EUROGRAPHICS - IEEE VGTC Symposium on Visualization (2005), pp. 1–8
K. W. Brodlie, D. J. Duke, K. I. Joy (Editors)

Dense Geometric Flow Visualization

Sung W. Park,[†] Brian Budge,[†] Lars Linsen,[†‡] Bernd Hamann,[†] and Kenneth I. Joy[†]

[†] Institute for Data Analysis and Visualization (IDAV)
Department of Computer Science
University of California, Davis
Davis, CA 95616, U.S.A.

[‡] Department of Mathematics and Computer Science
Ernst-Moritz-Arndt-Universität Greifswald
Greifswald, Germany

Abstract

We present a flow visualization technique based on rendering geometry in a dense, uniform distribution. Flow is integrated using particle advection. By adopting ideas from texture-based techniques and taking advantage of parallelism and programmability of contemporary graphics hardware, we generate streamlines and pathlines addressing both steady and unsteady flow. Pipelining is used to manage seeding, advection, and expiration of streamlines/pathlines with constant lifetime. We achieve high numerical accuracy by enforcing short particle lifetimes and employing a fourth-order integration method. The occlusion problem inherent to dense volumetric representations is addressed by applying multi-dimensional transfer functions (MDTFs), restricting particle attenuation to regions of certain physical behavior, or features. Geometry is rendered in graphics hardware using techniques such as depth sorting, illumination, haloing, flow orientation, and depth-based color attenuation to enhance visual perception. We achieve dense geometric three-dimensional flow visualization with interactive frame rates.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-dimensional Graphics and Realism

1. Introduction

Flow visualization has a long tradition in scientific data visualization. Approaches for 3D vector fields however have only recently experienced a boost due to the introduction of programmable graphics hardware with large texture memory. Consequently, volumetric flow visualization has entered many disciplines of science and engineering including mechanics, physics, chemistry, meteorology, geology, and medicine. Many applications are concerned with steady or unsteady flow over 2D or 3D domains.

Flow visualization approaches can be categorized into direct, geometric, texture-based, and feature-based approaches [LHD*04]. Early attempts such as arrow and hedgehog plots or color coding fall into the category of direct flow visualization [PLV*02]. They provide an intuitive image of local flow properties.

For a better understanding of global flow dynamics with respect to "long-term" behavior, integration-based approaches have been introduced. These integrate flow data leading to trajectories of no-mass particles moving over time. Geometric flow visualization approaches render the integrated flow using geometric objects such as lines, tubes, ribbons, or surfaces [PLV*02].

In texture-based flow visualization, a texture is used for a dense representation of a flow field. The texture is filtered according to the local flow vectors leading to a notion of overall flow direction [LHD*04]. The most prominent approaches are line integral convolution (LIC) [CL93] and texture advection [MBC93]. The LIC primitive is a noise texture, which is convolved in the direction of the flow using

[†] {sunpark|bcbudge}@ucdavis.edu, {hamann|joy}@cs.ucdavis.edu
[‡] linsen@uni-greifswald.de

filter kernels. In texture advection, the primitive is a "moving" texel, which is advected in the direction of the flow.

Feature-based flow visualization is concerned with the extraction of specific patterns of interest, or features. Various features such as vortices, shock waves, or separatrices have been considered. Each of them has specific physical properties, which can be used to extract the desired feature. Once a feature has been extracted, standard visualization techniques are applied for rendering [PVH*03].

All these approaches work well for 2D vector fields. While geometric flow visualization approaches generalize to volume data, direct and texture-based approaches have occlusion issues in 3D, an inherent problem for dense representations. Dense representations are desirable, as they provide information concerning overall flow behavior and serve as a context for chosen visualization methods [Wei04].

We introduce a dense geometric flow visualization technique based on streamlines for steady flow and on pathlines for unsteady flow. The approaches are described in Sections 3 and 4, respectively. They can be applied to 2D and 3D data sets. The lines are computed using a trajectory-based particle-advection method. We exploit the speed and parallelism of contemporary graphics hardware. In contrast to texture-based approaches, we use the graphics hardware as a general-purpose computing device. We achieve high numerical accuracy by enforcing short particle lifetimes and employing a fourth-order integration method.

Since we propose a dense flow representation, we must address occlusion problems, which we do by applying multi-dimensional transfer functions (MDTFs) based on physical flow properties [PBL*04]. The application of MDTFs to dense geometric flow visualization is described in Section 5.

We display our lines using various rendering techniques, described in Section 6. Essential to a correct visual perception is depth sorting of the geometric objects. Illumination is a key technique to enhance spatial perception. Adding halos clarifies the spatial relationships between overlapping lines. To orient lines, we highlight their tips, which increases the perception of motion. Depth perception is enhanced using depth-based color attenuation like desaturation and darkening. All rendering features have been implemented in graphics hardware to achieve interactivity.

