# 6

# Texture Advection

# *Texture advection*

Motivation: dense visualization of vector fields, no seed points needed.

Methods for static fields:

- LIC - Line integral convolution (Cabral/Leedom 1993)

Methods for time-dependent fields:

- LEA - Lagrangian-Eulerian Advection (Jobard et al. 2001)
- IBFV - Image-Based Flow Vis (van Wijk 2002)

Methods for vector fields on surfaces:

- IBFVS - IBFV for Surfaces
- ISA - Image-Space Advection (Laramee 2003)

# *Line integral convolution*

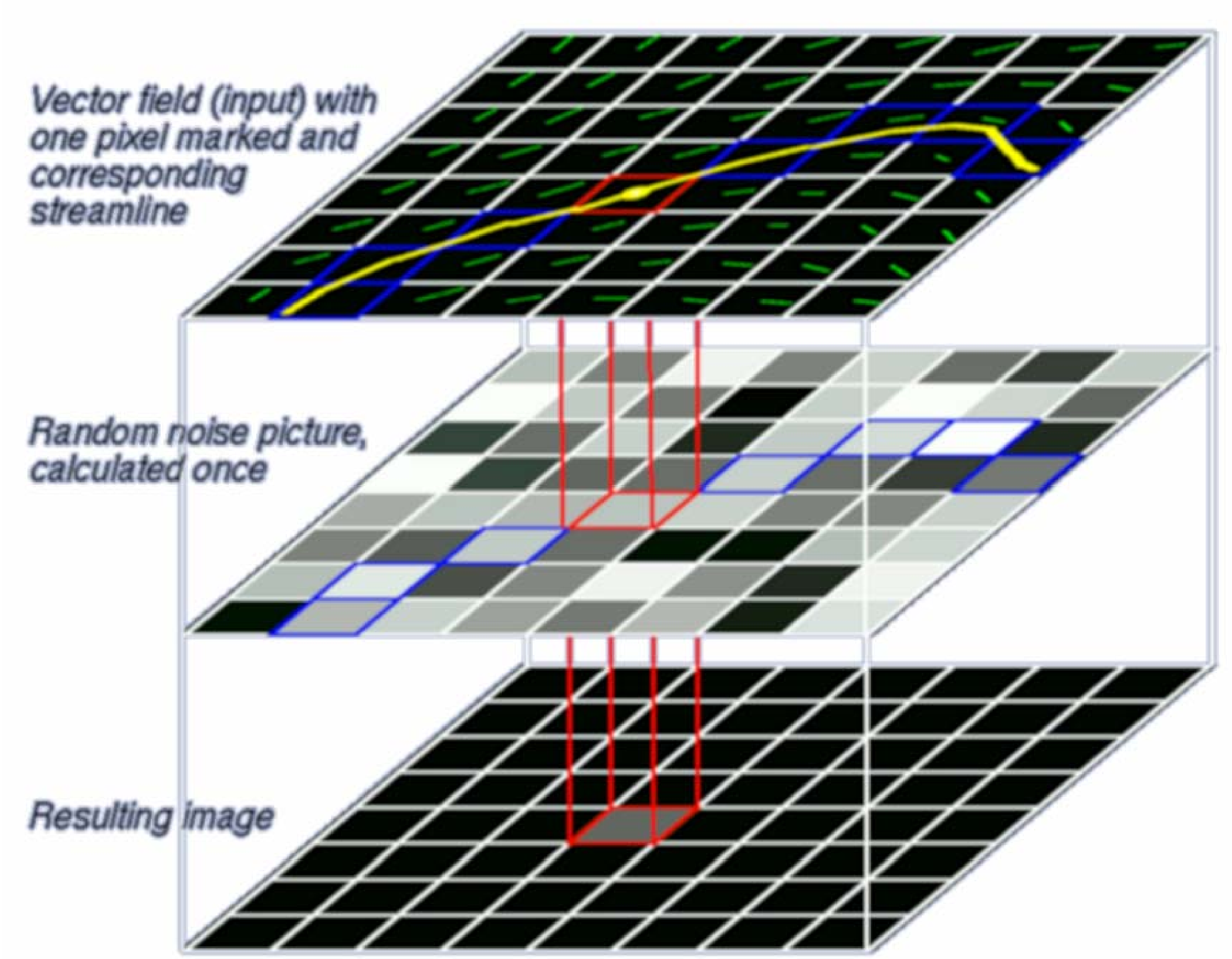
Line integral convolution (LIC) is a family of 10+ variants.

The original method by Cabral and Leedom assumes 2D vector fields on rectilinear grids.

