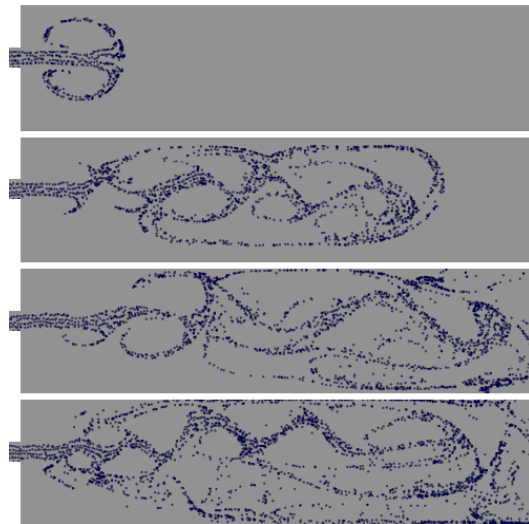

FLOW VISUALIZATION

CS 176 SPRING 2011

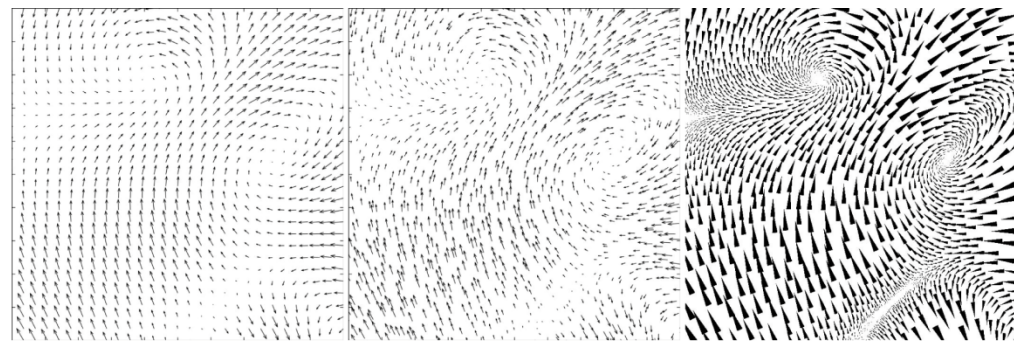
FLUID SIMULATION

Looking at vector fields...

■ how?



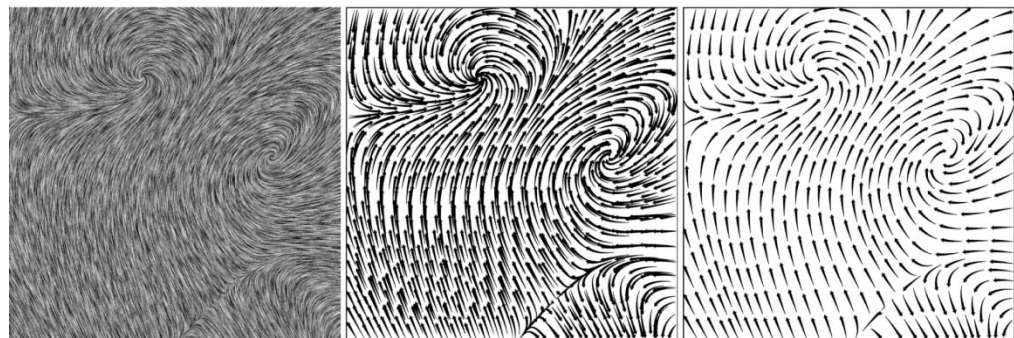
Bolz et al.



GRID

JIT

LIT



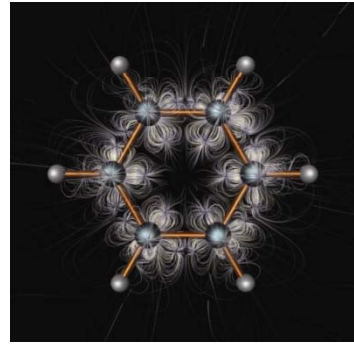
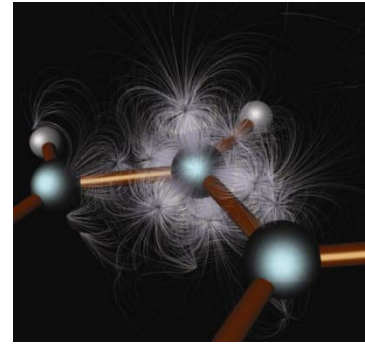
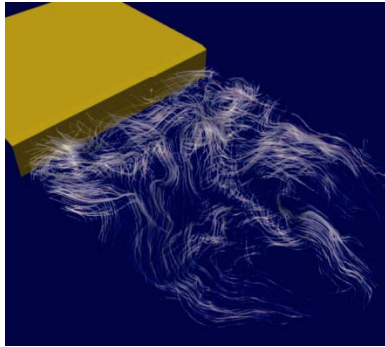
LIC