2. Related Work

Flow visualization based on streamlines provides an intuitive geometric approach for steady flow data sets, as the extracted line geometry represent trajectories of mass-less particles moving under the influence of the flow field. Streamlines have a long tradition in visualization and are used in various flow visualization systems [BMP*90]. The equivalent of streamlines for unsteady flow fields are pathlines, where the flow field induces the movement of the particle changes over time [SML04]. When compared to animated streamlines [JL00], pathlines better convey change over time.

Today texture-based approaches are more widely used than streamlines for flow field visualization, as they provide a dense representation and can be used to display the entire discrete domain. Texture-based approaches work well in 2D. For 3D applications the dense representation of texture-based approaches leads to occlusion problems, which is a major advantage of sparse representations.

In [ZSH96], an interactive 3D flow visualization using streamlines was presented. Illumination is used to improve the streamlines' perception. Texture-mapping capabilities of modern graphics hardware is exploited. Some approaches generalize streamlines to streamtubes or even stream ribbons [USM96]. In [FG98], animated opacity-mapped streamlines called dashtubes are used. These approaches only treat the visualization of steady flow.

A major challenge when using sparse representations like streamlines is seeding of starting points for flow integration. In [TB96], techniques for automated placing of seed points were developed to achieve a nearly uniform, dense distribution of streamlines for 2D flow fields. For 3D flow fields, seeding strategies typically involve analysis of the underlying flow field to visualize certain features using sparse distributions. In [Lar02], user interaction is involved for seeding and for interactive control over the evolution of streamlines for 3D flow fields. Our approach uses dense streamlines, which alleviates the problem of seeding at the expense of introducing occlusion problems. We tackle occlusion by applying multi-dimensional transfer functions [PBL*04].

A dense distribution of streamlines involves additional computations. To achieve interactive frame rates, we make extensive use of the programmability of contemporary graphics hardware. We adopt concepts from texture-based flow visualization. The rendering primitive for texture-based approaches is a texel. In texture-based approaches using line integral convolution (LIC) [CL93], the texture is filtered along the path of a streamline. An oriented version of the algorithm was discussed in [WGP97] and an extension to unsteady flow in [SK97]. In texture-based approaches using texture advection [JL97], texels are moved while the motion is directed by the flow field. In [vW02], a fast algorithm was presented based on advection and decay of texels in image space. Our approach uses particles as primitives instead of texels. The particles are advected and connected over subsequent frames to generate streamlines or pathlines.

3. Dense Geometric Flow Visualization

The concept of dense geometric flow visualization is adopted from texture-based flow visualization approaches. A rendering primitive is advected over time under the influence of the underlying flow field. In each time frame, a flow integration

step is applied to determine the position of the primitive in the subsequent frame. The motion of the primitive indicates local flow behavior. The motion of a large number of primitives distributed densely across the domain visualizes the behavior of the entire flow field.

While for texture-based approaches a rendering primitive is represented by color information, for example in form of noise, our approach uses geometry. Instead of advected texels, we advect particles and connect them over subsequent frames to form streamlines or pathlines. We first describe the details of our method for steady flow, before we generalize dense geometric flow visualization to unsteady flow in the subsequent section.

Let $\mathbf{f} : D \rightarrow R$ be the function of a steady flow field with a 2D or 3D domain D and a vector-valued range R. To visualize steady flow we render a large number n of streamlines distributed densely and nearly uniformly over the domain D. At each point in time (after a start-up phase), exactly n streamlines exist. Each streamline has a constant lifetime of k cycles. The lifetime of a streamline is divided into three phases: (i) seeding, (ii) advection, and (iii) expiration. The seeding and the expiration phases last only one cycle each, such that for the major part of its lifetime, the other $k-2$ cycles, a streamline is in its advection phase.

The streamlines are grouped into k groups of size $\frac{n}{k}$, where n is chosen to be a multiple of k. All streamlines of each group start their life simultaneously. The computation of all streamlines can be processed in a pipeline with k cycles. Figure 1 shows the processing pipeline.

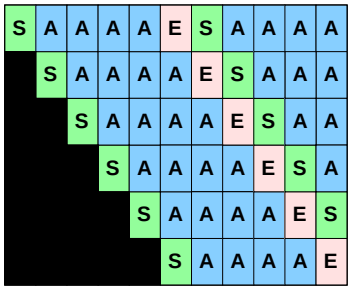


Figure 1: *Processing pipeline for streamline rendering. Lifecycle of streamlines is divided into phases seeding (S), advection (A), and expiration (E).*

3.1. Seeding

Our method makes use of quasi-random numbers for a uniform seeding across the domain. Uniform seeding is necessary in order for all regions in the flow to receive attention. One reason to use quasi-random number generation is that their computation is reasonably simple to perform in graphics hardware. Another reason is that they tend to provide a more well-distributed set of samples than pseudo-random generators [Nie92].

