Rendering Implicit Flow Volumes

Daqing Xue, Caixia Zhang, Roger Crawfis*

Department of Computer Science and Engineering
The Ohio State University

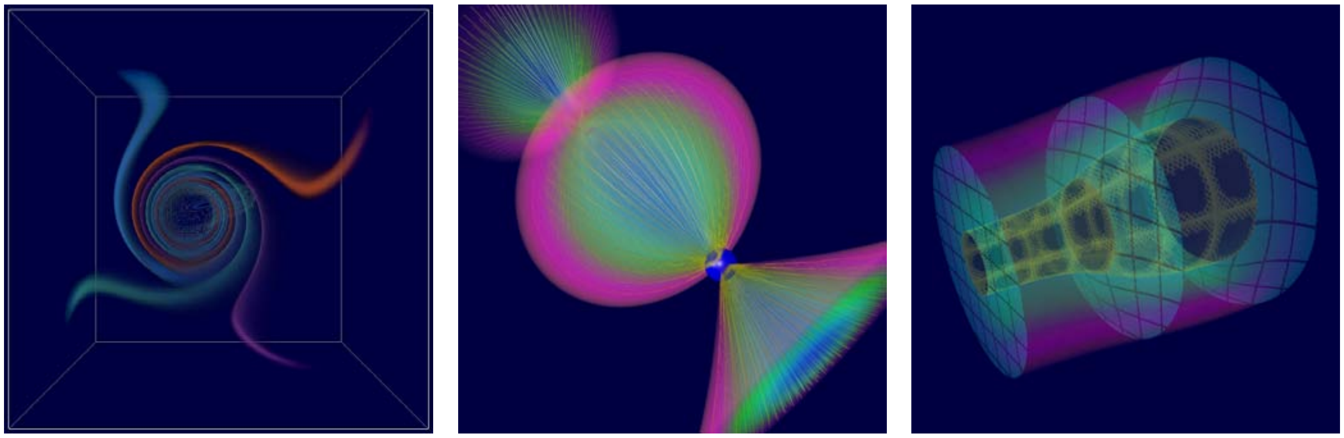


Figure 1: Visualization using implicit flow volumes. (a) Tornado dataset rendered by 3D texture mapping; (b) A coupled-charge dataset rendered using interval volumes with five interior stream surfaces textured by streamline-like texture; (c) A large flow volume in which the stream and time surfaces of the flow are textured for greater clarity.

ABSTRACT

Traditional flow volumes construct an explicit geometrical or parametrical representation from the vector field. The geometry is updated interactively and then rendered using an unstructured volume rendering technique. Unless a detailed refinement of the flow volume is specified for the interior, information inside the underlying flow volume is lost in the linear interpolation. These disadvantages can be avoided and/or alleviated using an implicit flow model. An implicit flow is a scalar field constructed such that any point in the field is associated with a termination surface using an advection operator on the flow. We present two techniques, a slice-based three-dimensional texture mapping and an interval volume segmentation coupled with a tetrahedron projection-based renderer, to render implicit stream flows. In the first method, the implicit flow representation is loaded as a 3D texture and manipulated using a dynamic texture operation that allows the flow to be investigated interactively. In our second method, a geometric flow volume is extracted from the implicit flow using a high dimensional iso-contouring or interval volume routine. This provides a very detailed flow volume or set of flow volumes that can easily change topology, while retaining accurate characteristics within the flow volume. The advantages and disadvantages of these two techniques are compared with traditional explicit flow volumes.

*{xue | zhangc | crawfis}@cse.ohio-state.edu

IEEE Visualization 2004
October 10-15, Austin, Texas, USA
0-7803-8788-0/04/$20.00 ©2004 IEEE

CR Categories and Subject Descriptors: I.3.3 [Computer Graphics]: Picture/Image Generation – Display Algorithms, Viewing Algorithms; I.3.6 [Computer Graphics]: Methodology and Techniques – Interaction techniques.

Keywords: interval volume rendering, implicit stream flow, flow visualization, graphics hardware

1. INTRODUCTION

Traditional flow volume rendering, as proposed by Max et al. [Max93], constructs an explicit geometrical representation of the separating volume using a streamline advection operator applied to the underlying vector field. The geometry is rendered using an unstructured rendering technique. Information within the flow volume boundary is usually incorrect unless a detailed refinement of the interior volume is specified. Van Wijk [Wijk1993] extracts implicit stream surfaces from a three-dimensional vector field which are rendered using traditional polygonal rendering methods. Although the flow structure can be well-tracked by the stream surfaces, the flow information inside or behind the surface is not visible. Multiple stream surfaces could be generated, but these would either occlude each other or a polygonal rendering system that can accurately support semi-transparent rendering of surfaces would be needed. We examine the problem of extending the implicit stream surfaces to that of implicit flow volumes. We develop new renderings of the implicit flow in which both the surface and the interior can be highlighted. By careful tracking and encoding of the advection parameters, we achieve highly parameterized surfaces which can be easily texture mapped for greater clarity. In this paper we present two techniques, slice-based three-dimensional texture mapping and interval volume rendering with tetrahedron projection, to render the underlying implicit flow.

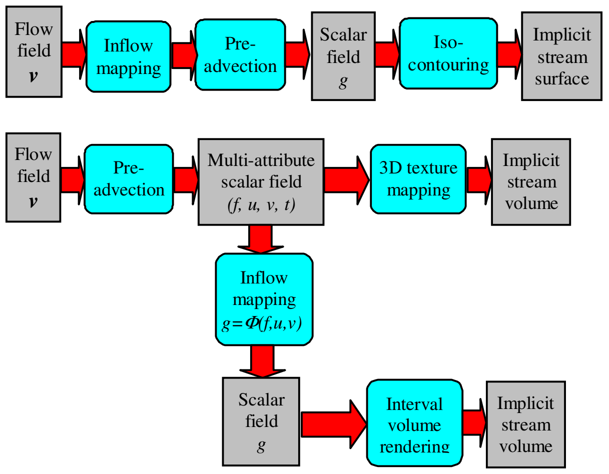


Figure 2: Visualization diagrams for van Wijk's implicit stream surfaces (top), and our implicit flow volumes (bottom).