GSTR

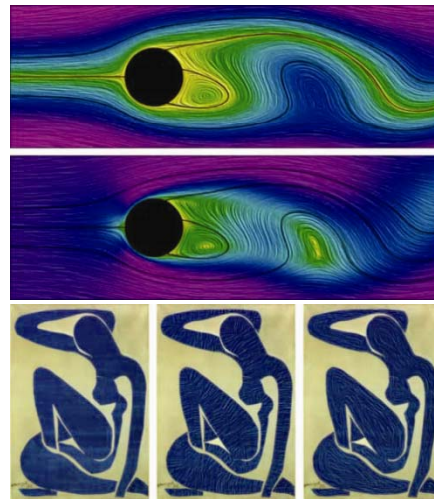
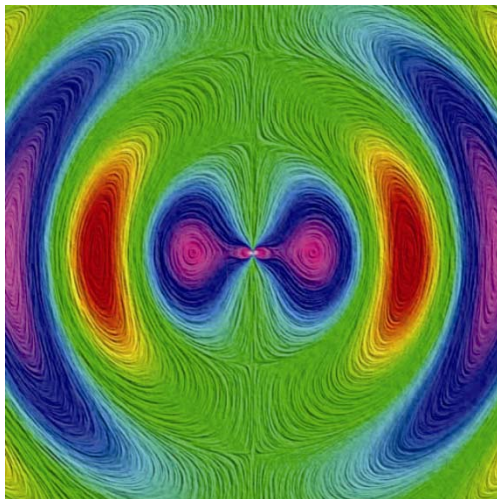
OSTR

Laidlaw et al.

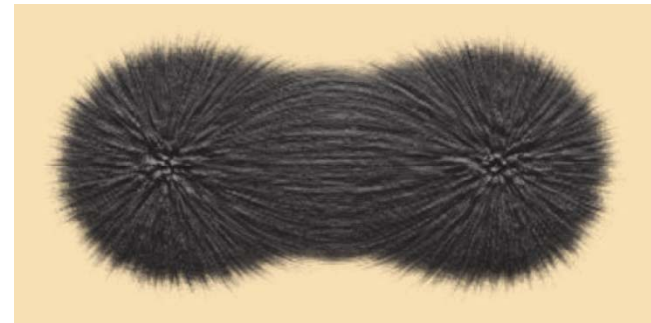
FIELD VISUALIZATION



Stalling et al.



Stalling et al.

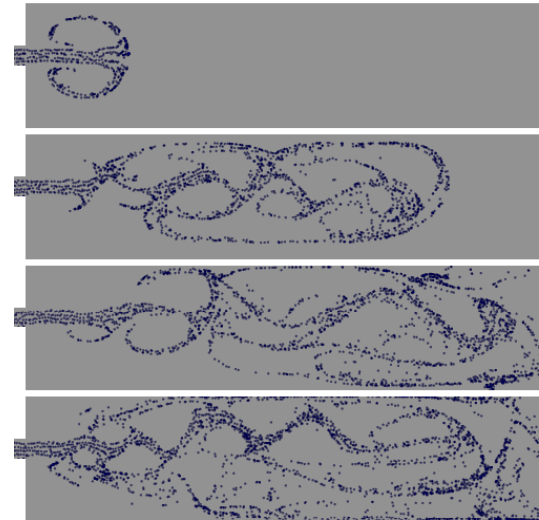


Cabral et al.

METHODS

Physical analogies

- particle advection
 - static or dynamic
- ink advection



Bolz et al.

Issues:

- missing important detail

TEXTURE METHODS

Advection

- advect texture coordinates with velocity; display regular grid with original texture
 - next time step: use previous texture
 - blend with exponential decay

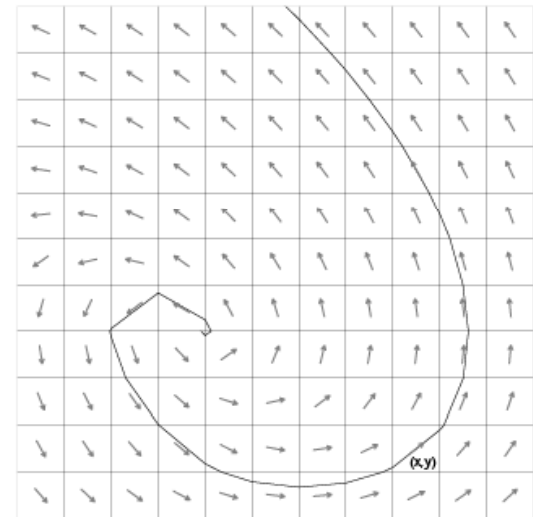
LINE INTEGRAL CONVLTN.