One of the best quasi-random methods is the *Hammersley sequence*. However, the Hammersley sequence is limited in that the user is restrained to a fixed number of points, and the order of generation matters. Instead, we use the closely related *Halton sequence* for seed generation, since the Halton sequence has a coverage approaching the Hammersley sequence, but has the nice property that new seeds can be adaptively generated [Kel96, Nie92].

In general, there are two methods for generating the Halton sequence. The first method is an iterative solution that has $O(\log i)$ complexity for each element calculation, where H_i is the ith element of the Halton sequence. The second technique involves generating H_i from H_{i-1}. This is ideal since the calculation is extremely fast, $O(1)$, and in most cases exceeds the calculation speed of pseudo-random number generation. Unfortunately, there is no easy way to keep state between successive calculations when working on graphics hardware. Thus, the second method is currently not possible.

3.2. Advection

Streamlines are generated by integrating flow over time. The approach we are following is to advect particles over time in the direction induced by the flow. The streamlines are determined as the paths of the particles advected over time.

Let $\mathbf{p}(t) \in D$ be the position of a particle at time t. The position $\mathbf{p}(0)$ at time $t = 0$ denotes the seed location for that particle. The path of the particle is defined as

$$\mathbf{p}(t) = \mathbf{p}(0) + \int_0^t \mathbf{f}\Big(\mathbf{p}(x)\Big)\,dx\,.$$

The integral equation must be solved using numerical integration methods. Most commonly a simple first-order Euler method or a second-order Runge-Kutta method is used.

The Euler method computes the particle position $\mathbf{p}(t+\Delta t)$ at time $t+\Delta t$ from the position $\mathbf{p}(t)$ at time t as

$$\mathbf{p}(t+\Delta t) = \mathbf{p}(t) + \Delta t \cdot \mathbf{f}\Big(\mathbf{p}(t)\Big)\,,$$

where Δt is a small time step. The second-order Runge-Kutta method computes a more precise approximation by utilizing the Euler approximation, denoted as $\mathbf{p}_E(t+\Delta t)$, as a look-ahead estimate and defining

$$\mathbf{p}(t+\Delta t) = \mathbf{p}(t) + \Delta t \cdot \frac{\mathbf{f}\Big(\mathbf{p}(t)\Big) + \mathbf{f}\Big(\mathbf{p}_E(t+\Delta t)\Big)}{2}\,.$$

Since keeping integration error low over time is one of our major goals, we use a fourth-order Runge-Kutta method based on the computations

$$\mathbf{x}_1 \;=\; \mathbf{f}\Big(\mathbf{p}(t)\Big)\,,$$

$$\begin{aligned}
\mathbf{x}_2 &= \mathbf{f}\Big(\mathbf{p}(t)+\frac{\Delta t}{2}\cdot\mathbf{x}_1\Big)\,,\\
\mathbf{x}_3 &= \mathbf{f}\Big(\mathbf{p}(t)+\frac{\Delta t}{2}\cdot\mathbf{x}_2\Big)\,,\\
\mathbf{x}_4 &= \mathbf{f}\Big(\mathbf{p}(t)+\Delta t\cdot\mathbf{x}_3\Big)\,,\ \text{and}\\
\mathbf{p}(t+\Delta t) &= \frac{\mathbf{x}_1+2(\mathbf{x}_2+\mathbf{x}_3)+\mathbf{x}_4}{6}\,.
\end{aligned}$$

3.3. Expiration

The lifetime of a streamline expires k cycles after its generation by the seeding process. The particle dies at this point, i. e., it is not advected any further. In the next cycle it will be replaced by a new particle at a new seed location. To achieve high integration accuracy during advection, we favor short lifetimes.

3.4. Rendering

A streamline is represented by a polyline define by connecting the successive discrete locations that the corresponding particle traverses during advection. Streamlines are rendered using line primitives. Line primitives offer the greatest simplicity and speed, yet they can provide good visual quality when coupled with appropriate rendering techniques (see Section 6).

3.5. Implementation

Since we compute and render many streamlines using a higher-order integration method, we exploit the parallelism of contemporary graphics hardware for efficiency. To implement the entire processing pipeline in graphics hardware, we also have to exploit the programmability of graphics hardware.

For particle seeding, we have implemented a Halton sequence generator in the vertex shader to not reduce the efficiency of the pipelining process using the fragment shader. As input, we stream a set of vertices, where each vertex holds a 2D index corresponding to a texel in a 2D texture. Each vertex streamed into the vertex shader generates two or three Halton numbers for 2D or 3D flow, respectively, and passes the fragment shader value. At the end of this stage, the rendered texture contains the location of a seed particle in each texel.

The advection component of the algorithm takes as input a set of particle positions, and generates a corresponding set of advected particles. We have implemented the advection by passing two textures, the vector field and the particle texture. These textures are used by a fragment shader to generate advected particles, which are written to a new render texture.

