

GPU-Based 3D Texture Advection for the Visualization of Unsteady Flow Fields

Daniel Weiskopf and Thomas Ertl

Institute of Visualization and Interactive Systems, University of Stuttgart
Universitätsstr. 38, 70569 Stuttgart, Germany
{weiskopf,ertl}@informatik.uni-stuttgart.de

ABSTRACT

We present an interactive visualization approach for the dense representation of unsteady 3D flow fields. The first part of this approach is a GPU-based 3D texture advection scheme that allows a slice of the 3D visual representation to be updated in a single rendering pass. In the second step, the result of the advection process is displayed by texture-based volume rendering. Since both parts are completely supported by the GPU, interactive frame rates are achieved for the visualization of time-dependent flow fields. Moreover, the noise and dye injection scheme of Image Based Flow Visualization (IBFV) is adopted and generalized to take into account a flexible combination of advected and newly injected values. In addition, the advection and rendering methods are extended to transport and display different materials instead of the aggregated colors and opacities. This approach leads to a unified description of noise and dye advection and allows the user to specifically emphasize or blend out regions of the flow.

Keywords

Flow visualization, unsteady flow, texture advection, volume rendering, GPU programming.

1. INTRODUCTION

The visualization of 3D vector fields has been investigated and used in various scientific and engineering disciplines for many years. Typical applications stem from simulations in computational fluid dynamics, calculation of physical vector fields, such as electromagnetic fields or heat flow, or from measurements of actual wind or fluid flows. As flow visualization has a long tradition, various techniques exist to visually represent steady and unsteady vector fields. Among the standard techniques for flow visualization is the class of methods based on particle tracing. 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, LIC-like visualization. This approach is popular and well investigated for 2D planar and curved surfaces (cf. the articles [Hau02, San00]).

Dense representations of a 3D flow, however, are

more 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. In addition, on the 2D image plane an accurate spatial perception of such a 3D scene is difficult. All these aspects are especially challenging for unsteady flow fields.

We think that interactivity plays a crucial role in improving the visual representation. In an interactive application, motion parallax is a good means to improve depth perception; and the problem of occlusion can be eased by exploring the scene from different viewpoints. Moreover, animated flows give a good impression of the direction and magnitude of the velocity field. Interesting regions of a flow can be investigated in detail by locally increasing the density of the visualization or injecting virtual dye at user-specified locations. Similarly, interactive volume clipping helps to examine interior regions.

Because of the high computational costs, however, there is only little previous work that deals with completely interactive techniques for a dense representation of unsteady flow. Most previous systems with interactive rendering rely on some non-interactive preprocessing step and therefore cannot

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

WSCG SHORT Communication papers proceedings
WSCG'2004, February 2-6, 2003, Plzen, Czech Republic.
Copyright UNION Agency – Science Press

handle time-dependent data on-the-fly. Recently, the increasing power and functionality of GPUs (graphics processing units) has been exploited to solve a large number of problems in computer graphics, visualization, and even simulation. So far, however, 3D Image-Based Flow Visualization (IBVF) [Tel03] is the only approach (known to the authors) which utilizes the power of GPUs to achieve the complete 3D flow visualization process to be running at interactive frame rates.

In this paper, we build upon the basic ideas of 3D IBVF, and improve and extend this approach. The main contributions of this paper are the following. First, a fully three-dimensional advection mechanism that deals with any input vector field is presented. Unlike 3D IBVF, the flow is not restricted to velocities with very small z component. Moreover, a slice of the 3D representation is updated in a single rendering pass, which is the basis for interactive frame rates. Second, the blending mechanism is enhanced to allow for more flexible noise and dye injection. Third, the advection and 3D texture-based volume rendering schemes are extended to transport and display different materials instead of combined colors and opacities. In this way, parts of the visualization can be specifically faded out or emphasized.

2. PREVIOUS WORK

Noise-based and dense texture representations are an important part of the research in flow visualization. A comprehensive overview on the field is given in [Hau02, San00]. An early texture-synthesis technique for vector field visualization is spot noise [Wij91]. LIC [Cab93] is another technique for the dense representation of streamlines in steady vector fields. Original LIC has been extended in various respects: animated LIC [For95], visualization of the orientation of flow [Weg97], the combination of animation and dye advection [She96], LIC for unsteady flow [She98], Fast LIC [Sta95], or Pseudo LIC [Ver99].

