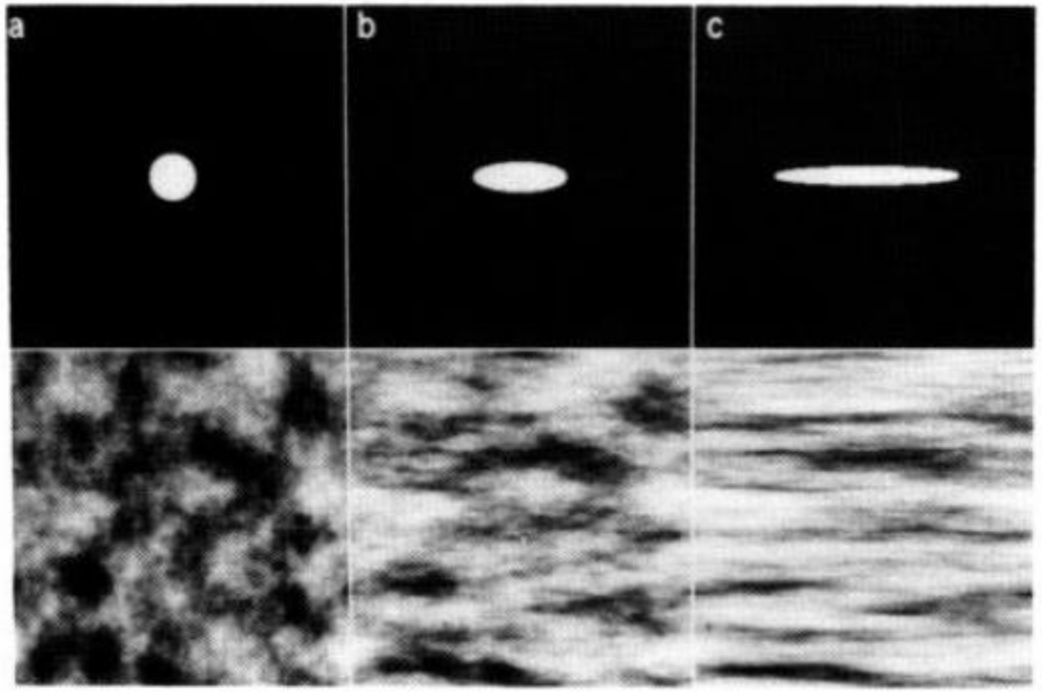


Fig. 6 Non-proportional scaling

- **Spot Noise**

- Different textures can be created using different spot sizes

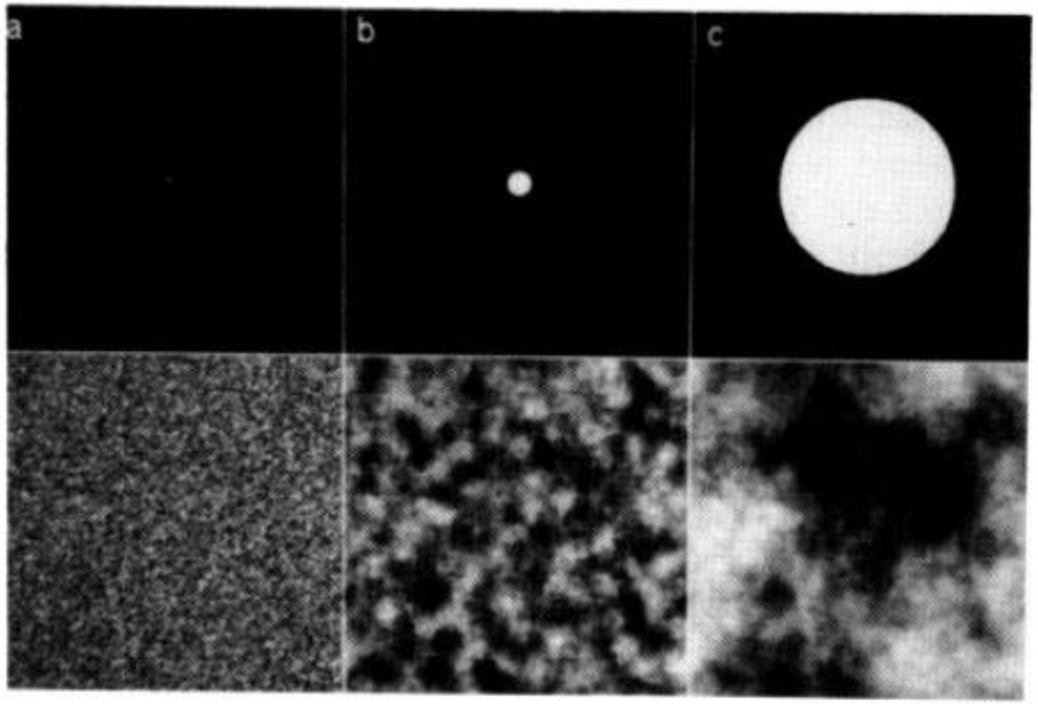
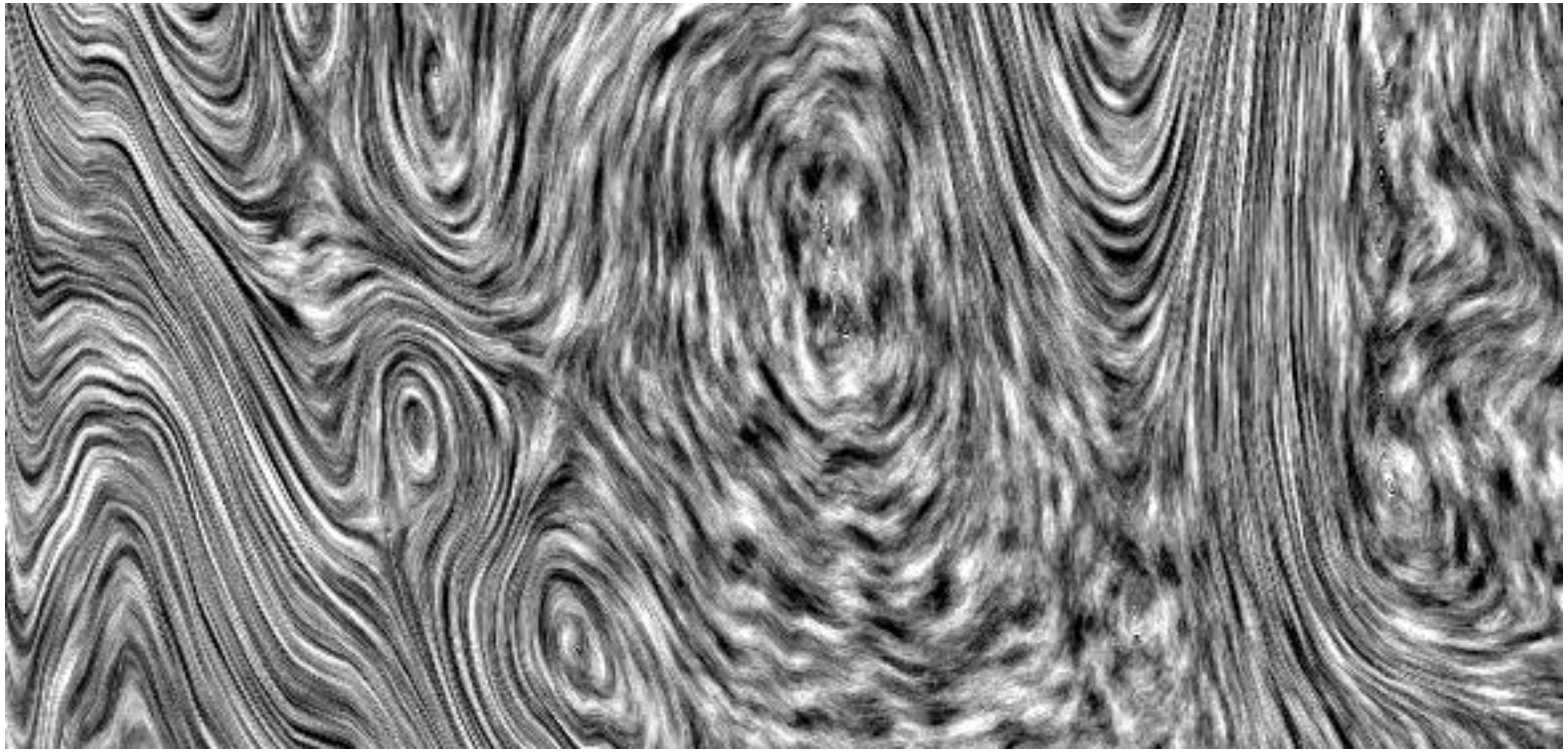


Fig. 2 Different sizes of spot

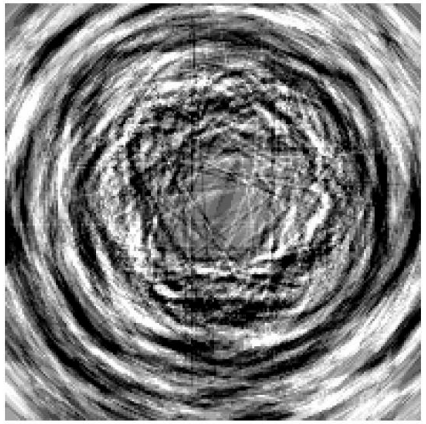
- **Spot Noise**
- Different textures can be created using different spot shapes
- Aligning the shape of the spot with the direction of flow gives a good visualization effect:
  - In direction of flow, scale proportional to (1 + |v|), where |v| = velocity magnitude
  - At 90 degrees to flow, scale proportional to 1 / (1 + |v|)

- **Spot Noise**

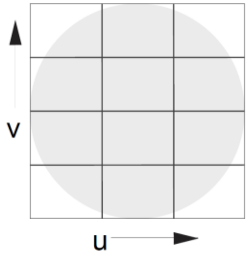


The velocity of the flow is encoded in the length of the streaks used to smear the texture.

- **Enhanced Spot Noise** de Leeuw & van Wijk, Vis 1995
  - Problem: The spot influences a region in the texture while its shape is based on data at a single point. If the velocity varies strongly over this region, the shape of the spot does not reflect the data properly.



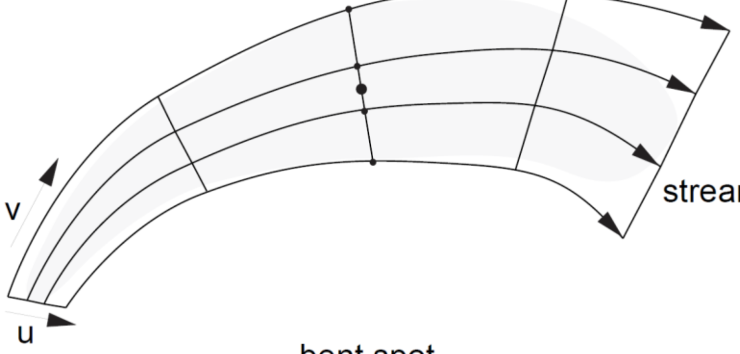
standard spot noise



standard spot

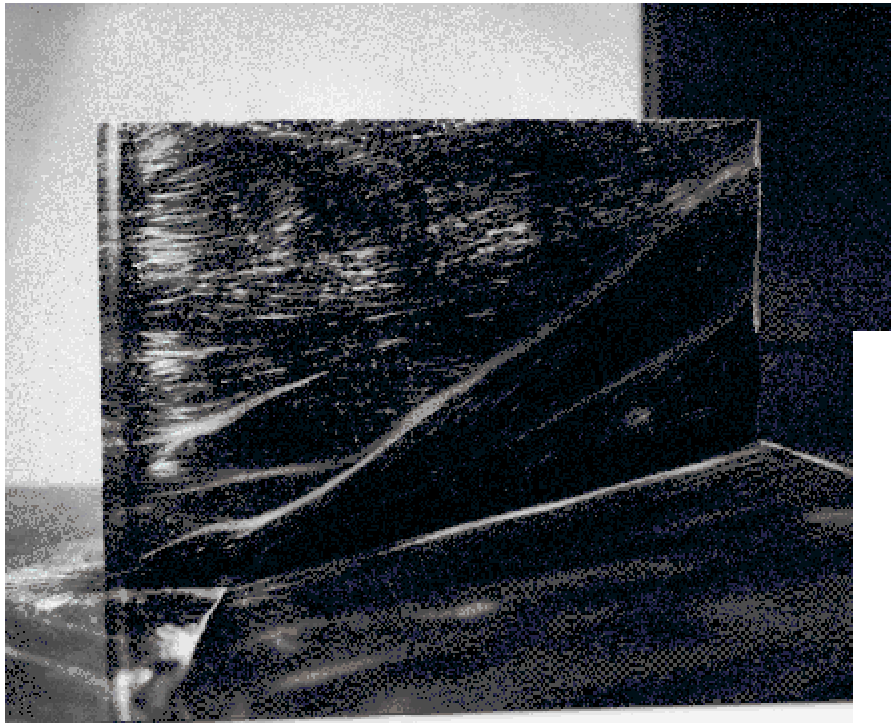
enhanced spot noise

  - Solution: Bend and deform spot following streamlines



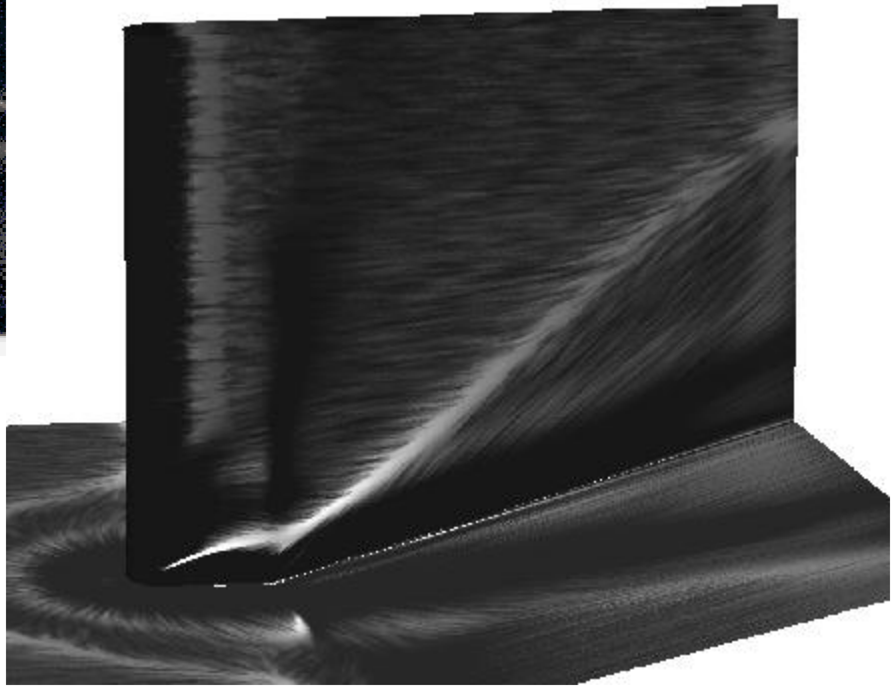
bent spot

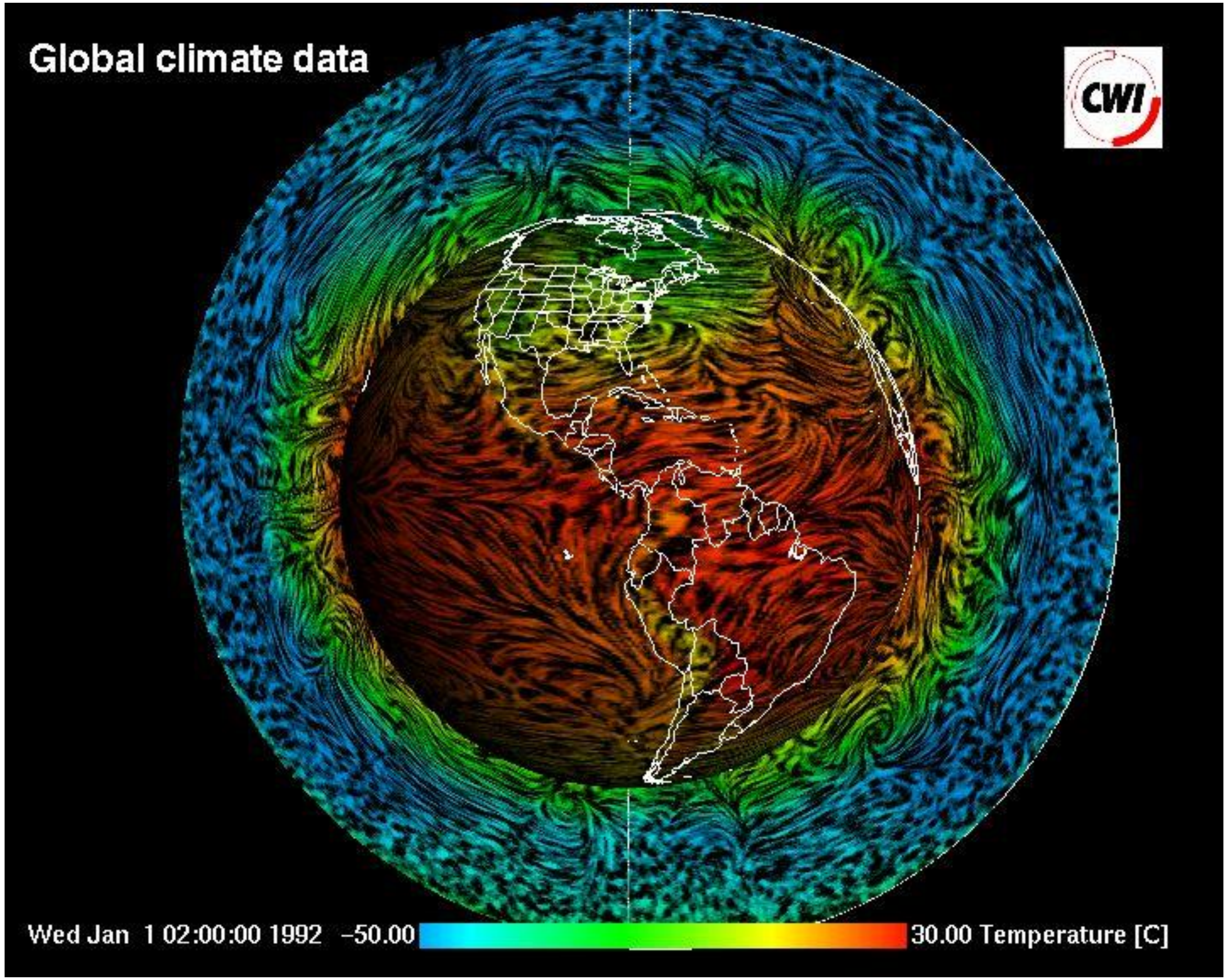
- **Spot Noise:** Relation to real-world experiments



Wall friction displayed using oil and paint - wind evaporates oil and paint leaves white traces

Numerical simulation of flow, visualized using spot noise



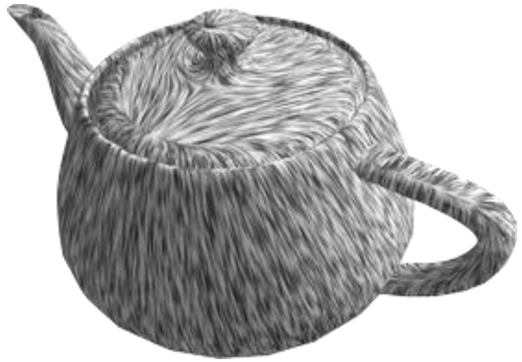


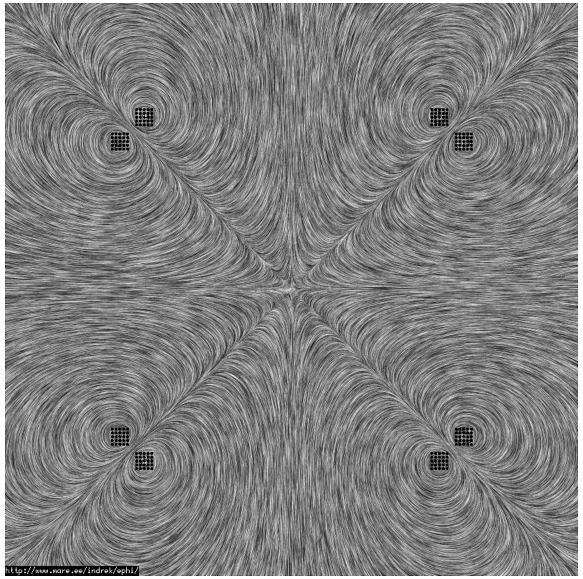
Global climate data
CWI
Wed Jan 1 02:00:00 1992 −50.00
30.00 Temperature [C]

very important

- **LIC – Line Integral Convolution**
  (Cabral/Leedom, Siggraph 1993)