A single advection is performed by first sampling each texel of the texture that holds the vector field at time t in the fragment shader. For each particle's position the vector field texture is looked up. The vector field texture is sampled three more times at different locations based on the particle's position according to the fourth-order Runge-Kutta integration method. Finally, the advected particles are written to a new texture, which is used for both rendering and further advection in the following time steps.

Rendering is performed by connecting particle positions over time using line primitives. We use a buffer that holds positions of each seed in its k cycles of life. The rendering component uses as input the result of the last particle advection. The advected particle positions are first extracted from the texture. Then, the particles are added into the buffer in its proper cycle. Finally, the geometry is rendered for each particle up to the last advected position.

4. Unsteady Flow

When function $\mathbf{f}$ describes an unsteady flow field with domain D being the Cartesian product of the two or three spatial dimensions and a temporal dimension, we replace streamlines by pathlines. Pathlines are defined as paths of particles in a flow field that changes over time, i. e., during pathline generation (or particle advection).

We visualize unsteady flow by rendering a dense, uniform distribution of pathlines with constant lifetime. The lifetime of a pathline consists again of the three phases seeding, advection, and expiration. The seeding phase, the expiration phase, and the line rendering are performed as done for streamlines.

During advection we have to account for the changing flow field. The flow integration for unsteady fields is described by

$$\mathbf{p}(t) = \mathbf{p}(0) + \int_0^t \mathbf{f}\Big(\mathbf{p}(x),x\Big)dx\,,$$

where $\mathbf{p}(t)$ describes the position of a particle at time t and function $\mathbf{f}$ depends on both position of the particle and time. We use fourth-order Runge-Kutta integration, which is generalized for time-dependent fields to the equations

$$\begin{aligned}
\mathbf{x}_1 &= \mathbf{f}\Big(\mathbf{p}(t),t\Big)\,,\\
\mathbf{x}_2 &= \mathbf{f}\Big(\mathbf{p}(t)+\frac{\Delta t}{2}\cdot\mathbf{x}_1,t+\frac{\Delta t}{2}\Big)\,,\\
\mathbf{x}_3 &= \mathbf{f}\Big(\mathbf{p}(t)+\frac{\Delta t}{2}\cdot\mathbf{x}_2,t+\frac{\Delta t}{2}\Big)\,,\\
\mathbf{x}_4 &= \mathbf{f}\Big(\mathbf{p}(t)+\Delta t\cdot\mathbf{x}_3,t+\Delta t\Big)\,,\ \text{and}\\
\mathbf{p}(t+\Delta t) &= \frac{\mathbf{x}_1+2(\mathbf{x}_2+\mathbf{x}_3)+\mathbf{x}_4}{6}\,.
\end{aligned}$$

For implementation purposes, we use a texture for the vector field at time $t+\Delta t$ in addition to the textures for the vector field at time t and the particle texture. The numerical integration is performed using six texture reads.

5. MDTFs

To capture the entire flow field we use a dense, uniform seeding strategy for streamline or pathline generation. For 3D fields, dense structures cause occlusion problems, which we tackle by applying MDTFs [PBL*04]. MDTFs can be used to select regions of a certain physical behavior, called *features*. We extract features with respect to five physical quantities of flow fields, the quantities being velocity magnitude, gradient magnitude, divergence, curl magnitude, and helicity. We have derived equations for each of the five magnitudes in [PBL*04]. We only visualize streamlines or pathlines, respectively, within the extracted features.

In order to define an MDTF, we map each of the $N = 5$ scalar magnitudes to color channels R, G, and B and opacity values α by using 1D transfer functions $\mathbf{T}_i : \mathbf{R} \rightarrow \mathbf{R}^4$, $i = 1,\ldots,N$. We combine the 1D transfer functions $\mathbf{T}_i$ to an N-dimensional transfer function $\mathbf{T} : \mathbf{R}^N \rightarrow \mathbf{R}^4$, according to the equations

$$\mathbf{T}_{RGB}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} \mathbf{T}_{i,RGB}(x_i) \cdot \mathbf{T}_{i,\alpha}(x_i)$$

and

$$\mathbf{T}_{\alpha}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} \mathbf{T}_{i,\alpha}(x_i) \ ,$$

where $\mathbf{x} = (x_1,\ldots,x_N) \in \mathbf{R}^N$. The color values $\mathbf{T}_{i,RGB}$ assigned by the individual 1D transfer functions are averaged in a weighted fashion to define the color $\mathbf{T}_{RGB}$ assigned by the MDTF, where the weights are given by the individual opacities $\mathbf{T}_{i,\alpha}$. The opacity values $\mathbf{T}_{i,\alpha}$ assigned by the individual 1D transfer functions are averaged as well, to define the opacity $\mathbf{T}_{\alpha}$ assigned by the MDTF. The function $\mathbf{T}$ allows one to visualize each component distinctly, but also makes possible blending of the values to extract a feature, whose behavior is defined by a combination of the five physical quantities/properties.