Its basic idea is:

- generate a gray level image of random pixels, at the desired resolution
- per pixel compute forward and backward streamline segments of fixed arc length
- sample the random image along the streamline and compute the average, i.e. convolve with a box filter
- use the computed values as the pixels of the output image
- stretch the range of the output image

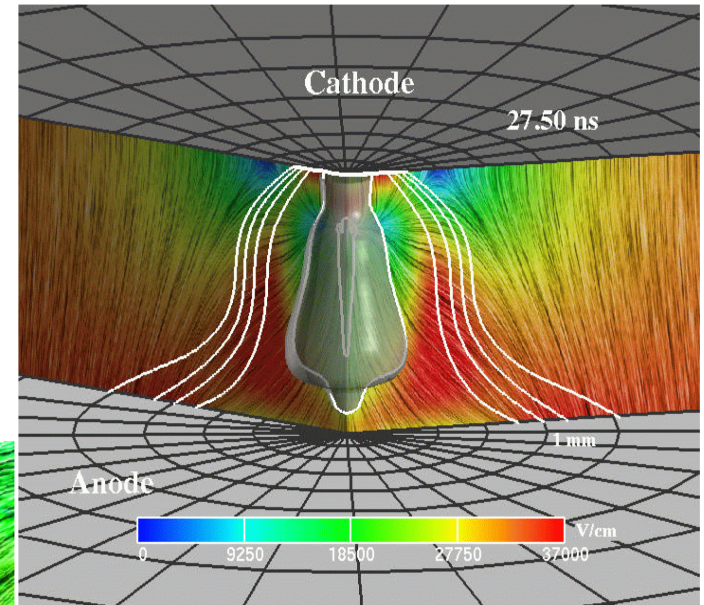
Illustration of the LIC principle:



LIC images can be combined with color coding of a scalar field.

in HSV space:

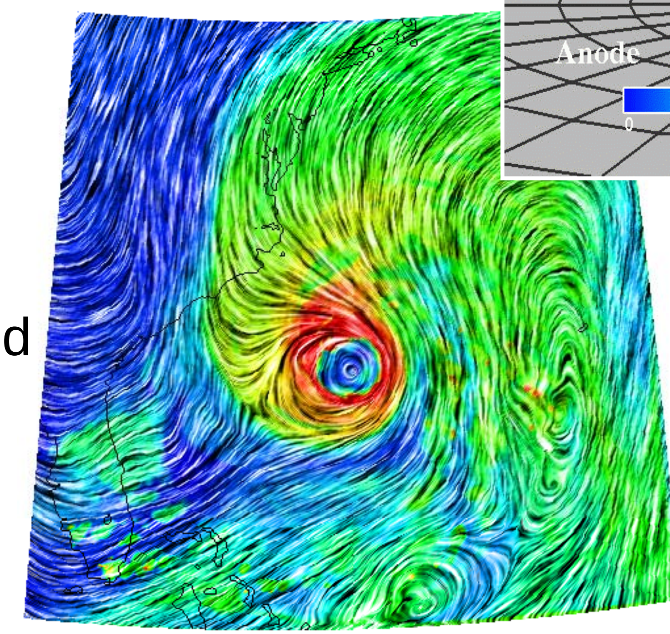
hue: scalar field
saturation: 1
value: LIC




(Image: J. Favre)

or in HLS space:

hue: scalar field
lightness: LIC
saturation: 1



The Fast LIC method (Stalling)

- is an order of magnitude faster by
- re-using parts of streamlines where possible

Fast LIC is the basis of most of the newer LIC methods.

LIC method for unstructured grids (Battke):

- use a procedural 3D random image
- compute a LIC image for each triangle as a separate texture map
- pack the triangle textures into texture memory

This method can be used also for vector fields on surfaces.

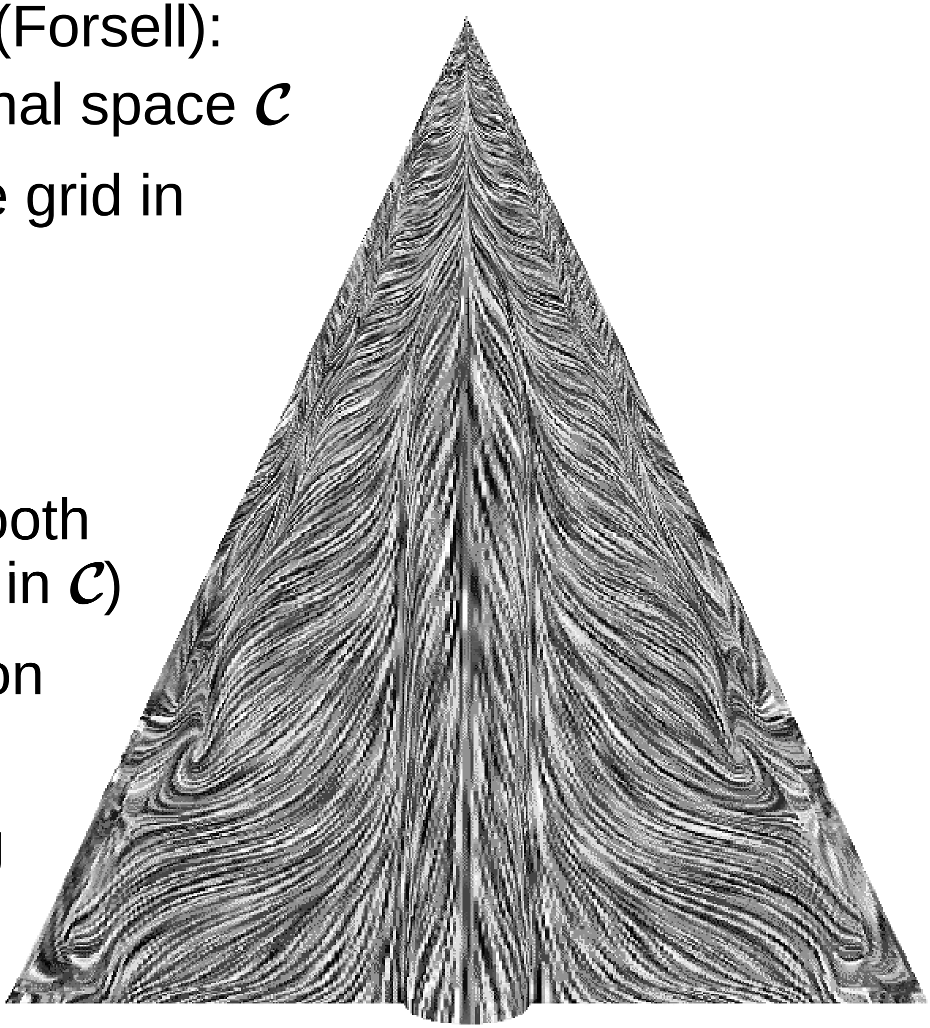
LIC method for curvilinear grids (Forsell):