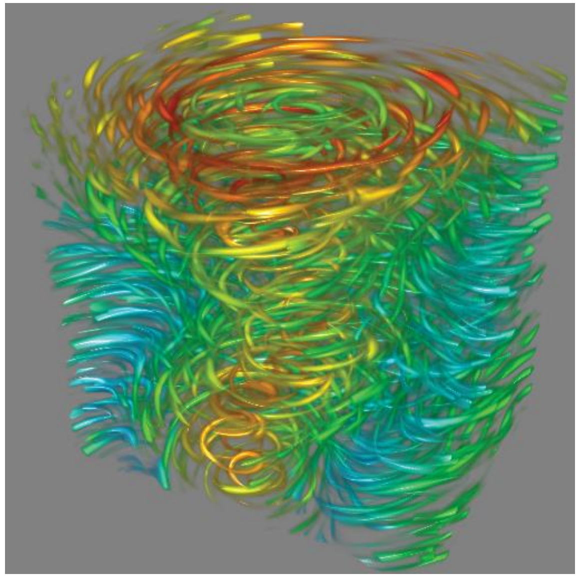
- A global method to visualize vector fields





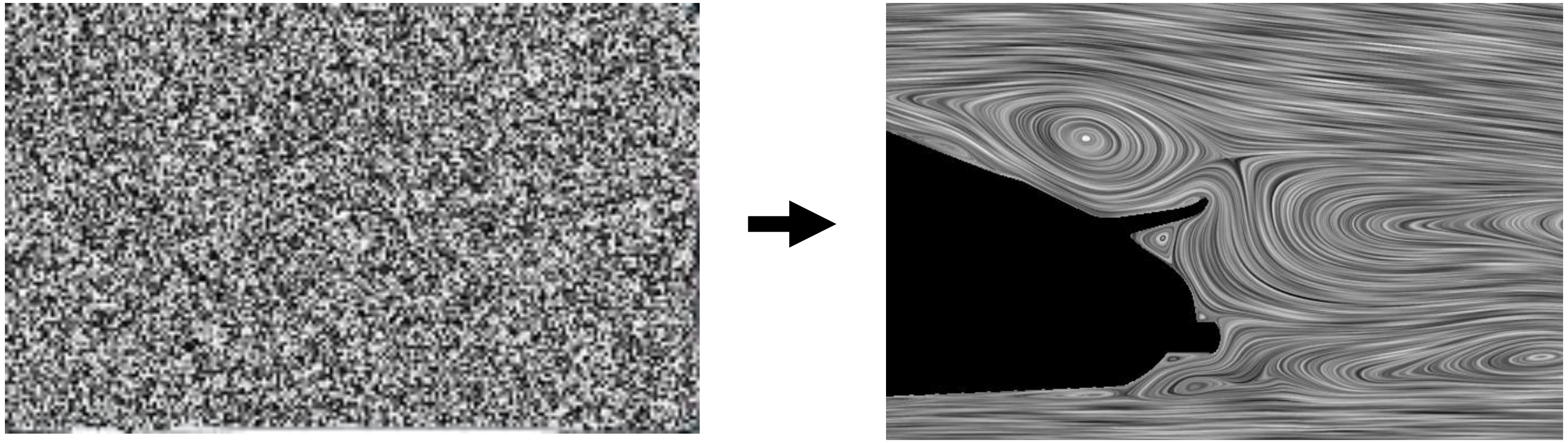
2D vector field

vector field on surface
(often called 2.5D)

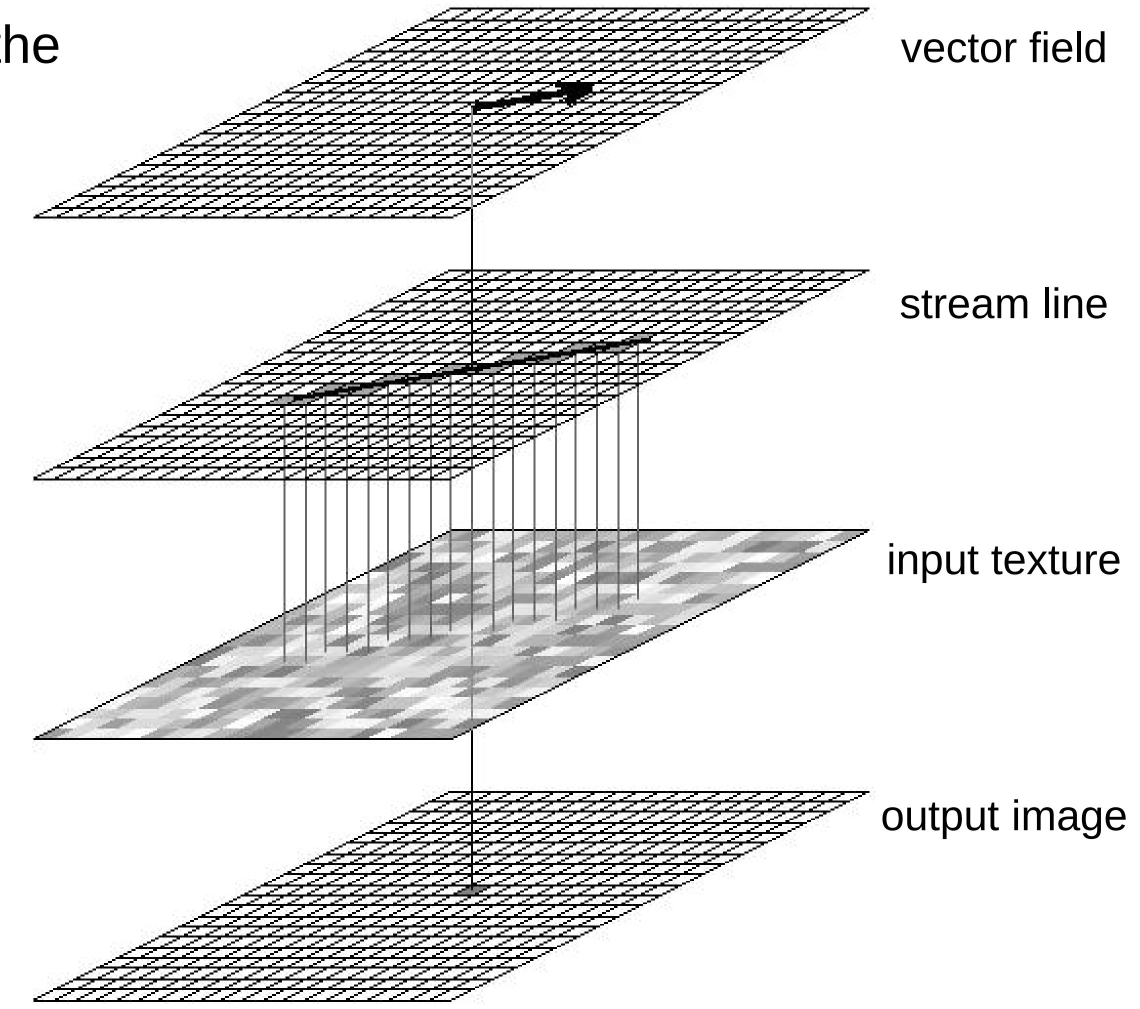


3D vector field

- Idea of Line Integral Convolution (LIC)
  - **Global** visualization technique; uses stream lines
  - Start with a **random texture**
  - **Smear** out this texture along the stream lines in a vector field
    - Results in
      - **low correlation** of intensity values **between** neighboring lines,
      - **but high correlation along** them



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

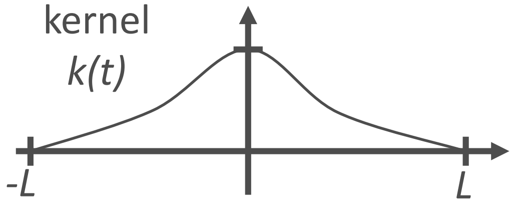


- Algorithm for 2D LIC
  - Let $t \rightarrow \Phi_0(t)$ the stream line containing the point (x_0, y_0)
  - $T(x,y)$ is the randomly generated input texture
  - Compute the pixel intensity as:

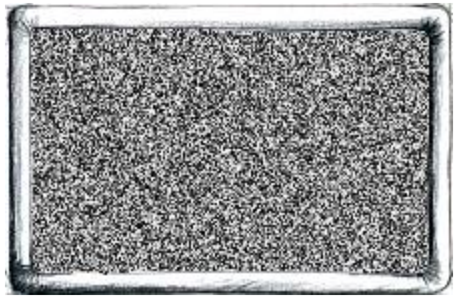
$$I(x_0, y_0) = \int_{-L}^{L} k(t) \cdot T(\varphi_0(t))\, dt$$

*convolution of the input texture T along the stream line $\Phi_0(t)$ with a kernel k(t), which has a length of 2*L.*

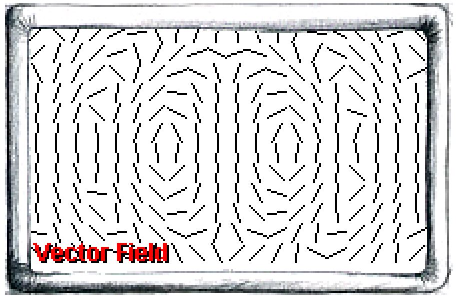
  - Kernel:
    - Finite support [-L,L]
    - Normalized
    - Symmetric



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

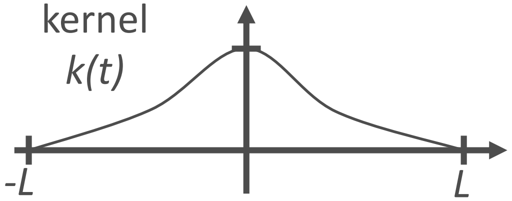


Input texture
(noise)

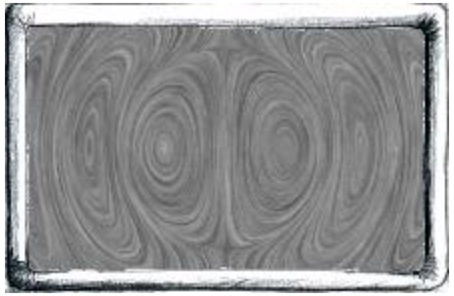


Vector field

Convolution



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

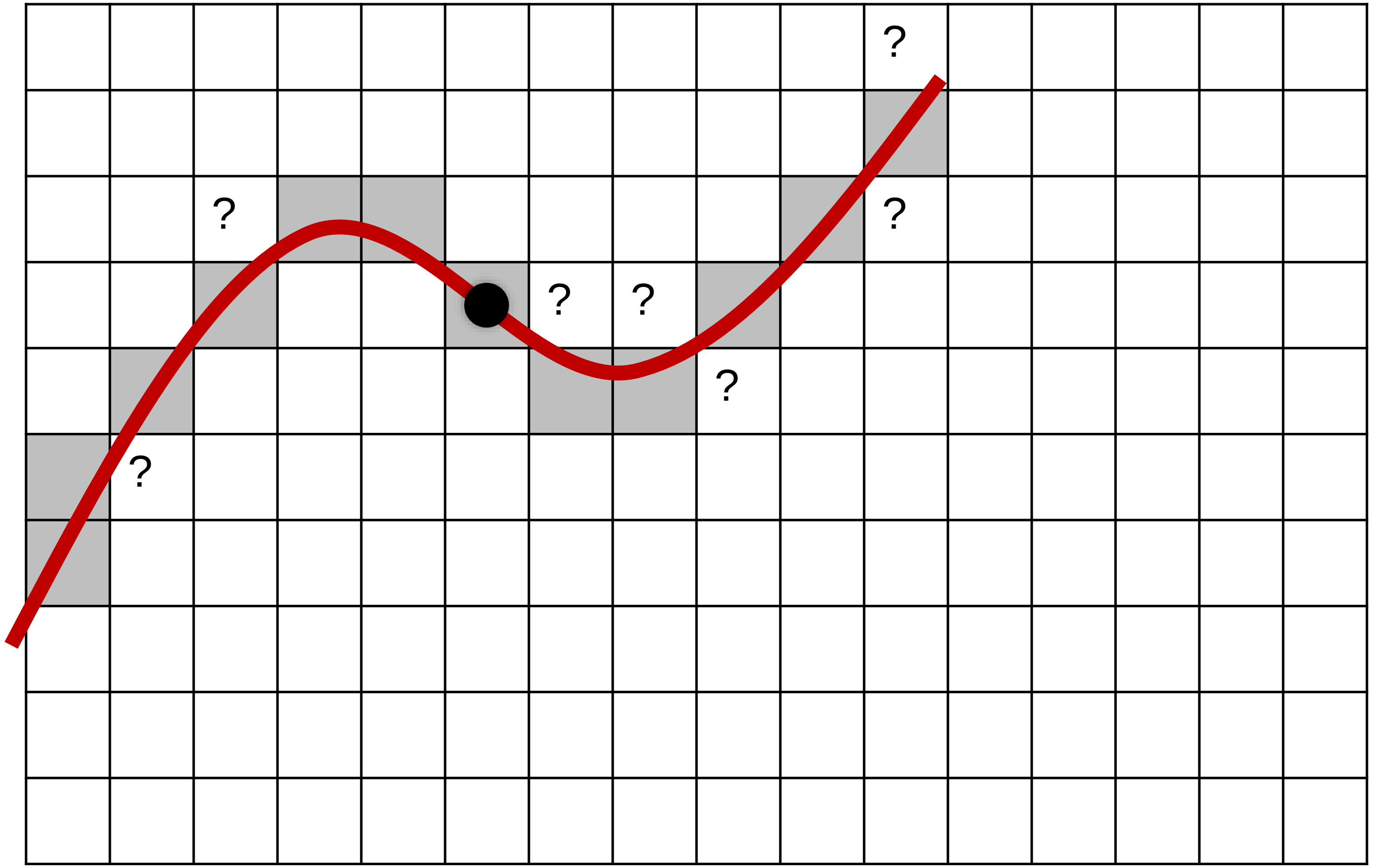


Final image

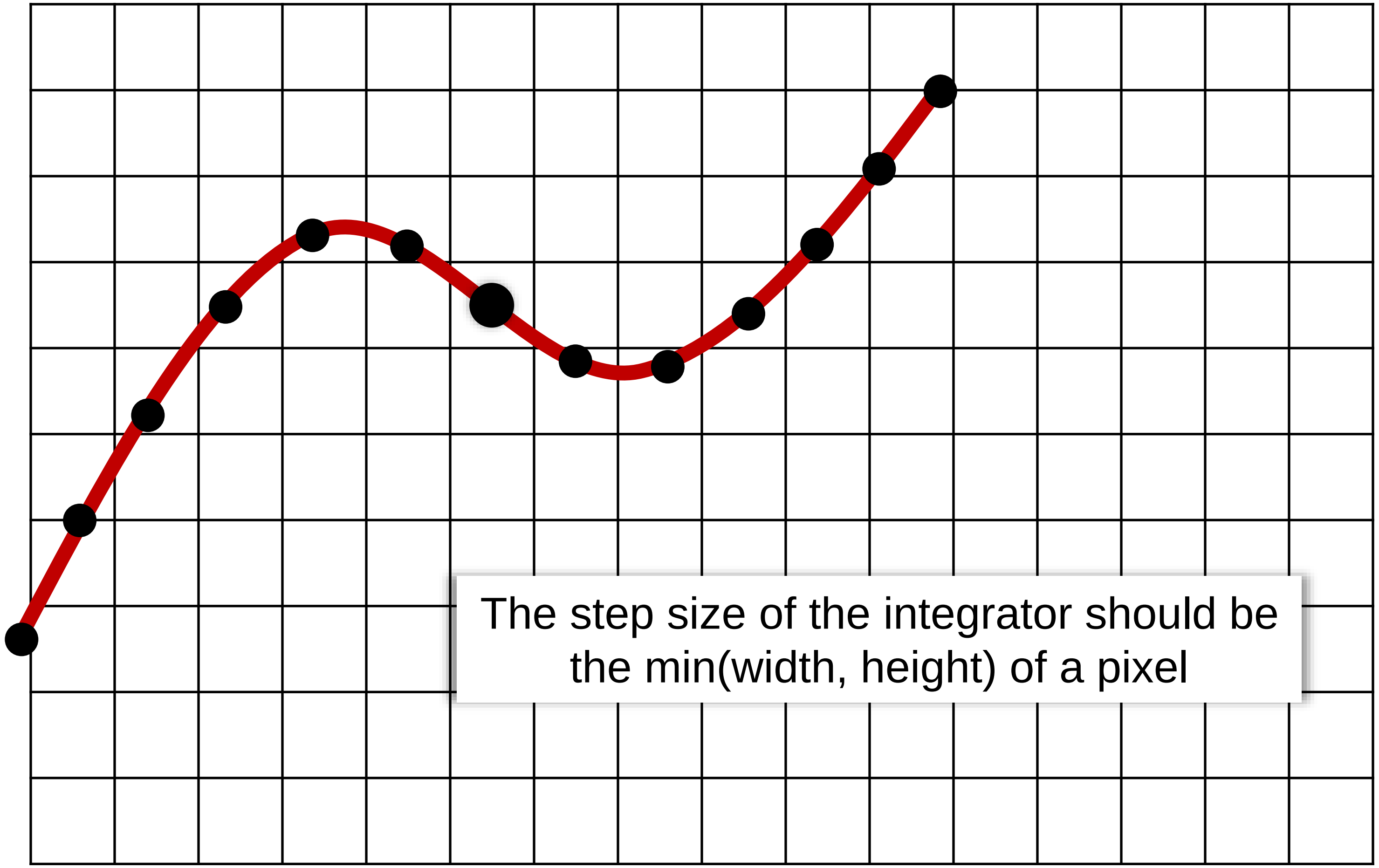
- Technical details:
- Rasterization of the stream line
- Type of kernel
- Length of kernel
- Contrast
- Performance
- Reproducibility

- Technical details: Rasterization of the stream line

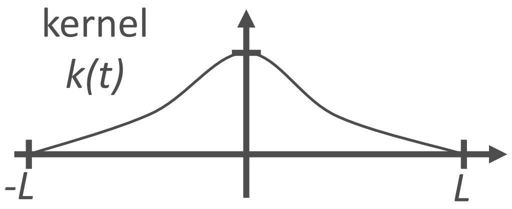
- Technical details: Rasterization of the stream line
  - How to do it?
  - Consider aliasing?