The basic idea of texture advection is to represent a dense collection of particles in a texture and to transport this texture according to the motion of the particles [Max95]. Lagrangian-Eulerian Advection (LEA) [Job02] visualizes 2D unsteady flows by integrating particle positions (i.e., the Lagrangian part) and advecting the color of particles based on a texture representation (i.e., the Eulerian aspect). Image-Based Flow Visualization (IBFV) [Wij02] is a variant of 2D texture advection that additionally blends a second texture into the advected texture at each time step. IBFV can be extended to flow on 2D curved hypersurfaces [Wij03] and to 3D flow [Tel03]. Laramee et al. [Lar03] propose an advection scheme for unsteady flow visualization on curved hypersurfaces

that, like [Wij03], works in image space. Implementations on GPUs are possible for many of the above techniques to increase visualization performance. For example, GPU-based implementations are known for LIC [Hei99], Eulerian texture advection [Wei01], LEA [Job00, Wei02], and IBFV [Wij02, Wij03, Tel03].

Dense 3D flow visualization is subject to perceptual and computational issues. Techniques to improve depth perception and reduce the problems of occlusion in 3D LIC are presented in [Int97]. Another approach is the interactive exploration of 3D LIC, making use of volume clipping [Rez99]. Recently, a texture-based framework for interactive rendering of 3D flow fields has been proposed [Li03]. All these systems for 3D flow visualization are either not interactive at all or require some time-consuming pre-processing for particle tracing. In contrast, 3D IBFV [Tel03] utilizes graphics hardware to achieve the complete visualization process at interactive frame rates.

3. BASIC TEXTURE ADVECTION

The standard approach to particle tracing adopts a Lagrangian point of view. Here, each single particle can be identified individually and the properties of each particle depend on time t . The path of a single massless particle is determined by the ordinary differential equation

$$\frac{d\mathbf{r}(t)}{dt} = \mathbf{u}(\mathbf{r}(t), t) \quad , \quad (1)$$

where $\mathbf{r}(t)$ describes the position of the particle at time t , and $\mathbf{u}(\mathbf{r}, t)$ represents the vector field to be visualized. All vector quantities (marked as boldface letters) are either 2D or 3D, depending on the dimensionality of the computational domain. For the actual flow visualization, the path could be directly rendered as a line-like graphical object.

For texture advection, however, an Eulerian approach is used. Particles lose their individuality and are rather represented by their property values (such as color or gray-scale values), which are stored in a property field. Typically, this property field is given on a uniform grid, i.e., on a texture. We denote the texture as $T(\mathbf{c})$, where \mathbf{c} describes the texture coordinates. Once again, the dimensionality of the texture depends on the dimensionality of the computational domain. Positions of particles are only given implicitly in the form of the texture coordinates of the respective texels. Using a first-order explicit Euler scheme for Eq. (1), we obtain

$$\mathbf{r}(t - \Delta t) = \mathbf{r}(t) - \Delta t \mathbf{u}(\mathbf{r}(t), t) \quad , \quad (2)$$

for an integration backward in time with a step size of Δt . Applying this numerical solution to the property field yields

$$T_t(\mathbf{c}) = T_{t-\Delta t}(\mathbf{c} - \Delta s \mathbf{v}_t(\mathbf{c})) \quad . \quad (3)$$

The physical positions \mathbf{r} and the corresponding texture coordinates \mathbf{c} are related by an affine transformation that takes into account that the physical space and computational space may have different units and origins. Accordingly, the step size Δs in computational space corresponds to the physical time step Δt , and \mathbf{v}_t corresponds to the vector field \mathbf{u} in physical space. The subscripts in T_t and \mathbf{v}_t denote the times associated with the property field and the vector field, respectively.

In Eq. (3), the texture T_t is updated only at grid points \mathbf{c} . However, the lookup in the property field at the previous time step is performed at position $\mathbf{c} - \Delta s \mathbf{v}_t$, which may differ from exact grid positions. Therefore, either a bilinear (in 2D) or trilinear (in 3D) interpolation is employed to determine the value of the property field at this position.

4. 3D TEXTURE ADVECTION

The previous discussion of texture advection leads to an efficient implementation of 3D texture advection on current GPUs, such as nVidia's GeForce FX or ATI's Radeon 9700/9800. Both the property field and the vector field are represented by 3D textures. The transport of the property field along one time step according to Eq. (3) can be realized by a dependent-texture lookup: First, modified texture coordinates $\mathbf{c}' = \mathbf{c} - \Delta s \mathbf{v}_t$ are computed; second, these texture coordinates \mathbf{c}' serve as the basis for the dependent lookup in the property field of the previous time step.