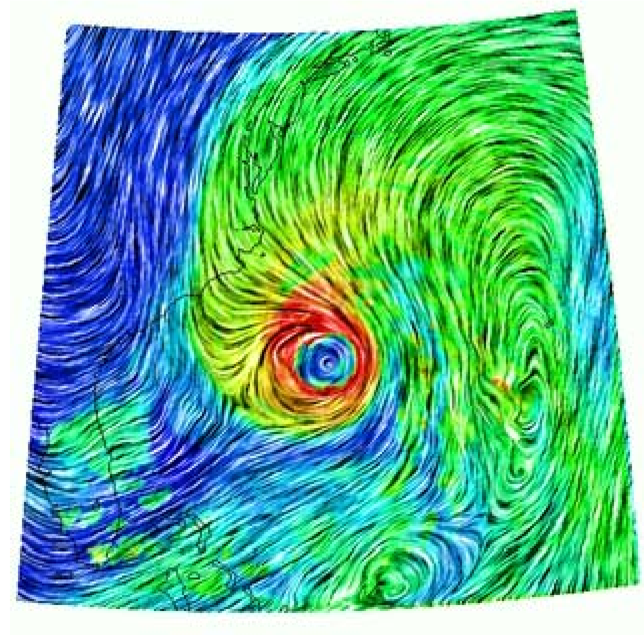
- generate a LIC in computational space $\mathcal{C}$
- use it as a texture map for the grid in physical space $\mathcal{P}$

Problems of this approach:

- if parameter lines are not smooth (cf. correctness of integration in $\mathcal{C}$)
- if cell sizes have large variation

Example: Flow over a delta wing

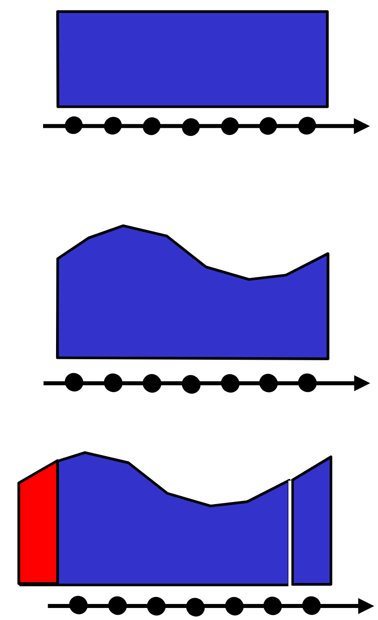




Animated LIC:

LIC of static vector fields can be easily animated to show the relative velocity magnitudes:

- use samples at constant time steps
- replace the box filter by a sinusoidal filter with exactly one period
- shift the kernel backward in steps of one sample
- this results in the texture moving forward



$t = t_0$

$t = t_0 + \Delta t$

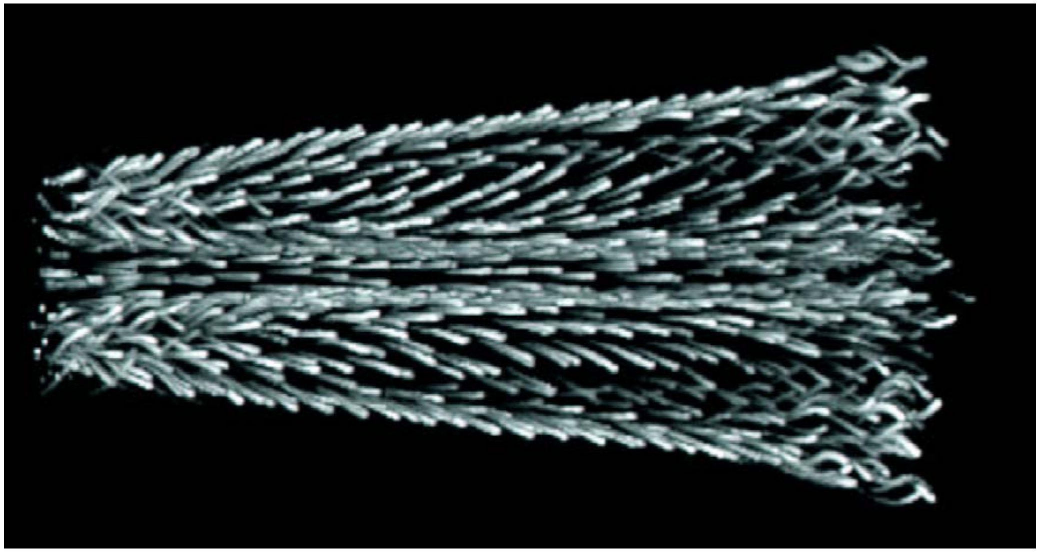
LIC can be computed easily also in 3D, but the result is a 3D image.

