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

Real-Time Advection and Volumetric Illumination for the Visualization of 3D Unsteady Flow

Daniel Weiskopf, Tobias Schafhitzel, and Thomas Ertl

Institute of Visualization and Interactive Systems, University of Stuttgart

Abstract

This paper presents an interactive technique for the dense texture-based visualization of unsteady 3D flow, taking into account issues of computational efficiency and visual perception. High efficiency is achieved by a novel 3D GPU-based texture advection mechanism that implements logical 3D grid structures by physical memory in the form of 2D textures. This approach results in fast read and write access to physical memory, independent of GPU architecture. Slice-based direct volume rendering is used for the final display. A real-time computation of gradients is employed to achieve volume illumination. Perception-guided volume shading methods are included, such as halos, cool/warm shading, or color-based depth cueing. The problems of clutter and occlusion are addressed by supporting a volumetric importance function that enhances features of the flow and reduces visual complexity in less interesting regions.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

1. Introduction

Vector field visualization is an important topic in scientific visualization and has been the subject of active research for many years. Typically, data originates from numerical simulations—such as those of computational fluid dynamics—or from measurements, and needs to be analyzed by means of visualization to gain an understanding of the flow. Particle-tracing methods are among the standard techniques for flow visualization. A fundamental problem is to choose appropriate seed points for particle tracing in order to visualize all important features of a flow. One solution to this issue is to employ a dense representation in the form of a texture-based visualization. This approach is well investigated for 2D planar and curved surfaces, but less well understood for 3D domains.

Dense representations of 3D flow are challenging because of two fundamental problems. First, the computational complexity increases significantly since computations have to be performed for all cells of a 3D grid. Second, it is difficult to find a good visual representation of a dense collection of particle traces because most particle traces will be occluded by others and the display becomes cluttered. We think that interactivity plays a crucial role in improving the visual representation because motion parallax is a good depth cue, the problem of occlusion can be eased by exploring the scene from different viewpoints, and an animated visualization gives a good impression of the direction and magnitude of the vector field. Moreover, interesting flow regions can be investigated in detail by selectively viewing those regions and locally increasing the density of the visualization.

This paper addresses issues of computational efficiency and visual perception alike. First, an efficient 3D GPU-based texture advection mechanism is proposed, where 2D textures are used for fast read and write access to the logical 3D memory. Second, we employ an on-the-fly computation of volume illumination to display the results of texture advection by slice-based volume rendering, which is accelerated by early ray termination via the early z test. Third, perception-guided volume shading methods are included, such as halos, cool/warm shading, or color-based depth cueing. Fourth, a volumetric importance function is supported to enhance features of the data set and reduce visual complexity in less interesting regions. All these elements of the visualization system are real-time capable and therefore support interactive visualization.

2. Previous Work

Texture-based representations are an important element of the research in flow visualization. A comprehensive overview is given in the survey article [LHD*04]. Early texture-based techniques are Spot Noise [vW91] and Line Integral Convolution (LIC) [CL93]. A related approach makes use of texture advection [MB95], which can be extended to 2D Lagrangian-Eulerian Advection (LEA) [JEH02] or 2D Image Based Flow Visualization (IBFV) [vW02]. One reason for recent advances in texture-based flow visualization is the increasing performance and functionality of GPUs, which can be used to improve the speed of 2D flow visualization [JEH00, WHE01, vW02].

Texture-based visualization can be extended to vector fields on curved surfaces [LJH03, vW03] and in 3D [TvW03, WHE01, WE04]. In the context of 3D LIC, dye visualization can be used to highlight features [SJM96]. 3D flow visualization is subject to perceptual issues, which can be addressed by a combination of interactive clipping and user intervention [RHTE99]. Alternatively, 3D LIC volumes can be effectively represented by selectively emphasizing important regions of the flow, enhancing depth perception, and improving orientation perception [IG97]. Perception of 3D flow can also be enhanced by shading according to limb darkening via transfer functions [HA04], by interactively changing the rendering style [LBS03], or by volume rendering of implicit flow volumes [XZC04]. All these systems for perception-guided 3D flow visualization are either not interactive at all or require some time-consuming pre-processing for particle tracing.

Finally, the extraction and selective display of flow features effectively reduces visual complexity. Background information on feature-based flow visualization can be found in the survey article [PVH*03]. In the context of this paper, interactive feature definition for focus-and-context 3D flow visualization [DGH03] is an interesting approach.