The property field for a subsequent time step is built in a slice-by-slice manner. Figure 1 shows the pseudo code for a single time step of the advection process. Each slice of the property field is updated by rendering a quadrilateral that represents this 2D subset of the full 3D domain. The quadrilateral is rasterized by the GPU, which leads to a filling with fragments. The dependent-texture lookup can be realized by a fragment program that computes the modified texture coordinates according to the Euler integration of the flow field. Figure 2 shows the corresponding OpenGL ARB fragment program. Note that the components of the vector field texture are stored as unsigned fixed-point numbers here and thus a bias/extend transformation is applied to achieve positive and negative values. The instructions have the following meaning: TXP and TEX stand for texture fetch operations, MAD for a multiplication and a subsequent adding operation.

```
load flow field into GPU memory
for i = 1 to max_slice
    render quad for slice i, with dependent
        texture lookup
    update slice i in new property field
end for
```

Figure 1: Pseudo code for 3D advection.

An animated visualization is built from iterative executions of the advection steps. In all computations, the property field is accessed for only two time steps: the current and the previous time step. Therefore, it is sufficient to provide two 3D textures to hold these two time steps. After each single advection computation, the roles of the two textures are exchanged, according to a so-called ping-pong scheme.

Our advection method is completely formulated in 3D and therefore takes into account any flow field. In contrast, 3D IBFV [Tel03] uses stacks of 2D textures with gathering along the z axis, and thus is restricted to a limited class of flows: The z component of the vector field has to be very small so that the absolute value of $\Delta s v_z$ is not larger than the slice distance along the z axis. Moreover, our approach makes use of built-in trilinear interpolation on 3D textures and thus allows for different resolutions of the particle and vector fields, whereas 3D IBFV supports only a bilinear resampling on 2D slices. Finally, 3D IBFV requires multiple render passes to update a single slice of the property field. The only advantage of 3D IBFV is that it does not require 3D textures and fragment programs. We think that the availability of

```
!!ARBfp1.0

# Parameters
PARAM  stepSize      = program.local[0];
PARAM  biasExtend    = {2.0, -1.0, 0.0, 0.0};
ATTRIB iTexCoord    = fragment.texcoord[0];
OUTPUT oColor       = result.color;

# Temporary variables (registers)
TEMP   velocity;
TEMP   oldPos;

# Fetch flow field
TXP velocity, iTexCoord, texture[0], 3D;
# Mapping to positive and negative values
MAD velocity, velocity, biasExtend.x, biasExtend.y;
# Compute previous position (Euler integration)
MAD oldPos, velocity, stepSize, iTexCoord;
# Dependent texture lookup:
# Advected property value from previous time step
TEX oColor, oldPos, texture[1], 3D;

END
```

Figure 2: ARB fragment program for 3D texture advection.

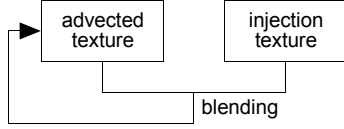


Figure 3: Basic visualization process.

this hardware features is no major issue today and will be none at all in a few years from now because GPUs with 3D texture and fragment program (or pixel shader 2) support are already in the low-cost market and thus will be ubiquitous in the near future.

5. NOISE AND DYE INJECTION

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 our case, this mapping is essentially restricted to an appropriate injection of property values. We adopt the IBFV approach, which introduces new property values at each time step, described by an injection texture. The injection mechanism of 2D IBFV [Wij02] is illustrated in Figure 3. The injection texture typically contains filtered noise to avoid aliasing artifacts. Moreover, the injection may be time-variant to introduce time-dependent visualizations even for steady flows. 2D IBFV is restricted to an affine combination of values from the previously advected texture $T_{t-\Delta t}$ and the injection texture I_t to yield the new property texture T_t :

$$T_t = (1 - \alpha)T_{t-\Delta t} + \alpha I_t \quad . \quad (4)$$

Here, α is a scalar blending parameter that is constant for the complete visualization domain. A repeated application of this α blending results in an exponential decay of the injected noise over time. The details of 2D IBFV are discussed in [Wij02].

3D IBFV [Tel03] introduces a slightly extended injection scheme that allows for space-variant scalar injection weights H_t :

$$T_t = (1 - H_t)T_{t-\Delta t} + H_t I_t \quad . \quad (5)$$

We think that the combination of a dense representation by noise injection and a user-guided exploration by injecting dye at isolated locations is very powerful because it combines both an overall view and a detailed visualization of specific features. Therefore, we further generalize the aforementioned injection schemes to allow for a unified description of both noise and dye advection. The extensions are: First, the restriction to an affine combination of the advected value and of the newly injected value is suspended and replaced by a generic combination of both; second, several materials can be advected and

blended independently. The extended blending equation is given by

$$T_t = W_t \circ T_{t-\Delta t} + V_t \circ I_t \quad , \quad (6)$$

where the two, possibly space-variant, multi-component weights W_t and V_t need not add up to one. The symbol “ \circ ” denotes a component-wise multiplication of two vector quantities, i.e., W_t , V_t , T_t , and I_t must have the same number of components. In this approach, the different components of each texel in the property field describe the density of different materials that are transported along the flow (rather than color or gray-scale values). The advantages of this extended blending scheme are: First, different materials are blended independently from each other and may therefore have different lengths for exponential decay; second, material can be added on top of existing material (e.g., additional dye), which is impossible with an affine combination as in Eqs. (4) and (5).