Rendering options:

- isosurfaces: no (sensitive to noise, LIC is a near worst case!)
- direct volume rendering: yes

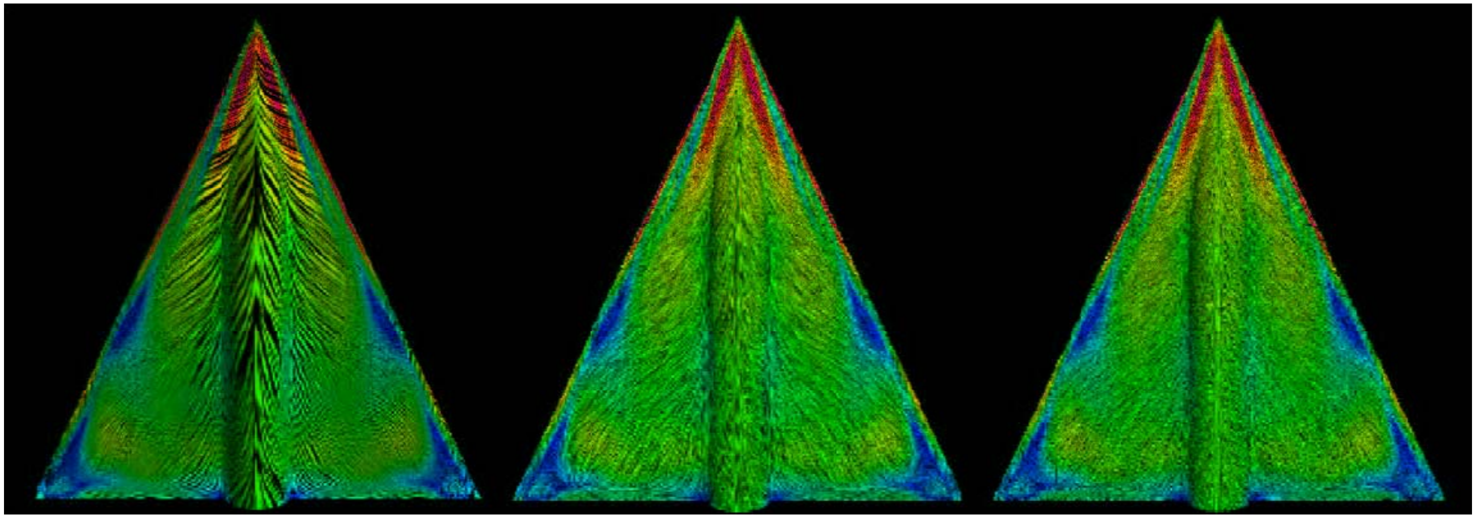
3D LIC method (Interrante/ Grosch):

- Fast LIC in 3D
- DVR
- TF with mostly low opacity
- "halos" in image space



Other LIC variants:

UFLIC, ELIC, PLIC



(image: Verma)

…, AUFLIC, OLIC, FROLIC, DLIC, GPULIC, ...

## *Lagrangian-Eulerian Advection*

Dynamic behavior can be expressed in either Eulerian or Lagrangian formulation:

Eulerian or grid-based: Fields are given at grid nodes

Lagrangian or particle-based: A set of particles is advected by the velocity field $\mathbf{v}(\mathbf{x},t)$, other fields are given at particle positions

The temporal change of a function $f(\mathbf{x},t)$ while following a particle is expressed by the material derivative (or convective derivative):

$$\frac{Df(\mathbf{x},t)}{Dt} = \frac{\partial f(\mathbf{x},t)}{\partial t} + \nabla f(\mathbf{x},t) \cdot \mathbf{v}(\mathbf{x},t)$$

Example: acceleration is $\frac{D\mathbf{v}(\mathbf{x},t)}{Dt} = \frac{\partial \mathbf{v}(\mathbf{x},t)}{\partial t} + \nabla \mathbf{v}(\mathbf{x},t) \cdot \mathbf{v}(\mathbf{x},t)$

Lagrangian-Eulerian methods are combinations.