3. Semi-Lagrangian 3D Texture Advection

The in-depth discussion of our 3D visualization approach begins with the underlying semi-Lagrangian transport mechanism. Here, particles or marker "objects" (such as dye) are modeled as massless material that is perfectly advected along the input vector field. From an Eulerian point of view, particles lose their individuality and are represented by their property values (such as color or gray-scale values), which are stored in a property field $\rho(\mathbf{x},t)$, where $\mathbf{x}$ denotes position and t denotes time. This property field is typically given on a uniform grid. The evolution of the property field is governed by the convection equation,

$$\frac{\partial \rho(\mathbf{x},t)}{\partial t} + \mathbf{v}(\mathbf{x},t) \cdot \nabla \rho(\mathbf{x},t) = 0 \quad ,$$

where $\mathbf{v}$ is the input vector field. We solve this equation by a semi-Lagrangian approach [Sta99, JEH00] that leads to a stable evolution even for large steps. Advection is performed along pathlines (for unsteady flow) or streamlines (for steady flow). Therefore, the ordinary differential equation,

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{v}(\mathbf{x}(t),t) \quad ,$$

for Lagrangian particle tracing needs to be solved. Backward explicit integration is employed to compute particle positions at a previous time step, $\mathbf{x}(t-\Delta t)$. For example, first-order Euler integration yields

$$\mathbf{x}(t-\Delta t) = \mathbf{x}(t) - \Delta t \mathbf{v}(\mathbf{x}(t),t) \quad .$$

Starting from the current time step t, an integration backwards in time provides the position along the pathline at the previous time step. The property field is evaluated at this previous position to access the value that is transported to the current position, leading to a backward advection scheme,

$$\rho(\mathbf{x}(t),t) = \rho(\mathbf{x}(t-\Delta t),t-\Delta t) \quad , \tag{1}$$

in the general case, or to

$$\rho(\mathbf{x},t) = \rho(\mathbf{x}-\Delta t \mathbf{v}(\mathbf{x},t),t-\Delta t) \quad , \tag{2}$$

in the case of Euler integration. This 3D advection is suitable for unsteady flow because the time dependency of the vector field is taken into account.

Equations (1) and (2) lead to a GPU implementation that represents the property and vector fields by 3D textures [WE04]. The physical position $\mathbf{x}$ and the corresponding texture coordinates are related by an affine transformation that takes into account that the physical and computational spaces may have different units and origins. Accordingly, the step size in computational (texture) space directly corresponds to the physical time step Δt. While the property texture is only updated at grid points, the lookup in the property field at the previous time step is performed at locations that may differ from exact grid positions. Therefore, trilinear interpolation is employed to reconstruct the value of the property field at the previous time step. Since any rendering operation is restricted to a 2D domain, the property field for a subsequent time step is constructed in a slice-by-slice manner. Each slice of the property field is updated by rendering a quadrilateral that represents this 2D subset of the full 3D domain. The dependent lookup in the "old" property texture can be realized by a fragment program that computes the modified texture coordinates according to the Euler integration along the flow field.

The main problem with this implementation is the slice-by-slice update of the 3D texture for the property field. In many cases, an update of a 3D texture is only possible via transfer of data to and from main memory. For example, Direct3D does not provide a mechanism to directly modify 3D textures from other data on the GPU. Although OpenGL allows us to update a slice of a 3D texture by `glCopyTexSubImage3D`, the speed of such an update can vary extremely between GPU architectures because of different in-

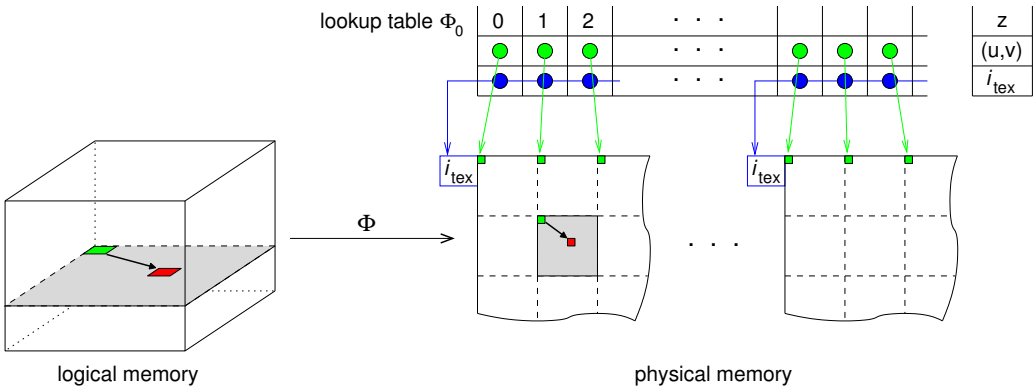

Figure 1: *Mapping between logical 3D memory and physical 2D memory by means of a lookup table.*

ternal memory layouts of 3D textures (see a related discussion on read access for 3D textures [WWE04]). This fundamental issue will most probably remain, even with the projected superbuffer extension [Per03]. Therefore, we propose an alternative approach that is based on 2D texture memory instead of 3D texture memory. 2D textures are available on any GPU, they provide good caching mechanisms and efficient bilinear resampling, and they support an extremely fast write access by the render-to-texture functionality.