A unified description of both dye and noise advection is supported by the extended blending. Typically, dye is faded out only very slowly or not at all. Since new dye has to be injected at seed points, both W_t and V_t need to be one or close to one, and therefore the sum is larger than one. A possible saturation of material can be controlled by clamping. Conversely to dye visualization, the streaklines generated by noise advection tend to be much shorter. Furthermore, the overall brightness of noise-based representations should be independent of the blending weights. Therefore, noise advection typically makes use of an affine combination of advected and newly injected material.

A time-varying noise injection texture allows the user to produce animated visualization even for steady flow. As in previous work [Wij02, Lar03, Tel03], we use a—possibly filtered—noise that is periodically switched on and off with slow-in and slow-out behavior. As long as the periodicity of this time dependency is the same for the complete domain, a single, time-independent texture that holds both the noise values and the random phases is sufficient. For a specific time step, the actual noise injection value is obtained from a lookup table that describes the temporal behavior of the slow-in and slow-out. The relative phase from the noise injection texture is added to the current global time to yield the local time of that noise. This time—modulo the temporal periodicity—serves as the basis for the aforementioned table lookup. The advantage of this approach is that a single noise-injection texture is sufficient to represent a time-dependent noise injection, following [Tel03]. The approach of [Wij02] is adopted to handle noise-based visualization in the vicinity of boundaries.

Dye injection is represented by an injection texture as well. The shape of the dye emitter is stored in a voxelized form; a random phase is not needed. Unlike [Tel03], we do not use stacks of 2D textures but 3D textures to represent the noise and dye injection, and the property fields. Moreover, two different 3D textures are used to represent dye and noise injection, respectively. In this way, different sampling rates, positions, and orientations with respect to the underlying property field can be used for both textures. Typically, only very small 3D textures are needed to specify dye injection. For example, a 4^3 texture with clamping at texture edges is sufficient for a cube-shaped dye emitter; the size, position, and orientation of the emitter is controlled by choosing an appropriate affine transformation of its 3D texture coordinates. Therefore, changes in size, position, and orientation do not need a re-voxelization with a subsequent update of the texture, and thus can be handled without any performance penalty.

The noise injection texture usually covers the same domain as the property field. However, the resolution of the noise injection may differ from that of the property field. For example, a smaller resolution for noise injection—in combination with the GPU-based trilinear interpolation—leads to an efficient reduction of the maximum spatial frequency in the noise, which is often needed to avoid aliasing artifacts in the final visualization. Since 3D IBFV [Tel03] builds the complete advection process based on 2D textures, it is restricted to bilinear interpolation in each slice and does not support the third linear interpolation along the principal axis of the slices. Therefore, GPU-based noise interpolation is restricted to, and only possible along, two axes.

6. RENDERING

Direct volume rendering allows the user to view a volume data set at different depth positions simultaneously by using semi-transparency. Therefore, direct volume rendering is appropriate to display the property fields that result from a 3D advection process. We adopt a rendering approach with viewport-aligned slices that directly works on 3D textures [Cab94].

Unlike previous work on texture advection, the property fields in this paper do not directly contain color values, but densities of different materials, which are coded into the RGBA channels of the property textures (i.e., a maximum number of four materials is possible with a single texture). A separate transfer function is applied to each material during the slice-based rendering in order to obtain color values for each material. Similarly to [Had03], the different transfer functions are evaluated on a per-fragment level. Since we have a restricted scenario with a

fixed assignment of materials, a collection of different 1D textures is used.

From a visualization point of view, it is very effective to interactively choose whether and how the different materials are displayed. For example, the noise part could be rendered very faintly to give an overall context and, at the same time, the dye part could be emphasized by bright colors to focus on this detail. In another scenario, dye could be completely removed and noise could be rendered more prominently. It is extremely important that these different visualization approaches can be interactively changed in order to address the fundamental problems of occlusion and clutter. In our approach, this is easily achieved by modifying the respective transfer functions.

