Real-Time Techniques for 3D Flow Visualization

Anton Fuhrmann and Eduard Gröller

Institute of Computer Graphics, Vienna University of Technology [*]

Abstract

Visualization of three dimensional flow has to overcome a lot of problems to be effective. Among them are occlusion of distant details, lack of directional and depth hints and cluttering. In this paper we present methods which address these problems for real-time graphic representations applicable in virtual environments. We use animated, opacity-mapped streamlines as visualization icon for 3D flow visualization. We present a texture mapping technique to keep the level of texture detail along a streamline nearly constant even when the velocity of the flow varies considerably. An algorithm is described which distributes the dashtubes evenly in space. We apply magic lenses and magic boxes as interaction techniques for investigating densly filled areas without overwhelming the observer with visual detail. Implementation details of these methods and their integration in our virtual environment conclude the paper.

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

1 Introduction and Motivation

Many visualization techniques for two dimensional flows have already been investigated in detail [10]. Visualization of 3D flow phenomena, however, tend to produce complex images with often heavily overlapping geometry. Occlusion, ambiguities in depth and orientation of flow strain the viewers abilities to interpret the visualized data.

The main point of this paper is to show how realtime graphics in a virtual environment can be used to overcome some of this problems. Stereo cues, interactive and intuitive changes of the viewpoint and the feeling of immersion allow users to get a better impression of the structure of the 3D flow in an virtual environment than on a desktop system.

We present selected visualization and interaction techniques which in combination enable the user to rapidly explore complex 3D vector fields.

Fast texture-based visualization techniques which utilize the graphics hardware to get realtime performance are applied to streamlines. A new parametrization scheme allows a direct mapping of a wide range of flow velocity to texture velocity without loss of detail. These techniques together with interactive 3D focussing enable the user to quickly identify and explore areas of interest. The focussed volume is selected with magic lenses and magic boxes which also use mainly hardware accelerated features. Animation is realized in the texture-coordinate domain with moving opacity maps. This reduces occlusion and cluttering by simulating particle traces. An automatic streamline placement algorithm is extended into the third dimension to generate an even distribution of streamlines in the virtual environment.

[*]Institute of Computer Graphics, Vienna University of Technology, Karlsplatz 13/186/2, A-1040 Vienna, Austria email:{fuhrmann, groeller}@cg.tuwien.ac.at

2 Related Work

Several techniques for the visualization of 2D and 3D flows inspired this work. Some examples of texture based techniques for the visualization of 2D flows are [4], [19], [13].

FROLIC [21] is a variation of LIC (Line Integral Convolution) [4]. LIC uses sparse textures and an asymmetric convolution kernel to encode the orientation of the flow in still images. Costly convolution operations as done in LIC are replaced in FROLIC by approximating a streamlet through a set of disks with varying intensity. The visualization icons we call dashtubes - basically streamlines with animated opacity (section 3) - apply similar techniques to 3D flows.

Interrante et. al. [7] use LIC for 3D flow volumes. Halos around streamlets offer additional depth and ordering cues. The high cost of volume rendering, however, precludes an interactive exploration.

Texture splats as discussed in [5] encode direction and orientation of 3D flows. Fast splatting operations are realized with hardware supported texture mapping. Animated texture splats illustrate the flow dynamics. While these techniques produce otstanding results, they are not easily applicable for realtime applications.

Max et. al. [9] presented various techniques for visualizing 3D flows close to contour surfaces. Motion-blurred particles are generated in the vicinity of the surfaces. Particles are started automatically on a lattice. Generation and deletion of particles is density based. Line bundles are realized as texture splats with antialiased lines as texture. Hairs are 3D particle traces originating at the surface. Additional information is encoded in the color, length and transparency of these hairs.

Streamline placement is an important task to achieve an approximately uniform coverage of phase space. An image-guided streamline placement has been presented in [18]. Another approach [8] for creating evenly-spaced streamlines uses a regular grid with lists of passing streamlines to determine whether there is space for the placement of another streamline. With queues of streamline vertices possible seedpoints for new streamlines are administered. Tapering of streamline widths produce hand-drawing effects. Directional glyphs illustrate flow orientation. A 3D variation of the streamline placement in [8] was used in our approach (see section 3).

3 Dashtubes: Streamlines with Animated Opacity

Streamlines are an intuitive way of visualizing flow. Their applicability in space however is limited, since they do not provide the visual cues needed. Normally, lines are rendered with the same width for all distances so they lack perspective distortion, a significant cue for judging distance.