Our overall goals in this study include the examination of more dynamic and detailed flow visualizations in three-dimensions, in particular, as it relates to advection-based flow volumes. Our criteria, thus include real-time interaction with the advection operation, support for animation through the flow, and visualization of large areas of the flow with minimal clutter.

The remainder of this paper is organized as follows. Section 2 examines related work in flow volume rendering. Section 3 provides an introduction of the implicit flow calculation, and our implicit flow volume representation. Section 4 gives an overview of the rendering of the implicit flow volume. Two rendering techniques, three-dimensional texture-based rendering and interval volume rendering are described in sections 5 and 6, respectively. In section 7, we compare these two methods with the traditional flow volume rendering technique.

2. RELATED WORK

Various techniques have been proposed to render vector fields. We focus on the work for rendering 3D vector fields. Crawfis and Max [Crawfis93] introduce a textured splat method to provide a dense global visualization of the 3D vector field. Line Integral Convolution, LIC [Cabral93], provides another dense visualization with more accurate local features. Rezk-Salama et al. [Rezk-Salama99] explore rendering volumetric LIC using 3D texture mapping hardware to examine the flow fields. Auxiliary clipping geometries are used to reveal the LIC pattern. Another dense visualization technique is the Image-Based Flow Visualization (IBFV) technique [Wijk01; Laramee03]. Telea and van Wijk [Telea03] extend the IBFV method to 3D IBFV to visualize three-dimensional flow fields. These methods provide fine-grain localization of the flow, with the aim of texture synthesis for a more global perception of the vector fields. Avoiding excessive clutter through tuning the various parameters, or restricting the range of the visualization can be difficult with these systems.

In contrast to the above dense visualization techniques, Zöckler et al. [Zöckler96] use illuminated streamlines to depict the 3D vector field. Other geometry-based methods include the stream polyhedron [Schroeder91], stream surfaces [Hultquist92], implicit stream surfaces [Wijk93], flow volumes [Max93], streamballs [Brill94] and saddle connectors [Theisel03]. Li et al. [Li03, Shen04] propose a hybrid method, where geometry is first constructed and voxelized into a coarse mesh. Each voxel in this coarse mesh is replaced with a dense 3D volume texture. The voxelized geometry, streamlines in their case, is fixed during the rendering and cannot be changed interactively.

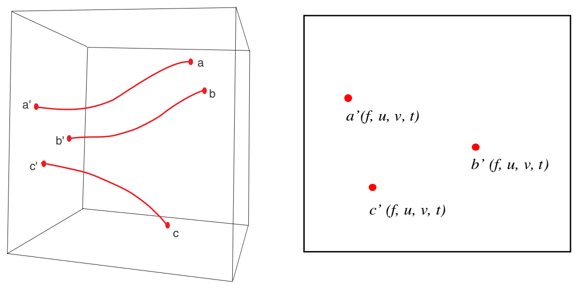


Figure 3: Backwards advection. Left: Three points, a, b and c, and their streamlines from the termination face. Right: each point (streamline) is assigned a 4-tuple, *(f,u,v,t)*, according to its advected (backwards) position on the termination face.

3. FUNCTIONAL MAPPING AND IMPLICIT FLOWS

Given a flow field (vector field), we first determine a multi-variate field in which each sample point in the field is assigned attributes from the flow. The basis of our implicit flow volumes extends from the implicit surface definition of van Wijk [Wijk93]. The visualization diagrams for van Wijk's implicit stream surface and our implicit flow volume are illustrated in Figure 2. He associates a scalar field with the inflow boundary of the computational grid, and then for each remaining grid point, he traces a streamline backwards in the flow until it reaches the boundary (ignoring critical points within the flow). The scalar field is evaluated at this location, and the grid point is assigned this scalar value. This amounts to a mapping of the 3D vector field to a 3D scalar field, $\mathbf{R}^3 \rightarrow \mathbf{R}$. An iso-contour surface is then extracted from this resulting scalar field to provide the stream surface. In addition to the obvious volume versus surface difference[1], three major differences exist between our technique and that of van Wijk. First, we either delay the specification of the scalar field on the inflow boundary, or eliminate the mapping onto a scalar field entirely. This allows us to develop new flow volumes without having to recompute the costly advection operations. Secondly, we associate several additional attributes with each sample point which allow for better user interaction, complex feature specification and enhanced surface representations. Finally, we allow for the user specification of many arbitrary boundary surfaces, which we call termination surfaces, indicating the termination of the backwards advection process. These can be used to place a termination surface around each critical point, allow for the specification of inflow and outflow boundaries [Mahrous04] or as a user-controlled segmentation of the flow. Westermann et al. [Westermann00] also use an implicit method to convert the vector field to the scalar field by storing the advection time. They render time surfaces using a level-set method by taking advantage of 3D texture mapping hardware. Their method is pretty similar to van Wijk's implicit method, but without the need for the inflow mapping.

A general functional mapping is associated with each sample point. Here we define a sample point as any location in three-

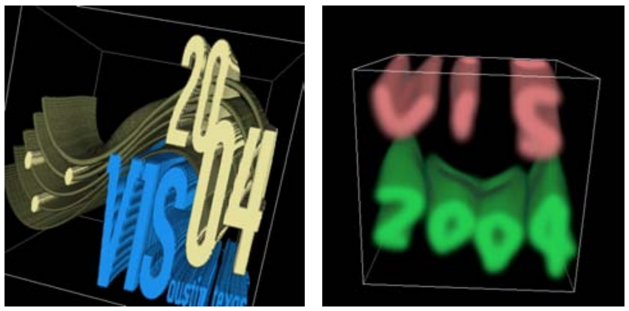
[1] Van Wijk actually points out the extension of implicit stream surfaces to stream volumes, as well as a flow of ink metaphor.

dimensional space, preserving a continuous mapping operation. In practice, we will generally associate a sample point with each vertex in either the underlying computation grid or a superimposed voxel grid. There are many attributes that can be derived or mapped onto each sample point. Local operations, such as velocity magnitude, vorticity, etc. provide simple filters. For implicit flows, we associate, at a minimum, a termination surface ID indicating which surface the backward streamline intersected first, the coordinates on the termination surface in a local coordinate frame to the surface, as well as the advection time required for the flow to reach the termination surface (backwards, or conversely, the time required for a point on the boundary to reach the sample point). This is illustrated in Figure 3. Additional attributes, such as the maximum velocity magnitude along the streamline, average density along the streamline, etc., can also be calculated and stored in this preprocessing stage. Thus, in general, we have an operation computing a mapping from $\mathbf{R}^3 \rightarrow \mathbf{R}^n$. The focus on this paper will be restricted to maintaining the four attributes mentioned above: termination surface ID, parametric position on the surface, and the advection time. Hence, for each sample point we store a 4-tuple, *(f,u,v,t)*, containing these values. This 4-tuple representation will be the basis for all of our future renderings.