To further reduce the visual complexity in dense 3D flow representations, additional scalar quantities can be mapped to the color and opacity values in the final rendering. Scalar quantities can either be derived from the vector field itself (e.g., velocity magnitude) or from other parameters of the data set (e.g., pressure or temperature in a fluid flow). For example, the velocity magnitude can be used to emphasize interesting regions of high flow magnitude and fade out parts with low speed.

7. IMPLEMENTATION

Our 3D texture advection application is implemented in C++; the GPU-based advection and rendering are based on OpenGL. Except for the `ARB_fragment_program` extension, we just use standard OpenGL 1.2. Our implementation runs and was tested on both nVidia's GeForce FX and ATI's Radeon 9700/9800 GPUs.

All the visualization techniques described in Sections 4-6 are directly mapped to GPU fragment programs. The fragment program source code for the pure advection is shown in Figure 2, the pseudo code for a complete advection step in Figure 1. Except for the color tables (explained in the following paragraphs), all textures in our implementation are three-dimensional: the two property textures (for ping-pong rendering), the noise and dye injection textures, and the texture that holds the vector field. 3D textures are always created with power-of-two extents that contain the advection domain, i.e., there might be empty, unused texture regions, which are not updated during the advection process. For an unsteady flow, the vector field texture is transferred from main memory to the GPU for each time step; for a steady flow, only once. For all other textures, no transfer between main memory and GPU is required during runtime because all texture updates are done completely on the GPU. Currently, all textures have color

Table 1: Performance measurements in frames per second.

size	advection only		complete visualization		rendering only	# slices
	steady	unsteady	steady	unsteady		
64^3	106.3	50.3	25.6	20.8	31.7	130
128^3	40.9	9.8	9.3	5.4	11.9	260
256^3	10.2	1.7	2.9	1.0	3.7	540

channels with fixed-point numbers with 8 bit resolution. Higher resolution can be easily achieved by just changing the texture format to 16 bit fixed-point or 32 bit floating-point numbers.

The core fragment program (Figure 2) is extended by the following steps to incorporate the noise/dye injection and blending schemes. First, an additional texture lookup provides the noise amplitude and phase for the different noise materials to be injected at the current voxel. The phase is added to the current global time to obtain a local time. This local time, modulo the temporal periodicity of noise injection, serves as the texture coordinate for a dependent lookup in a 1D texture that describes the slow-in and slow-out of noise. After multiplication with the above noise amplitude, the actual noise injection value is obtained. Dye injection is determined by a lookup in the respective injection texture.

To implement several different noise and dye materials, the above processes are computed for each noise and dye material independently. Each material is associated with a distinct color channel. Thus, a maximum number of four materials is supported in the current implementation (if required, more materials could be realized by using several property textures). The blending mechanism (Eq. (6)) is mapped one-to-one to numerical fragment program instructions (i.e., multiplication and summation).

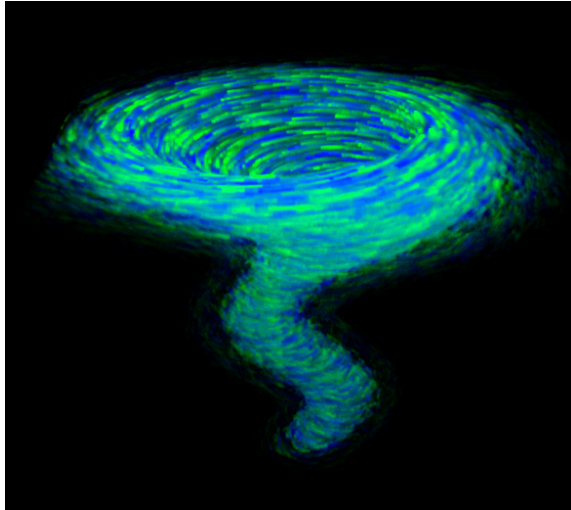
The volume visualization part adopts standard 3D texture-based volume rendering with viewport-aligned slices [Cab94]. For each material, post-classification is implemented by a dependent-texture lookup in the respective transfer function table. The resulting intermediate colors are added to obtain the final color.