Additional techniques like halos [7] are necessary to resolve the ambiguities of overlapping lines. When visualizing flow, streamlines need to be enhanced to convey the direction of the flow. This can be done by directional color variations or by placing icons along the streamline as shown in [8]. Texture based techniques like LIC can be modified to include directional variations as we have shown

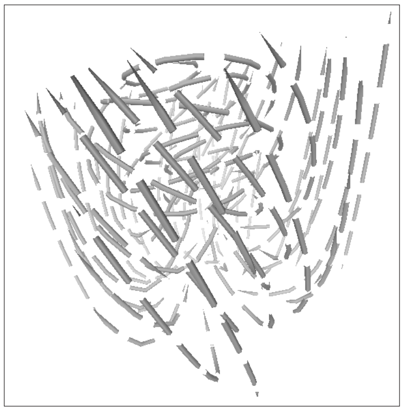

Figure 1: Dashtubes: opacity mapped streamlines

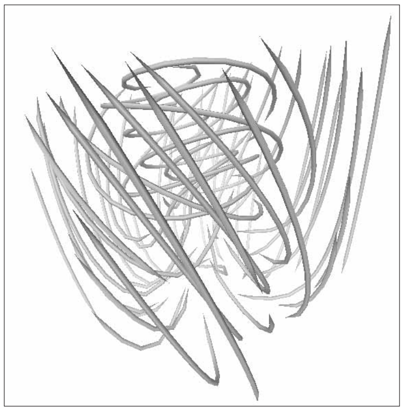

Figure 2: streamlines without opacity texture

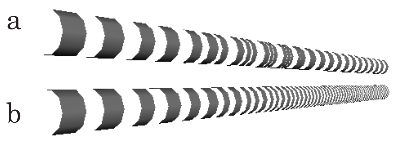

Figure 3: dashtubes with (a) and without (b) adaptive texturing

in [21]. A more direct approach however is the visualization of flow by animation. In the 2D case we use FROLIC combined with lookup-table animation to do this in realtime. Bryson [3] has succesfully used streaklines - 2D particles moving along the vector field - in the virtual windtunnel to animate flow in space. This technique depends on continually updating the position of all particles with every animation step, leading to a considerable consumption of processing power. Like FROLIC we would like to use the graphics hardware - which anyway has to update the image continuously when rendering for a virtual environment - to do the animation, leaving the CPU time for simulation and interaction.

Dashtubes meet the mentioned requirements. To avoid the occlusion of distant parts of the visualization by closer features and to generate the desired effect of particles moving along the dashtube we render it partially invisible. This is done by using an opacity texture which includes transparency information for the rendering hardware. Since transparency does not work well in combination with z-buffered visibility resolution, we only map completely opaque or complete transparent values to the geometry. Thereby we avoid artifacts produced by the order in which we render different parts of the scene. We supply the dashtubes with texture coordinates as time parametrization of the integration along the flow. When combined with an appropriate opacity texture, this leads to the desired dashed appearance, with opaque dashes intermitted by empty sections (figure 1). Since animating the texture image itself is a relatively time-expensive operation on most graphics hardware, we just transform the texture coordinates along the direction of the tube. In GL this can be done by modifying the texture-transform matrix, which has the additional advantage that it works even when more than one texture map is used. Animation is an essential part of the method, since otherwise structural information visible in (figure 2) would be lost in the opacity mapped representation in (figure 1).

4 Adaptive Texture-Mapping

When the flow covers a wide range of velocities, the texture resolution along the dashtube suffers. In sections of the tube where the velocity is high, long streaks of transparency convey no information and in sections where the velocity is extremely low, subpixel dashes lead to aliasing. In (figure 3) we show to dashtubes. Tube (a) uses adaptive texture mapping, tube (b) is non-adaptive textured. Since the flow velocity rises from left to right, the dashes of (b) quickly become unrecognizable and produce aliasing effects. In the middle of tube (a) our parametrization unites three short dashes to one long one, thereby reducing texture frequency while preserving the representation of flow velocity as texture speed. The transition is smoothly animated and produces a minimum of artifacts.

Below we describe two different approaches how this behavior can be implemented.

4.1 Mipmap Method

Aliasing artifacts can easily be reduced by using mipmaps [22], which are implemented in GL texture hardware. By specifying maps which convey an equal amount of visual detail on every level (figure 4) we can assure that the texture is visible independently of the parametrization. With this method it is not necessary to integrate all details in every level. High-resolution parts of the mipmap used when the parametrization is stretched do not have to contain the low-frequency components of lower-resolution maps, which are necessary to generate detail in densely parametrized sections of the tube. Such a map (figure 5) can be used to automatically switch textures when their projection to the screen would result in unpractical magnifications.