For our 2D texture-based implementation, we have to distinguish between logical memory and physical memory. Logical memory is identical as for 3D texture advection—it is organized in the form of a uniform 3D grid. Physical memory is a 2D uniform grid represented by a 2D texture. We denote the coordinates for addressing the logical memory by $\mathbf{x} = (x, y, z)$ and the coordinates for physical memory by $\mathbf{u} = (u, v)$. A slice of constant value z in logical memory corresponds to a tile in physical memory, as illustrated in Figure 1. Different tiles are positioned in physical memory with a row-first order. Since the maximum size of a 2D texture may be limited, several "large" 2D textures may be used to provide the necessary memory. These 2D textures are labeled by the integer-valued index i_{tex}.

Since all numerical operations of 3D advection are conceptually computed in logical 3D space, we need an efficient mapping from logical to physical memory, which is described by the function

$$\Phi : \ (x, y, z) \longmapsto ((u, v);\, i_{\text{tex}}) \quad .$$

The 2D coordinates can be separated into the coordinates for the origin of a tile, $\mathbf{u}_0$, and the local coordinates within the tile, $\mathbf{u}_{\text{local}}$:

$$\Phi : \ (x, y, z) \longmapsto (\mathbf{u}_0 + \mathbf{u}_{\text{local}};\, i_{\text{tex}}) \quad , \tag{3}$$

with

$$\mathbf{u}_0 = \Phi_{0,\mathbf{u}}(z), \ \ i_{\text{tex}} = \Phi_{0,i_{\text{tex}}}(z), \ \ \mathbf{u}_{\text{local}} = (s_x x, s_y y) \ . \tag{4}$$

The function Φ_0 maps the logical z value to the origin of a tile in physical memory and is independent of the x and y coordinates. The map Φ_0 can be represented by a lookup-table (see Figure 1), which can be efficiently implemented by a dependent texture lookup. Conversely, the local tile coordinates are essentially identical to (x, y)—up to scalings (s_x, s_y) that take into account the different relative sizes of texels in logical and physical memory. If more than one "large" 2D texture is used, multiple texture lookups in these physical textures may be necessary. However, multiple 2D textures are only required in tiles that are close to the boundary between two physical textures because the maximum difference vector for the backward lookup in Eq. (2) is bounded by $\Delta t \mathbf{v}_{\max}$, where $\mathbf{v}_{\max}$ is the maximum velocity of the data set. We use different fragment programs for boundary tiles and internal tiles to reduce the number of texture samplers for the advection in internal regions.

Trilinear interpolation in logical space is implemented by two bilinear interpolations and a subsequent linear interpolation in physical space. Bilinear interpolation within a tile is directly supported by built-in 2D texture interpolation. A one-texel-wide border is added around each tile to avoid any erroneous influence by a neighboring tile during bilinear interpolation. The subsequent linear interpolation takes the bilinearly interpolated values from the two closest tiles along the z axis as input. This linear interpolation is implemented within a fragment program.

While trilinear interpolation by the above mapping scheme is necessary for the read access in Eqs. (1) or (2), write access is more regularly structured. First, write access does not need any interpolation because it is restricted to grid points (i.e., single texels). Second, the backward lookup for Eqs. (1) or (2) allows us to fill logical memory in a slice-by-slice manner and, thus, physical memory in a tile-by-tile fashion. A single tile can be filled by rendering a quadrilateral into the physical 2D texture if the viewport is restricted to the corresponding subregion of physical memory.

4. Visual Mapping and Volume Rendering

So far, only the basic advection mechanism has been discussed. However, a useful visualization needs—besides the computation of particle traces—a mapping of the particle traces to a graphical representation. In this paper, the mapping is restricted to an appropriate injection of property values, adopting the basic idea of 2D IBFV (Image-Based Flow Visualization) [vW02]. IBFV introduces new property values at each time step, described by an injection texture I. The structure of the injection mechanism of 2D IBFV is illustrated in Figure 2.

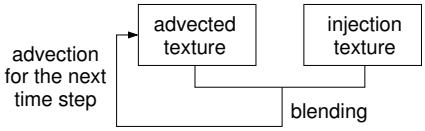

Figure 2: *Basic structure of the injection mechanism of IBFV.*

The original compositing schemes for 2D IBFV and 3D IBFV [TvW03] can be generalized to allow for a unified description of both noise and dye advection [WE04]: First, the restriction to an affine combination of the advected value and of the newly injected value is replaced by a generic linear combination of both and, second, several materials can be advected and blended independently. The extended blending equation is given by

$$\rho(\mathbf{x},t) = W(\mathbf{x},t) \circ \rho(\mathbf{x}(t-\Delta t), t-\Delta t) + V(\mathbf{x},t) \circ I(\mathbf{x},t) \ ,$$

with two, possibly space-variant and time-dependent, weights W and V. The symbol "$\circ$" denotes a component-wise multiplication of two vector quantities. The different components of each texel in the property field describe the density of different materials. Continuous blending of injected "particles" leads to streakline-like visual structures. The advantages of the extended blending scheme are: First, different materials are blended independently from each other and may therefore have different lengths of exponential decay; second, material can be added on top of existing material, which is the prerequisite for dye advection (see the discussion in [WE04]).