8. RESULTS

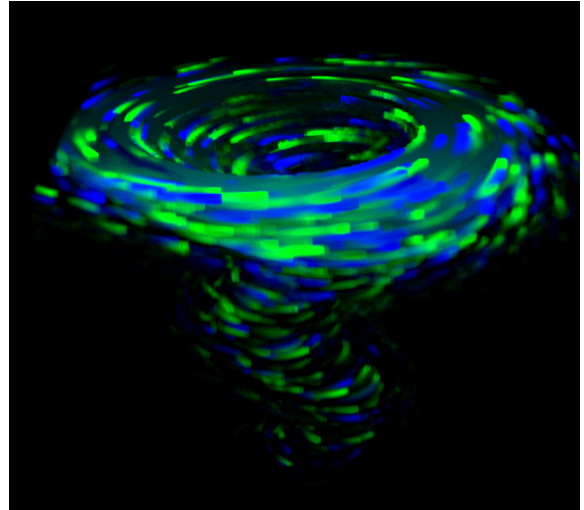
Figure 4 shows results produced by our visualization system. The underlying data set represents the behavior of a tornado (the data set is courtesy of Roger Crawfis). The size of the flow field is 128^3 , the particle textures are 256^3 , and the noise injection texture is 128^3 . Figures 4 (a)-(f) compare different visualization styles for the same viewpoint. Images (a)-(d) are rendered with velocity masking, i.e., only regions of high velocity magnitude are displayed. Image (a) employs a dense noise injection of two materials

(bluish and green) and (b) a sparse noise injection. In (c), additional red dye is injected at a user-specified position. The seed point is visualized by the intersection of three orange, axis-oriented, thin tubes. In this way, the user can easily identify the spatial position of dye injection. In (d), different lengths for blue and green noise material demonstrate that different blending weights can be used for each material. Image (e) shows a dense visualization without masking. Here, the velocity field is normalized to unit magnitude to achieve a visualization that is comparable to LIC. Due to the asymmetric, exponential filter kernel (based on iterative α blending), this image resembles oriented LIC [Weg97]. In this dense representation, interior parts of the flow are completely occluded by advected material in the front. Therefore, clipping approaches are required to view these inner regions, e.g., with a slanted clipping plane as in (f).

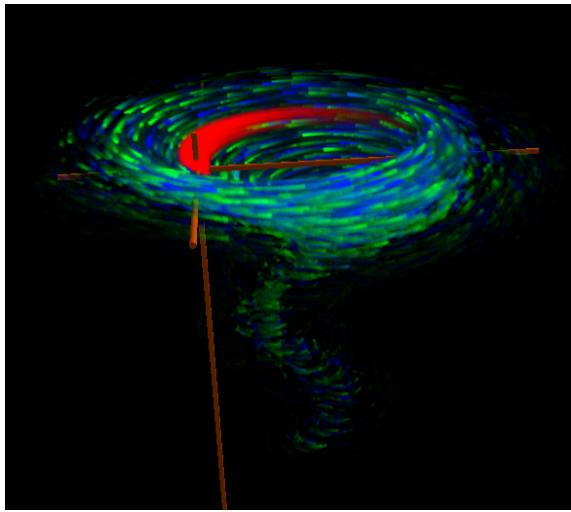
Table 1 shows performance measurements for a Windows XP machine with an ATI Radeon 9800 Pro GPU (256 MB). The sizes of the vector and property fields are stated in the first column. Two noise materials and one dye material are advected. The performance numbers show a strong dependency on the number of grid cells; the behavior could be very roughly considered linear. The viewport of size 600^2 was almost filled by the volume rendering; the number of slices is shown in Table 1. Comparing steady and unsteady visualization, it becomes clear that the download of 3D textures to the GPU is a bottleneck. Interestingly, the volume rendering part often is more time-consuming than even unsteady advection. Since the measurements for 3D IBFV [Tel03] are given for a GeForce 3 Ti 400 and for very asymmetric resolutions of property fields, it is hard to compare their numbers with ours. They achieve some 10 fps for steady flow advection with a property texture of resolution $256^2 \cdot 50$, which is roughly 20 percent of our performance for the same number of property field texels. Even when we take into account the higher rendering speed of the Radeon 9800, we still see some performance advantages of our single-pass advection scheme compared to the multi-pass approach of 3D IBFV. Moreover, our implementation already includes the effort of building a 3D texture, which is required for the subsequent volume rendering part anyway.



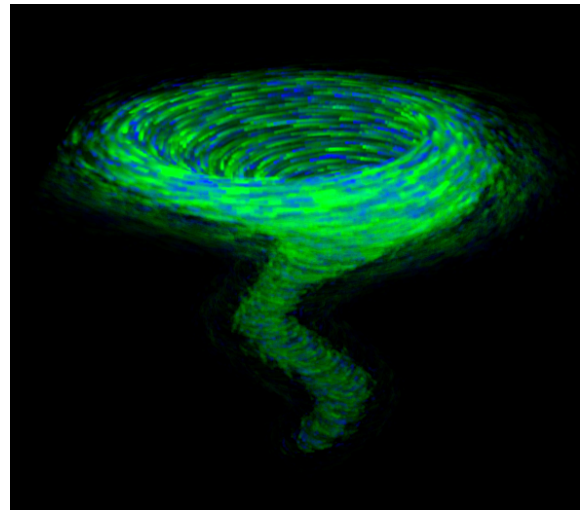
(a)