To reduce discontinuities when switching from one level of the mipmap to the next, GL implements continuous blending of levels. Such continous blending of opaque and transparent sections leads to semi-transparent parts on the tube, resulting in exactly the same z-buffer artifacts as stated in section 3. For opacity-mapped dashtubes we therefore have to use another technique or use the discrete (non-blending) mode of level switching, which produces considerable artifacts.

4.2 Texture-Coordinate Method

Switching between mipmap levels is done automatically in GL, based on the resolution of the texture map when projected to screen. With the method mentioned above we use this to automatically blend to a lower resolution texture when a sparse parametrization of the dashtube would lead to extremely elongated dashes. For our opacity-mapped tubes we need a method which implements a con-

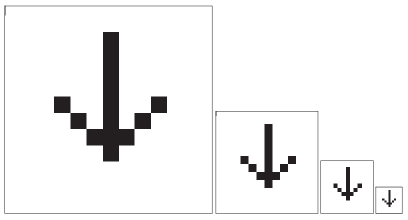

Figure 4: conventional use of mipmaps

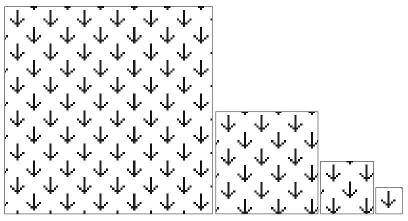

Figure 5: mipmaps for adaptive texturing

tinuous transition of dashwidth depending on parametrization density.

When using mipmaps, the different levels effectively add another dimension to the texture space, which can be used for this transition. This dimension is very coarse quantisized (10 levels for a 1024x1024 texture), therefore we need another approach to implement a smooth transition.

Since we are using an essentially onedimensional texture, mapped symmetrically around the tube, we can use the second, unused texture coordinate (v) to provide the transition dimension needed. By mapping level switches on this coordinate as shown in (figure 6a), we are able to apply onedimensional sections (figure 6b) of a 2D texture map (figure 7a) on the streamtube. The respective sections are shown in (6c), mapped onto a tube with nonlinear parametrization. Without adaptive mapping the tube looks like (6d). The velocity of the flow rises from left to right (6e). The mapping of u-texture coordinates to the tube is such that it yields a continuous transition from three short dashes to one long dash. When animating this texture the three short dashes slowly merge to one longer dash. By using a function like in (figure (6a), we restrict this behavior to short sections (figure (6: between points A and B) between longer tubes, where the dashwidth is constant. Additionally we can apply a hysteresis, switching between different dash-levels only when a reasonable change in velocity over a longer distance on the line is detected. This assures a sparse occurence of this transitions, which otherwise could lead to distracting behavior when the parameter density fluctuates heavily. Is the flow field well-behaved, a texture like (figure figure 7b) can be used to directly map velocity to adaptive dashes. The horizontal striped section map to section of the dashtube where the stripes do not split or unite.

5 Streamline Placement

When using streamlines for flow-visualization, the quality of the result depends heavily on the placement of the lines. Even when visualizing two dimensional flow fields an even distribution is desirable, but when extending the flow visualization to three dimensions the added complication of occlusions make it essential. Therefore our flow volume has to be depicted with sparsely but evenly distributed streamlines.

To accomplish this we use an algorithm based on an extension of [8] method to three dimensions.