4. RENDERING OF FLOW VOLUMES

Our task now is to examine methods for either rendering such a field or extracting more meaningful regions from this space. This suggests another mapping, one from the attribute space to optical properties for rendering. In the sections that follow, we will define a few such mappings. A primary criterion for such a mapping rests in providing flexible and robust mappings that provide an intuitive and simple interface. In order to better explore the flow, the user needs to be able to interactively adjust and control this mapping. This also suggests either techniques easily implemented in graphics hardware, or easily calculated and rendered.

In this paper, we present two different techniques to model and render the implicit flow volumes: slice-based 3D texture mapping, and interval volume segmentation coupled with tetrahedron volume and surface rendering. The first technique renders the implicit 4-tuple flow field directly without the inflow mapping to a scalar field, taking advantage of modern graphics hardware. With the support of the dependent texture, we can change the appearance and representation of the 3D flow volume using advanced volume shaders. The advantages of this rendering method are high interactivity and fine texture details rendered throughout the 3D flow volume. In order to illustrate the flow on stream surfaces and time surfaces, a second technique is used to incorporate semi-transparent (zero-thickness) surfaces with the flow volume. Similar to van Wijk's implicit stream surface, a flow mapping is necessary to obtain a scalar field on which an interval volume segmentation [Bhaniramka04b] is applied. Here, the flow volume is the extracted interval volume enclosed between two iso-surfaces, and the stream surfaces and time surfaces are iso-surfaces with respect to the scalar value and to the advection time. The extracted flow volume can be rendered using any tetrahedron rendering technique. In this paper, we chose the projection-based tetrahedron rendering implemented with the modern graphics hardware [Wylie02, Weiler02]. The 4-tuple can be used as texture coordinates to map textures onto the stream and time surfaces to illustrate the flow details. This interval volume rendering technique has the advantages of a flexible inflow mapping and incorporating textured stream and time surfaces with the flow volumes. This semi-transparent surface texturing is not possible with even the most advanced volume shaders. The next two sections explain these two rendering techniques in more detail.



(a) (b)

Figure 4: Two different inflow textures advected thru the volume.

5. SLICE-BASED 3D TEXTURE MAPPING

Traditional three-dimensional texture-based volume rendering takes as input a pre-shaded RGBA voxel grid. This is loaded into three-dimensional texture-memory and image-aligned proxy geometry is rasterized with three-dimensional texture coordinates specified, such that an interpolation of the texture-map values is painted across the proxy geometry. This set of proxy geometry is rendered in a back-to-front (or front-to-back) order, compositing the next slice over the partially computed image. Recent research [Westermann1998; Mueller1999; Kniss 2001] illustrate the benefits of using post-classification. Here, the original scalar field is mapped into the three-dimensional texture memory. The proxy geometry is then used to interpolate a slice of the underlying scalar field. Each interpolated value on this slice, then needs to be mapped to an appropriate RGBA value for compositing. This is supported through dependent textures in most modern systems.

Dependent textures allow both the representation and appearance of the 3D volume to change. Li et al. [Li03] use small three-dimensional dependent textures, indexed using a trace volume generated from voxelized streamlines. The dependent texture allows for colorful volumetric textures along the streamlines, and due to its small size is more easily replaced, allowing for animation effects along the streamlines. We extend their concept of a trace volume to an implicit flow volume in which each voxel is an *n*-tuple as defined in section 3. For interactive user-controlled exploration, the system must support re-painting the dependent texture in real time. In order to accomplish this, the number of texels being updated in the dependent texture needs to be limited. Our underlying functional mapping is a 4-tuple mapped to an RGBA normalized format. OpenGL supports up to four texture coordinates, but does not support four-dimensional textures, so a 4-component dependent texture is only theoretically possible. Besides, changing every value in a 4-dimensional texture becomes prohibitively expensive as the resolution of the dependent texture grows. Three-dimensional dependent textures are supported, but if our goal is to allow the user to control the appearance of the flow throughout the entire volume, a dependent texture of at least the same size as the underlying voxel grid would be required. This differs from Li et al. [Li03], in that our goal is to change the underlying trace volume dynamically. Updating the entire volume in real-time is not feasible for large volumes. This also would greatly reduce the amount of texture memory available for the implicit flow volume.

Our focus instead, has been on reducing the mapping down to a 2D parameterization of a single termination surface. Our approach, allows the user to paint the dependent texture colors and opacities directly on this surface [Hanrahan90]. We call this dependent texture, the *inflow texture* in the subsequent sections, as it dictates the paint that is carried from the termination surface into the flow. The next few sections provide details on a few of these choices, as well as additional techniques which extend this parameterization to utilize more attributes from the underlying 4-tuple in the implicit flow representation.

5.1 User-Controlled Painting

Without loss of generality, we consider an implicit flow volume in which only one termination surface exists, i.e., the backward advections of all sampling points terminate on this surface. The simplest such surface would be a single face of the bounding box for the flow field. A dependent texture mapped to this face is used as our lookup table. The *(u, v)* in the 4-tuple, *(f, u, v, t)*, is employed to index into the dependent texture, producing the current fragment's color and opacity. By changing the alpha mask, we can dynamically change the three-dimensional representation of the flow. With our user interface, we can brush the inflow texture to get arbitrary representations of the flow. Figure 4b shows an image in which the user hand-painted *vis 2004* on the inflow texture on one face of a bounding box. In addition to hand painting, the user can import any image for use as the inflow texture. Figure 4a, has the IEEE Visualization 2004 conference logo used as an opacity and color texture. The painting modes are supported for adding paint to the inflow texture. The previous texture can be cleared and new paint added from the user's current brush, providing a moving flow volume. The previous texture can simply be added to, building out regions of interest in the flow. An eraser (a brush that reduces the opacity) is also supported for refining these regions of interest. Finally, the previous texture can be *faded* out over time by first reducing its opacity and then adding new paint under the user's control. This provides a motion blur of the flow volume as it moves through the field.