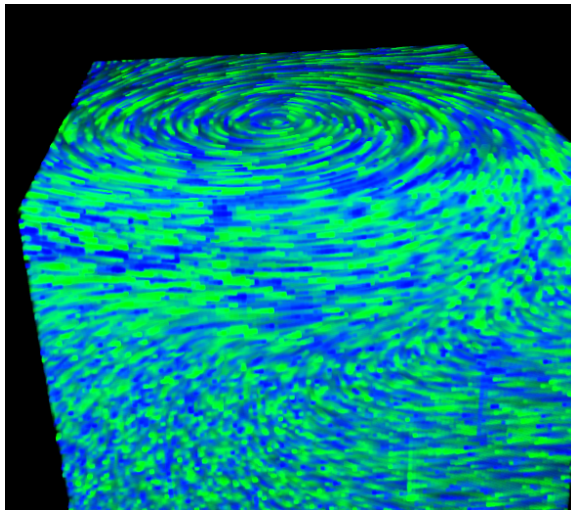
(b)



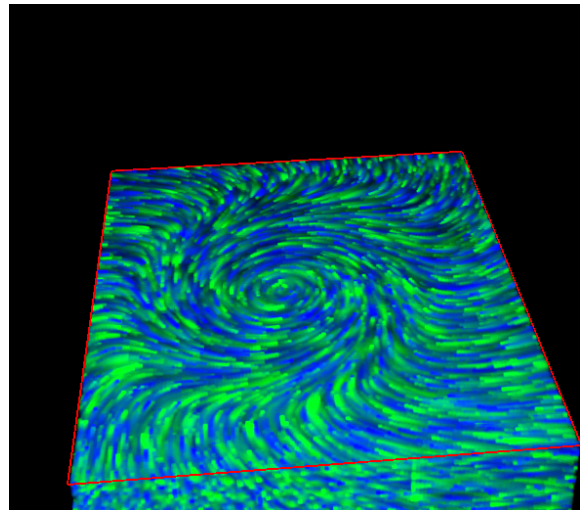
(c)



(d)



(e)



(f)

Figure 4: Visualization of a tornado dataset. Images (a)-(d) are rendered with velocity masking: (a) dense noise injection, (b) sparse noise injection, (c) additional red dye, (d) different lengths for blue and green material. Image (e) shows a dense visualization without masking; (f) the same with a slanted clipping plane.

9. CONCLUSION

We have presented an interactive texture-based system for the dense visualization of arbitrary unsteady 3D flow fields. The core of this system is an advection scheme that realizes all intrinsically three-dimensional objects, such as property and vector fields, as 3D textures. An implementation on current GPUs allows a slice of the 3D representation to be updated in a single rendering pass and therefore achieves interactive frame rates. The 3D texture approach with built-in trilinear interpolation allows the resolution, size, orientation and position of the property field, the vector field, and the injection textures to differ and to be separately controlled by specifying affine transformations of texture coordinates. Therefore, interactive changes of these parameters are possible without revoxelization. Moreover, we have proposed an enhanced blending scheme that abandons the restriction to an affine combination of advected and newly injected values. In addition, the advection and rendering schemes have been extended to transport and display different materials instead of combined colors and opacities. In this way, a unified description of a wide range of noise and dye advection approaches is possible, and parts of the visualization can be specifically faded out or emphasized.

In future work, more advanced interaction techniques for volume clipping could be incorporated to facilitate the exploration of interior details. The rendering part could be further improved by perception enhancing approaches, such as depth cueing or halos.

10. ACKNOWLEDGMENTS

We would like to thank Roger Crawfis for providing the tornado data set used in Figure 4. Thanks to Bettina Salzer for proof-reading. This project was supported by the “Landesstiftung Baden-Württemberg”.