The property field ρ is visualized by volume rendering with texture slicing—similarly to 2D texture-based rendering with axis-aligned slices. Traditional 2D texture slicing holds three copies of a volume data set, one for each of the main axes. In our approach, however, only a single copy of the property field is stored on GPU. The stacking direction is changed on-the-fly during advection if the viewing angle becomes larger than 45 degrees with respect to the stacking axis to avoid holes in the final display. (For example, these holes are present in 3D IBFV [TvW03]. Compared to 3D IBFV, our approach has further advantages: It avoids multiple render passes per slice, it facilitates the visualization

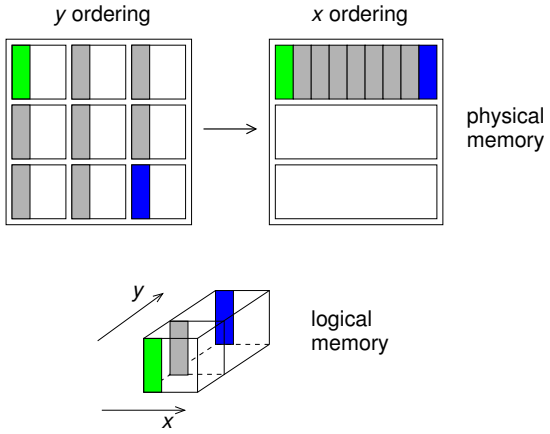

Figure 3: *Reordering of stack direction.*

of arbitrary vector fields without the restriction to velocities with small z components, it supports the independent transport of several materials, and it provides a more flexible blending scheme—see the related discussion in [WE04].) Figure 3 illustrates a reordering of the stacking direction from y to x axis. A tile in the new stacking order is rendered in a stripe-by-stripe fashion, according to portions of tiles from the old stacking order. The reordering process takes into account that the number of tiles, their sizes, and their positions may change.

Actual volume rendering accesses tile after tile in front-to-back order by rendering object-aligned quadrilaterals. Texture coordinates are issued to address a tile in physical memory of the "large" 2D texture. We employ a dependent-texture lookup in a fragment program to implement post-classification. For each material in the property field, density is mapped to optical properties (color and opacity) by its corresponding transfer function. The results of different transfer functions for different materials are added to obtain the final color and opacity.

For dense 3D flow representations, transfer functions are typically specified to render interesting flow regions with high opacities. Therefore, early ray termination is an effective way of accelerating volume rendering. Similarly to [RGW*03], the early z test is used to efficiently skip the execution of a fragment program when a user-specified maximum opacity has been accumulated. To this end, terminated rays are masked in a separate rendering pass by setting the z buffer to zero (near clipping plane); the z value is set to the far clipping plane for all other pixels. Then, the depth test skips terminated rays while a slice of the volume is rendered. As the initialization of the z buffer in the separate pass consumes additional rendering time, we choose to perform this initialization only for every n-th volume slice to achieve a compromise between perfect early ray termination and the additional costs for the separate pass. A typical value is $n = 10$.