More complex termination surfaces can easily be supported, provided a simple parameterization exists. This is in general, a hard problem. In addition to flat planes, we currently support spherical and cylindrical termination surfaces (useful for bounding a neighborhood of a source), and a rectangular box termination surface. The parameterization of the box is supported through the use of OpenGL's cube-maps. This allows for six independent termination surfaces, onto which the user can paint an inflow texture. Figure 1a shows an image generated with a cube map for inflow texture on the tornado dataset.

5.2 Inflow Texture Animation

Max, Becker and Crawfis [Max93] included a simple modulation of the opacity as a function of the advection time for their flow volumes. By phase shifting this modulation function, they were able to animate smoke puffs along their flow volumes. In our implicit flow volume representation, the advection time has been encoded into the 4-tuple for each sampling point. We define a one-dimensional opacity table containing an opacity modulation. For animation, we phase-shift this opacity modulation for each time step. The advection time information of the sample point is used to index into the opacity table. This is combined with the dependent textures for the inflow texture, using multi-textures. This produces a puff-like motion in the flow. By adding another dependent texture, we can also encode the *age* of the paint on the inflow texture. Adding this age to the advection time releases the paint from the inflow texture through the flow field. We can also automate this to provide flow representations using particles, animated streamlines and propagating time fronts (see the supplemental material).

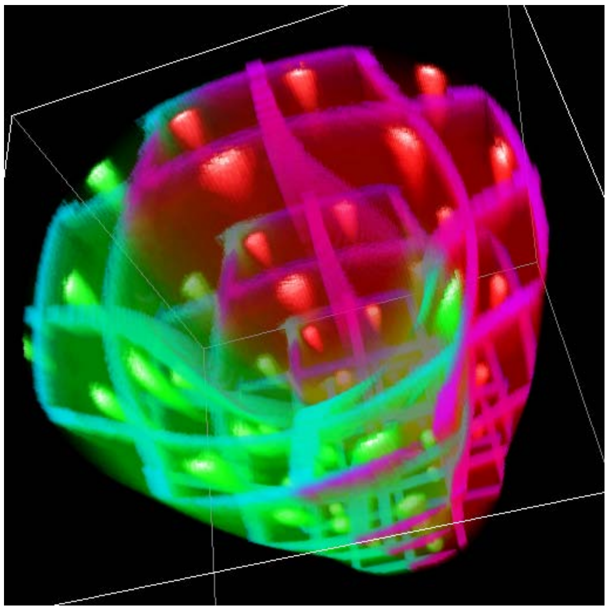


Figure 5: Complex cross-section for the inflow with dual-texture support.

5.3 Dual Inflow Textures

Periodic dye injection can help understand the interior structure and highlight local features in the flow. Shen et al. [Shen96] use a "smeared" noise texture to simulate dye. Instead, we use a multi-texture technique. In addition to the user defined inflow texture, we create a separate dual-inflow texture. The support for dual inflow textures, allows for a separate high-frequency texture without requiring the user to painstakingly paint in such details. A high-resolution dependent texture can be used for this purpose. This does not require a large amount of texture space, provided the high-frequency texture is periodic and tile-able. Figure 5, embeds a dual inflow texture with a regular grid pattern and a Poisson disc pattern. Each inflow texture can be animated separately. This allows one to model a periodic dripping or injection of colored dye into the flow volume. The underlying flow volume retains its global characteristics and multi-colored sub-flows are passed through the flow volume. For the images in Figure 6, an initial inflow texture, as shown in Figure 6a and a dual inflow texture as shown in Figure 6b were used to generate the image in Figure 6d. The image in Figure 6c shows the flow volume without a high-frequency detail texture. Here, two dependent textures are used, both indexed similarly, with a multi-texturing operation that replaces the paint from the inflow texture with the dual inflow texture.

6. INTERVAL VOLUME RENDERING

Our second technique for volume rendering the implicit flow representation utilizes interval volumes and projected tetrahedron rendering. To use this technique, a mapping from our 4-tuples into a single scalar value is needed. The following equation describes an implicit flow volume using this technique:

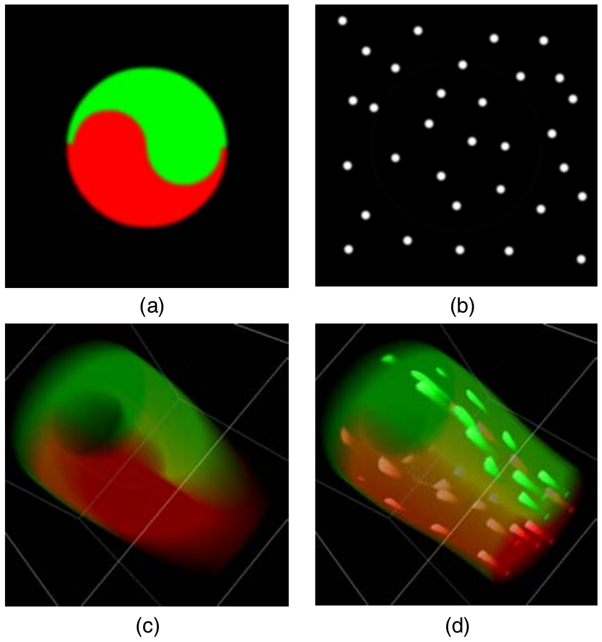


Figure 6: a) The inflow texture specified by the user. b) A particle distribution. c) The result from the inflow texture only. d) The result obtained by combing the inflow texture and texture b).

$$I_p = \{ p \mid \alpha \le \Phi(f(p), u(p), v(p), t(p)) \le \beta \} \quad (1)$$

This equation determines the continuous set of points, *p*, which lie on or inside the implicit flow volume, I_p, specified as an interval of the scalar field mapping. Here, *f* is the termination surface ID of the advection point, *u* and *v* are the normalized coordinates of the point on the termination surface, and *t* is the advection time with which the backwards advection for the sampling point reaches this surface. The function, Φ, provides a mapping of the implicit flow field to a single scalar distribution over the field, and α and β are two iso-values specifying the boundaries of the flow surface. We define our scalar field mapping as in [Wijk93], using a similar methodology from Section 5. Many such mappings for the same dataset are thus possible, each having a different iso-value distribution. We maintain the association between the sampling points and their 4-tuple, *(f, u, v, t)* from Section 3. This information will be used in later sections to enhance the flow volume appearance.