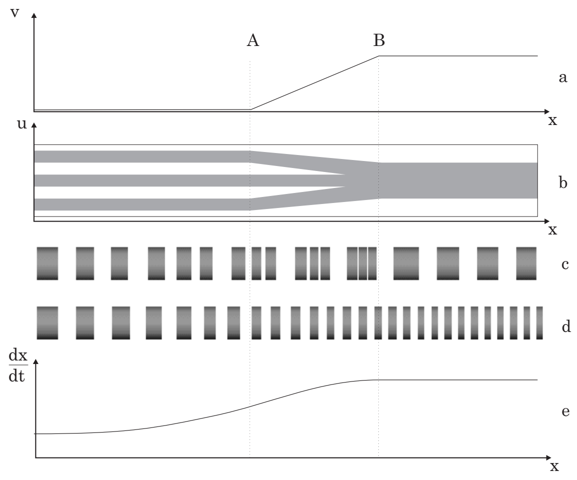

Figure 6: switching texture resolution using the u-coordinate

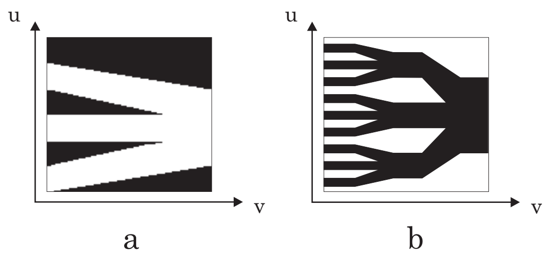

Figure 7: textures used for level switching

Jobard places the streamlines based on purely local criteria: the distance to the next streamline may not fall below a specified minimum. Since the speed of the algorithm depends mainly on this distance test, certain techniques are applied to accelerate the test. Firstly the distance between streamlines is defined as minimal distance between any of their sample points. This works reasonably well when the sample points are always closer spaced than the minimum distance between lines. Secondly all sample points are stored in a regular grid to reduce the set of points to be tested to the ones in the immediate neighbourhood of the new point. The distribution of seedpoints depends on the desired density of the illustration. For dense fields the seedpoints are distributed randomly, while for sparse fields they are introduced near the sample points of existing streamlines.

The adaption of this algorithm to our needs was quite straightforward. We extended the grid to three dimensions, making it necessary to check now a maximum of 27 neighbouring cells per test. The properties of the agglomerative seedpoint placement for sparse distributions mainly produce visually appealing results in 2D. In 3D, where streamlines can pass in front of each other and their visual distance depends mainly on the viewpoint it produces no distinctive advantage.

For that reason we chose to distribute the seedpoints on a jittered grid, a process which works faster than agglomerative seedpoint placement and produces acceptable results.

Since streamlines short in respect to the texture length can produce irritating ”blinking” artifacts, we reject them as soon as they are introduced, therefore allowing other streamlines to grow longer. This leads to an even greater average length than mere removal of short lines in an postprocessing step would produce.

6 Focussing and Context

One of the main problems when visualizing 3D flow fields is finding the right information density. Too much information per volume occludes features further away and too little information may hide important details. When using streamlines for 3D flow visualization, how much information a given volume contains depends directly on the number of streamlines through it. To a lesser degree it depends on the number of sample points along the streamline. An increase in points per line over the nyquist limit of the flow field does not contribute to the information content of the visualization. Since we place our streamlines as described in section 5 approximately equidistant to each other, the density of the visualization depends mainly on this user selected distance. Other factors contributing to the general appearance are width of the streamline and - in case of dashtubes - the length ratio of opaque to transparent sections.

When investigating 3D flow we first try to get an overview of the flow field. This includes investigation of global features, the identification of areas of special interest, like vortices, separatrices, cycles. Then, when an interesting local feature has been identified, we want to single this feature out and investigate it. We want to view it in great detail, without distractions or occlusions from other features.

In most practical cases, these two different goals exclude each other. So we tested focussing techniques where we use a coarser representation to identify interesting regions. Then we use one of the mechanisms described below to focus our attention on these regions and investigate them in higher detail.

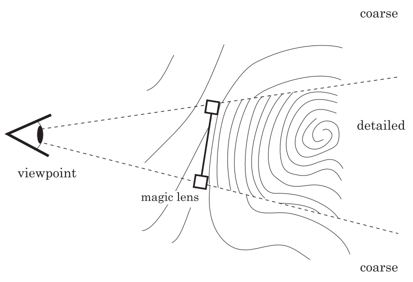

Figure 8: Volume defined by magic lens

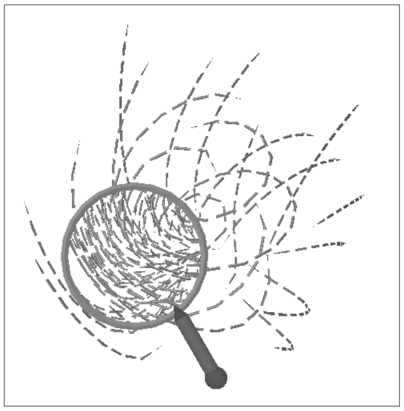

Figure 9: Focussing with magic lenses

7 Magic Lenses and Magic Boxes

Magic lenses, introduced by [2] are transparent user interface elemnts for conventional 2D windowing desktop environments. They are represented by special windows which do not display their own independent content but rather change the representation of underlying information. They can be used for filtering or otherwise modifying underlying image data but also for more abstract operations like showing additional information like comments. [20] applied this concept in virtual environments and also introduced their extension into three dimension as volumetric lenses, or ”magic boxes” as we call them.