- Technical details: Rasterization of the stream line
  - Easiest way: sample the arc-length-parameterized stream line equidistantly & evaluate the random texture as a bilinear scalar field



- Technical details: Type of kernel

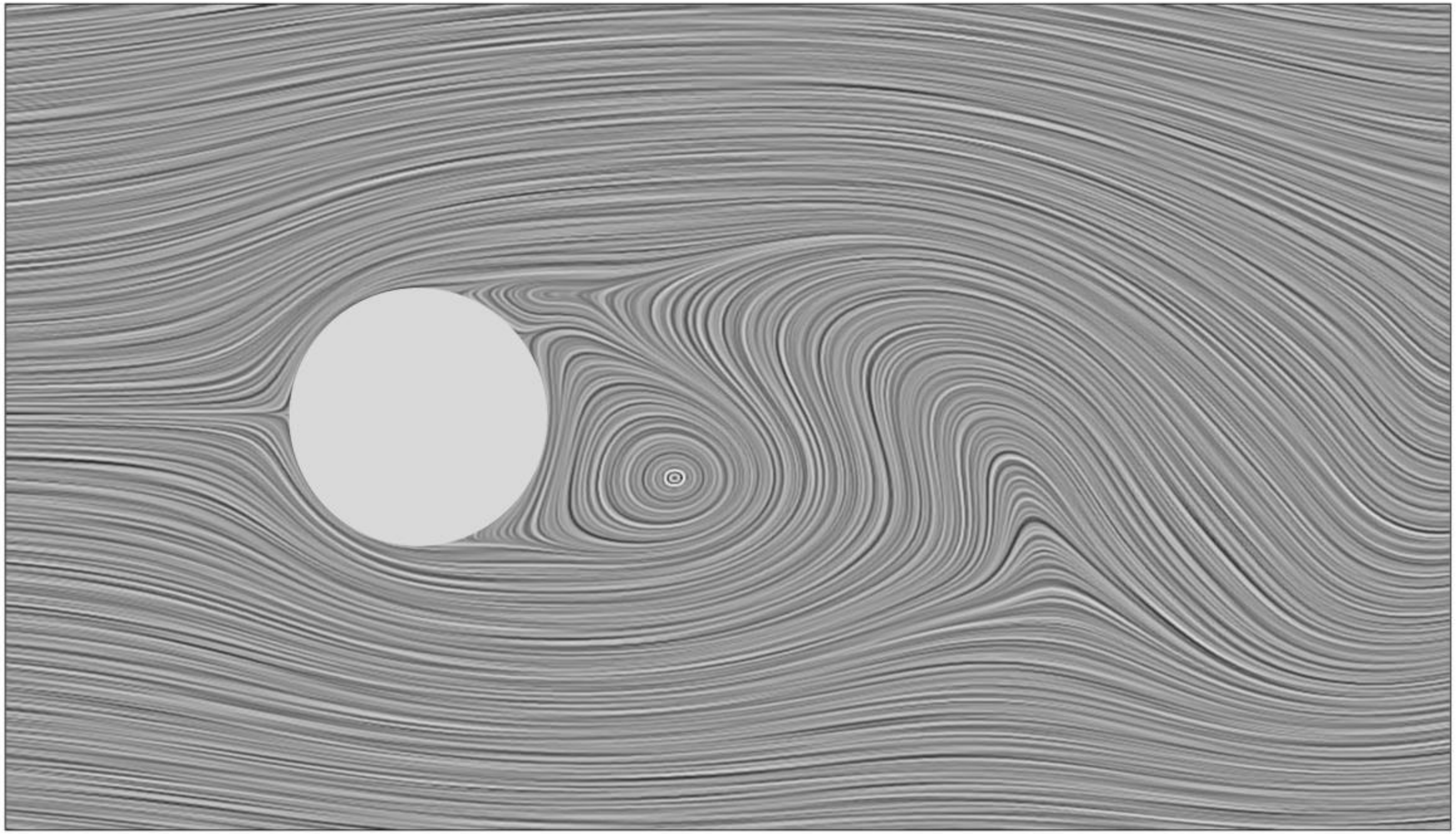


$$\int_{-L}^{L} k(t)dt = 1$$

- Gaussian kernel
- Triangle kernel
- Box kernel
    - Result is the arithmetic mean of all collected pixel values.

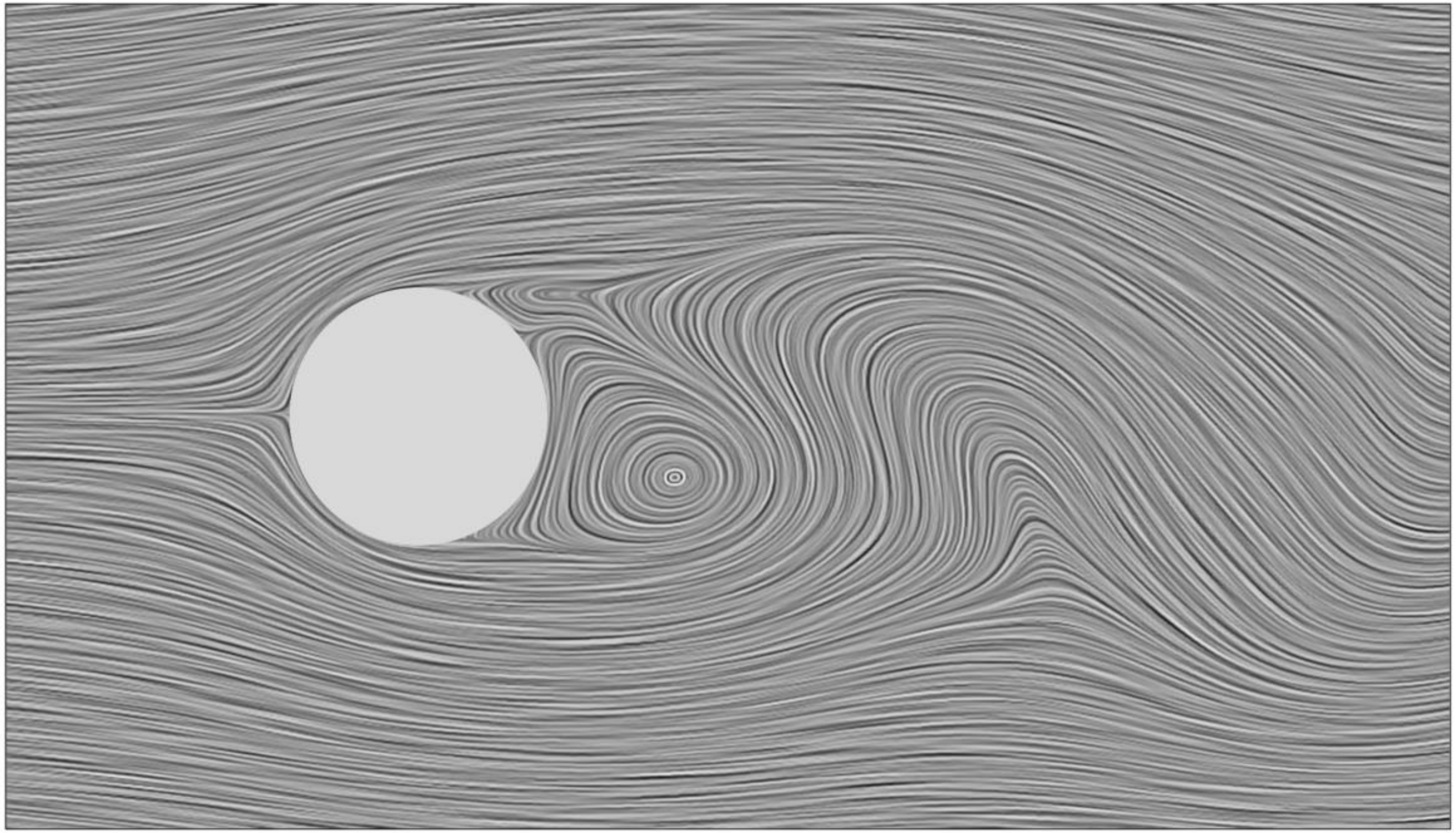
- Technical details: Length of kernel

- Longer kernel leads to longer lines, and less contrast
- Smaller kernel leads to shorter lines, and more contrast



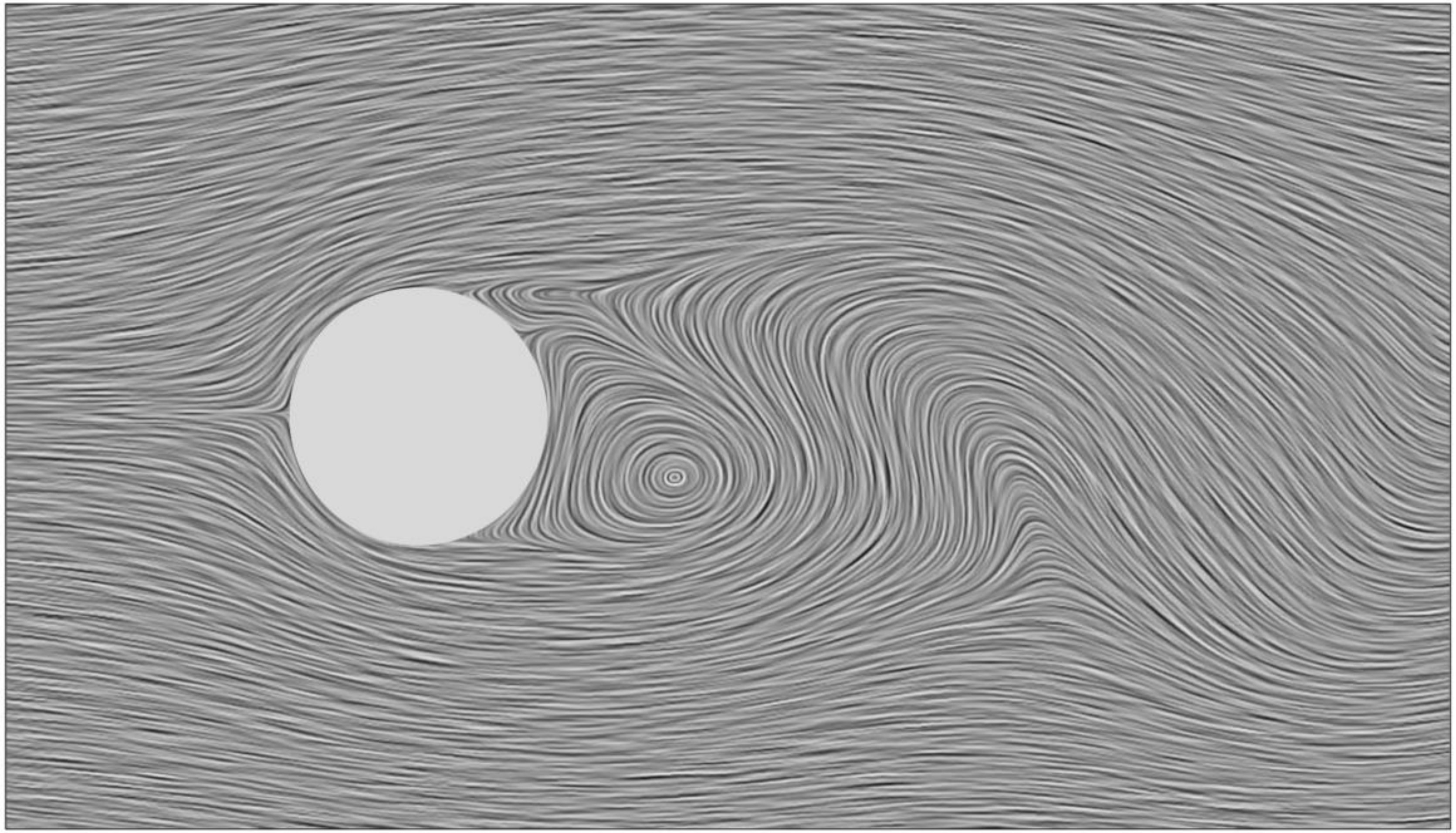
Filter length influences the quality of LIC images
filter length = 100

2D flow behind a cylinder



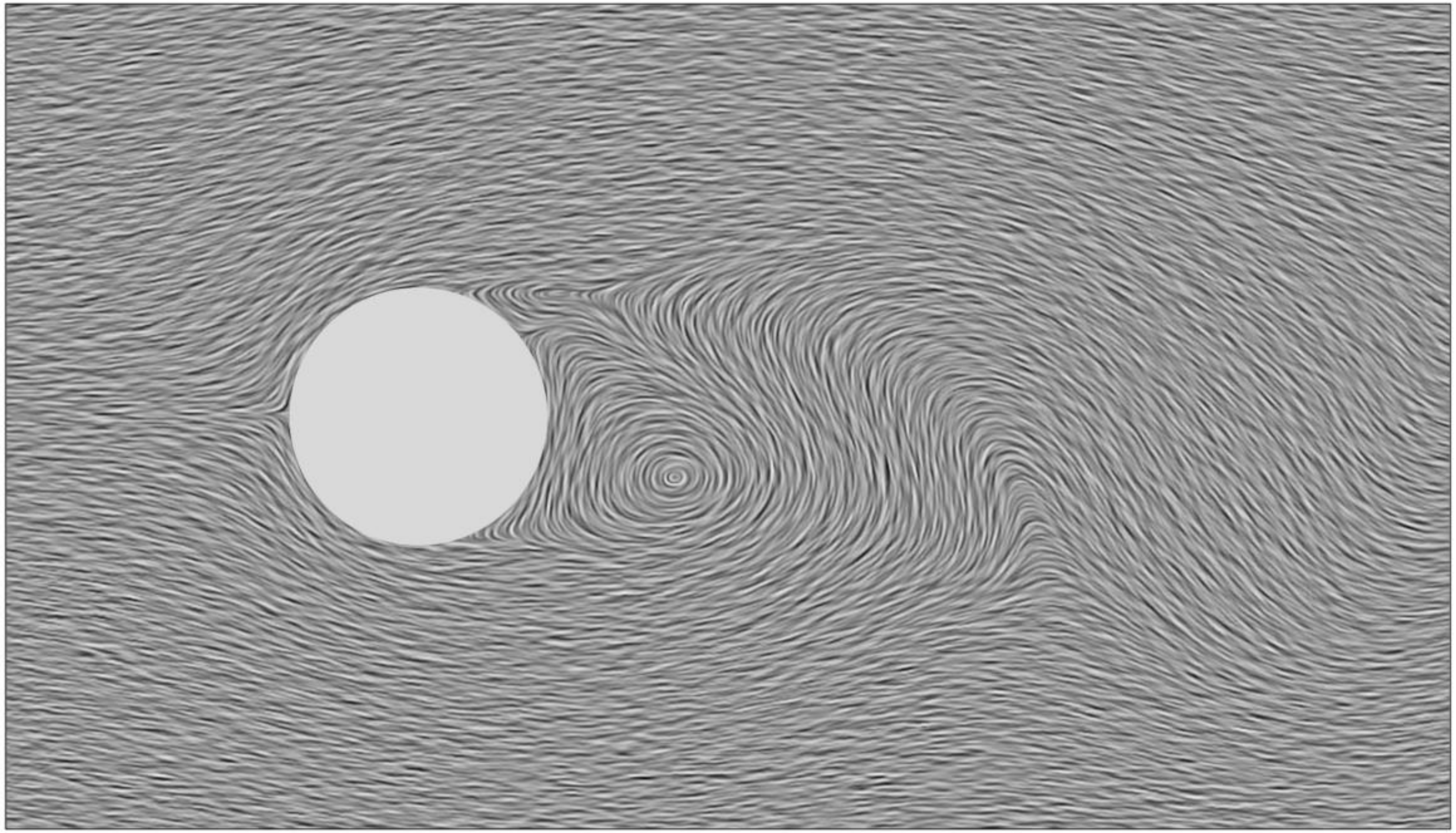
Filter length influences the quality of LIC images
filter length = 50

2D flow behind a cylinder



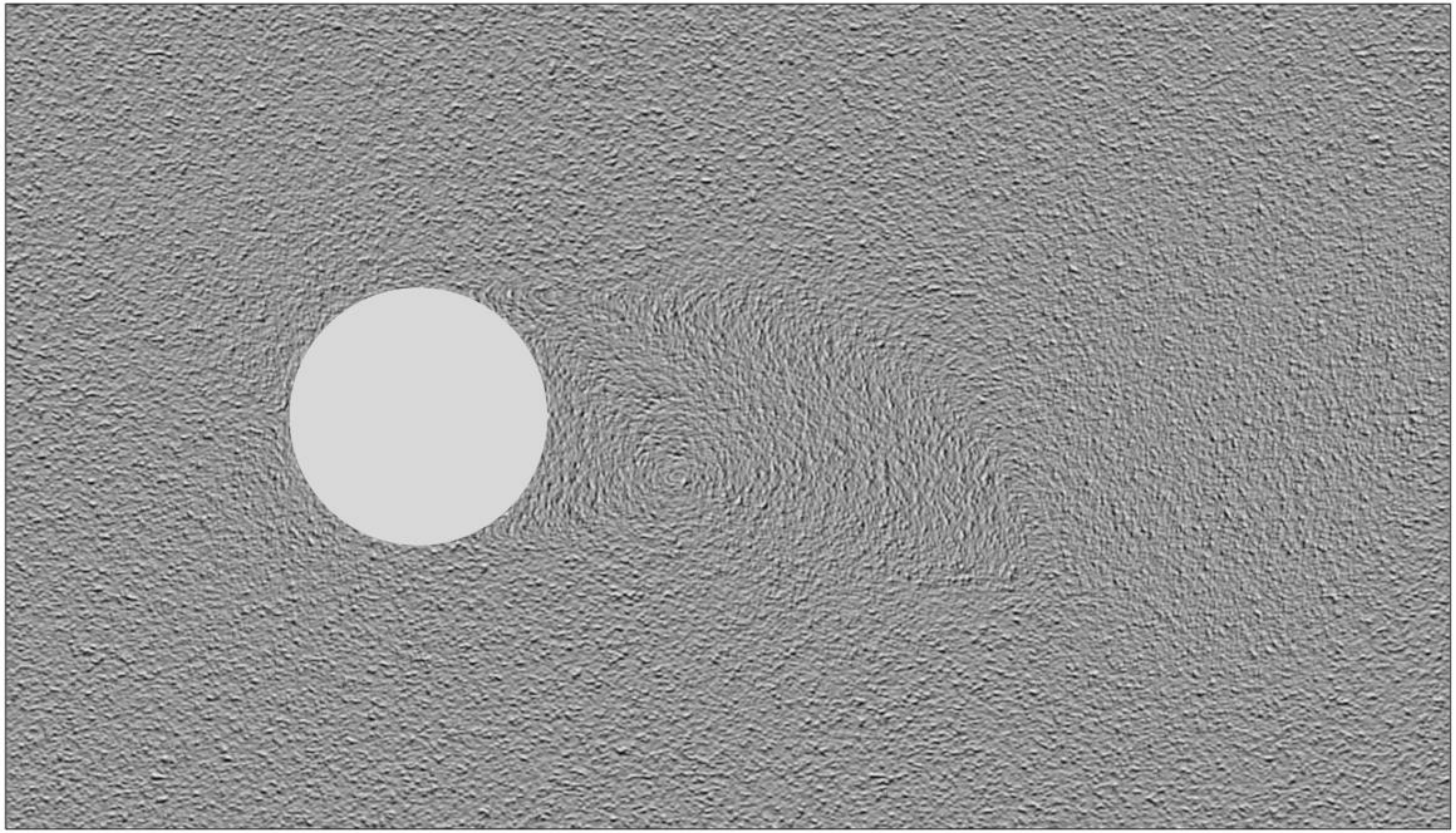
Filter length influences the quality of LIC images
filter length = 25

2D flow behind a cylinder



Filter length influences the quality of LIC images
filter length = 10

2D flow behind a cylinder



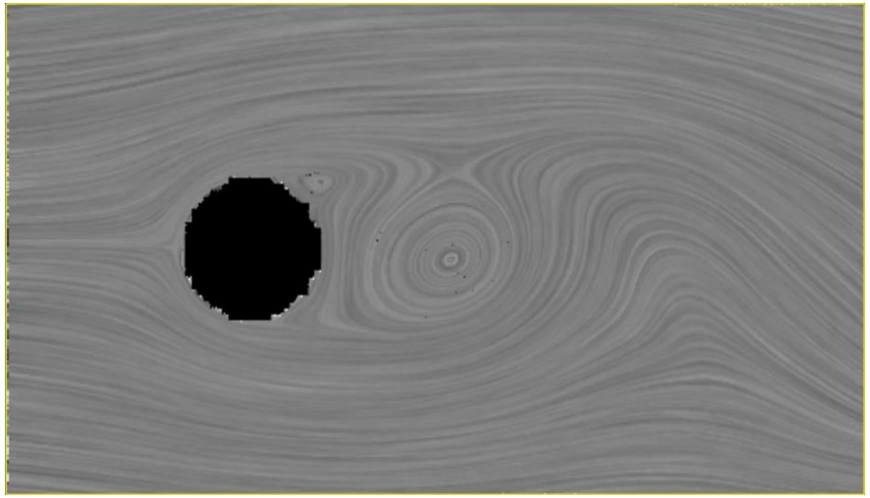
Filter length influences the quality of LIC images
filter length = 1