The scalar field distribution is used to extract the flow geometry using a high dimensional iso-contouring routine [Bhaniramka2000] [Bhaniramka2004a]. Interval volumes can be calculated efficiently, by adding an extra dimension to the underlying polyhedra, and duplicating each vertex in this dimension with a scalar value from the original vertex, shifted according to the interval width (see [Bhaniramka2004b]). The resulting tetrahedra are then rendered using a hardware-accelerated Projected Tetrahedra renderer [Wylie02][Weiler02]. The tetrahedra are sorted using the MPVONC algorithm [Williams92].

One of the advantages of this technique is that actual geometry is extracted, efficiently enough to allow for the surface to be interactively changed. Furthermore, individual stream surface and/or time surface renderings can be seamlessly integrated into the volume rendering of the flow volume. Since these surfaces are embedded in the 3D interval volume, typical problems in integrating polygonal geometry with volume rendering are

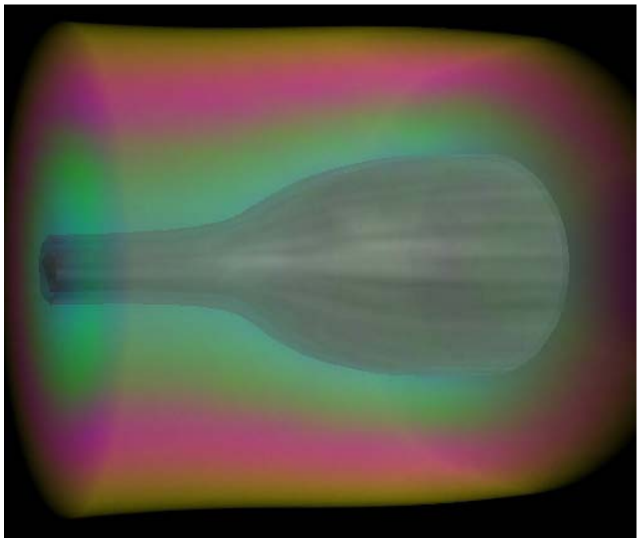


Figure 7: A stream surface inside the flow is textured using a 3D LIC texture.

avoided. Moreover, these surfaces can be semi-transparent, with many nested stream-surfaces composited together in the final rendering. The surfaces can also be easily texture-mapped, including opacity textures, to present additional depth cues, animation, surface reflections, and general clarity [Gorla03].

6.1 User Controlled Stream Volumes

Again, one of our primary concerns is ensuring an intuitive and easy-to-use interface for the user specification of the flow volumes. Interaction with traditional flow volumes was achieved using direct manipulation of a 3D widget. The shape of the widget was restricted to a small rectangle. Although additional shapes could be supported, no direct user control or specification was provided. We have developed two solutions to this problem. The user can select from a set of predefined geometric shapes, and then translate and scale these across the termination surface. A continuous scalar field is then derived using these curves as implicit basis functions. Alternatively, we can employ a methodology similar to that used in section 5.1, where the user directly paints on the termination surface. Here, rather than the user painting in RGBA mode, the inflow texture is a grey-scale image. A separate user control, allows for selecting a set of grey-values or iso-contours in the grey-scale texture. Interval volumes are extracted from each iso-value pair. This allows the user to loosely paint and then refine the initial stream surface contours.

6.2 Surface Shading and Textures

The interval volumes algorithm when applied to this implicit flow field, produces a geometric sub-volume, where the boundaries correspond to stream surfaces. Many additional shape cues arise from surface shading [Todd97]. In this section we examine techniques to incorporate surface (or curve) shading to aid in the volume visualization. Direct surface rendering is certainly possible, but occludes the interior of the flow we are trying to visualize, as well as portions of the stream surface which wrap behind the front most surface.

Since the iso-value dictates the stream surface boundary, we search the resulting tetrahedral mesh for any faces having all three vertex values equal to the iso-value. These faces are tagged as boundary faces and an additional surface rendering is performed

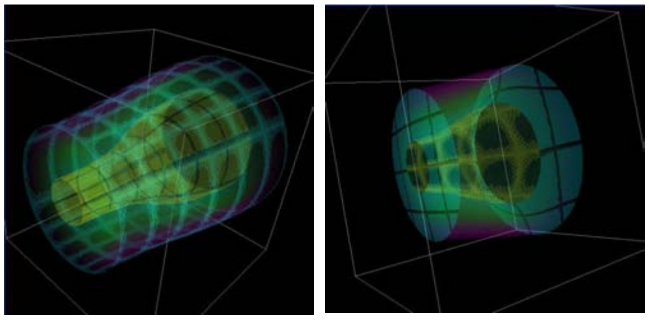


Figure 8: The textured stream and time surfaces. a) Multiple stream surfaces are mapped with different texture. b) Stream and time surfaces are both textured and the flow is clipped by the time steps.

during the projected tetrahedron rendering. To embed additional surfaces within the stream volume, we simply give the interval volume algorithm additional iso-values at which to segment the volume. During rendering, we first render each tetrahedron. If a face of the tetrahedron belongs to a boundary for which the user has chosen surface shading, we then render that face. We use back-face culling to ensure the polygonal face lies on top of the tetrahedron (in a back-to-front rendering order). An interior iso-value will have two adjacent tetrahedra which share a face, one of which will be a front-face and the other a back-face. This is our normal rendering operation.