Lagrangian-Eulerian Advection (LEA) method for vector field visualization (Jobard 2001) uses one particle per cell:

Initialize a white noise texture (as for LIC)

For each time step do:

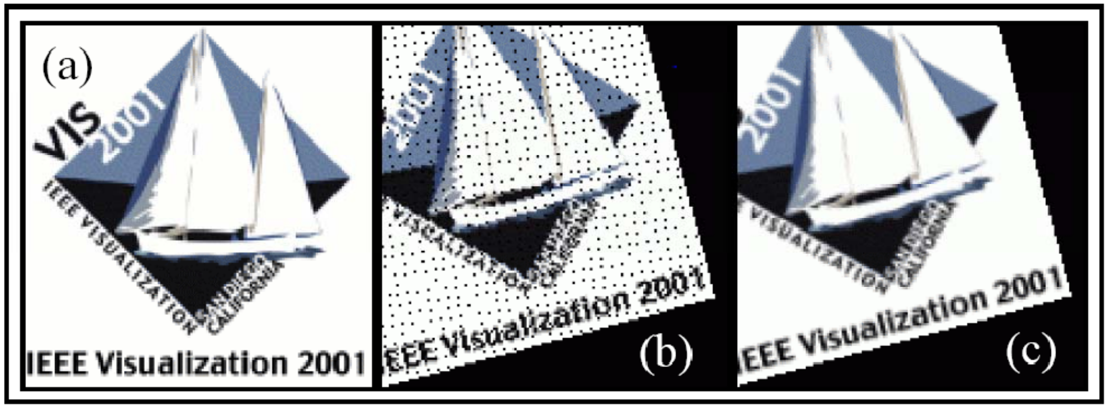
For each particle do:

- integrate backward pathline segment, giving new texel value for the cell
- integrate forward pathline segment, giving new particle position in same cell (local coordinates modulo 1)

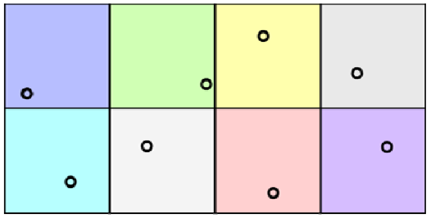
Why backward integration?

Texture advection can be done as forward mapping (Lagrangian scheme) or backward mapping (Eulerian scheme).

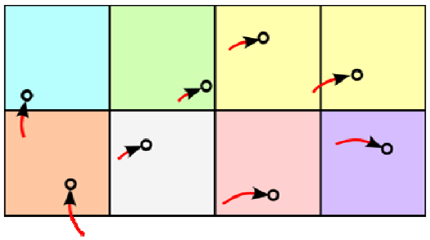
Example: Image rotation (a: original, b: forward, c: backward)



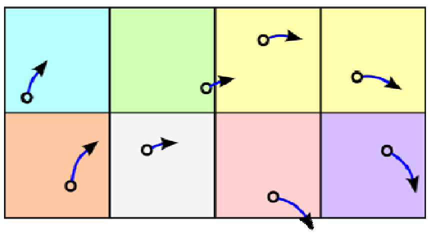

Backward mapping is the better choice, avoiding holes.



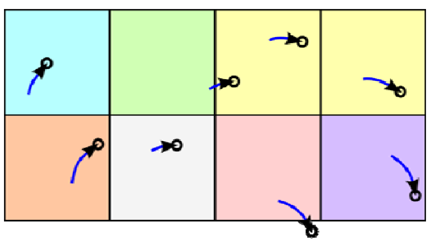
particle positions



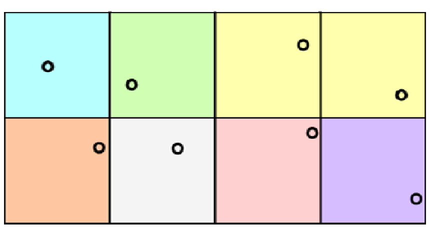
backward pathline segments, texture advection



forward pathline segments



new particle positions before resetting to their cells



after resetting (taking local coordinates modulo 1)

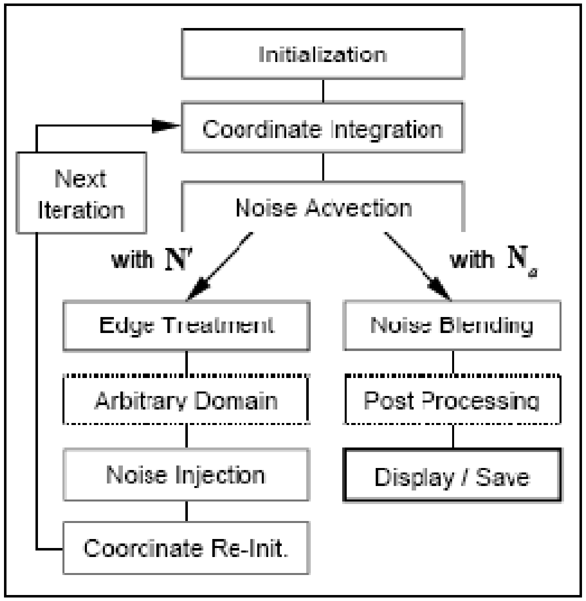
Special choices made by LEA:

- $1^{st}$ order integration
- simplification: forward segment = - backward segment
  (better would be: backward segment = - previous forward segment!)
- add buffer cells at grid boundaries
  - contain texture but no particles
  - allow texture advection at inflow boundaries
  - random texture is refreshed after each time step to avoid artifacts
- post-processing: apply a LIC filter to each image before outputting

Backward mapping scheme allows 2 interpolation choices:
nearest-neighbor or bilinear.