6. Rendering techniques

We render line primitives whose simplicity is advantageous for rendering many streamlines/pathlines at high frame rates. But, this qpproach requires us to apply certain rendering techniques to enhance visual perception in 3D settings.

6.1. Depth Sorting

Most importantly, we need to sort our line primitives according to their depth in viewing direction, which is used to render lines in the correct order. When reading data from the GPU, performing a depth sort on the central processing unit (CPU), and passing the data back to the GPU, data exchange becomes the bottle neck of our entire rendering system. Thus, we perform depth sorting on the GPU.

In a bitonic sort [Bat68], a sequence of numbers (representing depth in our application) is recursively divided into two subsequences of equal length, where the first subsequence is being sorted recursively in ascending and the second subsequence in descending order. The two subsequences are merged by comparing (and possibly switching) the ith value of the first subsequence with the ith value of the second subsequence for all positions i and performing the merging step recursively for both subsequences.

The beauty of the bitonic sort algorithm is that its mechanism is independent of the values to be sorted and thus can be implemented in hardware. Moreover, it is amenable to parallel architectures like GPUs. In [PDC*03], an implementation of a bitonic sort algorithm on a GPU was proposed. We use an improved version of this implementation as described in [KSW04]. The depth and indices of all particles are stored in a 2D texture. The sorting algorithm first sorts the depth values of all the rows of the texture simultaneously and then merges the rows.

6.2. Illumination

Flat shading algorithms of line primitives impair spatial perception of the rendered geometry. In [ZSH96], a method for rendering illuminated lines at interactive frame rates is described by utilizing texture mapping hardware. The light intensity $I(\mathbf{x})$ at each point $\mathbf{x}$ on a line segment is computed by applying a slightly modified Phong model. The model is given by

$$I(\mathbf{x}) = I_{ambient}(\mathbf{x}) + I_{diffuse}(\mathbf{x}) + I_{specular}(\mathbf{x}) \ ,$$

where the ambient portion is kept constant, $I_{ambient} = k_{ambient}$, the diffuse portion is defined as $I_{diffuse} = k_{diffuse}(\mathbf{l}\cdot\mathbf{n})$ with light direction $\mathbf{l}$ and normal vector $\mathbf{n}$, and the specular component is defined as $I_{specular} = k_{specular}(\mathbf{v}\cdot\mathbf{r})^m$ with viewing direction $\mathbf{v}$ and reflection vector $\mathbf{r}$. The illumination coefficients $k_{ambient}$, $k_{diffuse}$, and $k_{specular}$ and the exponent m are application-specific.

Using geometric properties the model is adjusted to

$$I_{diffuse} = k_{diffuse}\left(\sqrt{1-(\mathbf{l}\cdot\mathbf{t})^2}\right)^p$$

and

$$I_{specular} = k_{specular}\left(\sqrt{1-(\mathbf{l}\cdot\mathbf{t})^2}\sqrt{1-(\mathbf{v}\cdot\mathbf{t})^2}-(\mathbf{l}\cdot\mathbf{t})(\mathbf{v}\cdot\mathbf{t})\right)^m .$$

The surface-specific normal vector $\mathbf{n}$ and reflection vector $\mathbf{r}$ are replaced by expressions using tangent vector $\mathbf{t}$, such that the model can be applied to line segments. To compensate for excessive brightness, we raise the diffuse portion to the power of an exponent p. We implement the illumination algorithm in graphics hardware using a 4×4 texture transformation matrix for the computation of diffuse and specular reflection as described in [SZH97].

6.3. Haloing

When overlapping lines occur, their spatial relationship can be clarified using haloing [ARS79]. We render a dark halo with a width of six pixels around each line that is rendered with a width of 1.5 pixels. The halo of a line obstructs the view of any line behind it.

6.4. Flow Orientation

For any flow visualization it is important to indicate the orientation of a flow direction. In our system flow direction is visualized by motion. In addition, we intuitively emphasize orientation by highlighting the tip of each streamline or pathline, respectively. Even in still images the flow orientation can be perceived.

6.5. Depth-based Attenuation

Although depth sorting, illumination, and haloing effectively support visual perception, it may be desirable to further enhance depth perception by the use of color attenuation. We have implemented the two attenuation methods of desaturation and darkening [SM02]. Desaturation reduces the saturation of a color while maintaining hue and brightness, which leads to the visual impression of light being absorbed or scattered by particles (like smog or fog) in the atmosphere. Darkening reduces the brightness of a color while maintaining hue and saturation. The two attenuation methods can also be combined.

7. Results and Discussion

We tested our approach for steady and unsteady data sets. For steady flow data, we examined a tornado data set [CM93] of size 128^3. For the unsteady case, we examined a CFD simulation of five jets consisting of 2000 timesteps of 128^3 vector field data.[†]

Figure 3 shows a dense geometric flow visualization for the tornado data set using streamlines. In Figure 3(a), we have rendered polylines using flow orientation, depth-based attenuation, and haloing. The streamlines are not lit nor are they sorted by depth. Figure 3(b) shows the importance of using illumination and depth sorting. The visual perception has increased significantly. In Figure 3(c), we have used an MDTF that extracts regions of high curl (or vorticity) magnitude. The streamlines are only drawn within the vortex area.