6.2.1 Textured Stream Surface Boundaries

Explicit stream surfaces or flow volumes are easily parameterized. One can think of these surfaces as swept surfaces resulting from some initial curve. The linear approximation of the curve has a specific ordering of the vertices in order to construct the triangulated surface (see [Hultquist92]). Texture parameterization or mapping for implicit surfaces is a difficult problem [Turk01]. The simplest texture parameterization is to use three-dimensional textures and the vertex locations of the stream surface for the texture coordinates. We use a 3D LIC to demonstrate non-parametrical space texture mapping. A 3D LIC texture for the underlying flow field is first generated. The coordinates of the surfaces' vertices are indexed into the 3D LIC texture. These stream and time surfaces function similar to the clip planes in [Rezk-Salama99]. Figure 7 shows a stream surface mapped with an underlying 3D LIC texture.

To support 2D texture mapping for our implicit stream surfaces, we have a slightly easier problem, as opposed to generic implicit surface mapping. We desire a parameterization where the curve length of the initial shape or iso-contour is mapped to one texture coordinate, and the advection time is mapped to the other texture coordinate. The latter mapping is embedded in our implicit flow volume representation and falls out easily. The former is more difficult, but extracting curves from 2D images or scalar fields is well-studied. We use a contour search algorithm, which finds a point on the contour and then visits neighboring cells on the contour [Itoh95]. Figure 8a shows stream surfaces mapped with different textures. Notice the nested stream surfaces in this figure. In figure 1c and figure 8b, both the stream surface and time front surface are textured to convey more flow information. The flow in figure 8b is also clipped by the time steps.

An interesting application of texture, adds streamlines to the stream surface. Constant colored streamlines are achieved using a

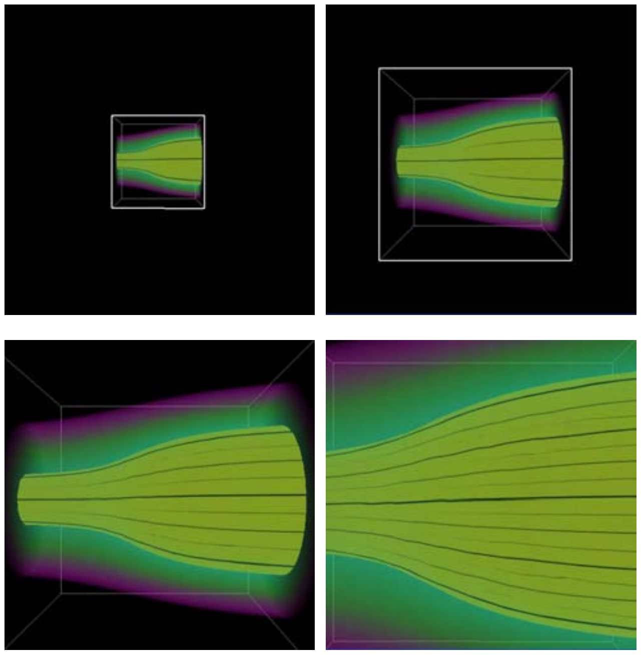


Figure 9: Streamlines using a non-averaged 1D Mip-map texture. As we zoon in, more streamlines are automatically added.

1D texture, whose coordinates are mapped to the 2D iso-contour curve length as describe above. A simple texture, which modulates the surface color, is used to encode black stripes (a square wave). We use a non-averaged mip-map technique that adds more and more streamlines as we zoom into the scene. This is constructed by halving the square wave's frequency for each higher mip-map level. With interpolation between the mip-map levels enabled, the streamlines gracefully fade in and out as the local projected stream surface area changes. Several snapshots, taken as we zoom in on the stream surface, are shown in Figure 9. Figure 1b shows the streamlines from a source to a sink inside the flow. Five stream surfaces are embedded within the flow volume.

6.2.2 Texture Animation

The high-frequency texture animation used in Section 5.3 provides powerful visual cues into the flow. Animation is a natural representation for flow fields. The dynamic texturing used in Section 5.3 will not work with projected tetrahedron, due to the multi-valued function at the *thick* vertex of the projection. The stream surface parameterization however provides a rich avenue for exploring animation. Animation on this surface is easily achieved by translating the advection time texture coordinate. More complex animations can also be achieved by cycling through a set of predefined textures. The supplemental material illustrates the integration of stream volumes with many different textures animated across the embedded stream surfaces.

7. RESULTS AND COMPARISONS

In this paper, we have described two techniques of implicit flow volume rendering. In this section, we compare the advantages and disadvantages of these two techniques with respect to explicit flow volumes [Max93]. Table 1 summarizes our comparison between these three techniques.

The implicit flow is generated by pre-advecting the flow field and storing the advection information for each voxel in the implicit flow. When we subsequently construct a stream volume, no integrator is required to compute streamlines through the flow field. Since this is a pre-computation, care can be taken to ensure accurate streamline advection. We use an adaptive fourth-order Runga-Kutta algorithm. The 3D texture mapping method renders stream volumes using a dependent texture, while the interval volume technique extracts a stream volume using a high dimensional iso-contouring routine. Explicit flow volumes are constructed using an advection algorithm during run-time. The 3D texture mapping has an advantage when the inflow boundary is changed, as it does not require any re-computation. The other two techniques need to re-compute their flow volumes, one through advection the other through iso-contouring, both of which can be costly operations. In our experiments, the 3D texture mapping can achieve roughly 10 FPS for 128^3 implicit flow dataset. The interval volume rendering offers about 3.5 FPS for the interval volume with 346.5K tetrahedra. More performance results about interval volume rendering can be found in [Bhaniramka04b]. All experiments are performed on a PC with a QuadroFX 3000 graphics card and a Pentium IV 3.2 GHz processor. Although, it should be pointed out that all of these techniques run fairly interactive for the datasets we have tested. A true performance comparison is not provided, due to the many parameters each technique requires for the specification. For any given technique, we can find a case where it would be the fastest, or the slowest. Nevertheless, there are three key differentiating factors that we wish to highlight among these flow volume techniques.