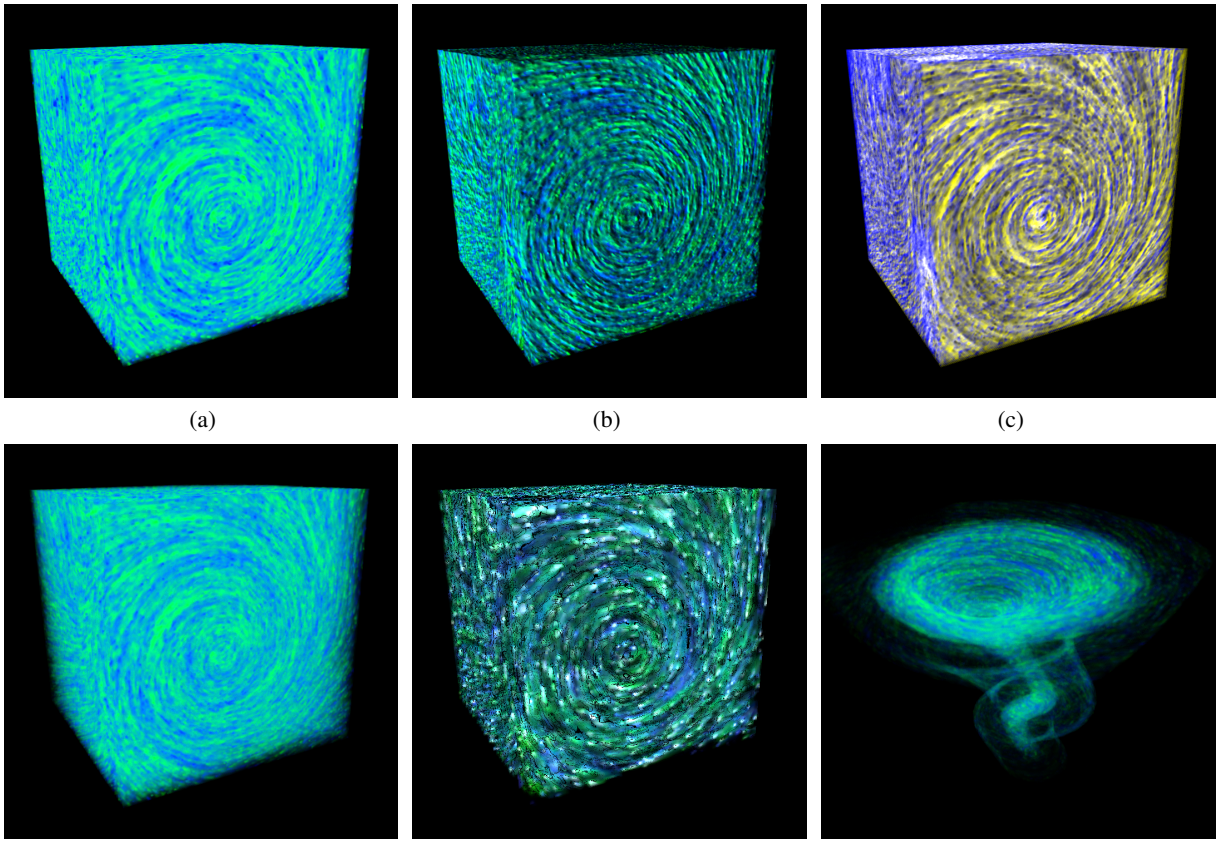

Figure 4: *3D advection for a tornado data set with volume rendering based on the emission-absorption model (a), Phong illumination (b), cool/warm shading (c), depth cueing (d), halos (e), and velocity masking (f).*

5. Illumination and Visual Perception

The previous section has described volume rendering according to the emission-absorption model, which leads to a display similar to a self-emitting gas cloud. Although this model allows us to view different semi-transparent depth layers of a 3D flow field, it fails to explicitly visualize the orientation and relative depth of streakline structures. Figure 4 (a) shows a visualization according to the emission-absorption model with high opacities. The underlying data set represents the wind flow in a tornado. To improve the final display, we additionally apply volume shading to the property field. In general, volumetric illumination needs gradients, which serve as normal vectors for local illumination.

Our goal is to incorporate volume shading into the above real-time advection system and, therefore, gradients have to be computed in real time as well. We employ a numerical computation by central differences, which delivers gradients of acceptable quality at a high speed. Gradients are stored in a grid that has the same logical and physical memory layout as the property field. Central differences exhibit a uniform access to 3D logical memory because data is fetched from neighboring grid points along the three main axes. Accordingly, a well-structured accessing scheme is also applied in 2D physical texture memory: Neighboring texels in the current tile yield the x and y partial derivatives whereas the partial derivative along the z axis is based on the two closest tiles in z direction.

The mapping from logical to physical memory according to Eq. (3) needs only to be computed at the four vertices that describe a single tile in physical memory. Six pairs of texture coordinates are attached to the vertices and interpolated by scanline conversion. Linear interpolation and the mapping from Eq. (3) are commutative: The respective tile origins $\mathbf{u}_0$ and texture indices i_{tex} are constant and the local coordinates $\mathbf{u}_{\text{local}}$ vary linearly because the relative distance between a central grid point and its neighbors is constant for a complete tile.

Gradients are computed after each advection and blending iteration and before volume rendering. Therefore, any gradient-based volume shading method may be applied. Fig-

ure 4 (b) shows an example of volume illumination by the Phong model with diffuse and specular components, which are added on top of the emissive part that is determined by the transfer function from the previous section. We use the same transfer function value as material color for diffuse and specular illumination, but other definitions of material colors could be easily incorporated, if required. Phong illumination greatly improves the perception of streakline orientation by shading—highlights in combination with camera motion are particularly effective in revealing streakline orientation.

Another illumination model implements cool/warm shading [GGSC98, ER00]. Here, orientation with respect to light direction is encoded by warm or cool colors, respectively. Figure 4 (c) shows an example image. The advantage of cool/warm shading is that orientation is represented by (almost) isoluminant colors. Therefore, brightness can still be used to visualize another attribute or property.