Continuous version

- integral curves of the flow

$$\frac{d}{dt}\sigma(t) = v(\sigma(t))$$

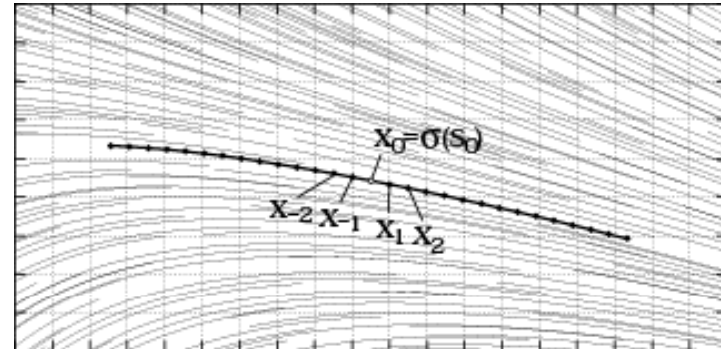
- particles in flow
- follow one traj.



INTEGRATE ALONG FLOW

Integrate *noise*

- output pixel:



$$I(x_0) = \int_{s_0-L}^{s_0+L} k(s - s_0) T(\sigma(s)) ds$$

noise texture

- high correlation along flow
- low correlation orthogonal to flow

MAKING IT FAST

Pixels along path are correlated

- curve tracing for each output pixel far to expensive
- step along curve!

$$I(x_2) = I(x_1) - \int_{s_1-L}^{s_1-L+\Delta s} T(\sigma(s))ds + \int_{s_1+L}^{s_1+L+\Delta s} T(\sigma(s))ds$$

- accumulate results in pixels crossed
- renormalize at end

DETAILS

Streamline integration

- high order integrator recommended
 - may make LARGE steps
 - need to interpolate along path
- aliasing...
 - step size $1/2$ of texture cell
- path length: $1/10$ image size

ANIMATION

Slide integration along streamline

- blend multiple images with different phases
- contrast and brightness (renormalize)

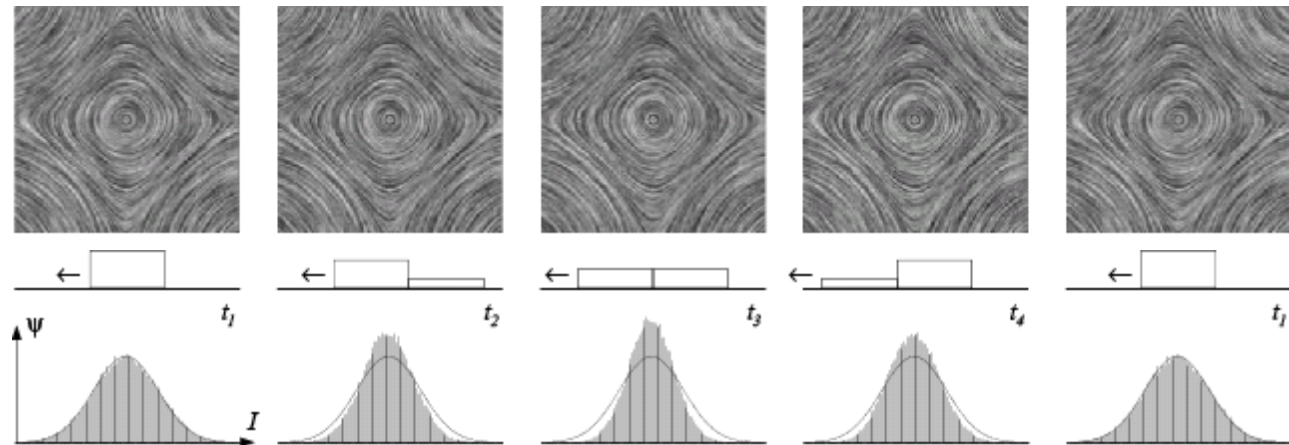


IMAGE BASED FLOW VIZ

Treat images as basic primitive

- LIC as blending of advected images
 - setup mesh with advected texture coordinates
 - render and blend; repeat