We use these interface elements to view a higher-resolution representation of our visualization. This representation contains more streamlines per volume and the streamlines are thinner and generated with closer spaced vertices than the representation used for coarse navigation. We found that both focussing techniques have specific advantages.

7.1 Magic Lenses

A magic lens is presented as a flat objects with arbitrary boundaries (e.g, circle, square) which can be positioned with a 6DOF input device, normally a 3D mouse or tracked pen (figure 9). When looking through the lens, the user sees the high resolution representation. The main difference two pure 2D magic lenses is that our lens acts additionally acts as a clipping plane, allowing only the parts of the high resolution scene behind the lens to be seen. Without this clipping plane, the lens would also display features of the detailled representation between lens and viewpoint, resulting in the same occlusion problems as if the whole visualization were to be investigated. Together with the current viewpoint a magic lens effectively defines a viewing frustrum with its near clipping plane lying in the plane of the lens and its cross-section defined by the shape of the lens (figure 8).

Work with the magic lens is easy and intuitive. The user positions it in front of interesting features and views them through it like through a magnifying glass (figure 9). The scene behind the lens has always the same orientation and scale as the surrounding coarse representation. Discontinuities between resolutions resulting from the additional streamlines inside the lens are masked by the lensframe. These discontinuities only concern the representation of the 3D flow, not the flow itself. A position in space the same coordinates in the 3D flow in- and outside the lens. While presenting an effective and visual appealing investigation mechanism, magic lenses have one distinctive disadvantage compared to their 3D counterparts: the focussed volume depends strongly not only on the position of the lens but also on the viewpoint. This does not matter when a single user is looking for a local feature. The user typically sweeps the lens through space, positioning it and himself until the area of interest is located. When this has been accomplished, the investigation technique normally changes. When the detail has been located, it has to be examined from different angles, a procedure for which magic lenses are not well suited. They have to be dragged around the feature when the viewpoint changes.

7.2 Magic Boxes

Magic Boxes overcome the above mentioned disadvantages of viewpoint dependencies. Instead of only implicitly defining the focussed volume depending on the current viewpoint (figure 8), they explicitly define a volume inside their boundaries to contain the detailled representation of the 3D flow figure 10). The user positions the box with a 6DOF input device until it contains the local feature (figure 11). Then it may be viewed from all directions.

This is especially important when there are several users viewing the same visualization at the same time like in our multi-user virtual environment STUDIERSTUBE [17]. When using magic lenses ev-

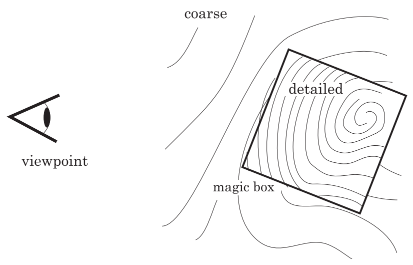

Figure 10: Volume defined by magic box

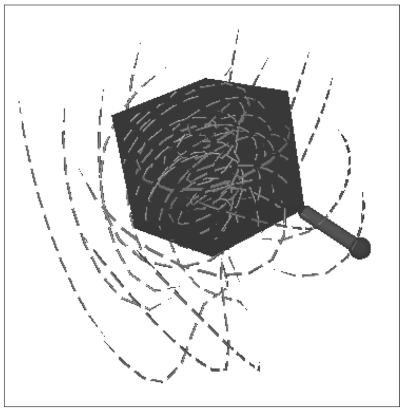
Figure 11: Focussing with magic boxes

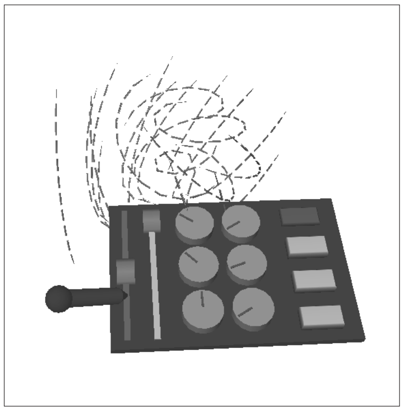
Figure 12: Interaction using the PIP

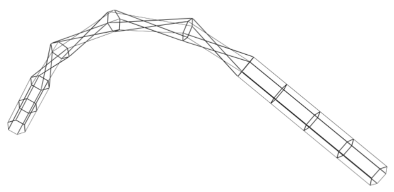
Figure 13: Contraction artifacts due to torsion

ery user has to position his own lens according to his position, or different users have to trade places when looking through a single lens.

Discontinuities between resolutions are more noticeable when using boxes instead of lenses, since the whole surface of the box acts as border between low- and high-resolution representations and cannot be masked by a frame like the image-aligned border of the lens.