2D flow behind a cylinder

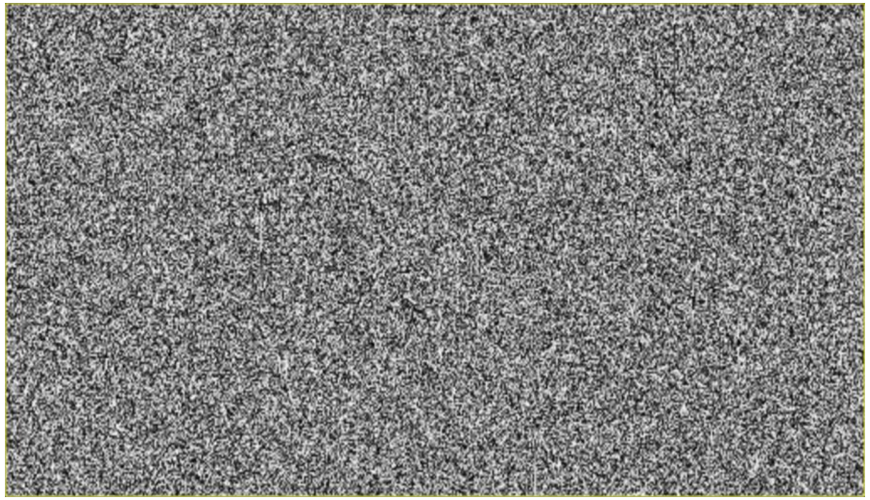
- Technical details: Contrast

- Problem:
  Convolution reduces contrast of the grayscale image

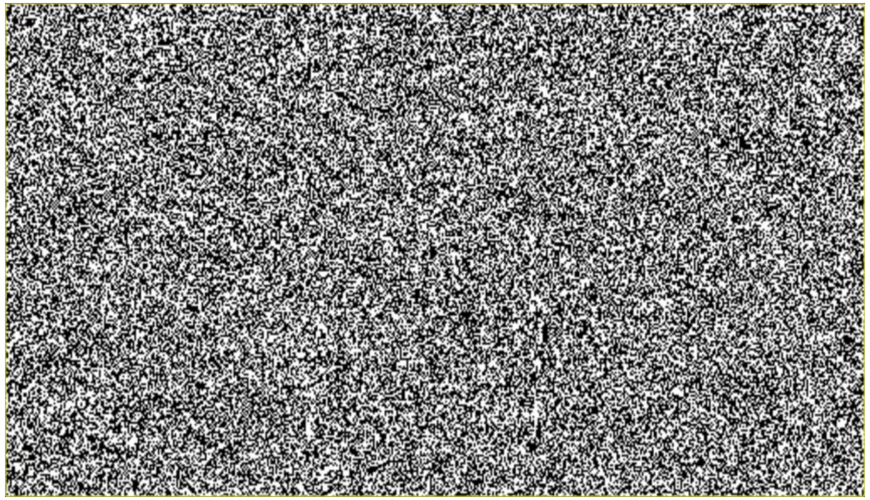
- Solutions:
  - Use black-white image as input
  - Enhance contrast after convolution



- Technical details: Contrast

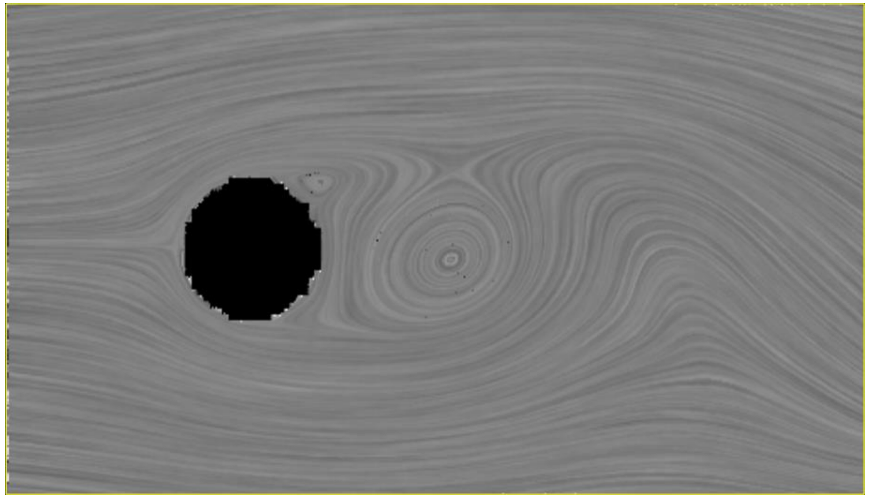


grayscale input texture

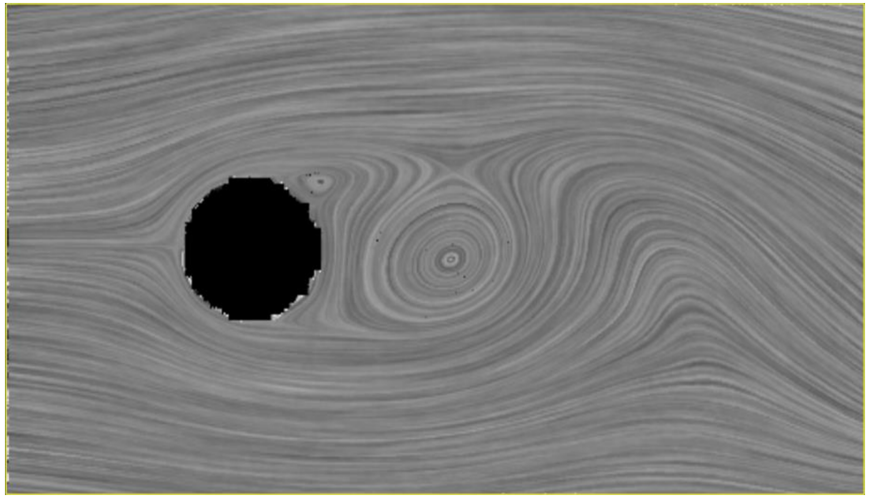


black-white input texture

- Technical details: Contrast

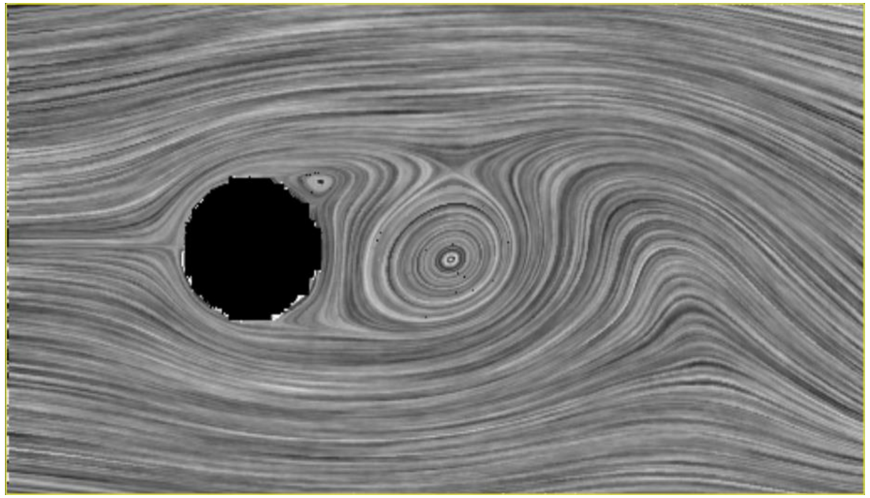


grayscale input texture



black-white input texture

enhanced contrast



Enhancing the contrast

Compute mean and standard deviation of the convolved texture

only for all non-black pixels !

$$\mu = \frac{\sum_i^n p_i}{n} \qquad P = \sum_i^n p_i^2 \qquad \sigma = \sqrt{\frac{P - n\mu^2}{n-1}}$$

arithmetic mean *standard deviation*

Adjust the mean and standard deviation to desired values

$$\sigma' \,,\, \mu' \quad \Leftarrow \text{ new desired values}$$

$$f = \frac{\sigma'}{\sigma} \qquad p_i' = \mu' + f(p_i - \mu)$$

stretching factor
(restrict to a maximum value !)

new grayscale pixel values

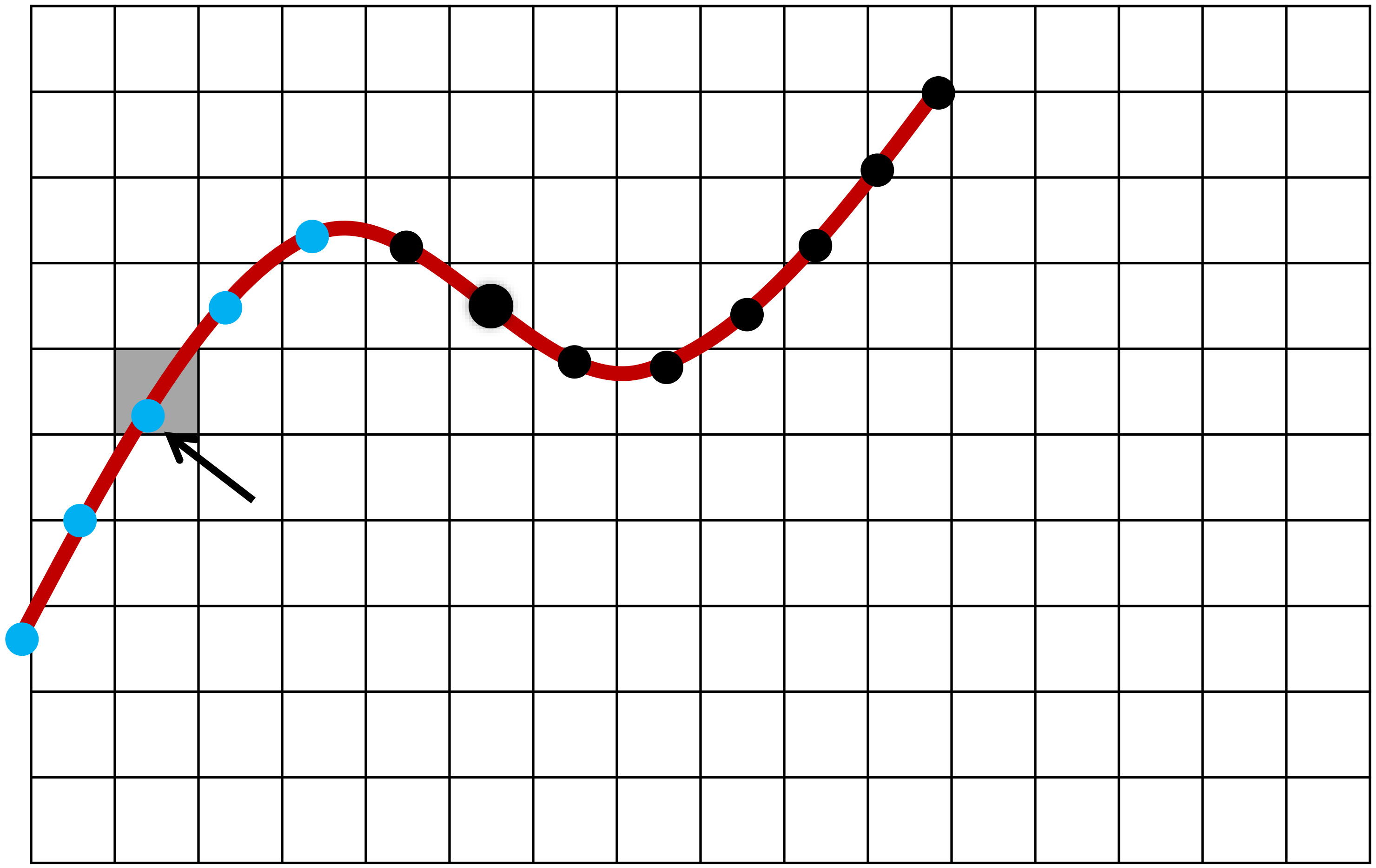
- Technical details: Enhancing the contrast
- Good defaults for desired mean and standard deviation:
  - considering a range [0, 1] with 0=black and 1=white
  - mean ➔ 0.5
  - standard deviation ➔ 0.1

- Technical details: Performance ➔ FastLIC
- Fast Line Integral Convolution (FastLIC)
  Stalling/Hege 1995
- ➔ Increase performance of LIC
  - apply convolution on all pixels on a particular stream line
  - store all pixels for which convolution is carried out
  - for untouched pixel: start a new tangent curve from there
- Significant speed up in comparison to original LIC

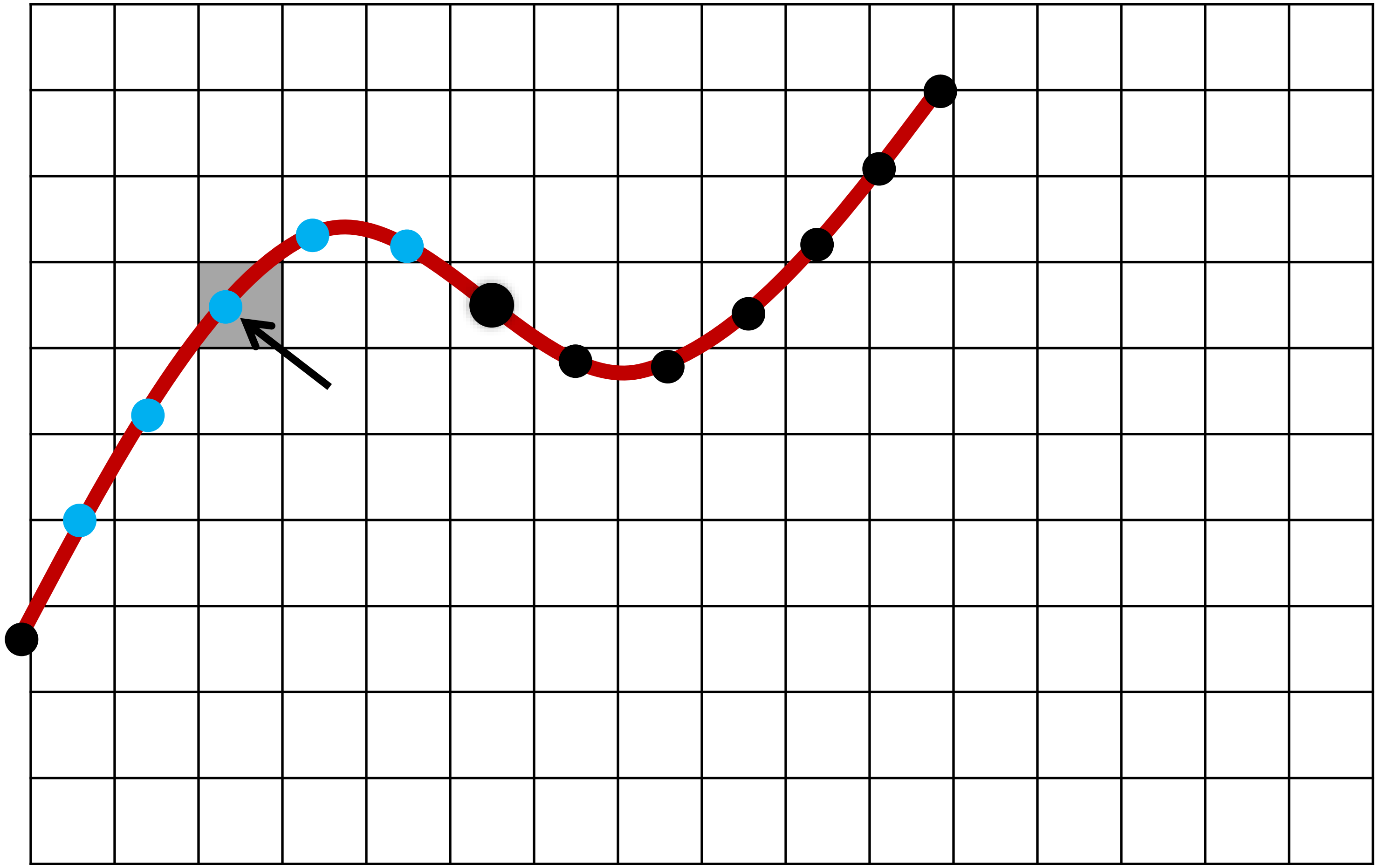
- Current implementations of LIC follow these ideas instead of the original algorithm.

- Technical details: Performance ➔ FastLIC
- General idea: exploit the coherence along the stream line
- Classic LIC:
  Visit every pixel, Integrate stream line, Convolve
- FastLIC:
  Visit every pixel that has not yet been visited
  Integrate stream line for as long as possible
  Convolve for every pixel covered by the stream line

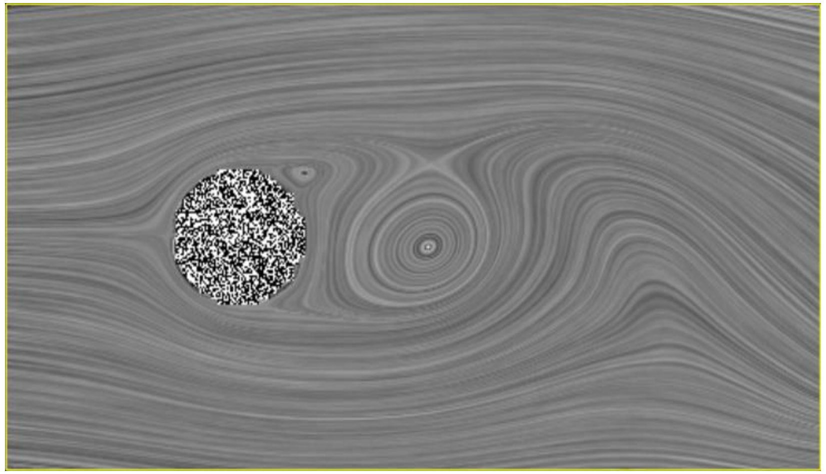
- Technical details: Performance ➔ FastLIC



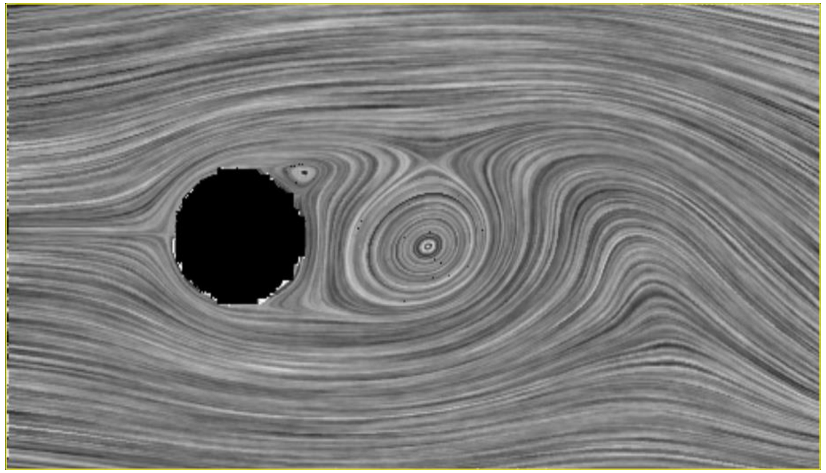
- Technical details: Performance ➔ FastLIC
  - Shifting the kernel: fast update possible for most kernels



- Technical details: Performance ➔ FastLIC
- Significantly fewer stream lines required
- Significantly better anti-aliasing



131072 stream lines were required to compute the classic LIC texture. 9 seconds for 512x256 texture.