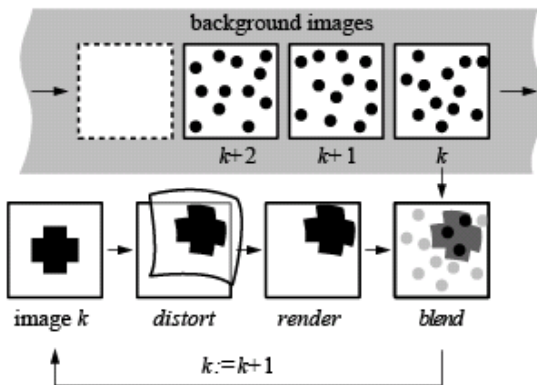
$$F(p_k; k) = (1 - \alpha)F(p_{k-1}; k - 1) + \alpha G(p_k; k)$$

- what image?
 - random noise (gives standard LIC)
-

WHAT IMAGE?

Issues to consider

- aliasing in time and space
 - pink, not white noise
- contrast, boundaries



```

ibvf_sample.c
/*-----*/
/* ibvf_sample.c - Image Based Flow Visualization */
/* Jaska J. van Wijk, 2002 */
/* Technische Universiteit Eindhoven */
/*-----*/
#include <GL/glut.h>
#include <math.h>

#define NPH 64
#define NMBH 100
#define DM ((float) (1.0/(NMBH-1.0)))
#define NPIX 512
#define SCALE 4.0

int iframe = 0;
int NPH = 32;
int alpha = (int)12345;
float aa;
float tmax = NPIX/SCALE/NPH;
float tmax = SCALE/NPIX;
void initGL(void)
{
    glViewport(0, 0, (GLsizei) NPIX, (GLsizei) NPIX);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluLookAt(-1.0, -1.0, 0.0);
    gluLookf(2.0, 2.0, 1.0);
    glEnable(GL_TEXTURE_2D);
    glTexParameters(GL_TEXTURE_2D,
        GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameters(GL_TEXTURE_2D,
        GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexParameters(GL_TEXTURE_2D,
        GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameters(GL_TEXTURE_2D,
        GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexEnvf(GL_TEXTURE_ENV,
        GL_TEXTURE_ENV_MODE, GL_REPLACE);
    glEnable(GL_TEXTURE_2D);
    glDisable(GL_DEPTH_TEST);
    glEnable(GL_SRC_ALPHA);
    glClear(GL_COLOR_BUFFER_BIT);
}
void makeDistort(void)
{
    int i, j;
    for (i = 0; i < NPH; i++) {
        for (j = 0; j < NPH; j++) {
            int i1 = i + tmax * ((float) i * tmax + 0.5);
            int j1 = j + tmax * ((float) j * tmax + 0.5);
            int i2 = i1 + tmax * ((float) i1 * tmax + 0.5);
            int j2 = j1 + tmax * ((float) j1 * tmax + 0.5);
            int i3 = i2 + tmax * ((float) i2 * tmax + 0.5);
            int j3 = j2 + tmax * ((float) j2 * tmax + 0.5);
            float aa = ((float) i * tmax + 0.5) * ((float) j * tmax + 0.5);
            float r = ((float) i * tmax + 0.5) * ((float) j * tmax + 0.5);
            float g = ((float) i * tmax + 0.5) * ((float) j * tmax + 0.5);
            float b = ((float) i * tmax + 0.5) * ((float) j * tmax + 0.5);
            float a = ((float) i * tmax + 0.5) * ((float) j * tmax + 0.5);
            glDrawTexi(GL_TEXTURE_2D, i1, j1, i2, j2);
            glDrawTexi(GL_TEXTURE_2D, i2, j2, i3, j3);
            glDrawTexi(GL_TEXTURE_2D, i3, j3, i4, j4);
        }
    }
}
void display(void)
{
    int i, j;
    float x0, y0, px, py;
    aa = 0.01 * cos(iframe * 2.0 * M_PI / 200.0);
    for (i = 0; i < NMBH; i++) {
        aa = sin(i * M_PI / NMBH);
        x0 = alpha;
        y0 = alpha;
        for (j = 0; j < NMBH; j++) {
            int i1 = i;
            int j1 = j;
            getDistort(x0, y0);
            getDistort(px, py);
            glDrawTexi(i1, j1, i2, j2);
        }
        iframe = iframe + 1;
        glEnable(GL_BLEND);
        glCallList(iframe * NPH + 1);
        glMatrixMode(GL_PROJECTION);
        glTexEnvf(GL_TEXTURE_ENV, GL_REPLACE);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_BLEND);
        glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
        glClear(GL_COLOR_BUFFER_BIT);
    }
}
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(NPIX, NPIX);
    glutCreateWindow(argv[0]);
    glutDisplayFunc(display);
    glutIdleFunc(idle);
    glutMainLoop();
    return 0;
}