11. REFERENCES

- [Cab93] Cabral, B. and Leedom, L.C. Imaging vector fields using line integral convolution. *Proc. ACM SIGGRAPH 93*, pp. 263-272, 1993.
- [Cab94] Cabral, B., Cam, N., and Foran, J. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. *Proc. Symp. Volume Visualization*, pp. 91-98, 1994.
- [For95] Forssell, L.K. and Cohen, S.D. Using line integral convolution for flow visualization: curvilinear grids, variable-speed animation, and unsteady flows. *IEEE Transactions on Visualization and Computer Graphics* 1, No. 2, pp. 133-141, 1995.
- [Had03] Hadwiger, M., Berger, C., and Hauser, H. High-quality two-level volume rendering of segmented data sets on consumer graphics hardware. *IEEE Visualization '03*, pp. 301-308, 2003.
- [Hau02] Hauser, H., Laramée, R.S., and Doleisch, H. State-of-the-art report 2002 in flow visualization. Technical report TR-VRVis-2002-003, VRVis Research Center, 2002.
- [Hei99] Heidrich, W., Westermann, R., Seidel, H.-P., and Ertl, T. Applications of pixel textures in visualization and realistic image synthesis. *ACM Symp. Interactive 3D Graphics*, pp. 127-134, 1999.
- [Int97] Interrante, V. and Grosch, C. Strategies for effectively visualizing 3D flow with volume LIC. *IEEE Visualization '97*, pp. 421-424, 1997.
- [Job00] Jobard, B., Erlebacher, G., and Hussaini, M.Y. Hardware-accelerated texture advection for unsteady flow visualization. *IEEE Visualization '00*, pp. 155-162, 2000.
- [Job02] Jobard, B., Erlebacher, G., and Hussaini, M.Y. Lagrangian-Eulerian advection of noise and dye textures for unsteady flow visualization. *IEEE Transactions on Visualization and Computer Graphics* 8, No. 3, pp. 211-222, 2002.
- [Max95] Max, N. and Becker, B. Flow visualization using moving textures. *Proc. ICASW/LaRC Symp. Visualizing Time-Varying Data*, pp. 77-87, 1995.
- [Lar03] Laramée, R., Jobard, B., and Hauser, H. Image space based visualization of unsteady flow on surfaces. *IEEE Visualization '03*, pp. 131-138, 2003.
- [Li03] Li, G.-S., Bordoloi, U.D., and Shen, H.-W. Chameleon: an interactive texture-based rendering framework for visualizing three-dimensional vector fields. *IEEE Visualization '03*, pp. 241-248, 2003.
- [Rez99] Rezk-Salama, C., Hastreiter, P., Teitzel, C., and Ertl, T. Interactive exploration of volume line integral convolution based on 3D-texture mapping. *IEEE Visualization '99*, pp. 233-240, 1999.
- [San00] Sanna, A., Montrucchio, B., and Montuschi, P. A survey on visualization of vector fields by texture-based methods. *Recent Res. Devel. Pattern Rec.* 1, pp. 13-27, 2000.
- [She96] Shen, H.-W., Johnson, C., and Ma, K.-L. Visualizing vector fields using line integral convolution and dye advection. *Proc. Symp. Volume Visualization*, pp. 63-70, 1996.
- [She98] Shen, H.-W., Kao, D.L. A new line integral convolution algorithm for visualizing time-varying flow fields. *IEEE Transactions on Visualization and Computer Graphics* 4, No. 2, pp. 98-108, 1998.
- [Sta95] Stalling, D. and Hege, H.-C. Fast and resolution independent line integral convolution. *Proc. ACM SIGGRAPH 95*, pp. 249-256, 1995.
- [Tel03] Telea, A. and van Wijk, J.J. 3D IBFV: hardware-accelerated 3D flow visualization. *IEEE Visualization '03*, pp. 233-240, 2003.
- [Ver99] Verma, V., Kao, D.L., and Pang, A. PLIC: Bridging the gap between streamlines and LIC. *IEEE Visualization '99*, pp. 341-348, 1999.
- [Weg97] Wegenkittl, R., Gröller, E., and Purgathofer, W. Animating flow fields: rendering of oriented line integral convolution. *Computer Animation '97*, pp. 15-21, 1997.
- [Wei01] Weiskopf, D., Hopf, M., and Ertl, T. Hardware-accelerated visualization of time-varying 2D and 3D vector fields by texture advection via programmable per-pixel operations. *Proc. VMV '01*, pp. 439-446, 2001.
- [Wei02] Weiskopf, D., Erlebacher, G., Hopf, M., and Ertl, T. Hardware-accelerated Lagrangian-Eulerian texture advection for 2D flow visualization. *Proc. VMV '02*, pp. 77-84, 2002.
- [Wij91] van Wijk, J.J. Spot noise - texture synthesis for data visualization. *Computer Graphics (Proc. ACM SIGGRAPH 91)* 25, pp. 309-318, 1991.
- [Wij02] van Wijk, J.J. Image based flow visualization. *ACM Transactions on Graphics* 21, No. 3, pp. 745-754, 2002.
- [Wij03] van Wijk, J.J. Image based flow visualization for curved surfaces. *IEEE Visualization '03*, pp. 123-130, 2003.