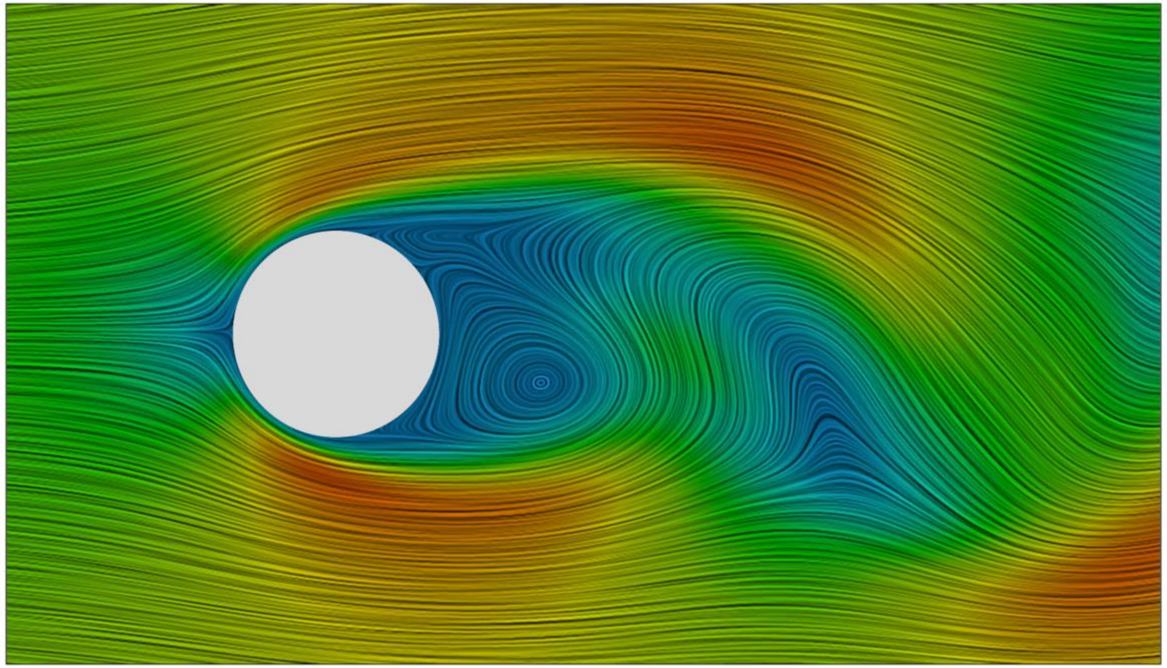


<1 second for 512x256 texture.

- Technical details: Reproducibility
- Using a random texture makes it difficult to reproduce the result from a previous run of the program
- ➔ Initialize the random number generator using a seed.
  - The seed could be a parameter given by the user.
  - srand() is the corresponding C function.
  - Then, the same random texture is generated with each run of the program. Unless you change other parameters such as the size of the texture.

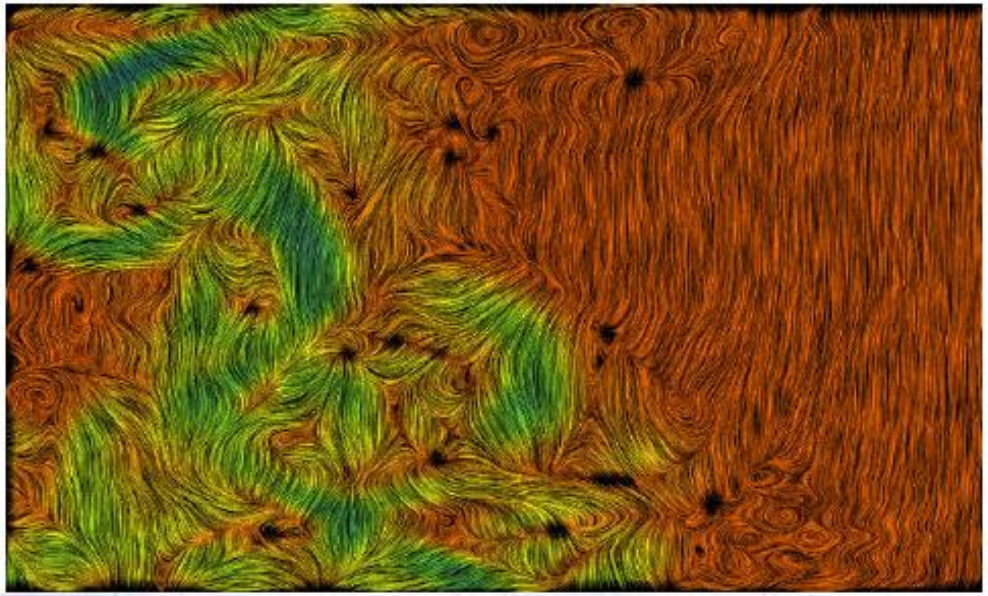
- **LIC – Line Integral Convolution**
- Improving LIC in the following directions:
- ➔ combination with color coding
- ➔ special applications (motion blur…)
- ➔ adding flow orientation
- ➔ LIC on surfaces
- ➔ LIC for 3D flows
- ➔ LIC for unsteady flows

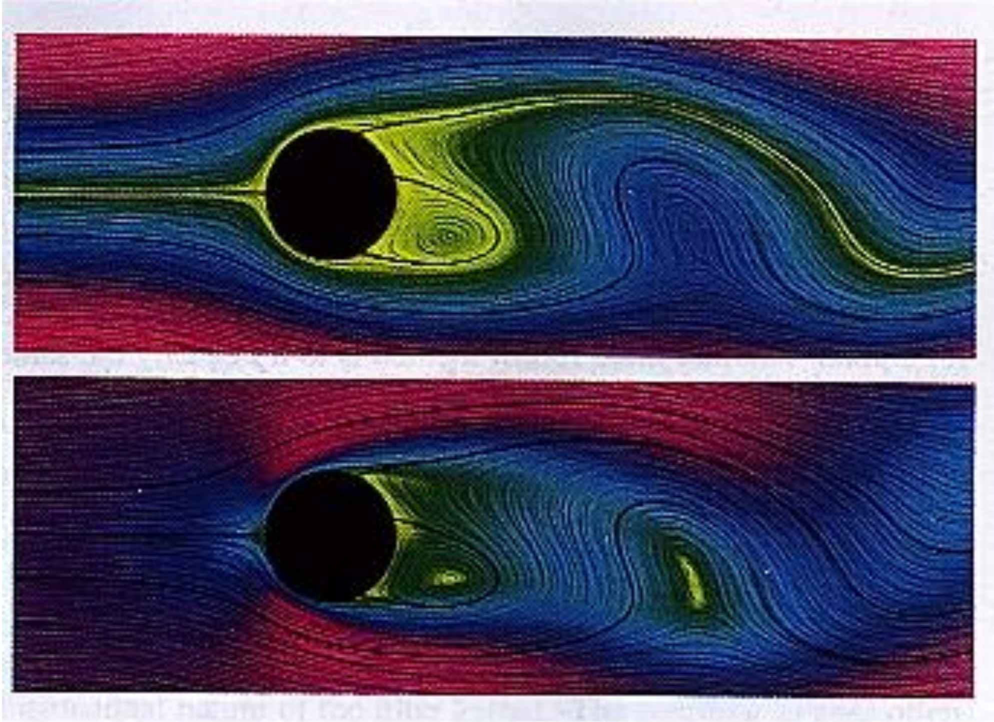
- **LIC – Line Integral Convolution**
- Combination with color coding
- Usually, LIC does not use the color channel
  - ➔ Use color to encode scalar quantities



Velocity magnitude encoded using color

2D flow behind a cylinder

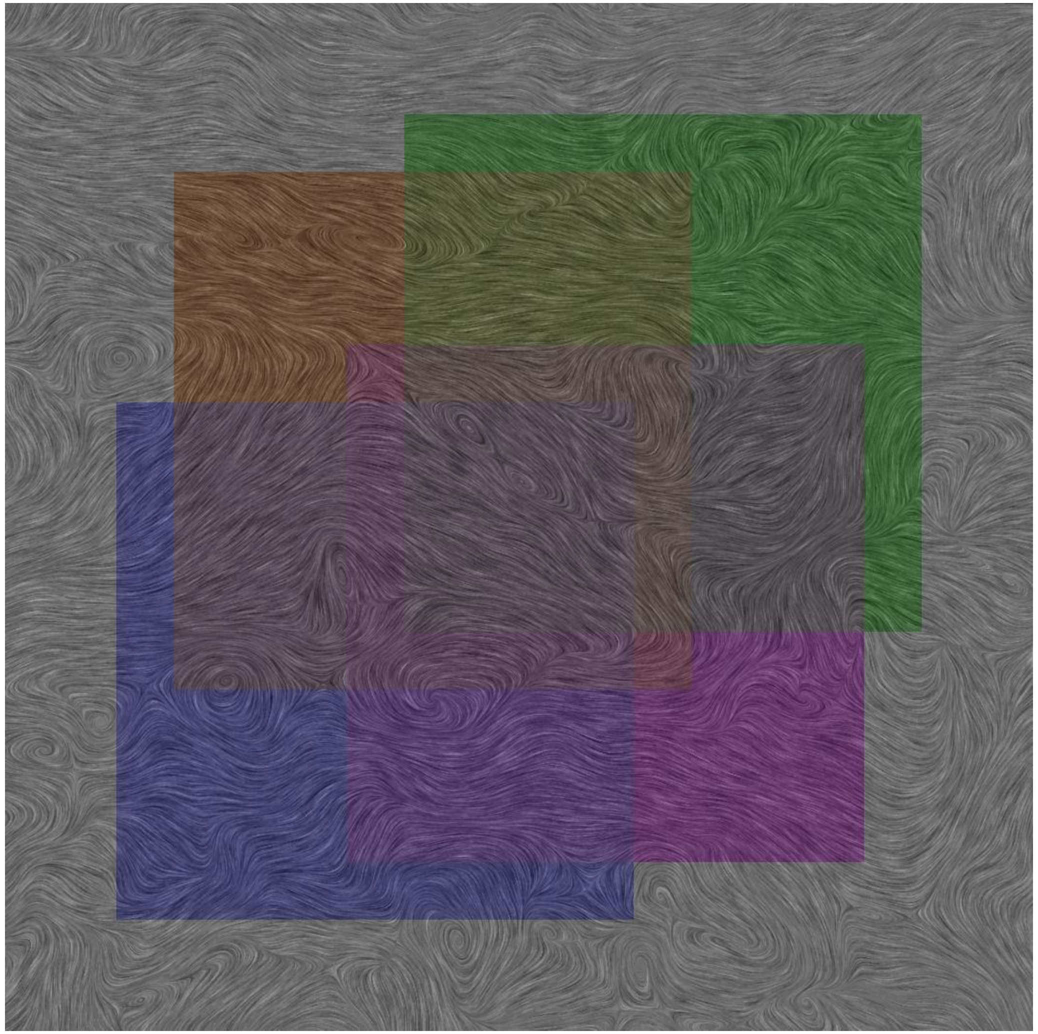




LIC and color coding of velocity magnitude

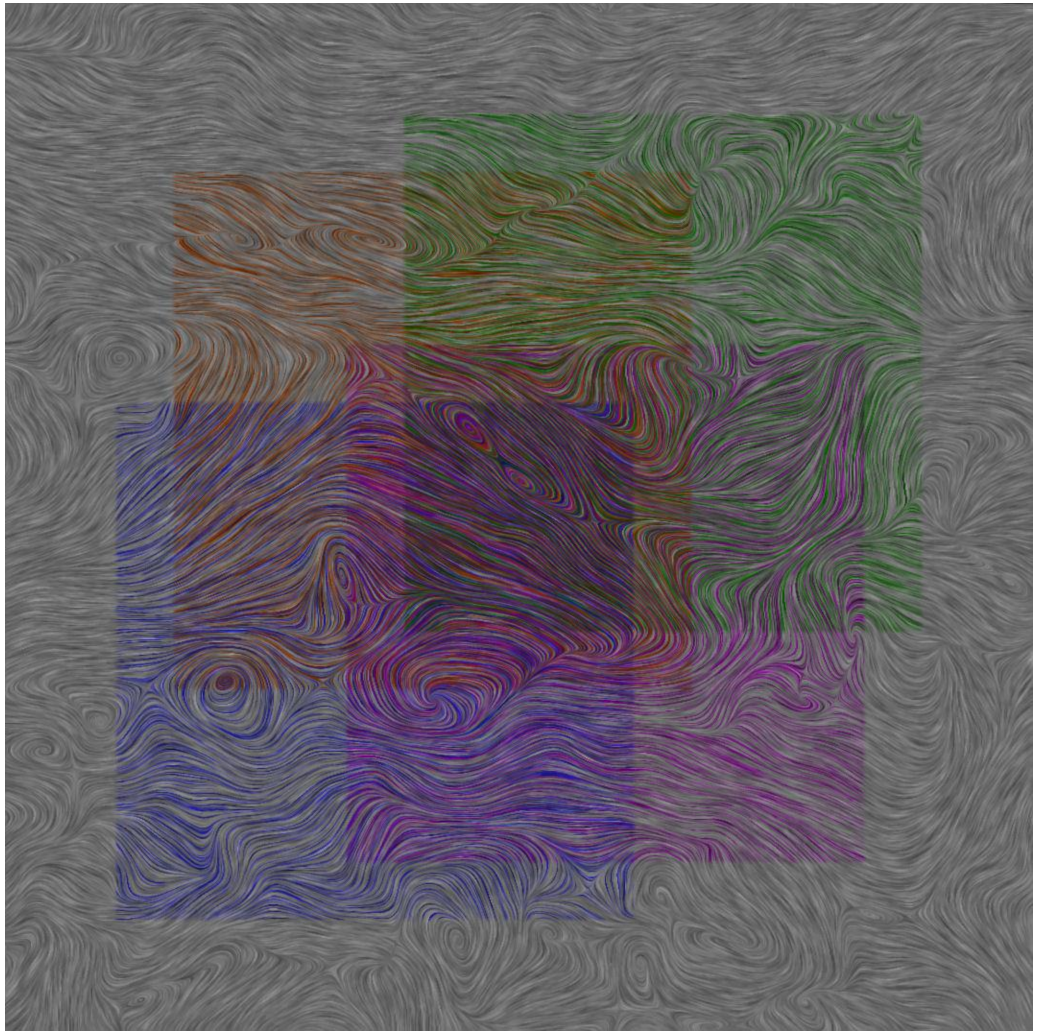
- **Color Weaving** (Urness et al., Vis 2003)
- How to encode different scalar quantities in a single LIC image?

Four artificially defined, mutually overlapping regions, overlaid on a LIC image. The color combinations in the overlap regions are obtained by averaging in RGB colorspace.

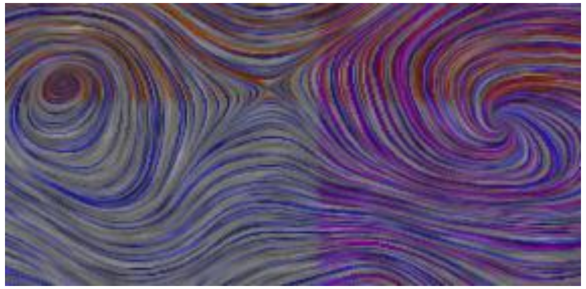


- **Color Weaving** (Urness et al., Vis 2003)
- How to encode different scalar quantities in a single LIC image?

The same four regions, represented across the same LIC image via *color weaving*. Note the continuity of color along individual streamlines within each region, and the ability to accurately perceive combinations of component colors in the areas of high overlap (characterized by the presence of three or more layers).

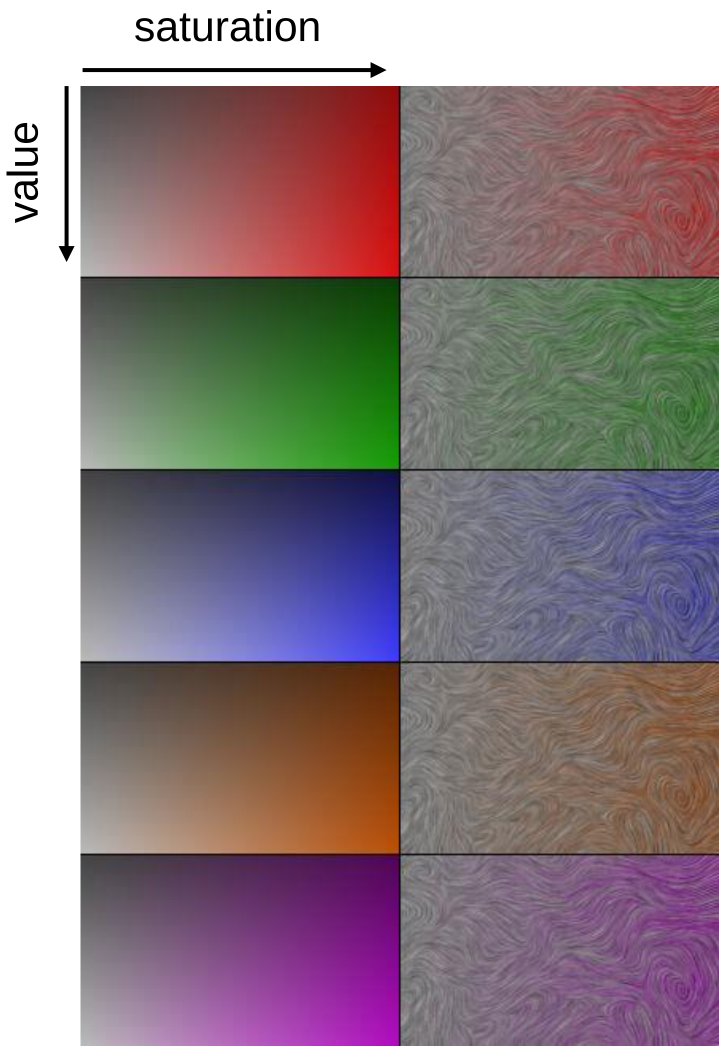


- **Color Weaving** (Urness et al., Vis 2003)
- Idea: Visualize multi-variate data on top of LIC by encoding different scalar quantities into the color of neighboring stream lines

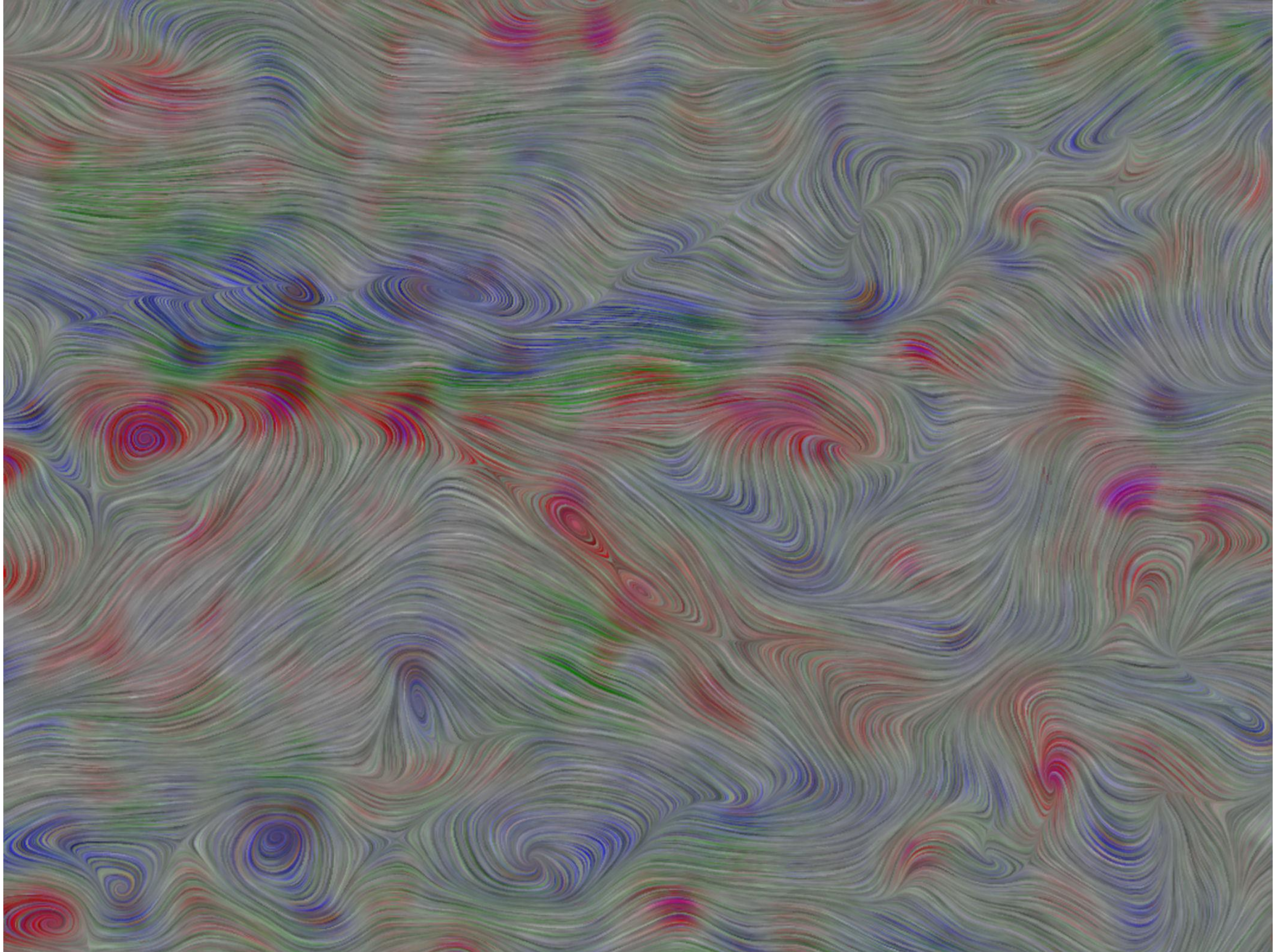


Close-ups

- **Color Weaving** (Urness et al., Vis 2003)
- 5 base colors; highly saturated and perceptually iso-luminant
- Select colors that are easy to discriminate
- Create two-dimensional color maps:
    - Vary saturation along horizontal axis
    - Vary value along vertical axis