```

NICE EXAMPLE

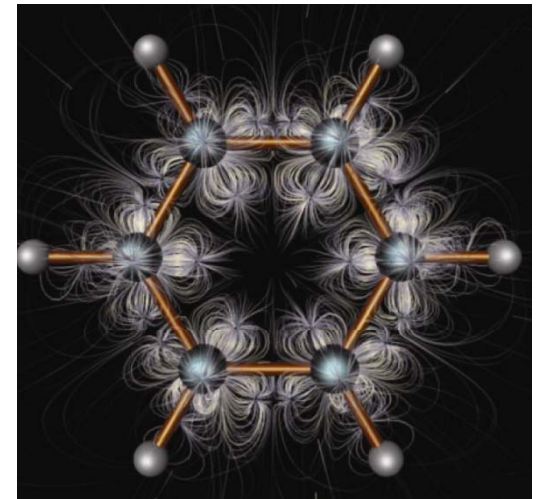
van Wijk's [ibfv](#) program

- testbed for visualization

DIRECT VISUALIZATION

Integral curves as shaded lines

- “hair” like analogy
- how to shade lines?
- usually objects co-D 1



New shading model

- objects of co-D 2
- think cylinders of infinitesimal size

SHADED LINES

What normal?

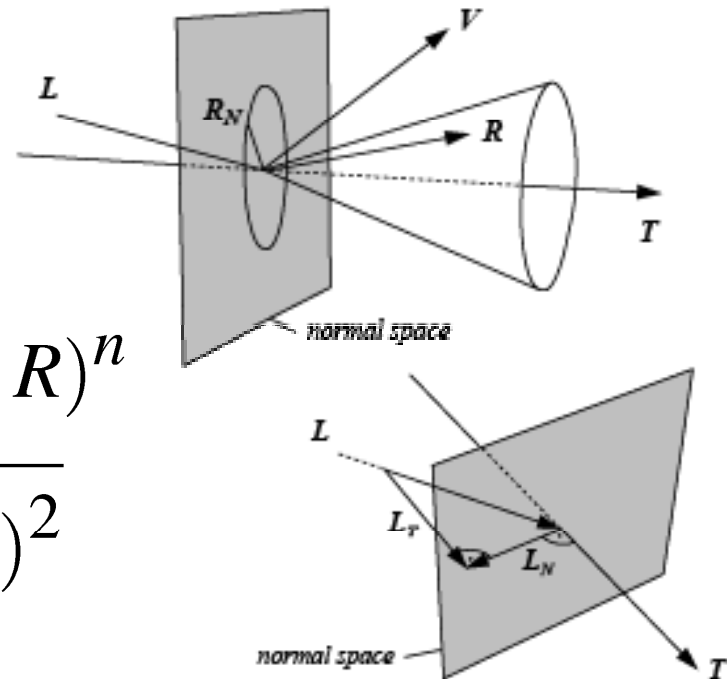
- standard model

$$I = k_a + k_d(L \cdot N) + k_s(V \cdot R)^n$$

$$L \cdot N = |L_N| = \sqrt{1 - (L \cdot T)^2}$$

$$V \cdot R = V \cdot (L_T - L_N)$$

$$= \sqrt{1 - (L \cdot T)^2} \sqrt{1 - (V \cdot T)^2} - (L \cdot T)(V \cdot T)$$

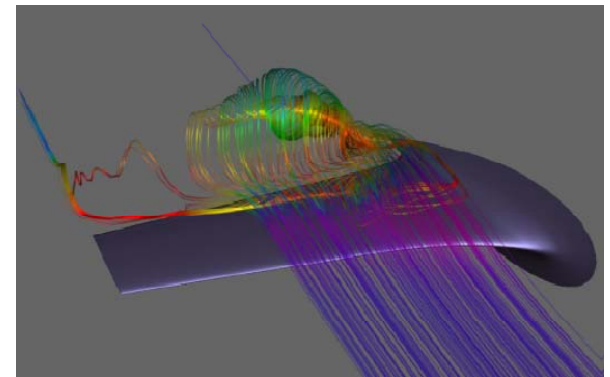
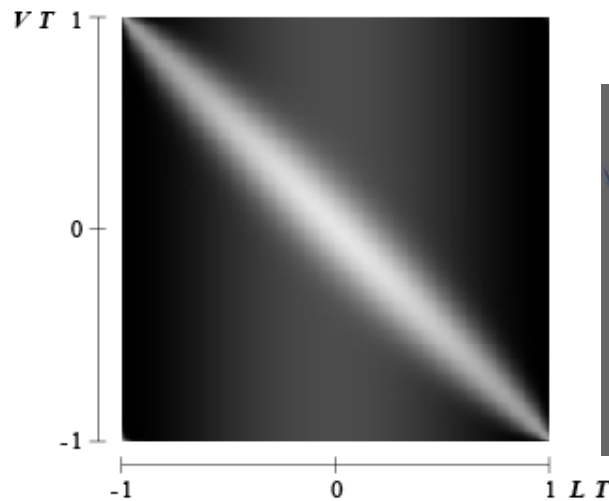


