Real-Time Techniques For 3D Flow Visualization

Anton Fuhrmann and Eduard Gröller

Vienna University of Technology*

ABSTRACT

Visualization of three-dimensional steady 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 occlusion. In this paper we present methods which address these problems for real-time graphic representations applicable in virtual environments. We use dashtubes, i.e., 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 densely 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.

Additional Keywords: virtual environments, flow visualization, texturing, interaction, magic lens, focussing

1 INTRODUCTION AND MOTIVATION

Many visualization techniques for two-dimensional flows have already been investigated in detail [Post93]. 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 viewer's abilities to interpret the visualized data.

The main point of this paper is to show how real-time graphics in a virtual environment can be used to overcome some of these 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 a virtual environment as compared to a desktop system.

We present a combination of selected visualization and interaction techniques, which enable the user to rapidly explore complex 3D vector fields. Fast texture-based visualization techniques, which utilize the graphics hardware to get real-time performance, are applied to streamlines. A new parameterization 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 that 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 [Joba97] 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, A-1040 Vienna, Austria
email: {fuhrmann|groeller}@cg.tuwien.ac.at
URL: http://www.cg.tuwien.ac.at/home/

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 [Cabr93], [Wijk93], [Stal95]. We already applied texture-based visualization techniques in [Löff97].

FROLIC [Wege97a] is a variation of OLIC (Oriented Line Integral Convolution, [Wege97b]) based on LIC [Cabr93]. OLIC 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 and OLIC are replaced in FROLIC by approximating a streamlet by a set of disks with varying intensity. The visualization icons we call dashtubes are basically 3D streamlines with animated texturing and apply similar techniques to 3D flows.

Interrante et. al. [Inte97] 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 [Craw93] 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 good results, their rendering times are prohibitive for real-time applications.

Max et.al. [Max94] presented various techniques for visualizing 3D flows close to contour surfaces. Motion-blurred particles are generated in the vicinity of 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 on 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 [Turk96]. Another approach for creating evenly spaced streamlines uses a regular grid [Joba97]. For each grid element a list of passing streamlines determines whether there is still space for the placement of another streamline.

Queues of streamline vertices administer possible seedpoints for new streamlines. Tapering of streamline widths produce hand-drawing effects. Directional glyphs illustrate flow orientation. A 3D variation of the streamline placement in [Joba97] 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 3D-space however is limited, since they do not

provide the visual cues needed. Normally, lines are rendered with the same width regardless of distance to the viewpoint so they lack perspective distortion, which is a significant cue for judging distance.

Additional techniques like halos [Inte97] 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 [Joba97]. Texture based techniques like LIC can be modified to include directional variations as we have shown in [Wege97a] and [Wege97b]. 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 real-time. Bryson [Bry97] has successfully 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.

In this paper we present dashtubes as visualization tool for steady 3D-flow. Dashtubes are generalized cylinders extruded along the direction of the flow (Figure 1). Their geometry is displayed with animated, opacity mapped texturing to visualize velocity and direction of flow. They appear as "dashes" - short opaque segments - moving along the direction of the flow.

Our requirements for dashtubes are:

Volume-filling properties: the dashtubes have to exhibit an even distribution over the volume of interest. Otherwise the omission of interesting features would be probable.

Reduced occlusion: we need a method which reduces the occlusion of distant features by features near the observer. This demand is almost the opposite from the volume-filling properties mentioned above.

Animation of flow: the velocity and direction of the flow should be visible in the animation. Dash velocity should directly correspond to flow velocity.

Visibility of dashes: the length of the dashes should not vary too much with velocity. High velocity areas would otherwise produce long dashes and gaps, which do not give the desired appearance, while low velocity would lead to very short dashes and eventually to aliasing artifacts.

Fast Rendering: Since one of our main points is the real-time applicability, we want a fast, hardware-assisted rendering method. Like FROLIC we would like to use the graphics hardware to do the animation, leaving the CPU time for simulation and interaction. This is advantageous as the graphics hardware has anyhow to update the image continuously when rendering a virtual environment.

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 semi-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. The dashtubes are assigned texture coordinates, which correspond to a temporal parameterization along streamlines.

Figure 1: Streamline geometry without texturing

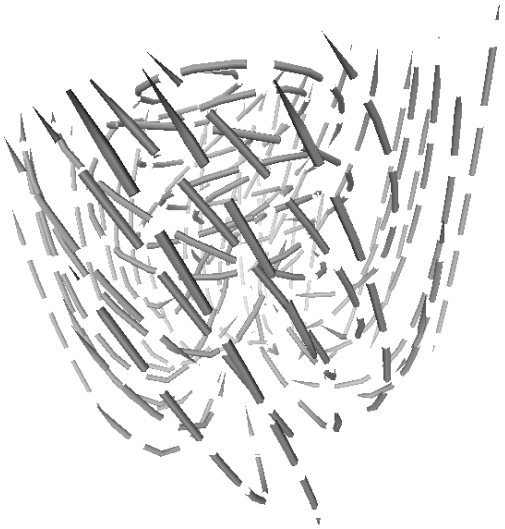

Figure 2: Dashtubes with opacity-texture

When combined with an appropriate opacity texture, this leads to the desired dashed appearance, with opaque dashes intermitted by empty sections (Figure 2). 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 OpenGL 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 1 would be lost in the opacity-mapped representation (Figure 2).