In our current implementation, the six pixel-wide haloing lines cannot be depth-sorted in conjunction with our 1.5 pixel-wide streamlines/pathlines, as they represent different types of geometric objects. Thus, haloing only looks correct when we extract features with low complexity, i. e., when the artifacts caused by omitting depth-sorting are minimal.

[†] Data set courtesy of Kwan-Liu Ma, IDAV, University of California, Davis

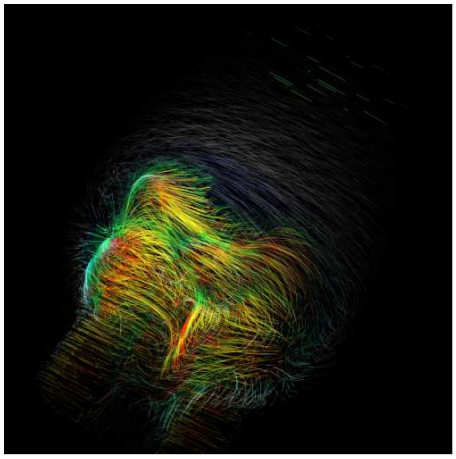


Figure 2: *Dense geometric flow visualization applied to time step 1500 of jets data set. MDTFs extract regions of high curl magnitude and high velocity magnitude.*

# cycles	# seeds	# active streamlines	frame rate
16	64	1024	66 fps
16	256	4096	66 fps
32	256	8192	22 fps
32	1024	32768	6.69 fps
64	256	16384	7.43 fps
128	64	8192	7,43 fps

Table 1: *Frame rates for varying number of seeds, cycles, and active streamlines.*

Figures 2 and 4 show dense geometric flow visualization for the jets data set. In Figure 2, we have used depth-sorting and illumination in conjuction with depth-based attenuation to render streamlines at time step 1500. MDTFs are used to extract regions with high curl magnitude and high velocity magnitude. Figure 4 shows three steps of an animation of unsteady flow using pathlines. The pictures have been captured at time steps 1120, 1400 and 1800. The images have been rendered using haloing, illumination, flow orientation, and depth-based attenuation.

The computation times for our rendering depend on the rendering techniques used and on the number of streamlines or pathlines, respectively. Table 1 lists frame rates for steady data when rendering illuminated streamlines. When applying depth sorting frame rates decrease by a factor five to ten. For unsteady data when rendering pathlines, we have achieved frame rates up to ten frames per second. Contrary to texture-based flow visualization approaches, the frame rates achieved when using dense geometric flow visualization are (almost) independent of the size of the data set. Thus, for

larger data sets that still fit in the texture memory, frame rates are close to the ones listed above. For unsteady data, data transfer, i. e. copying the individual time steps to the GPU, becomes a bottle-neck.

Another nice property of our dense geometric flow visualization approach when compared to texture-based methods is that no spatial-temporal coherence problems occur. When coupling dye advection with texture-based flow visualization techniques, blurring based on numerical diffusion artifacts can be observed, which has only been fixed recently in [Wei04]. As our advection primitives are pathlines, i. e., geometric objects, our advection step is not subject to diffusion.

8. Conclusions and Future Work

We have presented a flow visualization approach based on rendering geometry in a dense, uniform distribution. We have integrated flow using particle advection to generate streamlines (for steady flow) and pathlines (for unsteady flow). Ideas from texture-based flow visualization have been adopted. Pipelining is used to manage seeding, advection, and death of streamlines/pathlines with constant lifetime. Our method uses a fourth-order Runge-Kutta method, which has been efficiently implemented in hardware by exploiting parallelism and programmability of graphics hardware. We have addressed the occlusion problem inherent to dense volumetric representations by applying MDTFs restricting particle attenuation to regions of certain physical behavior, features. Geometry is rendered using several techniques to enhance visual perception. We have applied our approach to steady and unsteady 3D flow fields achieving interactive frame rates.

Future research efforts will be directed at employing a depth-sorted haloing strategy, using adaptive time steps for flow integration, experimenting with different types of geometry (requires programmability of the graphics card's geometry engine), and extending the approach to unstructured and irregularly grid-structured data.

Acknowledgments

This work was supported by the National Science Foundation under contract ACI 9624034 (CAREER Award), through the Large Scientific and Software Data Set Visualization (LSSDSV) program under contract ACI 9982251, through the National Partnership for Advanced Computational Infrastructure (NPACI) and a large Information Technology Research (ITR) grant; the National Institutes of Health under contract P20 MH60975-06A2, funded by the National Institute of Mental Health and the National Science Foundation; by a United States Department of Education Government Assistance in Areas of National Need (DOE-GAANN) grant #P200A980307; and through a Hewlett-Packard contribution to a Graduate Student Fellowship. We thank the members of the Visualization and Computer Graphics Research Group at the Institute for Data Analysis and Visualization (IDAV) at the University of California, Davis.