IMPLEMENTATION

Shading is function of L, T, V

- texture lookup
- inner products via texture xform

$$M = \frac{1}{2} \begin{pmatrix} L_1 & V_1 & 0 & 0 \\ L_2 & V_2 & 0 & 0 \\ L_3 & V_3 & 0 & 0 \\ 1 & 1 & 0 & 2 \end{pmatrix}$$



DETAILS

Implementation

- shading too bright: for diffuse

$$I_d = k_d(L \cdot N)^p$$

- add transparency: “wispy” tails
- how to seed stream lines?
 - user driven
 - Monte Carlo; Voronoi
 - divergence... (reseed)

Image Based Flow Visualization

Jarke J. van Wijk^{*}
Technische Universität Lindhoven
Dept. of Mathematics and Computer Science

Abstract

A new method for the visualization of two-dimensional fluid flow is presented. The method is based on the advection and decay of dye. These processes are simulated by defining each frame of a flow animation as a blend between a warped version of the previous image and a number of background images. For the latter a sequence of filtered white noise images is used. Filtered in time and space to remove high frequency components. Because all steps are done using images, the method is named Image Based Flow Visualization (IBFV). With this a wide variety of visualization techniques can be emulated. Flow can be visualized as moving textures with line integral convolution and spot noise. Arrow plots, streamlines, particles, and topological images can be generated by adding extra dye to the image. (Usually, flow, defined on arbitrary meshes, can be handled. IBFV achieves a high performance by using standard features of graphics hardware. Typically, fifty frames per second are generated using standard graphics cards on PCs. Finally, IBFV is easy to understand, analyze, and implement.

In the next section related work is discussed, and in section 3 the method is described and analyzed extensively. The implementation and application are presented in section 4. In section 5 the results are discussed. Finally, conclusions are drawn.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation; I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques; I.6.6 [Simulation and Modeling]: Simulation Output Analysis

Keywords: Flow visualization, texture mapping, line integral convolution

1 Introduction

Fluid flow plays a dominant role in many processes that are important to mankind, such as weather, climate, industrial processes, cooling, heating, etc. Computational Fluid Dynamics (CFD) simulations are carried out to achieve a better understanding and to improve the efficiency and effectiveness of man-made artifacts. Visualization is indispensable to achieve insight in the large datasets produced by these simulations. Many methods for flow visualization have been developed, ranging from arrow plots and streamlines to dense texture methods, such as Line Integral Convolution [Cabral and Leedom 1992]. The latter class of methods produces very clear visualizations of two-dimensional flow, but is computationally expensive. We present a new method for the visualization of two-dimensional vector fields in general and fluid flow fields in particular.

*e-mail: vanwijk@win.tue.nl

particular: The method provides a simple framework to generate a wide variety of visualizations of flow, varying from moving particles, streamlines, moving textures, to topological images. Three other features of the method are: handling of *anisotropic flow*, efficiency and ease of implementation. More specific, all 512 × 512 images presented in this paper are snapshots from animations of unsteady flow fields. The animations were generated at up to 50 frames per second (fps) on a notebook computer, and the accompanying DVD contains a simple but complete implementation in about a hundred lines of source code.

The method is based on a simple concept: Each image is the result of warping the previous image, followed by blending with some background image. This process is accelerated by taking advantage of graphics hardware. The construction of the background images is crucial to obtain a smooth result. All operations are on images, hence we coined the term Image Based Flow Visualization (IBFV) for our method.

In the next section related work is discussed, and in section 3 the method is described and analyzed extensively. The implementation and application are presented in section 4. In section 5 the results are discussed. Finally, conclusions are drawn.

2 Related work

Many methods have been developed to visualize flow. Arrow plots are a standard method, but it is hard to reconstruct the flow from discrete samples. Streamlines and advected particles provide more insight. A disadvantage of these techniques is that the user has to decide where to position the startpoints of streamlines and particles, hence important features of the flow can be overlooked.