A prominent additional attribute is depth with respect to the camera. We apply color-based depth cueing to imitate aerial perspective. Figure 4 (d) shows an example in which increasingly black background color is added with increasing depth (fogging with black color). Variations are also feasible, e.g., a subtle blue shift [ER00] can be included by adding a blue component to the fog color.

Halos are effective in visualizing relative depth between line-like structures [IG97]: Objects behind a closer streakline are partly hidden by dark halos. We implement halos as thick silhouette lines. Silhouette lines are detected by examining the dot product of gradient and viewing directions because an ideal silhouette has a gradient perpendicular to the viewing vector. An additional transfer function is included to specify halos. This transfer function maps the above dot product to color and opacity. Thick silhouette lines are implemented by mapping a finite range of input values (around zero) to high opacities. Figure 4 (e) shows an example with halos and Phong illumination. For purely geometric reasons, limb darkening through high opacities in the transfer function [HA04] also contributes to the halo effect. In addition, the gradient criterion further enhances halos.

Here, we would like to point out that we always advect several materials because different streakline colors are an effective means of visualizing continuity along lines [IG97].

6. Feature-Based Visualization

Another issue of dense 3D representations is clutter and occlusion. This problem can be addressed by selectively fading out uninteresting flow regions. The important parts can be regarded flow features in a general sense. The derivation of new feature definitions is beyond the scope of this paper. In fact, we assume that any useful feature description can be condensed into a scalar-valued importance function. Our visualization method directly supports such an importance function—the importance value can be used to modify the transfer functions by a nonlinear mapping.

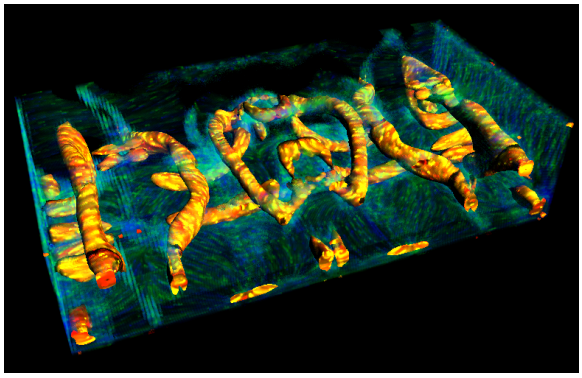

Figure 5: *Benard flow with λ_2 vortex visualization.*

Velocity magnitude can be an interesting feature measure (see the discussion for 2D flow in [JEH02]). Figure 4 (f) shows the visualization of the tornado data set with velocity masking—only regions with large velocity magnitude are visible. A more sophisticated feature is used in Figure 5, where λ_2 vortex detection [JH95] is applied. This vortex definition is widely used in fluid dynamics and, e.g., has the advantage of being Galilei invariant. Vortex regions are displayed by materials with red or yellow colors while the surrounding flow is still visible as semi-transparent material with blue or green color. These two examples demonstrate that different features can be used to emphasize flow regions within our system.

7. Implementation

Our implementation is based on C++ and DirectX 9.0, and was tested on Windows XP machines with ATI Radeon X800 Platinum Edition (256 MB) and NVIDIA GeForce 6800 Ultra (256 MB), respectively. GPU states and shader programs are configured within clear-text effect files. All shader programs are formulated with high-level shading language (HLSL) to achieve a code that is easy to read and maintain. A comparable implementation should be feasible with OpenGL and its vertex and fragment program support.

Semi-Lagrangian advection can be implemented by fragment programs because the backward advection from Eqs. (1) or (2) and the mapping between logical and physical memory from Eqs. (3) and (4) can be directly mapped to GPU instructions. Since only short streaklets are used in our visualizations, the accuracy of first-order Euler integration is sufficient. Higher-order methods, however, could be readily included at the cost of decreased advection speed. All computations take place on a texel-by-texel level, which essentially reduces the role of the surrounding C++ program

Table 1: *Performance for steady flow visualization on a 600^2 viewport (in fps).*

	Radeon X800		GeForce 6800	
Domain size	128^3	256^3	128^3	256^3
Advection	114.2	13.0	96.4	16.7
+Volrend w/ early ray	71.9	11.6	53.8	12.7
+Volrend w/o early ray	51.2	9.9	53.3	10.0
Reorder stacking axis	2.4	0.5	2.7	0.4
Gradient computation	167.4	12.1	124.4	20.2
Advection & volrend:				
Phong	34.5	5.4	29.1	6.7
Halo & Phong	33.5	5.3	23.0	6.0
Depth cue	66.0	11.3	12.7	53.4

Table 2: *Performance for steady vs. unsteady flow visualization on a 600^2 viewport (in fps).*

	Radeon X800		GeForce 6800	
Domain size	128^3	256^3	128^3	256^3
Advection only:				
Steady	114.2	13.0	96.4	16.7
Unsteady	51.2	9.9	39.0	6.7
Advection & Phong:				
Steady	34.5	5.4	29.1	6.7
Unsteady	22.5	4.1	19.7	5.9

to allocating memory for the required textures and executing the fragment programs by drawing domain-filling quadrilaterals. Textures are updated by using ping-pong rendering and the render-to-texture functionality of DirectX. An analogous GPU implementation is feasible for gradient computation and volume shading.