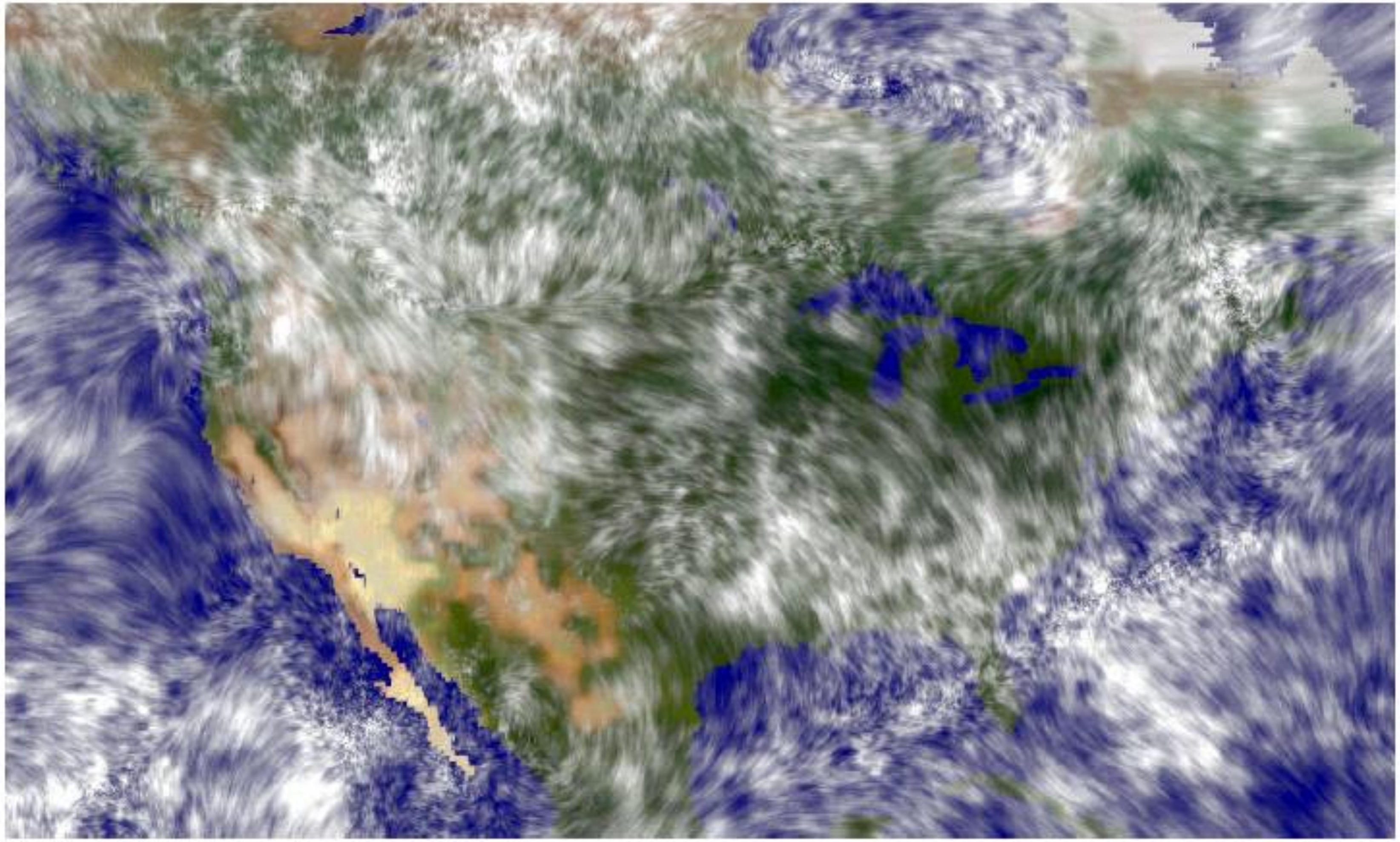


- **Color Weaving** (Urness et al., Vis 2003)
- Each scalar quantity gets its own colormap
- Final pixel color:
  - Hue: scalar quantity to be encoded, i.e., its colormap
  - Value: gray value obtained from LIC
  - Saturation: value of the scalar quantity
- One has to be careful when several stream lines run across the same pixel



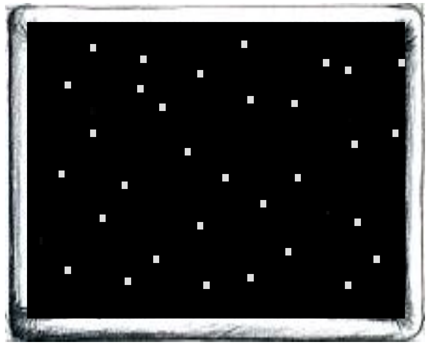
Here, the chosen vector field is the gradient field of the image.

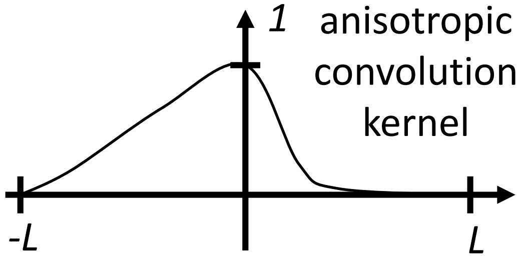
Add motion blur by means of variable length LIC

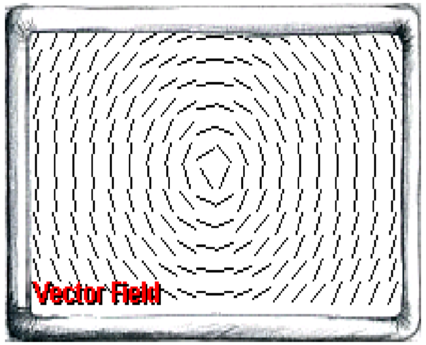


Length of convolution integral with respect to magnitude of vector field

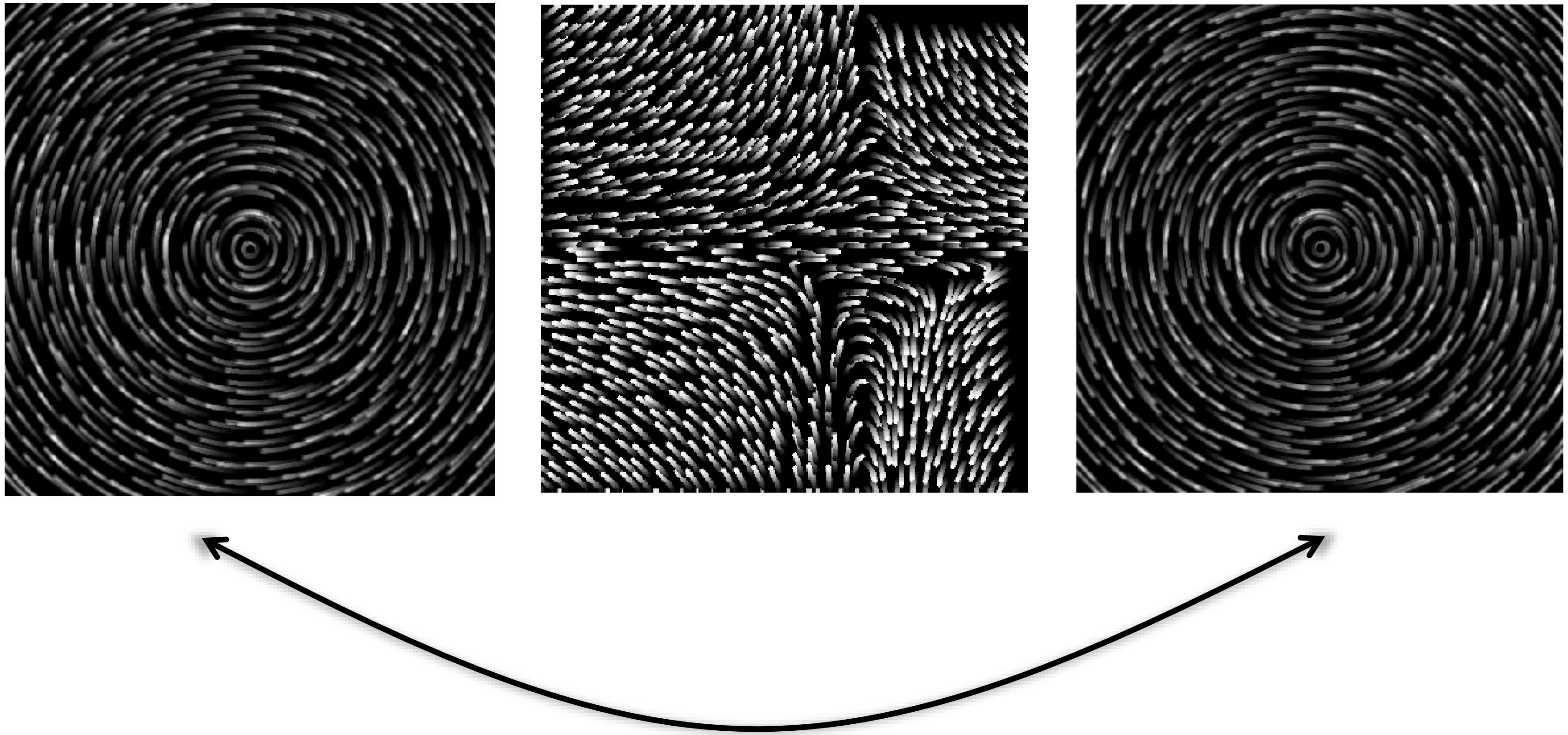
- Oriented LIC (OLIC):
  - Visualizes **orientation** (in addition to direction)
  - Uses a **sparse texture**, i.e., smearing of individual drops
  - **Asymmetric** convolution kernel



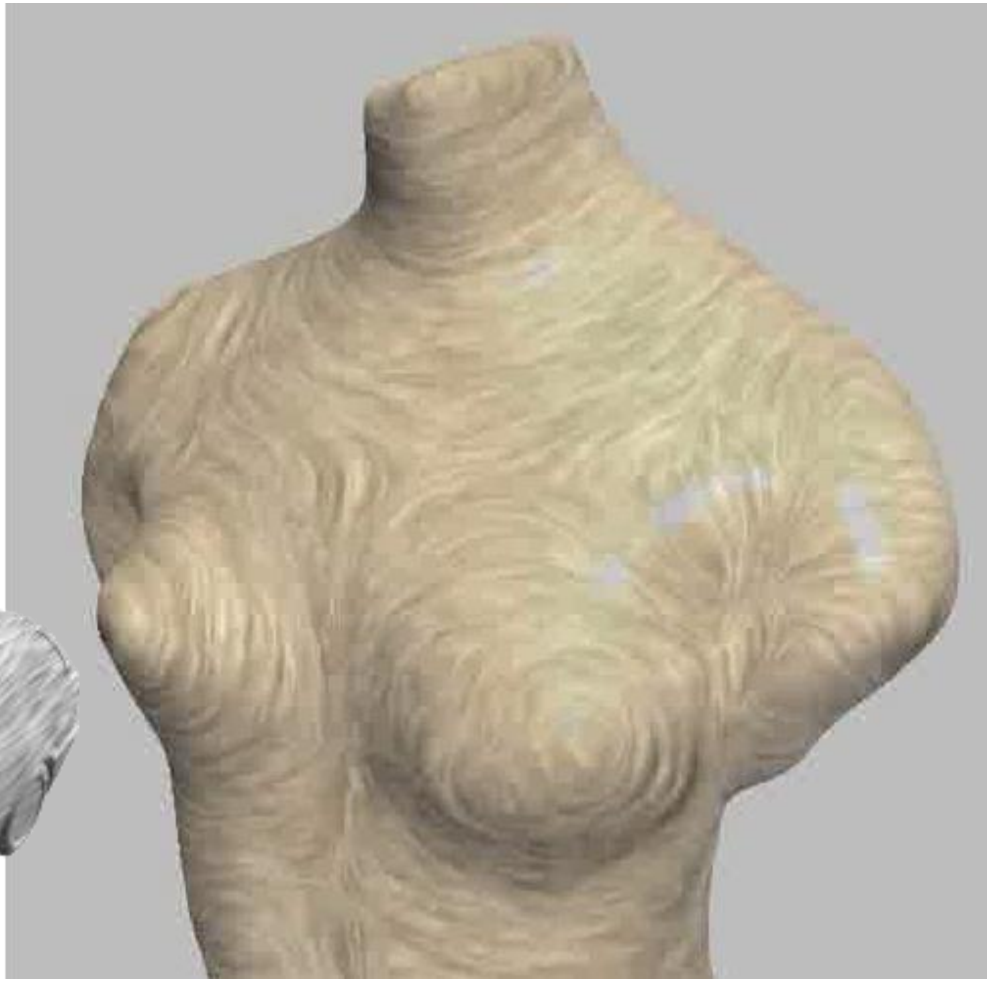


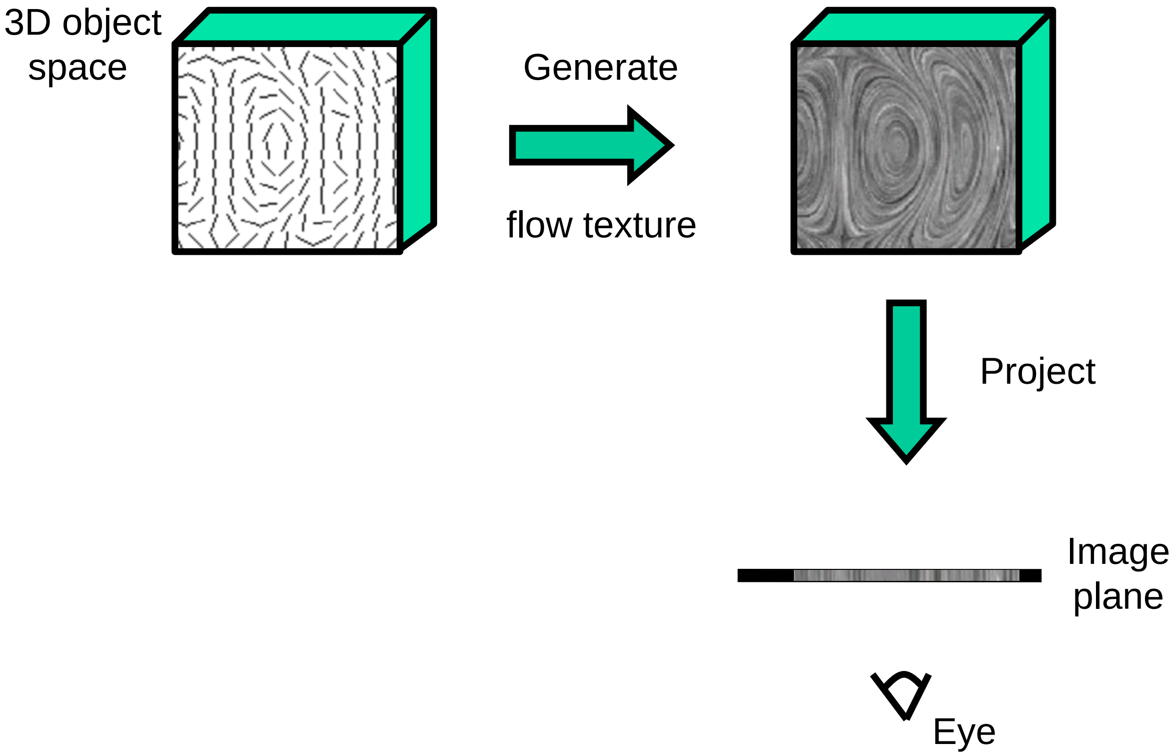


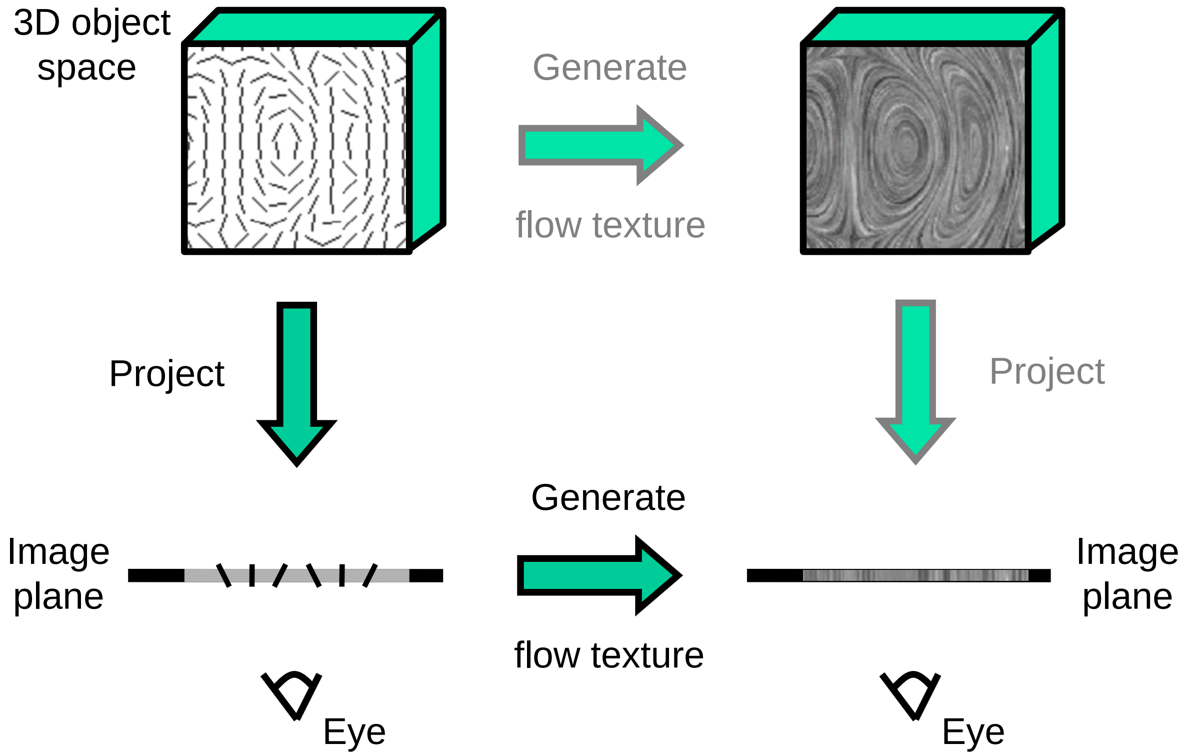
- Oriented LIC (OLIC)