The visualization community has spent much effort in the development of more effective techniques. Van Wijk [1991] introduced the use of texture to visualize flow. A *per noise texture* is generated by inserting disoriented spots with a random intensity at random locations in the field, resulting in a dense texture that visualizes data. Cabral and Leedom [1993] introduced *Line Integral Convolution* (LIC), which gives a much better image quality. For grid data streamlines are traced, both upstream and downstream, along this streamlines a random noise texture field is sampled and convolved with a filter. Many extensions to and variations on these original methods, and especially LIC, have been published. The main issue is improvement of the efficiency. Both the standard spot noise and the LIC algorithm use a most or least brute force approach. In spot noise, many spots are required to achieve a dense coverage. In pure LIC, for each pixel a number of points on a streamline (typically 20-50) have to be computed. As a result, the computing time per frame is in the order of tens of seconds. One approach is to develop more efficient algorithms. Stalling and Hegge [1995] achieved a higher performance by using a faster numerical method and a more efficient, streamline-oriented scheme for the integration.

Another approach to achieve a higher efficiency is to exploit hardware. Finally, parallel processing can be used [de Looze and van Lier 1997; Zöckler et al. 1997; Shen and Kao 1998]. Secondly, graphics hardware can be used. De Looze et al. [1995] use texture

Fast and Resolution Independent Line Integral Convolution

Detlev Stalling Hans-Christian Hegge

Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB)¹

Abstract

Line Integral Convolution (LIC) is a powerful technique for generating striking images and animations from vector data. Introduced in 1993, the method has rapidly found many application areas, ranging from computer arts to scientific visualization. Based upon locally filtering an input texture along a curved stream line segment in a vector field, it is able to depict directional information at high spatial resolutions.

We present a new method for computing LIC images. It employs simple box filter kernels only and minimizes the total number of stream lines to be computed. Thereby it reduces computational costs by an order of magnitude compared to the original algorithm. Our method utilizes fast, error-controlled numerical integrators. Decoupling the characteristic lengths in vector field grid, input texture and output image, it allows computation of filtered images at arbitrary resolution. This feature is of significance in computer animation as well as in scientific visualization, where it can be used to explore vector data by smoothly enlarging structure of details.

We also present methods for improved texture animation, again employing box filter kernels only. To obtain an optimal motion effect, spatial decay of correlation between intensities of distant pixels in the output image has to be controlled. This is achieved by blending different phase-shifted box filter animations and by adaptively rescaling the contrast of the output frames.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation; I.3.6 [Computer Graphics]: Methodology and Techniques; I.4.3 [Image Processing]: Enhancement

Additional Keywords: vector field visualization, texture synthesis, periodic motion filtering

1 Introduction

Generation of textured images from various kinds of vector fields has become an important issue in scientific visualization as well as in animation and special effects. In 1993 Cabral and Leedom presented a powerful technique for imaging vector data called *line integral convolution* [1]. Their algorithm has been used as a general tool for visualizing vector fields. Additionally it has broad applications for image enhancement. A major drawback of the original algorithm, however, is its high computational expense and its restriction to a fixed spatial resolution.

In this paper we present an improved algorithm for line integral convolution, in which computation of streamlines is algorithmically separated from that of convolution. This allows us to exploit economies and to provide wider functionality in each of the computational steps. The new algorithm

is *almost an order of magnitude faster* than original line integral convolution, making interactive data exploration possible.

- is more accurate by employing an adaptive, error-controlled streamline integration technique;
- is resolution independent, enabling the user to investigate image details by smooth detail enlargement (zooming);
- improves texture convolution using shifted box filter kernels together with a simple blending technique.

In recent years a number of methods for artificially generating textures have been suggested. These methods cover a variety of applications. In the field of scientific visualization texture-based methods are of special interest because they allow the display of vector fields in an untruncated spatial resolution. Traditionally, vector data has been represented by small arrows or other symbols indicating vector magnitude and direction. This approach is restricted to a rather coarse spatial resolution. More sophisticated methods include the display of stream lines [9], stream surfaces [10], flow volumes [14] as well as various particle tracing techniques [19, 9, 11]. These methods are well suited for revealing characteristic features of vector fields. However, they strongly depend on the proper choice of seed points. Experience shows that interesting details of the field may easily be missed.

Texture-based methods are not affected by such problems. They depict all parts of the vector field and thus are not susceptible to missing characteristic data features. In addition they achieve a much higher spatial resolution, which in some cases can be viewed as the maximum possible resolution since the minimum possible feature size of a textured image is a single pixel. In an early method introduced by van Wijk [18] a random texture is convolved along a straight line segment oriented parallel to the local vector direction. Line integral convolution (LIC) [1] modifies this method, so that convolution takes place along curved stream line segments. In this way field structure can be represented much more clearly. Forewell [5] describes another extension that allows for a map flat LIC image onto curvilinear surfaces in three dimensions.