Property and gradient fields are represented by physical 2D textures with 8-bit fixed-point color channels. Vector field and particle injection textures, however, are stored in 3D textures because they do not need to be modified on the GPU. Particle injection textures have 8-bit color channels, vector field textures have 8-bit fixed-point or 16-bit floating-point resolution, depending on the required accuracy. The example images and performance measurements in this paper are based on 8-bit vector fields. Additional material can be found on the accompanying web page[†], which contains the source code of the effect file for the core advection routine and electronic videos with animated visualizations.

Table 1 shows performance measurements for our implementation on ATI Radeon X800 Platinum Edition (256 MB) and NVIDIA GeForce 6800 Ultra (256 MB). Viewport size for volume rendering is 600^2, the size of the property and gradient fields are given in the table. Here, we use a steady vector field of size 128^3 (tornado from Figure 4). The measurements indicate that advection speed is roughly proportional to the number of texels. Sophisticated volume shading tends to be slower than pure emission-absorption volume rendering. Interactive visualization is feasible with property fields up to 256^3. A slight problem is the slow reordering of the stacking direction. Switching between stacks, however, does not occur very often in interactive applications and, therefore, this rendering bottleneck typically does not disturb the user.

Table 3: *Comparison between 2D texture-based advection of this paper and 3D texture-based advection from [WE04] on Radeon X800 with 600^2 viewport (in fps).*

	2D Texture		3D Texture	
Domain size	128^3	256^3	128^3	256^3
Advection only	114.2	13.0	44.4	9.7
Advection & volrend	71.9	11.6	16.8	4.4

Table 2 compares the visualization performance for unsteady and steady flow under the same conditions as for Table 1. For unsteady flow, a new 3D texture for the vector field is transferred for each frame. Although the rendering speed is reduced for unsteady flow, the overall performance still facilitates interactive visualization for property fields up to 256^3. Table 3 compares the advection method of this paper (implemented in DirectX) and the previous 3D texture-based method from [WE04] (implemented in OpenGL). Here, only ATI Radeon X800 is considered because the rendering speed of the OpenGL version is much slower on NVIDIA GPUs due to slow `glCopyTexSubImage3D` (e.g., 2.0 fps on GeForce 6800 Ultra for 256^3 property field, advection only). Even on X800, however, the new 2D texture-based method outperforms the 3D texture-based method, i.e., the benefit of fast read and write access to 2D texture outweighs the additional operations for the mapping of texture coordinates from logical 3D space to physical 2D space.

8. Conclusions and Future Work

We have presented an interactive technique for the dense texture-based visualization of unsteady 3D flow. Efficient 3D texture advection has been achieved by mapping from logical 3D memory to 2D physical memory implemented by 2D textures. 2D texture memory has the advantage of fast read and write accesses that are independent of GPU architecture. Streakline-like structures are constructed by a flexible particle injection and blending scheme that allows for different materials as well as for noise and dye advection. We have presented a GPU-based real-time computation of gradients as a basis for volume illumination. The perception of orientation and depth of streaklines is improved by Phong

[†] http://www.vis.uni-stuttgart.de/texflowvis

illumination, cool/warm shading, halos, or color-based depth cues. A generic volumetric importance function is used to address the issue of clutter and occlusion: Important flow regions are emphasized and visual complexity is reduced in less interesting parts by modifying the transfer function. All steps of our visualization system are capable of real-time visualization. We believe that interactivity is one of the major building-blocks for achieving an appropriate volumetric visualization of 3D flow because interaction helps to address perceptual issues of occlusion and spatial perception.

Since our visualization method is "orthogonal" to feature descriptions, advanced interactive feature definitions [DGH03] could be incorporated in future work. Another interesting line of research could include techniques from multi-field visualization to simultaneously visualize vector data (by advection) and additional attributes, such as pressure or temperature.

Acknowledgments

We would like to thank Roger Crawfis for providing the tornado data set used in Figure 4, Simon "nine-to-go" Stegmaier for help with the λ_2 code, and Mark Segal (ATI) for the Radeon X800 Platinum Edition graphics board.