8 Implementation

The visualization and investigation methods described above were implemented C++ using Open Inventor [14]. This OpenGL based graphics toolkit enabled us to efficiently realize our methods providing high-level graphics concepts like a scenegraph and sophisticated desktop interaction elements which we used in the early phases of our tests. The main advantage when implementing our methods was Inventors ability to suppy this high-level concepts while simultaneously enabling direct access to all OpenGL functions. This was essential when manipulating rendering sequences for magic lens and magic box. Since our virtual environment STUDIERSTUBE is also based on Inventor, the transfer from desktop evaluation implementation to the application in our VE was straightforward. The following sections describe implementation details of the our techniques and their integration in our virtual environment.

8.1 Interaction in the Virtual Environment

STUDIERSTUBE [17] is a multi-user augmented environment which we use for scientific visualization [6]. It implements basic interaction methods like positioning objects by dragging them with a 6 DOF pen (figures 9 and 11) as well as conventional 2D interaction elements like sliders, dials and buttons for parametrization of the visualization methods. These purely virtual interface elements are positioned on the PIP [16], a handheld tablet. Users hold this board in their non-dominant hand while make adjustments to the interface elements with the same pen they use for 6 DOF interaction (figure 12). In our application we used the PIP to adjust parameters of the visualized dynamical system (section 8.5) as well as properties of the dashtubes and the magic box. The speed, length of dashes and distance between dashes was adjustable with dials. Sliders on the PIP adjust the overall size of the magic box and allow independent scaling of one dimension of the box. This transforms the box to a ”slab”, allowing the user to use it to cut slices of arbitray width out of the flow field. Buttons on the PIP were used to switch between lens and box and to disable the coarse representation on demand.

8.2 Dashtubes

Dashtubes are realized as textured polygonal extrusions along the direction of the flow. Ideally the cross-section of the extrusion should be a circle to provide a symmetric appearance from all directions, but we found that the polygonal approximation can be reduced down to 3-6 edges dependending on the resolution of the display and the reqired quality of the image. By using Goraud shading the resulting discontinuities of the approximation are only visible along the silhouette edges. Coarse tesselations like these are prone to generating artifacts when the geometry is twisted along the extrusion axis. The resulting radial contractions lead to irritating variations in the width of the dashtube (figure 13). To avoid this, we generate the segments of the extrusion not by following the frenet-frame but by an modified algorithm, which tries to minimize the torsion by projecting the orientation of the cross-sections along the segments.

Early tests showed that the animated texture produces annoying visual artifacts at the ends of the dashtube. The opaque segments entering and leaving the surface of the extrusion exhibited an irritating ”blinking” behavior, comparable to the pulsation we had to overcome in FROLIC. We treated this by reducing the radius of the first and last cross-section of the dashtubes to null, thereby making the extrusion ”pointed” at the ends (figure 2). This yields smooth transitions at both ends, comparable to a ”fade-in” effect.

The geometry of the dashtubes where implemented as Inventor Shapekits, containing fields for the vertices of the extrusion axis and the texture-parameters and the geometric parameters of the cross-section. The Shapekit produces GL trianglestrip which give a better rendering performance than other GL primitives. Rendering the dashtubes with culled backfaces produces a the ”halfpipe” appearance on the ends of the opaque segments visible in (figure 3). Since this is only evident in extreme closeup, we decided that the rendering speedup justifies this artifact.

8.3 Magic Lenses

Magic lenses act as window from the coarse scene to the detailed scene. Our implementation uses SEAMs [12] as rendering primitives. Our ”magnifying glass” uses a circular SEAM inside a ring geometry as frame (figure **??**). According to the nomenclature of [12], the coarse scene outside the lens would be the ”primary world” and the detailed scene seen through the lens the ”secondary world”.

Rendering a SEAM uses only a single pass, the image of the secondary world is created on the fly when the SEAM is rendered in the primary world. To restrict the rendering of the secondary world to the area covered by the SEAM we use the GL stencil buffer, an additional layer used for masking areas of the screen during rendering. The geometry of both worlds is given as a directed acyclic graph (scene graph). The scene graph of the primary world is traversed and rendered. When a SEAM is encounted, the associated polygon - in our case the ”lens” - is passed to the rendering hardware for scan conversion.

For all pixels that the Z-test for the SEAM polygon finds to be visible:

- The frame buffer is set to the background color of the secondary world (clear screen),
- the Z-buffer is set to infinity (clear Z-buffer),
- the mask (stencil buffer) is set to 1.