Vector fields are not only of relevance in science and engineering. Many objects of our natural environment exhibit characteristic directional features which are naturally represented by vector data. Consequently algorithms for turning such data into pictorial information are of great importance for synthetic image generation, image post-processing, and computer arts [6, 16]. The variety of directional filters offered by commercial image processing software is just one evidence for this.

The remainder of the paper is organized as follows. Section 2 provides mathematical background and basic notation. The basic idea of the new algorithm are outlined in section 3. In the following three sections we present algorithms for fast and accurate streamline integration, discuss some optimization issues, and sketch strategies for fast texture map sampling. We then discuss periodic motion filtering and smooth detail enlargement. Finally, we present some results and give an outlook concerning various aspects of LIC methods.

Interactive Visualization Of 3D-Vector Fields

Using Illuminated Stream Lines

Malte Zöckler, Detlev Stalling, Hans-Christian Hegge

Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB)¹

Abstract

A new technique for interactive vector field visualization using large numbers of properly illuminated stream lines is presented. Taking into account ambient, diffuse, and specular reflection terms as well as transparency, we employ a realistic shading model which significantly increases quality and realism of the resulting images. While many graphics workstations offer hardware support for illuminating surface primitives, usually no means for an accurate shading of line primitives are provided. However, we show that proper illumination of lines can be implemented by exploiting the texture mapping capabilities of modern graphics hardware. In this way high rendering performance with interactive frame rates can be achieved. We apply the technique to render large numbers of integral curves in a vector field. The impression of the resulting images can be further improved by making the curves partially transparent. We also describe methods for controlling the distribution of stream lines in a scene. These methods enable us to use illuminated stream lines within an interactive visualization environment.

1 Introduction

The visual representation of vector fields is subject of ongoing research in scientific visualization. A number of sophisticated methods has been proposed to tackle this problem, ranging from particle tracing [7, 15, 11] over icon based methods [8, 13] to texture based approaches [3, 2, 4, 14, 9]. A straightforward, popular and still very powerful method is the concept of depicting stream lines. However, when using stream lines for visualization the user is confronted with a number of problems. First, on a common graphics workstation stream lines either have to be displayed using flat-shaded line segments, impairing the spatial impression of the image, or they have to be represented by polygonal tubes, strongly limiting the number of stream lines that can be displayed in a scene. Second, it is usually not quite obvious how to distribute stream lines in space in order to get expressive pictures without missing important details of the field.

In this paper we present ideas that can help to overcome both problems. To achieve a fast and accurate illumination

of line segments we exploit the texture mapping capabilities of modern graphics hardware. We apply this new shading technique to render large numbers of stream lines distributed throughout a vector field. Taking into account light reflection on stream lines is of great significance for scientific visualization because it very much increases the spatial impression of the resulting images. Image quality can be further improved by making parts of a stream line semi-transparent. This allows us to get a better understanding of the inner structure of a field. It also makes it possible to distinguish between forward and backward direction. To facilitate the placement of a large number of stream lines we employ statistical methods. Given some scalar quantity that loosely describes the degree of interest in the vector field at some location, stream lines are placed automatically such that the relative degree of interest is matched qualitatively.

It is a well-known fact that quality and realism of computer generated images depend to a high degree on the accurate modeling of light interacting with the objects in a scene. Shading effects are perhaps the most important cue for spatial perception. Consequently much research has been performed to develop realistic illumination and reflection models in computer graphics. A widely used compromise between computational complexity and resulting realism is Phong's reflection model [12] which assumes point light sources and approximates the most important reflection terms by simple expressions. Traditionally the model is applied to surface elements. Today many graphics workstations offer hardware support for this kind of illumination. However, the model can also be generalized to line primitives, and in this paper we will make direct use of such a generalization.

In scientific visualization the goal is not to render natural scenes in a photo-realistic way, but to generate images which provide maximal insight into numerical or experimental data. Nevertheless, shading effects are at least as important for the spatial interpretation of artificial images as in traditional computer graphics. Shading provides the observer with a minimum of realism in a world of cutting planes, isosurfaces, and symbols. Unfortunately there are a number of visualization techniques which aren't based on surface primitives, and which therefore can't make use of the hardware shading capabilities of current graphics workstations. As an example consider the various volume rendering techniques. While interactive frame rates can be achieved

¹Takustr. 7, D-14195 Berlin, Germany
E-mail: {zoeckler, stalling, hegge}@zib1.zib-berlin.de