References

[CL93] CABRAL B., LEEDOM L. C.: Imaging vector fields using line integral convolution. In *Proc. ACM SIGGRAPH* (1993), pp. 263–270. 2

[DGH03] DOLEISCH H., GASSER M., HAUSER H.: Interactive feature specification for focus+context visualization of complex simulation data. In *EG / IEEE TCVG Symp. Vis.* (2003), pp. 239–248. 2, 8

[ER00] EBERT D., RHEINGANS P.: Volume illustration: Non-photorealistic rendering of volume models. In *IEEE Vis.* (2000), pp. 195–202. 6

[GGSC98] GOOCH A., GOOCH B., SHIRLEY P., COHEN E.: A non-photorealistic lighting model for automatic technical illustration. In *Proc. ACM SIGGRAPH* (1998), pp. 101–108. 6

[HA04] HELGELAND A., ANDREASSEN O.: Visualization of vector fields using seed LIC and volume rendering. *IEEE Trans. Vis. and Comp. Graph. 10*, 6 (2004), 673–682. 2, 6

[IG97] INTERRANTE V., GROSCH C.: Strategies for effectively visualizing 3D flow with volume LIC. In *IEEE Vis.* (1997), pp. 421–424. 2, 6

[JEH00] JOBARD B., ERLEBACHER G., HUSSAINI M. Y.: Hardware-accelerated texture advection for unsteady flow visualization. In *IEEE Vis.* (2000), pp. 155–162. 2

[JEH02] JOBARD B., ERLEBACHER G., HUSSAINI M. Y.: Lagrangian-Eulerian advection of noise and dye textures for unsteady flow visualization. *IEEE Trans. Vis. and Comp. Graph. 8*, 3 (2002), 211–222. 2, 6

[JH95] JEONG J., HUSSAIN F.: On the identification of a vortex. *J. Fluid Mech. 285* (1995), 69–94. 6

[LBS03] LI G. S., BORDOLOI U., SHEN H. W.: Chameleon: An interactive texture-based framework for visualizing three-dimensional vector fields. In *IEEE Vis.* (2003), pp. 241–248. 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. *Comp. Graph. Forum 23*, 2 (2004), 143–161. 2

[LJH03] LARAMEE R. S., JOBARD B., HAUSER H.: Image space based visualization of unsteady flow on surfaces. In *IEEE Vis.* (2003), pp. 131–138. 2

[MB95] MAX N., BECKER B.: Flow visualization using moving textures. In *Proc. ICASW/LaRC Symp. Visualizing Time-Varying Data* (1995), pp. 77–87. 2

[Per03] PERCY J.: OpenGL extensions. ATI presentation at ACM SIGGRAPH 2003, http://www.ati.com/developer, 2003. 3

[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. *Comp. Graph. Forum 22*, 4 (2003), 775–792. 2

[RGW*03] RÖTTGER S., GUTHE S., WEISKOPF D., ERTL T., STRASSER W.: Smart hardware-accelerated volume rendering. In *EG / IEEE TCVG Symp. Vis.* (2003), pp. 231–238. 4

[RHTE99] REZK-SALAMA C., HASTREITER P., TEITZEL C., ERTL T.: Interactive exploration of volume line integral convolution based on 3D-texture mapping. In *IEEE Vis.* (1999), pp. 233–240. 2

[SJM96] SHEN H.-W., JOHNSON C. R., MA K.-L.: Visualizing vector fields using line integral convolution and dye advection. In *Vol. Vis. Symp.* (1996), pp. 63–70. 2

[Sta99] STAM J.: Stable fluids. In *Proc. ACM SIGGRAPH* (1999), pp. 121–128. 2

[TvW03] TELEA A., VAN WIJK J. J.: 3D IBFV: Hardware-accelerated 3D flow visualization. In *IEEE Vis.* (2003), pp. 233–240. 2, 4

[vW91] VAN WIJK J. J.: Spot noise – texture synthesis for data visualization. *Comp. Graph. (Proc. ACM SIGGRAPH 91) 25* (1991), 309–318. 2

[vW02] VAN WIJK J. J.: Image based flow visualization. *ACM Trans. Graph. 21*, 3 (2002), 745–754. 2, 4

[vW03] VAN WIJK J. J.: Image based flow visualization for curved surfaces. In *IEEE Vis.* (2003), pp. 123–130. 2

[WE04] WEISKOPF D., ERTL T.: GPU-based 3D texture advection for the visualization of unsteady flow fields. In *Proc. WSCG Short Comm. Papers* (2004), pp. 181–188. 2, 4, 7

[WHE01] WEISKOPF D., HOPF M., ERTL T.: Hardware-accelerated visualization of time-varying 2D and 3D vector fields by texture advection via programmable per-pixel operations. In *Proc. VMV* (2001), pp. 439–446. 2

[WWE04] WEISKOPF D., WEILER M., ERTL T.: Maintaining constant frame rates in 3D texture-based volume rendering. In *Proc. IEEE CGI* (2004), pp. 604–607. 3

[XZC04] XUE D., ZHANG C., CRAWFIS R.: Rendering implicit flow volumes. In *IEEE Vis.* (2004), pp. 99–106. 2