References

[ARS79] APPEL A., ROHLF F. J., STEIN A. J.: The haloed line effect for hidden line elimination. *Computer Graphics 13*, 2 (1979), 151–157. 6

[Bat68] BATCHER K. E.: Sorting networks and their applications. In *Proceedings of AFIPS Spring Joint Computing Conference* (1968), vol. 32, pp. 307–314. 5

[BMP*90] BANCROFT G. V., MERRITT F. J., PLESSEL T. C., KELAITA P. G., MCCABE R. K., GLOBUS A.: FAST: a multi-processed environment for visualization of computational fluid dynamics. In *VIS '90: Proceedings of IEEE conference on Visualization '90* (1990), IEEE Computer Society Press, pp. 14–27. 2

[CL93] CABRAL B., LEEDOM L.: Imaging vector fields using line integral convolution. In *Computer Graphics Proceedings* (1993), pp. 263–269. 1, 2

[CM93] CRAWFIS R. A., MAX N.: Texture splats for 3d vector and scalar field visualization. In *Proceedings of IEEE Conference on Visualization 1993* (1993), Nielson G. M., Bergeron D., (Eds.), IEEE, IEEE Computer Society Press, pp. 261–266. 6

[FG98] FUHRMANN A., GRÖLLER E.: Real-time techniques for 3d flow visualization. In *VIS '98: Proceedings of IEEE conference on Visualization '98* (1998), IEEE Computer Society Press, pp. 305–312. 2

[JL97] JOBARD B., LEFER W.: The motion map: efficient computation of steady flow animations. In *VIS '97: Proceedings of IEEE conference on Visualization '97* (1997), IEEE Computer Society Press, pp. 323–328. 2

[JL00] JOBARD B., LEFER W.: Unsteady flow visualization by animating evenly-spaced streamlines. *Computer Graphics Forum 19*, 3 (2000), 21–31. 2

[Kel96] KELLER A.: Quasi-monte carlo methods in computer graphics: The global illumination problem. *Lectures in Applied Mathematics 32* (1996), 455–469. 3

[KSW04] KIPFER P., SEGAL M., WESTERMANN R.: Uberflow: A GPU-based particle engine. In *Proceedings of Eurographics Conference on Graphics Hardware* (2004). 5

[Lar02] LARAMEE R. S.: Interactive 3d flow visualization using a streamrunner. In *CHI 2002 Conference on Human Factors in Computing Systems, Extended Abstracts* (2002), pp. 804–805. 2

[LHD*04] LARAMEE R. S., HAUSER H., DOLEISCH H., VROLIJK B., POST F. H., WEISKOPF D.: The state of the art in flow visualization: Dense and texture-based techniques. *Computer Graphics Forum 23* (2004). 1

[MBC93] MAX N., BECKER B., CRAWFIS R.: Flow volumes for interactive vector field visualization. In *VIS '93: Proceedings of IEEE conference on Visualization '93* (1993), pp. 19–24. 1

[Nie92] NIEDERREITER H.: *Random Number Generation and quasi-Monte Carlo Methods*. SIAM, 1992. 3

[PBL*04] PARK S., BUDGE B., LINSEN L., HAMANN B., JOY K. I.: Multi-dimensional transfer functions for interactive 3d flow visualization. In *Proceedings of The 12th Pacific Conference on Computer Graphics and Applications - Pacific Graphics 2004* (2004), Cohen-Or D., Ko H.-S., Terzopoulos D.,, Warren J., (Eds.). 2, 5

[PDC*03] PURCELL T. J., DONNER C., CAMMARANO M., JENSEN H. W., HANRAHAN P.: Photon mapping on programmable graphics hardware. In *HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (2003), Eurographics Association, pp. 41–50. 5

[PLV*02] POST F. H., LARAMEE R. S., VROLIJK B., HAUSER H., DOLEISCH H.: Feature extraction and visualisation of flow fields. In *Eurographics 2002, State of the Art Reports* (2002), Fellner D., Scopigno R., (Eds.), The Eurographics Association, IEEE Computer, pp. 69–100. 1

[PVH*03] POST F. H., VROLIJK B., HAUSER H., LARAMEE R. S., DOLEISCH H.: The state of the art in flow visualization: Feature extraction and tracking. *Computer Graphics Forum 22*, 4 (2003), 775–792. 2

[SK97] SHEN H., KAO D.: UFLIC: A line integral convolution algorithm for visualizing unsteady flows. In *Proceedings of IEEE Conference on Visualization 1997* (1997), Yagel R., Hagen H., (Eds.), IEEE Computer Society Press, pp. 317–322. 2