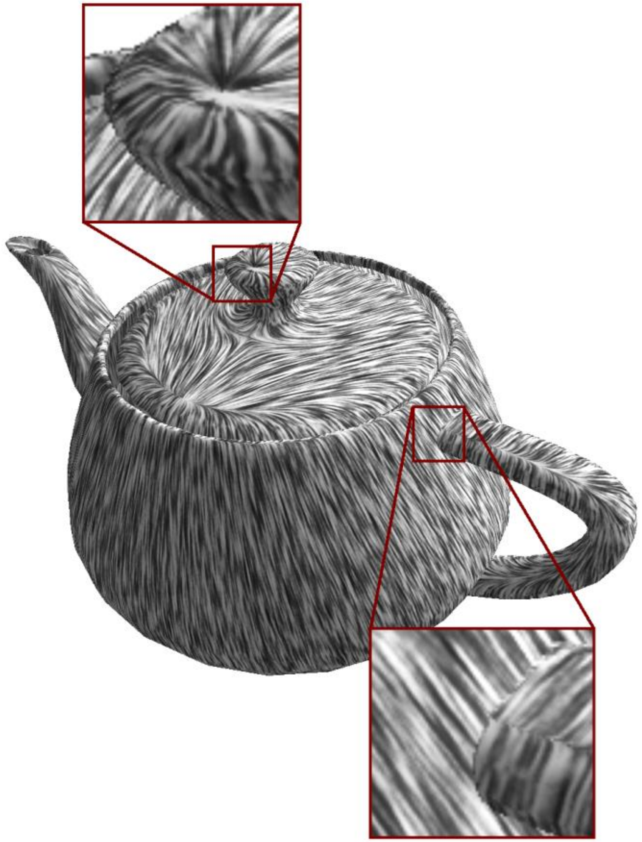
- LIC on Surfaces
- 2 possibilities:
- Integrate on surfaces
- Integrate in image space
- Show in Amira

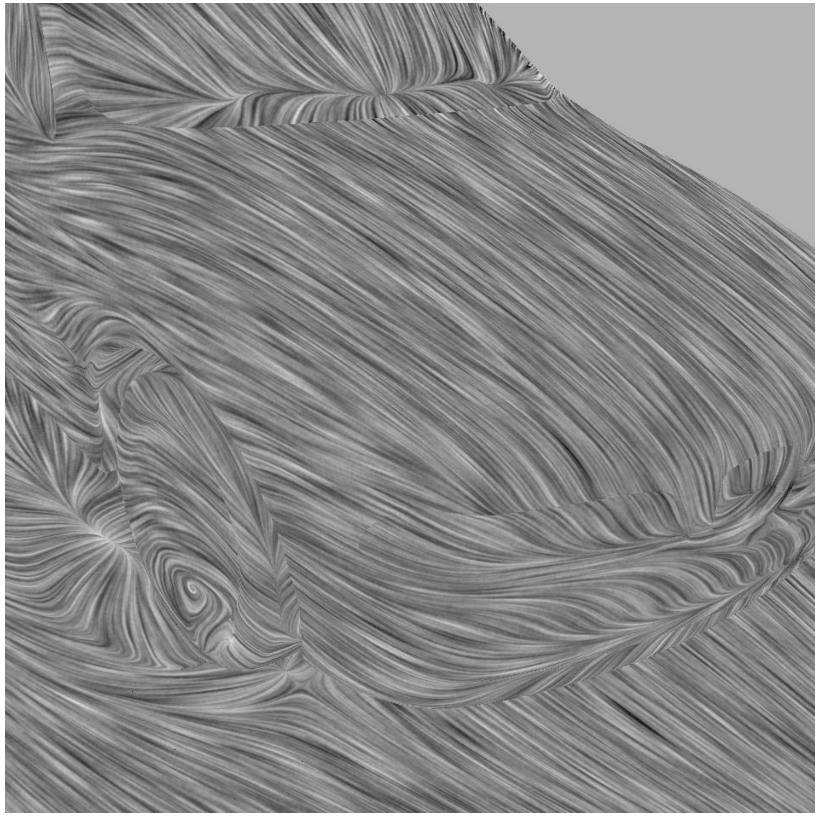



3D object
space
Generate
flow texture
Project
Image
plane
Eye

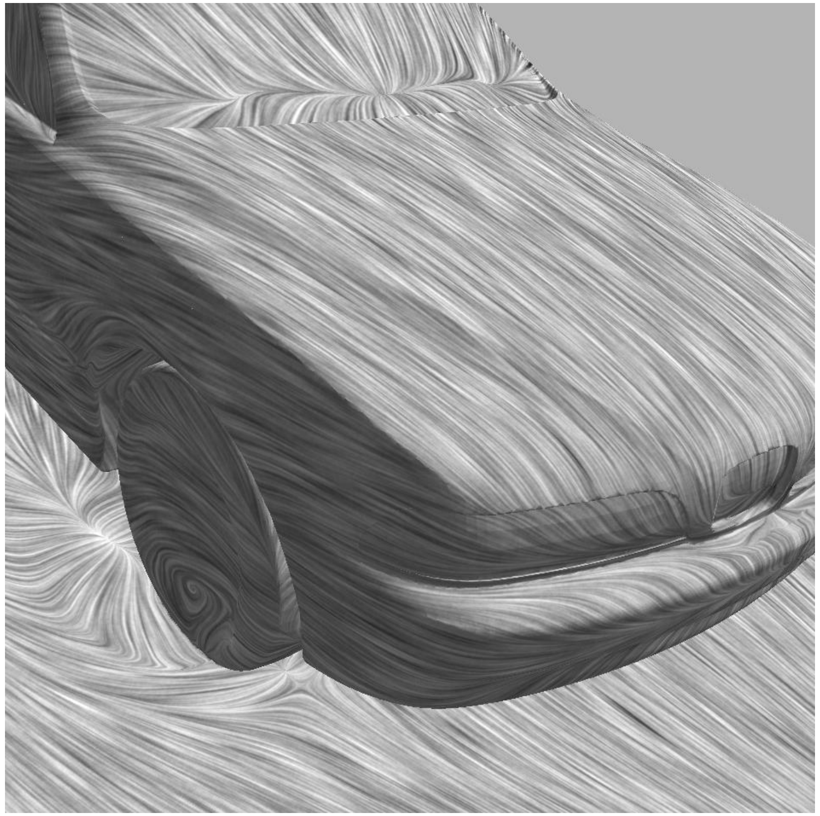


3D object space
Generate
flow texture
Project
Project
Generate
flow texture
Image plane
Image plane
Eye
Eye

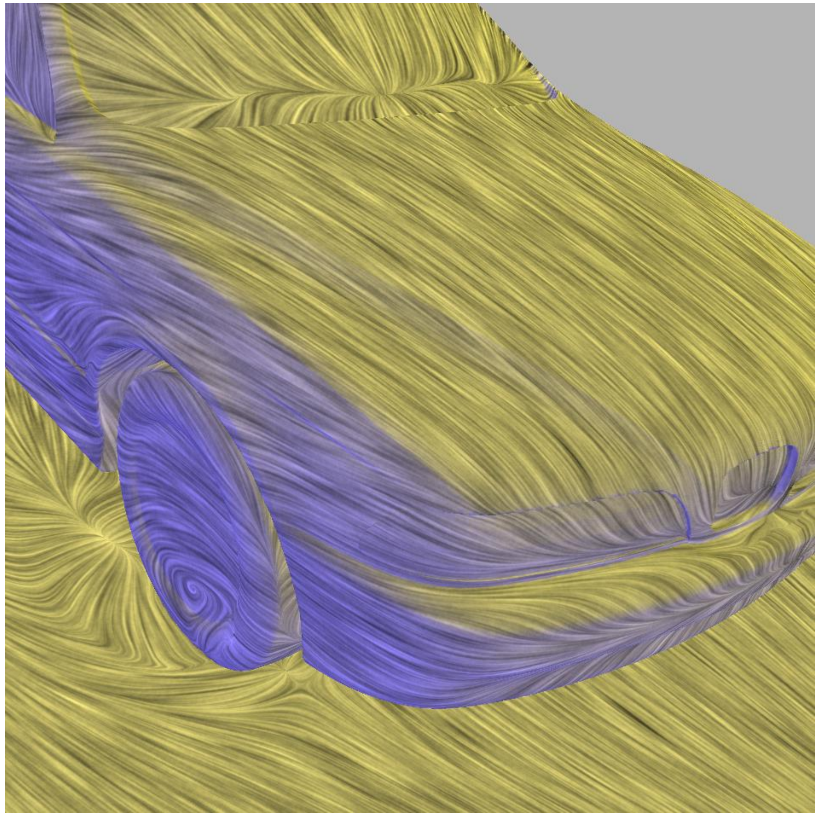




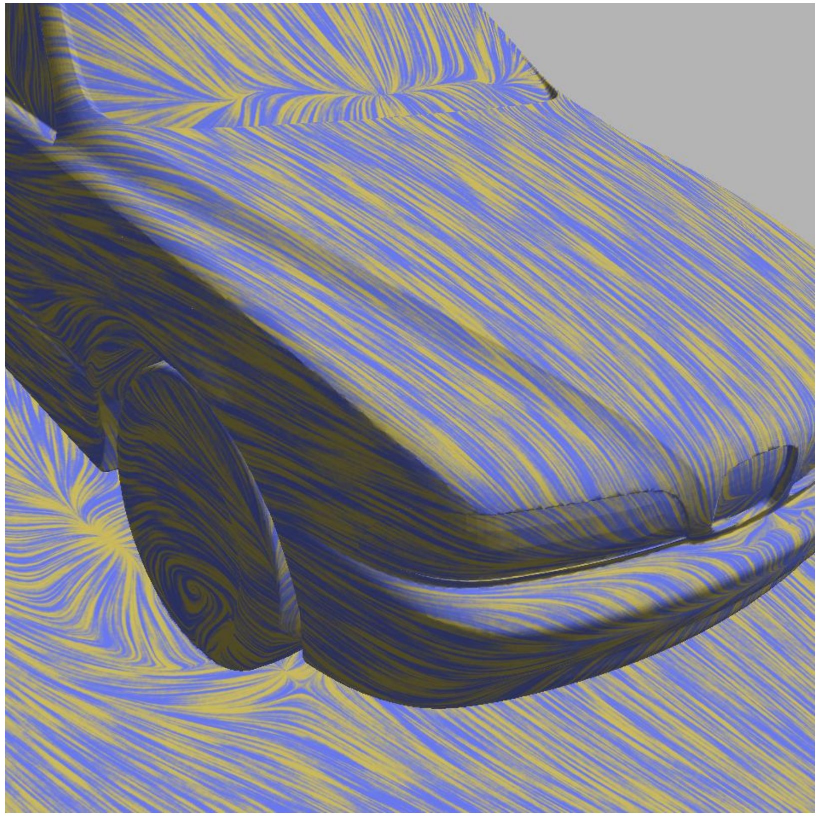
No surface shading



Gray surface

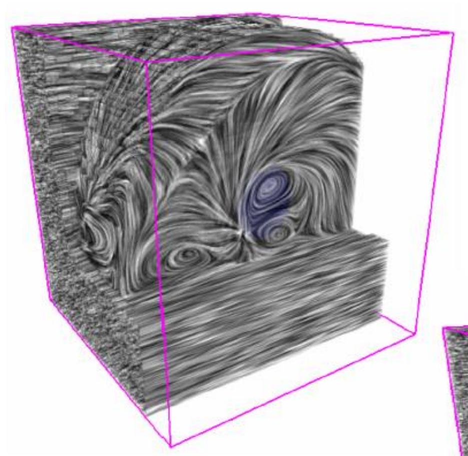


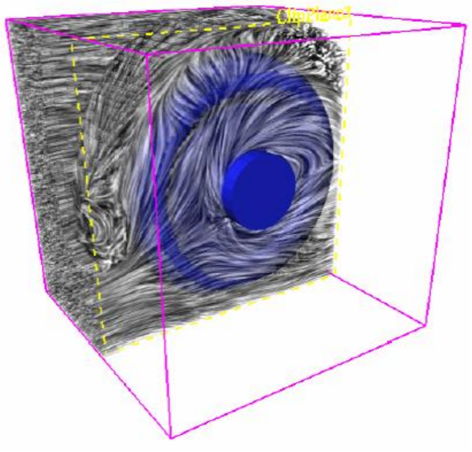
Cool/warm shading
for surface

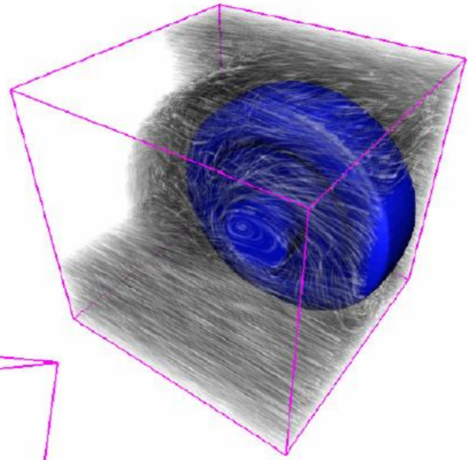


Gray surface
Yellow/blue flow

- LIC for 3D Flows
  - LIC concept easily extendable to 3D
  - Problem: rendering!

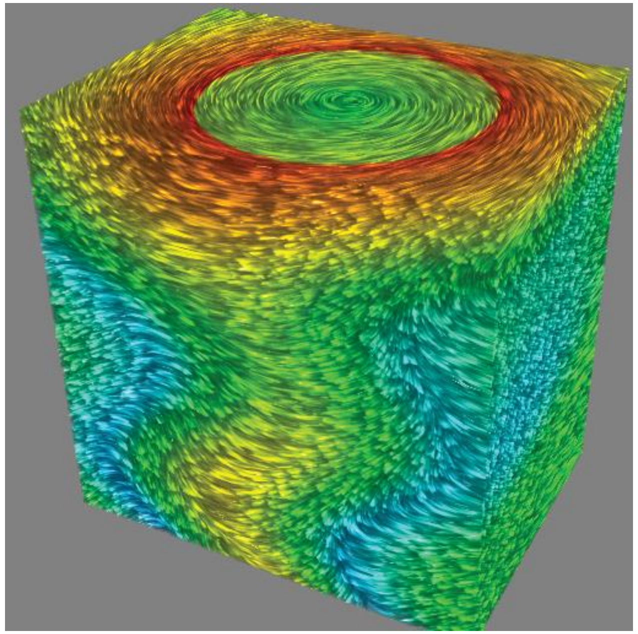




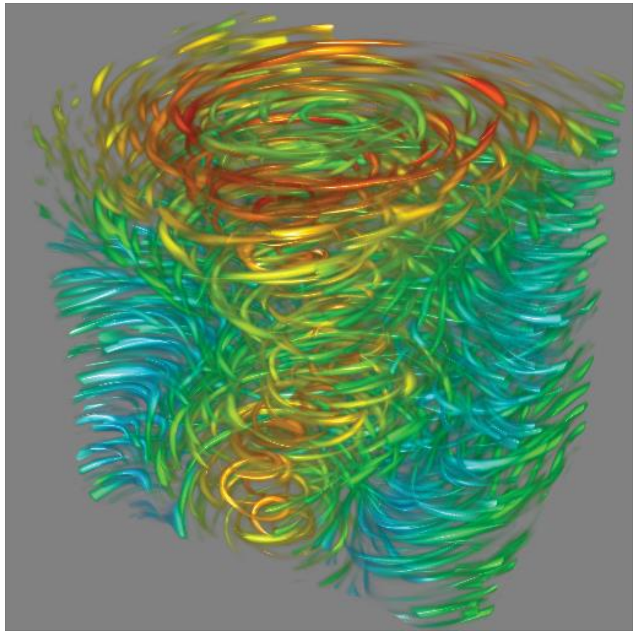


3D LIC can only reveal interesting structures, if some data is discarded.

Images courtesy of
[Rezk-Salama et al. 99]



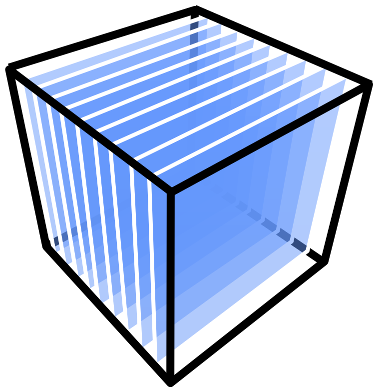
Dense Noise

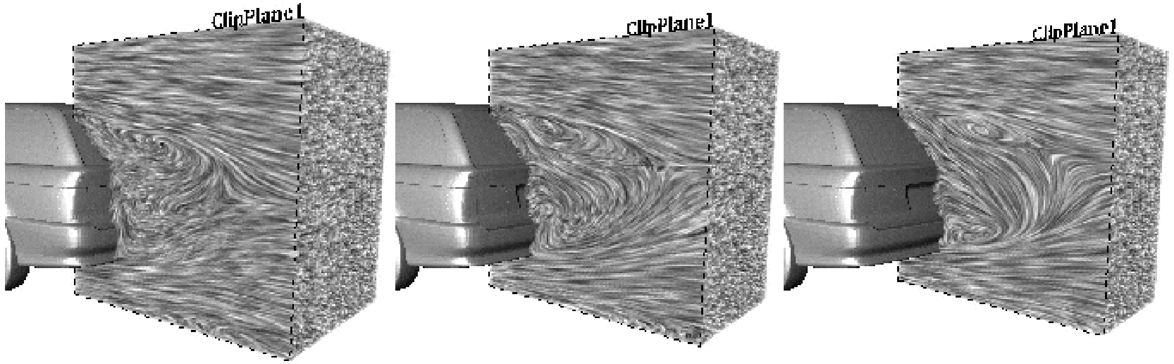


Sparse Noise

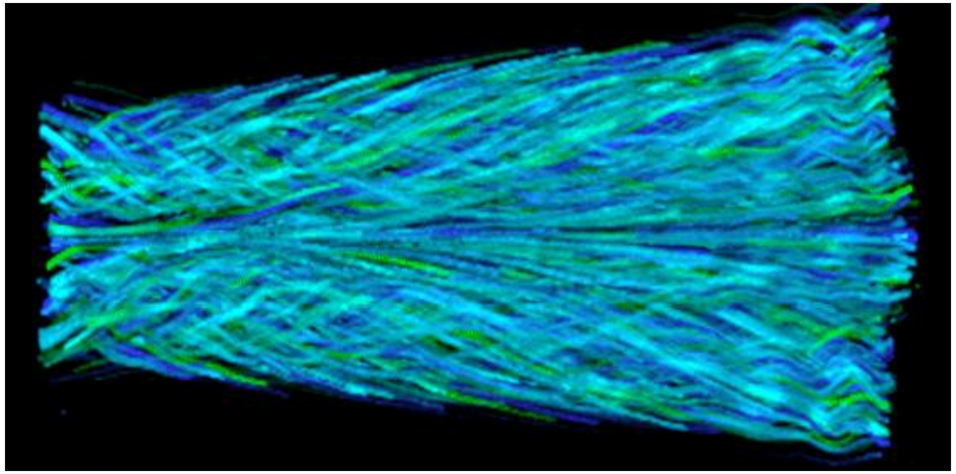
- Scalar Volume Rendering via
  - Slice Based Volume Rendering
  - Raycasting