Note that these image modifications are only carried out for the visible portion of the SEAM surface. After this preparation step, rendering of the secondary world is performed inside the stencil mask created in the previous step (so that the secondary world is not drawn outside the SEAM area), and with a clipping plane coincident with the SEAM polygon (so that the secondary world does not protrude from the SEAM). Finally - before rendering of the primary world proceeds - the SEAM polygon is rendered again, but only the computed depth values are written into the Z-buffer. Thereby the SEAM is ”sealed”. The resulting Z-values are all smaller than any Z-value of the secondary world. This asserts that no geometric primitive of the primary world located behind the SEAM will overwrite a visible pixel from the secondary world rendering.

8.4 Magic Boxes

We found that displaying only the contents of the magic box without visual representation of its boundaries makes it difficult to locate and position the box and tends to confuse the user. Therefore we added as geometric representation of the focussing volume a cube. The front faces of the cube are culled, leading to an ”open front” appearance regardless of the viewpoint.

Magic boxes are rendered using the same SEAM algorithm as described above, but use a cube instead of a plane to define the ”windows” between the two worlds. Six clipping planes coincident with the faces of the cube clip the secondary world.

Our implementation renders the complete scene (primary and secondary) only once, while [20] needs six rendering passes, one for each half-space defined by a clipping plane. This leads to a significant improvement in the framerate. The disadvantage of our method is the inability of displaying any parts of the secondary world (the detailed representation) that lie behind the box, which for our application would anyway only be distractive.

8.5 Application: Visualization of a Dynamical System

A dynamical system is a system whose temporal evolution from some initial state is dictated by a set of rules. Dynamical systems are found in many areas of research and application. Examples are fluid flow analysis, economic processes (e.g., stock market models), physics, medicine, and population growth models [1]. Dynamical systems are either given as an analytical specification or as sampled data.

Continuous systems (also called flows or vector fields) are given by a set of differential equations $\dot{x} = \nu(x)$. Vector $\nu(x)$ describes the direction, orientation and velocity of the flow at position x.

The behavior of dynamical systems can be investigated in *phase space*, where each state variable (i.e., each coordinate component of x) corresponds to a coordinate axis. A point in phase space completely describes the state of the system at one point in time. The temporal evolution from an inital state is called *trajectory*. Starting from an initial state x_0 the solution (i.e., trajectory) of a continuous dynamical system is given as curve $x(t)$ in phase space (see Equation (1)).

$$x(t) = x_0 + \int_0^t \nu(x(u))\,du \qquad (1)$$

Equation (1) is an integral equation which can be solved analytically only in very simple cases. Typically numerical integration is used to determine the trajectory of such a dynamical system [11]. In case of a time-invariant dynamical system a trajectory corresponds to a streamline (curve where the tangent at every point coincides with the flow direction of the underlying flow field).

The dynamical system we investigated in our evaluation is the well known forced Duffing oscillator [15] (see equation ...). It describes the damped motion of a mass attached to a nonlinear spring. The driving force is periodic. Due to the nonlinearity it can have quite an intricate behaviour. We generated the streamlines using a Runge-Kutta integrator with adaptive stepsize.

9 Evaluation and Results

Animated dashtubes produce a visual appealing and intuitive visualization of a 3D flow field. The main problem when applying them lies in finding the correct density for coarse and detailled representation.

Most users applied magic lenses without any problems, but needed some experimentation to grasp the concept of volumetric magic boxes.

When testing lenses and boxes with different detailed representations we found that magic boxes work better than lenses with very dense scenes. Since lenses do not clip distant parts of the detailed scene they are only applicable to scenes of higher density when they are rendered with strong depth cues (haze, fog).

The method of slicing the flow field with ”slabs” as mentioned in section 8.1 was implemented after users started to experiment with the distances of near and far clipping plane of the view volume to achieve this slicing view-dependent.

When using magic boxes, most users applied the following technique:

- position coarse representation
- position box until interesting features visible
- switch of coarse representation
- magnify box with included details for investigation

Since our application allowed independent positioning of focussing element and flow field, some users preferred positioning the flow field and keeping the box or lens stationary.

During the design phase of the dashtubes we used shutter glasses to produce stereoscopy. While this works very well for the examination of the flow field, interaction with the magic lens and box using the 2D desktop mouse is cumbersome and non-intuitive compared to interaction in the virtual environment.

10 Conclusion

In this paper we discussed several techniques which facilitate 3D flow visualization within a virtual environment. The newly introduced adaptive texture-mapping shows that texture hardware can be efficiently used to produce dashtubes with uniform spatial resolution. The approach ensures that velocity variations are still encoded in the animation. Dashtubes are positioned uniformly in phase space.