LEA uses both:

- nearest-neighbor is used for updating stored texture
- bilinear interpolation is used for displayed texture

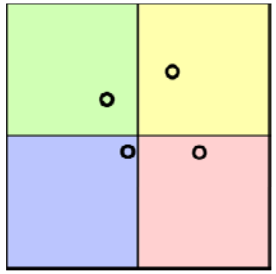


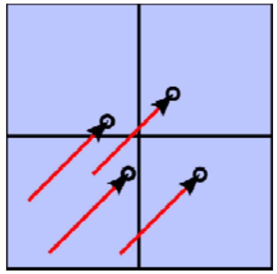
Backward mapping can have a duplication effect.

Causes are:

- divergence of the vector field
- nearest-neighbor interpolation

Example: a texel is copied to three neighbors in a single step (uniform flow, no divergence).





Noise injection: a small percentage of noise is added after each step.

Trade-off: keep high frequencies, but also temporal correlation.

LEA algorithm applied to wind prediction data:

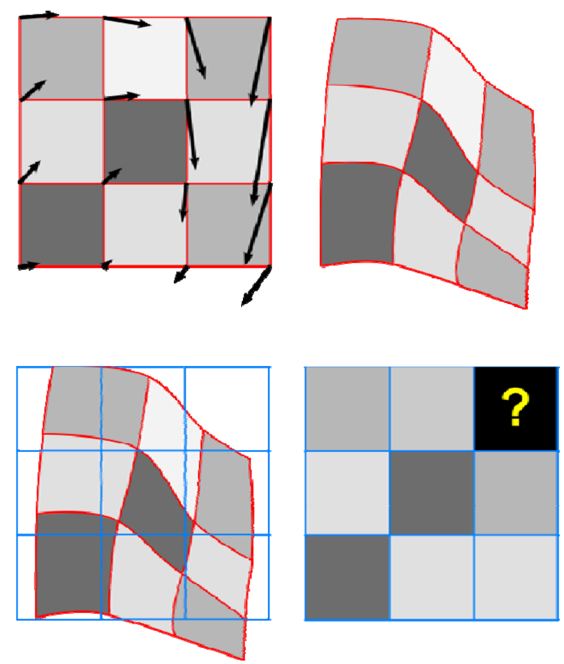

`http://srnwp.cscs.ch/Gallery/texture_loop.html`

# *Image-Based Flow Visualization*

IBFV algorithm (van Wijk 2002):

Main idea:

- Initialize a noise texture image
- For each time step do:
  - advect nodes of the texture image, resulting in a warped grid
  - render the warped grid, texture-mapped
  - resample image to original mesh, i.e. read back the rendered image to texture memory,
  - use as next texture image



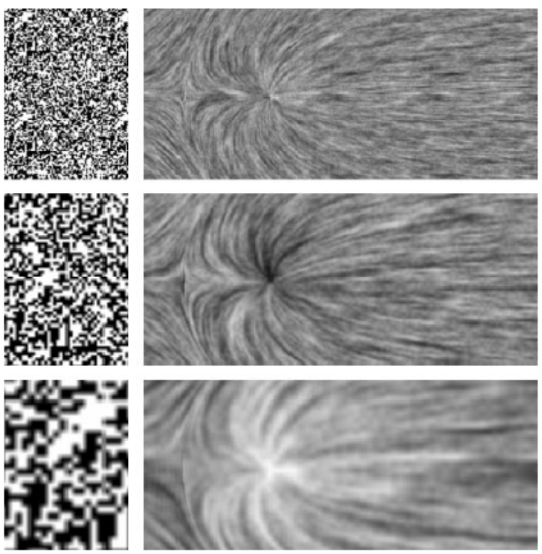
Detailed algorithm:

- Initialize a noise texture image
- For each time step do:
  - advect nodes of the texture image, resulting in a warped grid
  - render the warped grid, texture-mapped
  - blend with noise image
  - apply dye injection
  - resample image to original mesh
  - use as next texture image
  - draw overlaid graphics

Noise image:

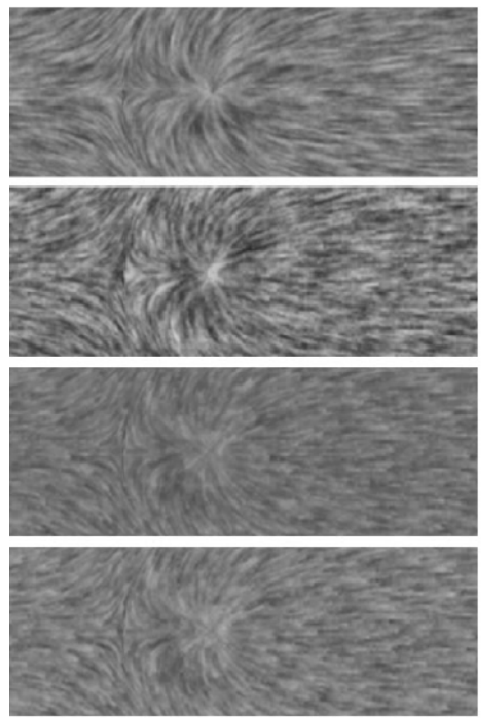
- static, resulting in static image for steady flow, or
- temporally coherent, using spot noise texture