[SM02] SCHUSSMAN G., MA K.-L.: Scalable self-orienting surfaces: A compact, texture-enhanced representation for interactive visualization of 3d vector fields. In *Proceedings of Tenth Pacific Conference on Computer Graphics and Applications – Pacific Graphics 2002* (2002), Coquillart, Shum,, Hu, (Eds.), IEEE, IEEE Computer Society Press. 6

[SML04] SCHROEDER W., MARTIN K., LORENSEN B.: *The Visualization Toolkit An Object-Oriented Approach To 3D Graphics*, 3 ed. Kitware, Inc. publishers, 2004. 2

[SZH97] STALLING D., ZÖCKLER M., HEGE H.-C.: Fast display of illuminated field lines. *IEEE Transactions on Visualization and Computer Graphics 3*, 2 (1997), 118–128. 5

[TB96] TURK G., BANKS D.: Image-guided streamline placement. In *International Conference on Computer Graphics and Interactive Techniques* (1996), pp. 453–460. 2

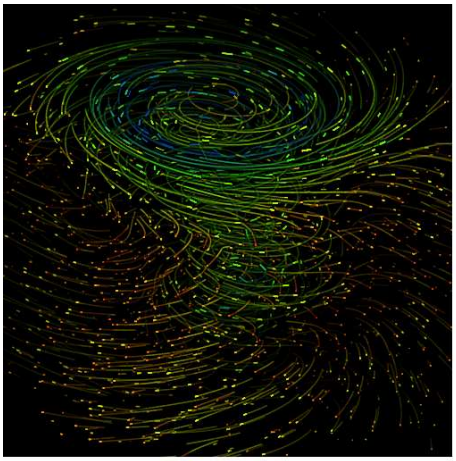
[USM96] UENG S. K., SIKORSKI K., MA K.-L.: Efficient streamline, streamribbon, and streamtube constructions on unstructured grids. *IEEE Transactions on Visualization and Computer Graphics* (1996), 100–110. 2

[vW02] VAN WIJK J.: Image based flow visualization. In *Proceedings of the 29th Conference on Computer Graphics and Interactive Techniques* (2002), Spencer S., (Ed.), vol. 21 of *ACM Transactions on Graphics*, ACM Press, pp. 745–754. 2

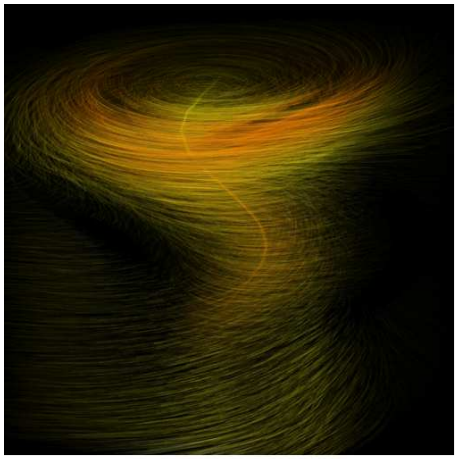
[Wei04] WEISKOPF D.: Dye advection without the blur: A level-set approach for texture-based visualization of unsteady flow. *Computer Graphics Forum (Eurographics 2004) 23*, 3 (2004), 479–488. 2, 7

[WGP97] WEGENKITTL R., GRÖLLER E., PURGATHOFER W.: Animating flow fields: Rendering of oriented line integral convolution. In *CA '97: Proceedings of Computer Animation* (1997), IEEE Computer Society, p. 15. 2

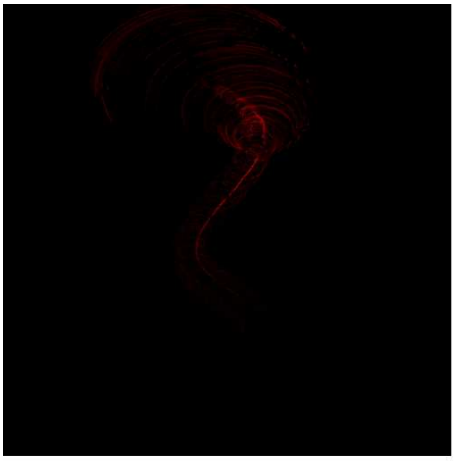
[ZSH96] ZÖCKLER M., STALLING D., HEGE H.-C.: Interactive visualization of 3d-vector fields using illuminated stream lines. In *VIS '96: Proceedings of IEEE conference on Visualization '96* (1996), IEEE Computer Society Press. 2, 5



(a)



(b)



(c)

Figure 3: *Dense geometric flow visualization applied to steady flow of tornado data set. Streamlines are rendered using (a) haloing and flow orientation or (b) depth sorting and illumination in conjuction with depth-based attenuation. MDTFs are used for feature extraction, e. g., regions of high curl magnitude (c).*

(a)

(b)

(c)

Figure 4: *Animation with dense geometric flow visualization applied to unsteady flow of jets data set. Pathlines are rendered at time steps 1120, 1400, and 1800.*