7.1 Volumetric Texture or Detail

In the traditional method, a cross section is specified by a low-resolution polygon. The quality of the cross section and the flow boundary is limited by user specification. Typically, the quality is poor. Furthermore, the distribution of any optical properties across the cross section is ill-specified. In order to allow for changes of the optical properties across the initial smoke generator, a subdivision of the cross section (and hence the resulting explicit flow volume) is required. Most explicit flow volume renderings utilize a constant color and extinction coefficient.

For the implicit methods, the cross section is specified using a general inflow texture. The complexity of the cross section and the resulting flow boundary is thus determined by the resolution of the dependent texture for the 3D texture mapping technique, and by the underlying voxel grid of the implicit volume for the interval volume technique, respectively. An extremely high virtual resolution is possible with the dependent textures. No assumptions about the underlying volume rendering model are made in our system. In fact, an arbitrary fragment program can be used to compute the volume rendering. This allows for volumetric straw textures, etc.

The implicit stream volume includes flow properties at all sampling points within the flow. This flow detail information is stored in the implicit flow volume representation and can be used to modify the color and opacity of the tetrahedral rendering. Embedding exotic volumetric textures into projected tetrahedron is an interesting future research topic.

7.2 Stream Surface Texture or Detail

One advantage of the interval volume method is that the stream surfaces and the time surfaces are modeled during the interval volume extraction process without extra computation cost. Texture mapping and surface shading are then applied on these surfaces to highlight internal features and provide a pleasing and more informative flow visualization. Although the traditional flow volume method did not add these surfaces, they are easily incorporated, since the representation is a true parametric representation. While algorithms exist to display contour surfaces using 3D texture-mapping based volume renderers [Engel01], applying parametric textures to these surfaces is an unsolved problem. Initial experiments are very prone to aliasing and blur. The 3D texture mapping technique cannot generate these texured surfaces.

7.3 Rendering Complexity

For the rasterization and rendering, the traditional flow volume method renders only the flow area, and the rendering performance depends on the number of tetrahedra. This is also true for the interval volume rendering technique. But the interval volume method requires the iso-contouring over the entire volume for the interval volume extraction. Furthermore, there may be many internal teatrahedra within the implicit flow volume. This is advantageous for multi-colored flow volumes, but is extra computation and rasterization for constant colored flow volumes. Both of these techniques of course, require sorting of the tetrahedra and a special rendering algorithm. The 3D texture mapping method is different from the above two methods in these aspects. It rasterizes the entire volume and the rendering performance is strictly dependent on the number of the voxels (and the complexity of the volume shader). Volume rendering the unstructured grids generated from the explicit flow volumes or interval volumes techniques can be much more expensive than volume rendering using 3D texture mapping. However, if the flow volume area is kept small, the reduced rasterization operations can be an advantage. An area of future research is to use the geometry from the interval volumes to compute and slice bounding hulls of the resulting flow volumes for the 3D slice rasterizer.

8. ACKNOWLEDGEMENTS

The authors would like to thank the reviewers for their time and hard work. This research is supported in part by NSF grant ACR 0222903.

References:

BHANIRAMKA, P., R. WENGER, AND R. CRAWFIS, *Isosurfacing In Higher Dimensions*, in *Proc. of IEEE Visualization 2000*, IEEE CS Press, 15-22.

BHANIRAMKA, P., R. WENGER AND R. CRAWFIS, *Isosurface Construction in Any Dimension Using Convex Hulls*, IEEE Transactions on Visualization and Computer Graphics, Vol. 10, No. 2 (March 2004), pp. 130-141.

BHANIRAMKA, P., C. ZHANG, D. XUE, R. CRAWFIS AND R. WENGER, *Volume Interval Segmentation and Rendering,* to appear in Volume Visualization and Graphics Symposium 2004.

BRILL, M., H. HAGEN, H.-C. RODRIAN, W. DJATSCHIN, S. KLIMENKO, *Streamball Techniques for Flow Visualization,* In *Proc. of IEEE Visualization '94*, IEEE CS Press, 225-231.

CABRAL, B., AND LEEDOM, C. 1993. *Imaging vector fields using line integral convolution.* In *Proceedings of SIGGRAPH '93*, ACM SIGGRAPH, 263.270.

CRAWFIS, R., AND MAX, N. 1993. Texture splats for 3d vector and scalar visualization. In *Proceedings Visualization '93*, IEEE CS Press, 261.266.

ENGEL, K., M. KRAUS, T. ERTL, High-quality pre-integrated volume rendering using hardware-accelerated pixel shading, Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop On Graphics Hardware, pp. 9-16, 2001, Los Angeles, California

FUJISHIRO, I., Y. MAEDA, AND H. SATO, *Interval volume: a solid fitting technique for volumetric data display and analysis,* in IEEE Visualization '95, Atlanta, GA, 1995.

GORLA, G. V. INTERRANTE, G. SHAPIRO, *Texture synthesis for 3D Shape Representation,* IEEE Transactions on Visualization and Computer Graphics, Vol. 9, No. 4 (Oct-Dec. 2003), pp. 217-242.

HANRAHAN, P., P. HAEBERLI, *Direct WYSIWYG painting and Texturing on 3D Shapes*, Computer Graphics (SIGGRAPH 90), Vol 24, pp. 215-223.

HULTQUIST, J. 1992. Constructing stream surfaces in steady 3d vector fields. In *Proc. IEEE Visualization '92*, IEEE CS Press, 171-178.

ITOH, T., K. KOYAMADA, *Automatic isosurface propagation using an extrema graph and sorted boundary cells,* IEEE Transactions on Visualization and Computer Graphics, Vol. 1, No. 4 (Dec. 1995), pp. 319-327.

KNISS, J., KINDLMANN., G., AND HANSEN., C. 2001. Interactive Volume Rendering Using Multi-Dimensional Transfer Functions and Direct Manipulation Widgets. In *Proc. IEEE Visualization '01*, IEEE CS Press, 241-248.

LI, G.-S., BORDOLOI, U., AND SHEN, H.-W., 2003. Chameleon: An Interactive Texture Based Rendering Framework for Visualizing Three-Dimensional Vector Fields. In *Proc. IEEE Visualization '03*, IEEE CS Press, 241-248.