- see Lecture on Volume Rendering

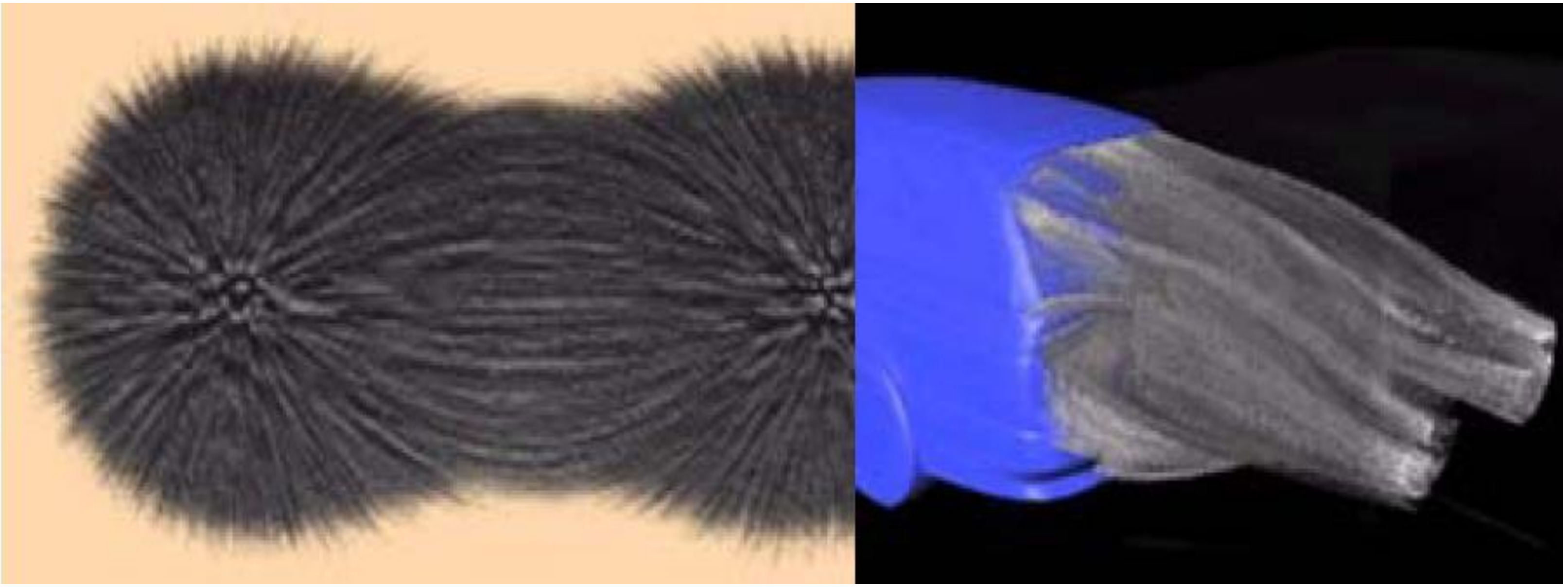




3D LIC volume is explored by interactively moving a clip plane (from Rezk et al 97)

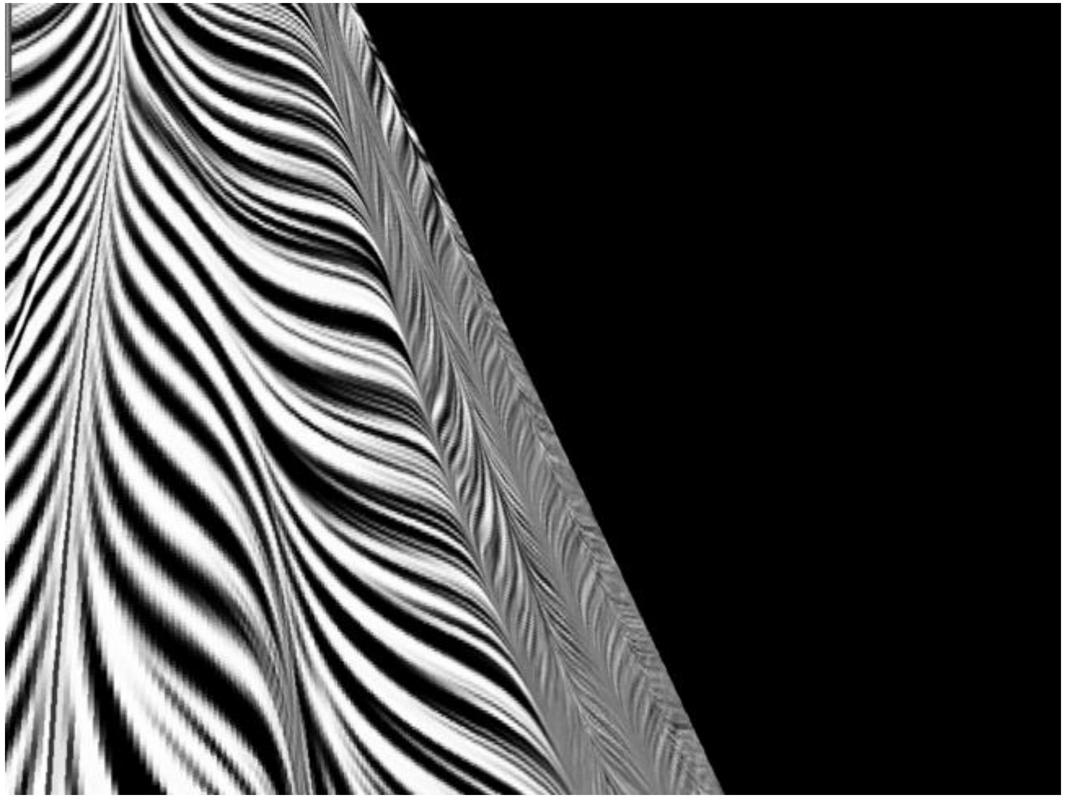


3D LIC using volume rendering (from Interrante 97)

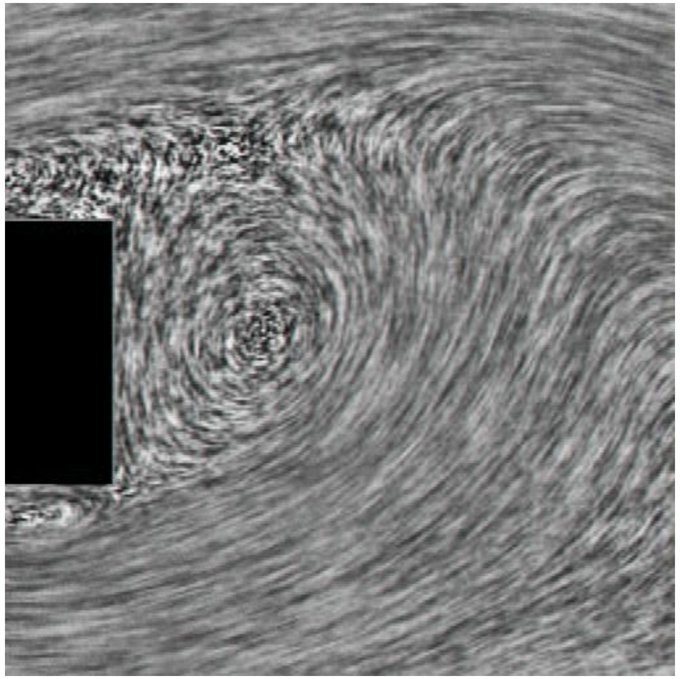


Use scalar quantities such as vorticity magnitude to define opacity for the final rendering.

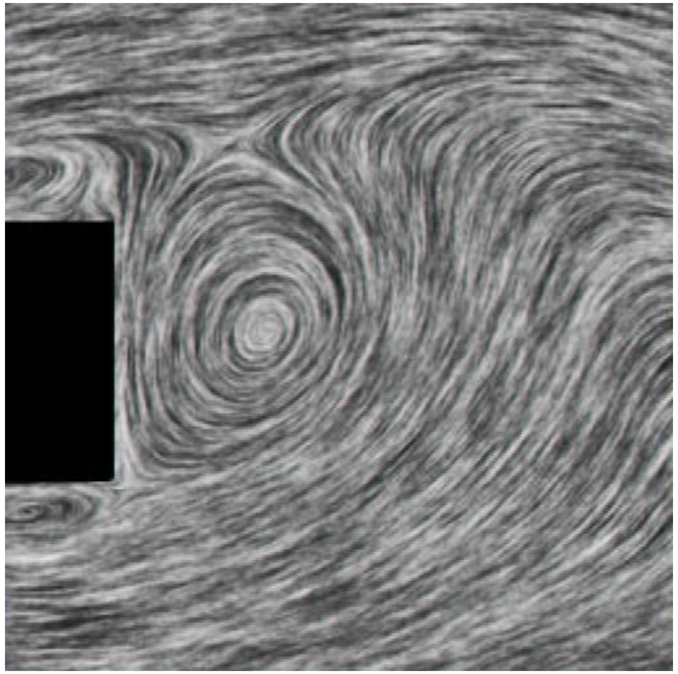
- **LIC for unsteady flows:**
- UFLIC [Shen, Kao 97]
- convolution along path lines (instead of stream lines)



- **Comparison of LIC and Spot Noise:**
- Spot noise: better encoding of velocity (magnitude)
- LIC: better encoding of critical points

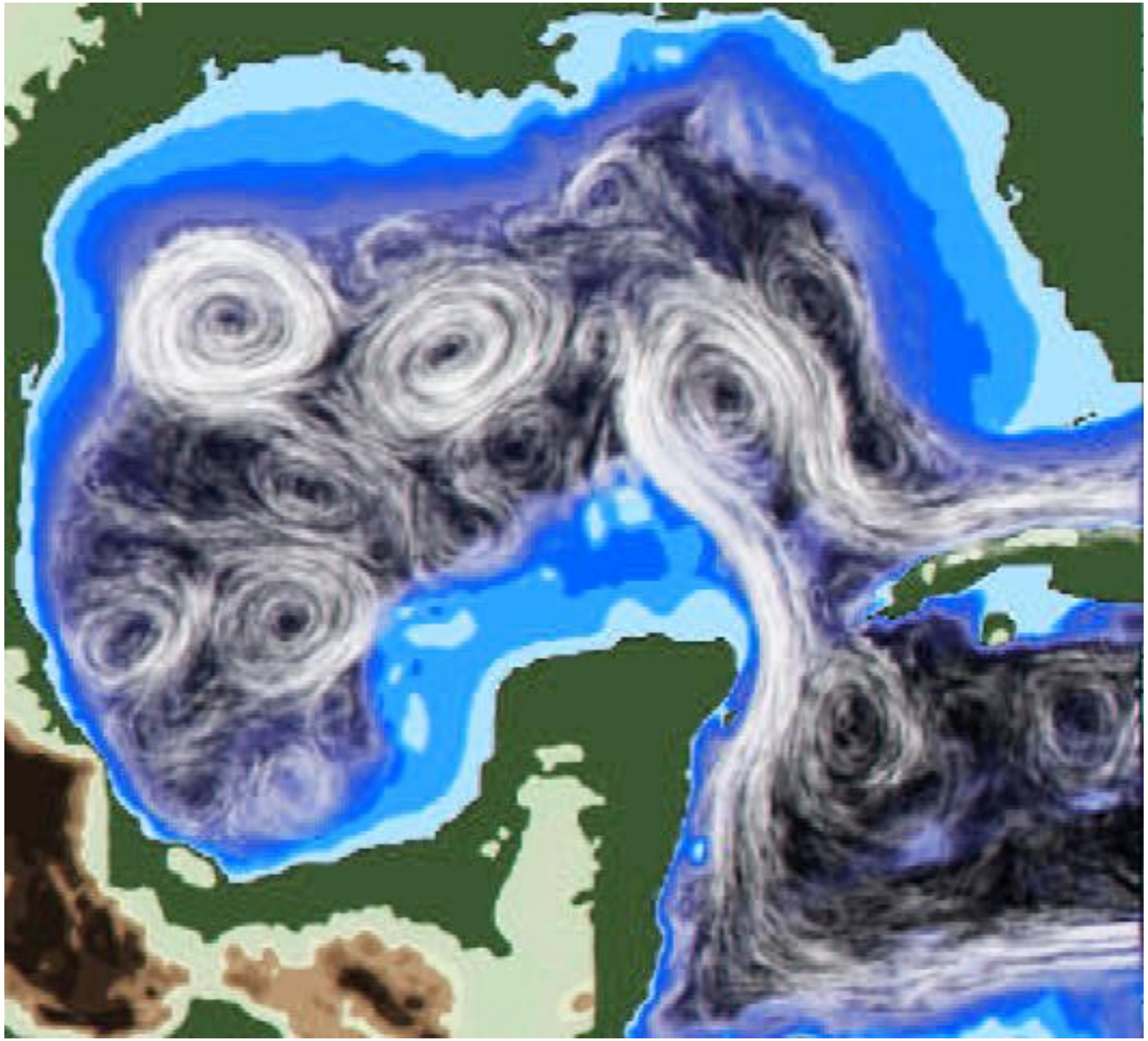


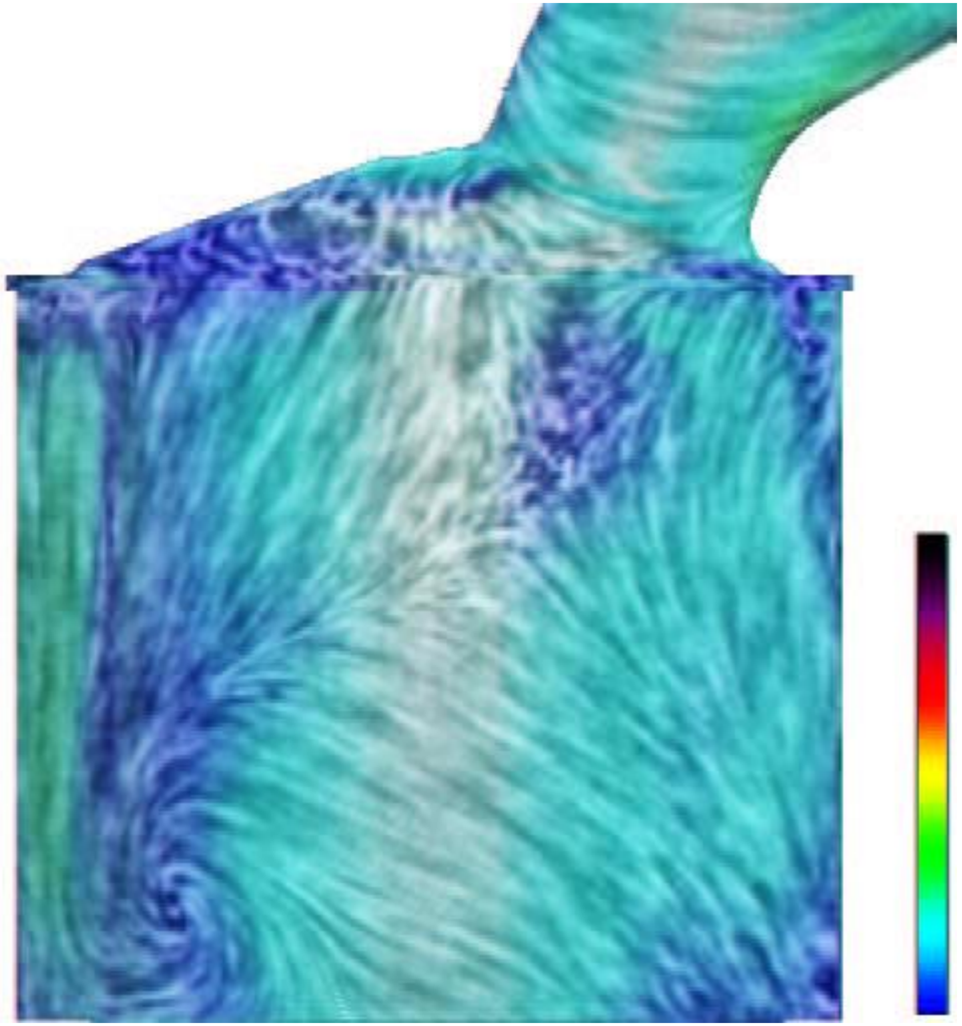
Spot noise



LIC

- **Texture advection**
- Based on moving (groups of) texels (texture elements)
- Classification by
  - Advection scheme: forward / backward
  - Advection primitive: texel / textured polygon
- Backward integration:
  - To get pixel color, integrate backwards over a certain time, get color from there
  - Iteration: backward integration / update
- usually comes with GPU implementation





Texture Advection on Stream Surfaces: A Novel Hybrid Visualization Applied to CFD Simulation Results

Robert S. Laramee
Christoph Garth
Juergen Schneider
Helwig Hauser

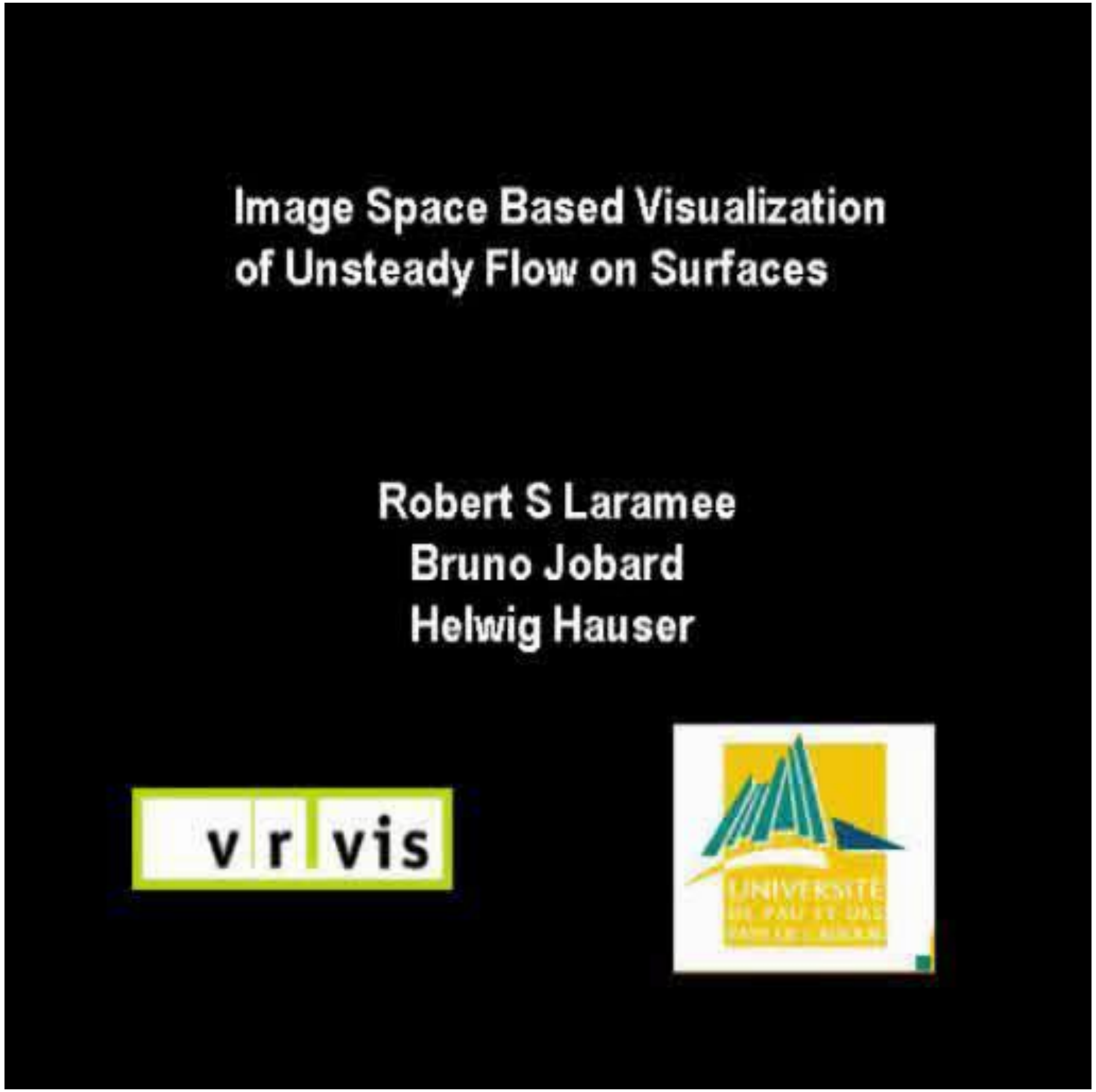


Image Space Based Visualization
of Unsteady Flow on Surfaces
Robert S Laramee
Bruno Jobard
Helwig Hauser
vrvis
UNIVERSITE

Investigating Swirl and Flow Motion

Robert S Laramee
Daniel Weiskopf
Jürgen Schneider
Helwig Hauser

Summary

- Image-based flow visualization
  - similar to real-world experiments
- Spot noise
- LIC
  - algorithmic details
  - extensions
- Texture advection