Interactive tools like magic lenses and magic boxes the proved to be valuable in the investigation of local features.

In our investigations 3D phase space contains spatially complex structures which are difficult to interpret. The added cues of a virtual environment (e.g., stereoscopic viewing, interactive and intuitive viewpoint change) are extremely helpful when inspecting these structures.

Acknowledgement

Special thanks to Hermann Wurnig, who did more than his share to implement the interaction methods, and to Dieter Schmalstieg and Helwig Löffelmann for their help and suggestions.

References

[1] D. K. Arrowsmith and C. A. Place. *An Introduction to Dynamical Systems*. Cambridge University Press, 1990.

[2] E. Bier, M. Stone, and K. Pier. Enhanced illustration using magic lens filters. *IEEE Computer Graphics and Applications*, 17(6):62–70, November/December 1997.

[3] Steve Bryson and Creon Levitt. The virtual windtunnel: An environment for the exploration of three-dimensional unsteady flows. In *Visualization '91*, pages 17–24, 1991.

[4] B. Cabral and L. C. Leedom. Imaging vector fields using line integral convolution. In *Proceedings SIGGRAPH '93*, pages 263–272, 1993.

[5] R. Crawfis and N. Max. Texture splats for 3d scalar and vector field visualization. In *IEEE Visualization '93 Proceedings*, pages 261–266. IEEE Computer Society, October 1993.

[6] A. Fuhrmann, H. Löffelmann, and D. Schmalstieg. Collaborative augmented reality: Exploring dynamical systems. In *IEEE Visualization '97 Proceedings*, pages 459–462. IEEE Computer Society, October 1997.

[7] V. Interrante and Ch. Grosch. Strategies for effectively visualizing 3D flow with volume LIC. In *Proceedings of Visualization '97*, pages 421–424, 1997.

[8] B. Jobard and W. Lefer. Creating evenly-spaced streamlines of arbitrary density. In Wilfrid Lefer and Michel Grave, editors, *Visualization in Scientific Computing*, pages 43–55. Springer-Wien-New-York, 1997.

[9] N. Max, R. Crawfis, and Ch. Grant. Visualizing 3D velocity fields near contour surfaces. In *IEEE Visualization '94 Proceedings*, pages 248–255. IEEE Computer Society, October 1994.

[10] F. H. Post and T. van Walsum. Fluid flow visualization. In H. Hagen, H. Müller, and G. M. Nielson, editors, *Focus on Scientific Visualization*, pages 1–40. Springer, 1993.

[11] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, 1988.

[12] Gernot Schaufler and Dieter Schmalstieg. Sewing worlds together with seams. Technical Report TR-186-2-98-11, Institute of Computer Graphics 186-2, Technical University of Vienna, Vienna, Austria, August 1998.

[13] D. Stalling and H.-Ch. Hege. Fast and resolution independent line integral convolution. In Robert Cook, editor, *Computer Graphics (SIGGRAPH '95 Proceedings)*, pages 249–256, August 1995.

[14] P. Strauss and R. Carey. An object oriented 3d graphics toolkit. In *Proceedings SIGGRAPH '92*, pages 341–347, 1992.

[15] S. Strogatz. *Nonlinear Dynamics and Chaos*. Addison Wesley, New York, 1994.

[16] Z. Szalavari and M. Gervautz. The personal interaction panel — A two-handed interface for augmented reality. *Computer Graphics Forum*, 16(3):C335–346, Sep 1997.

[17] Zsolt Szalavári, Dieter Schmalstieg, Anton Furhmann, and Michael Gervautz. Studierstube - An Environment for Collaboration in Augmented Reality. *Virtual Reality: Research, Development & Applications*, 1998.

[18] G. Turk and D. Banks. Image-guided streamline placement. In *Proceedings SIGGRAPH '96*, pages 453–459, 1996.

[19] J. J. van Wijk. Flow visualization with surface particles. *IEEE Computer Graphics & Applications*, 13(4):18–24, July 1993.

[20] J. Viega, M.J. Conway, G. Williams, and R. Pausch. 3d magic lenses. In *ACM UIST'96 Proceedings*, pages 51–58. ACM, 1996.

[21] R. Wegenkittl and E. Gröller. Fast oriented line integral convolution for vector field visualization via the internet. In *IEEE Visualization '97 Proceedings*, pages 309–316. IEEE Computer Society, October 1997.

[22] L. Williams. Pyramidal parametrics. *Computer Graphics*, pages 1–11.