LARAMEE, R., JOBARD, B., AND HAUSER, H., 2003. Image Space Based Visualization of Unsteady Flow on Surfaces. In *Proc. IEEE Visualization '03*, IEEE CS Press, 131-138.

MAHROUS, K., J. BENNETT, G. SCHEUERMANN, B. HAMANN, K. JOY, *Topological Segmentation in Three-Dimensional Vector Fields*, IEEE Transactions on Visualization and Computer Graphics, Vol. 10, No. 2 (March 2004), pp. 198-205.

MAX, N., BECKER, B., AND CRAWFIS, R., 1993. Flow Volumes For Interactive Vector Field Visualization, In *Proc. of IEEE Visualization '93*, IEEE CS Press, 19-24.

MUELLER, K., MOELLER, T., AND CRAWFIS, R. 1999. Splatting without the Blur. In *Proc of IEEE Visualization '99*, IEEE CS Press, 363-370.

NIELSON, G., J. SUNG, *Interval Volume Tetrahedrization,* In *Proc. of IEEE Visualization '97.* IEEE CS Press, 221-228.

REZK-SALAMA, C., HASTREITER, P., TEITZEL, C., AND ERTL, T., 1999. Interactive Exploration of Volume Line Integral Convolution Based on 3D-Texture Mapping. In *Proc. of IEEE Visualization '99.* IEEE CS Press, 233-240.

SCHROEDER, W. J., C. R. VOLPE, W. E. LORENSEN, 1991. *The Stream Polygon: A Technique for 3D Vector Field Visualization.* In *Proc. IEEE Visualization '91*, IEEE CS Press, 126-132.

SHEN, H.-W., JOHNSON, C., AND MA, K.-L. 1996. Visualizing Vector Fields Using Line Integral Convolution and Dye Advection. 1996 *Symposium on Volume Visualization,* IEEE Computer Society and ACM SIGGRAPH, California.

SHEN, H.-W., LI, G.-S., BORDOLOI, U. 2004. Interactive Visualization of Three-Dimensional Vector Fields with Flexible Appearance Control, IEEE Transactions on Visualization and Computer Graphics, Vol. 10, No. 4 (July 2004), pp. 434-445.

TELEA, A. J. VAN WIJK. 2003. *3D IBFV: Hardware-Accelerated 3D Flow Visualization* In Proceedings. IEEE Visualization '03, IEEE CS Press, 225-232.

THEISEL, H., WEINKAUF, T., HEGE, H.-C., AND SEIDEL, H.-P. 2003. *Saddle Connectors – An Approach to Visualize the Topological Skeleton of Complex 3D Vector Fields.* In Proceedings IEEE Visualization '03, IEEE CS Press, 225-232.

TODD, J., F. NORMAN, J. KOENDERINK, A. KAPPERS, *Effects of Texture, Illumination, and Surface Reflectance on Stereoscopic Shape Perception,* Perception, 26, pp. 807-822, 1997.

TURK, G., *Texture Synthesis on Surfaces*, Computer Graphics Proceedings (SIGGRAPH 2001), pp. 347-354.

VAN WIJK, J.J.., 1993. Implicit Stream Surfaces. In *Proc. of IEEE Visualization'93*. IEEE CS Press, 245-252.

VAN WIJK, J. J., 2001. Image based flow visualization. *Computer Graphics (Proc. SIGGRAPH '01),* ACM Press, 263-279.

WESTERMANN, R., AND ERTL, T. 1998. Efficiently Using Graphics Hardware in Volume Rendering Applications. In *Proc. of SIGGRAPH ' 98*, ACM Press, 169-177.

WESTERMANN, R., JOHNSON, C., AND ERTL, T. 2000. A Level-Set Method for Flow Visualization. In *Proc. of IEEE Visualization 2000*, IEEE CS Press, 147-154.

WEILER, M., M. KRAUS, T. ERTL, *Hardware-Based View-Independent Cell Projection,* in Symposium on Volume Visualization, 2002, Boston, MA., pp. 13-22.

WILLIAMS, P. *Visibility Ordering of Meshed Polyhedra*, in ACM Transactions on Graphics, 11 (4), 103-126, April 1992.

WYLIE, B., K. MORELAND, L. A. FISK, AND P. CROSSNO, *Tetrahedral projection using Vertex Shaders*, in Symposium on Volume Visualization, 2002, Boston, MA., pp. 7-12.

ZÖCKLER, M., STALLING, D., AND HEGE, H.-C. 1996. Interactive visualization of 3d-vector fields using illuminated stream lines. In *Proc. of Visualization '96*, IEEE CS Press, 107-114.

	Traditional Flow Volume	**Implicit Stream Volume**	
		3D texture mapping	**Interval volume rendering**
Requires pre-processing	No	Yes	Yes
Advection	Advection during the volume construction	Pre-advection	Pre-advection
Flow volume construction	Through advection	Using dependent textures	Using high dimensional iso-contouring routine
Representation	Explicit	Implicit	Explicit (reconstructed from the implicit flow volume)
Initial Starting Location	Anywhere	User-defined Termination surfaces (pre-computed)	User-defined Termination surfaces (pre-computed)
Stream surface / time surface	Easily added	No	Yes
Rasterization / rendering range	Render only the flow area	Rasterize the entire volume	Iso-contouring the entire volume, render only the flow area
Requires recomputation	Yes	No	Yes
Non-regular grids	Easily supported	Requires voxelization	Easily supported
Cross section specification	Polygon	Per-pixel mask function	Mask function on the termination surface(s)
Cross section quality	Limited by user specification, typically poor	Resolution of the dependent texture	Dependent on voxelization
Boundary quality	Dependent on polygon	Dependent on dependent texture	Dependent on voxelization
Details / correctness	Without mesh refinement, misses details in flow.	More accurate	More accurate
Rendering performance	Dependent on the number of tetrahedra	Dependent on voxel grid size	Dependent on the number of tetrahedra
Volume size	Arbitrarily large volume	Limited by the texture memory of the display card	Arbitrarily large volume, limited by system memory
Source, sink critical points	Fine	Requires critical point detection	Requires critical point detection

Table 1: Comparison of the flow volume visualization techniques.