different spot sizes



(images: van Wijk)

different temporal profiles



cosine

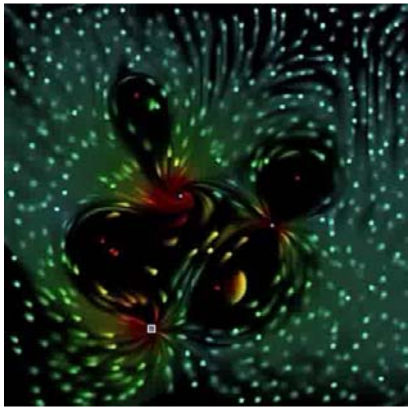
box

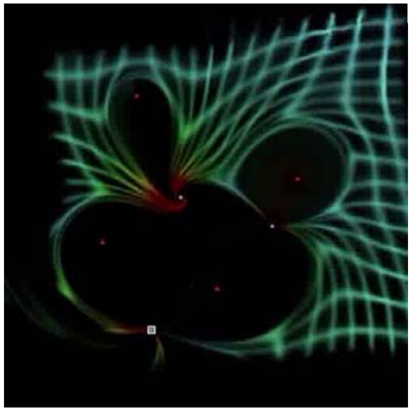
exponential decay

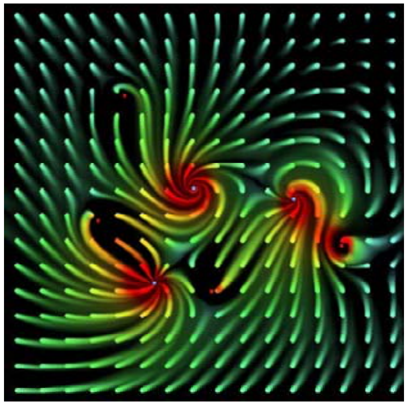
saw tooth

Dye injection:

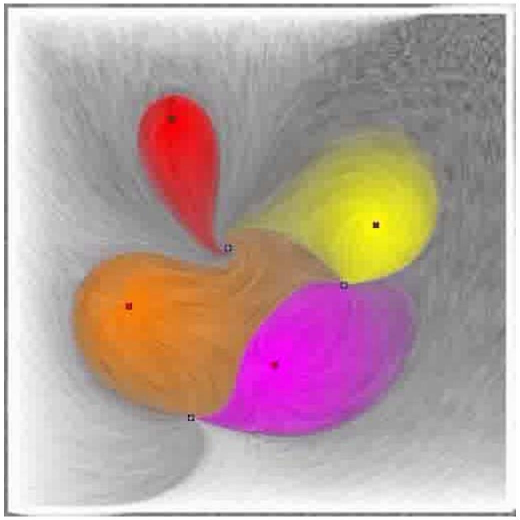
- once vs. continually (streaklines)





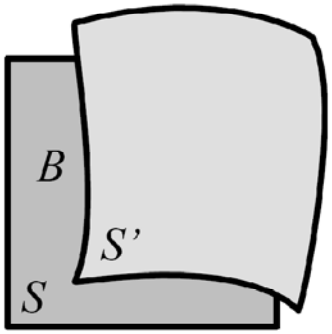


- together with texture



Boundary areas:

- special handling is needed (area B = S - S')
- simple solution: just don't clear the screen before redrawing!

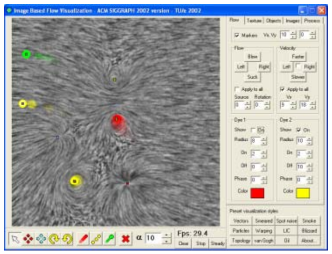



Comparison with LEA:

- much faster (grid to advect can be coarser than texture image)
- coherence is not as good (IFBV can simulate LIC, but only with exponentially decreasing filter weights)

Website and demo tool

`http://www.win.tue.nl/~vanwijk/ibfv`



# *Texture advection on surfaces*

Texture advection on surfaces can be used for:

- boundary flow (wall shear stress)
- flow on streamsurfaces
- less meaningful: projected flow on other surfaces (isosurfaces,…)

Possible but expensive:

- work in object space
- use 3D texture

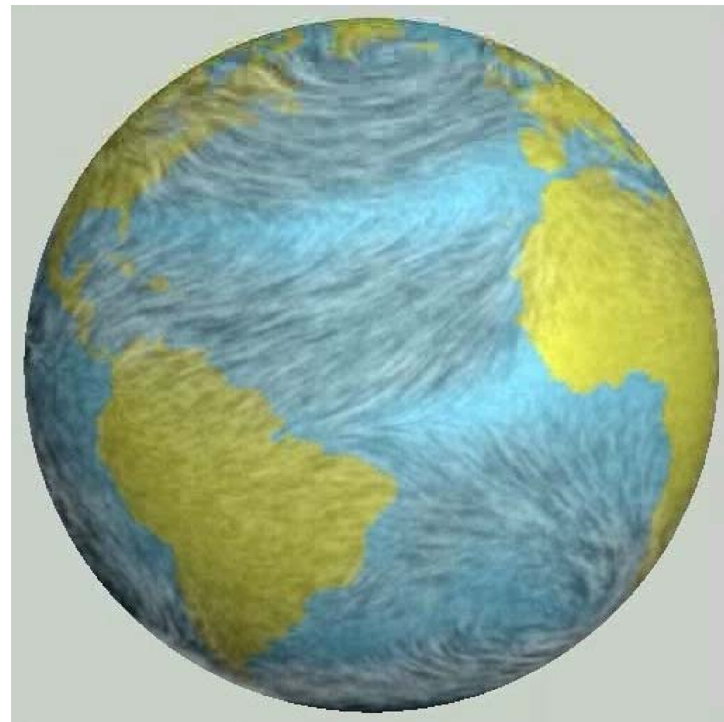
Alternative:

- work in image space
  - IBFV for surfaces (van Wijk)
  - Image-space advection (Laramee)

# *IBFV for surfaces*

Idea of IBFVS

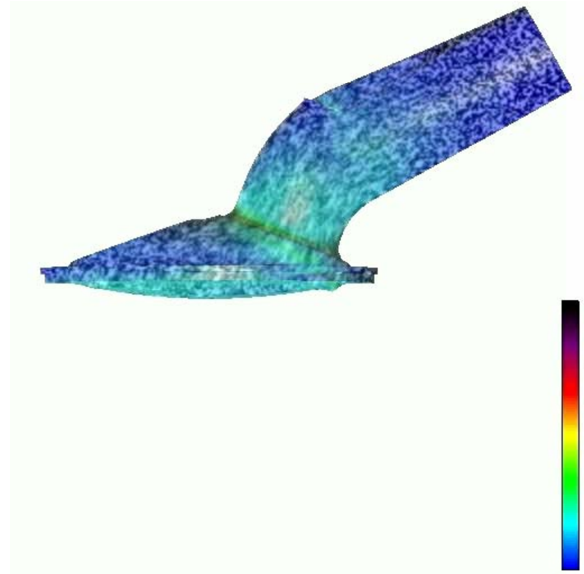
- use screen coordinates from previous rendering as texture coordinates
- advect in object space, i.e. distort the surface mesh
- render the distorted mesh, keeping texture coordinates
- apply noise injection and blending
- overlay image

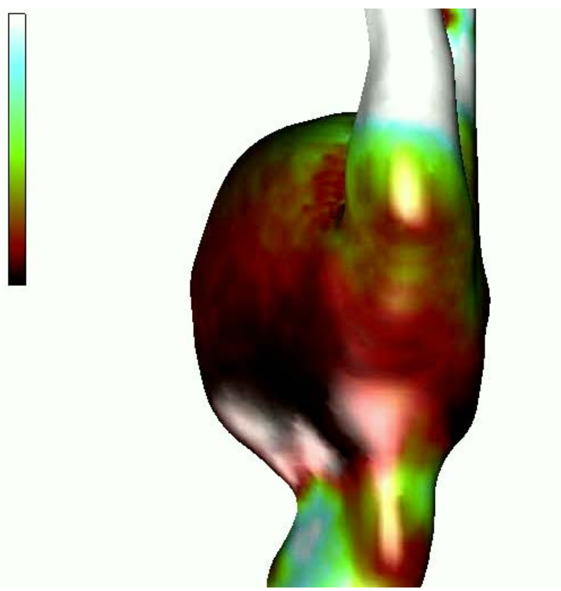


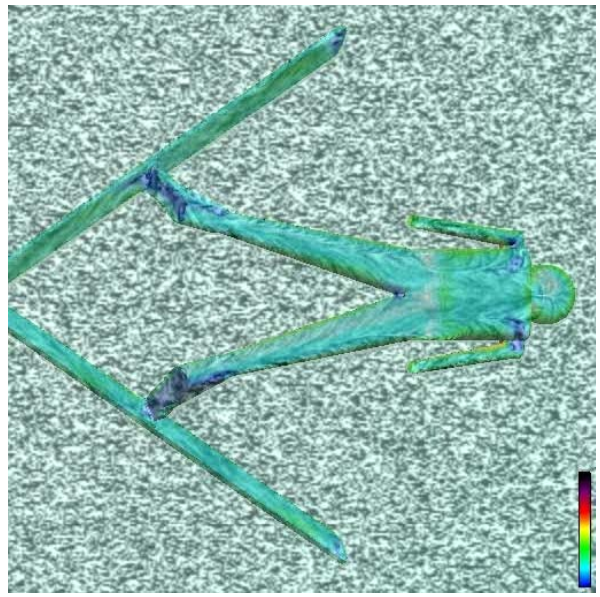
# *Image-space advection*

Idea of ISA:

- project the velocity field to image space
- do IBFV within boundary silhouette, i.e. advect rectangles
- apply noise injection and blending
- overlay image







Videos: Laramee (http://www.vrvis.at/scivis/isa-ibfvs/)

Comparison with IBFVS:

Advantages of ISA:

- projected velocity field simplifies advection
- no computation time is spent for polygons smaller than a pixel

Problems of ISA:

- artificial continuity across interior silhouettes
  - ISA uses edge detection (depth discontinuities)
- texture is not attached to surface when camera is moving