Early tests showed that the animated texture produces annoying visual artifacts at the ends of the dashtubes. 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 tapering the extrusion at the ends (Figure 1 and Figure 2). This yields smooth transitions at both ends, comparable to a ”fade-in” effect.

4 ADAPTIVE TEXTURE-MAPPING

As most texture mapping techniques, our method is prone to aliasing. When the length of one dash in image-space is reduced to a single pixel width or below annoying artifacts make the distinction of dashes difficult. Even worse, the speed and direction of the visualized flow becomes impossible to observe.

These effects appear when the viewpoint moves away from the texture-mapped tube or when the flow velocity is very low. In both cases a large texture area is mapped to a small region of the screen.

Therefore we need a method to reduce aliasing while preserving the essential properties of dashtubes: a high contrast texture moving along the tube at the speed of the visualized flow which contains distinctive opaque and completely transparent sections. Furthermore the maximum length of a dash and gap sequence must not be too big. Otherwise the appearance of the dashtube as a line of moving particles would suffer. Long dashes, while giving a good impression of the direction of the flow, occlude much of the scene farther away and reduce the impression of a volume flow. Ideally we want approximately uniform spaced dashes in image space, independent of viewpoint and flow velocity.

4.1 Mipmap Method

Most commonly the mipmap-method [Will83] is used to reduce texture aliasing. Thereby the texture is filtered to consecutively lower resolutions. A single texture is represented by a series of texture maps with decreasing size and texture-frequency (Figure 3a). Since this method does the filtering in a pre-processing step, the additional expenses at runtime are relatively low making it the method used by most real-time graphics hardware.

To apply mipmaps on dashtubes we have to alter the standard algorithm of producing the reduced texture maps. Normally each sub-map of a mipmap contains a filtered version of the level above, thereby halving resolution and maximum frequency of the texture (Figure 3a) This clashes with our rejection of semi-transparent objects in section 3: the contrast of lower resolution maps is reduced due to filtering, which also produces semi-transparent areas. The flow velocity is preserved, but the dashes get shorter in areas with reduced velocity or a more distant viewpoint.

So we have to produce sub-maps which do not exhibit this undesired properties. Figure 3b shows an example how such maps could look like: They posses the fractal property of having the same amount of detail on every level.

Using such a mipmap produces the following effects: If the viewpoint moves farther away, the texture (the dashes) shrink continuously in length until a new mipmap level is reached, where the switch to a coarser resolution is performed and the dashes again have a distinguishable length. Sections of dashtubes where the low flow velocity would produce very short dashes are also mapped to coarser resolutions and therefore exhibit longer dashes (Figure 4).

An important aspect of this approach is that only the contents of the texture are switched, not the mapping of texture to object. This leads to the intended effect when animating the texture: the velocity of the texture along the tube directly corresponds to the flow velocity independently of the length of the dashes.

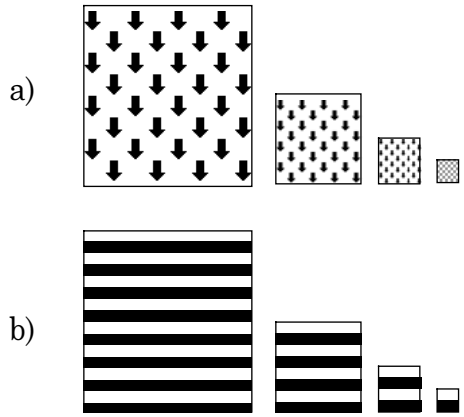

Figure 3: Mipmap textures

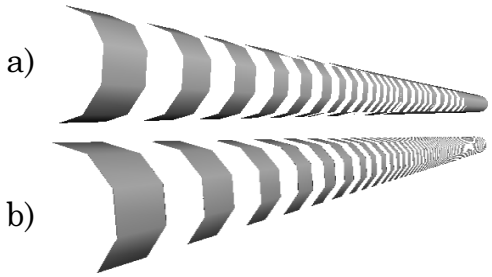

Figure 4: dashtubes with (a) and without (b) mipmap-texturing

The main problem when using mipmaps for adaptive texture-mapping arises from the small number of available mipmap levels: Since every two-dimensional sub-map has exactly the double resolution of the next lower level, the memory consumption exponentially rises with the number of levels. Additionally the switch between one level and the next leads to discontinuities in the texture appearance. Since the textures move along the tubes these discontinuities are especially annoying when they appear along a single tube.

Most mipmap implementations - including the one used in OpenGL - reduce the artifacts due to level switching by interpolating between two adjacent sub-maps. We can not apply this method in our case since this strongly reduces contrast because the interpolation is performed between two essentially different maps, not between maps only differing in the high-frequency components. Furthermore the interpolation produces semi-transparent areas causing the already in section 3 discussed problems when rendering into the z-buffer.

These problems occur since the transition between one level and the next is relatively coarse. If we could reduce the difference between level resolutions and increase the number of levels we would be able to exactly specify how switches between longer (less) and shorter (more) dashes are performed. The vast memory consumption of mipmaps with a high number of levels denies us the trivial solution of this problem, so we have to approach it differently.

In section 4.2 we show a different approach to adaptive texture-mapping, which overcomes some of the problems mentioned above.

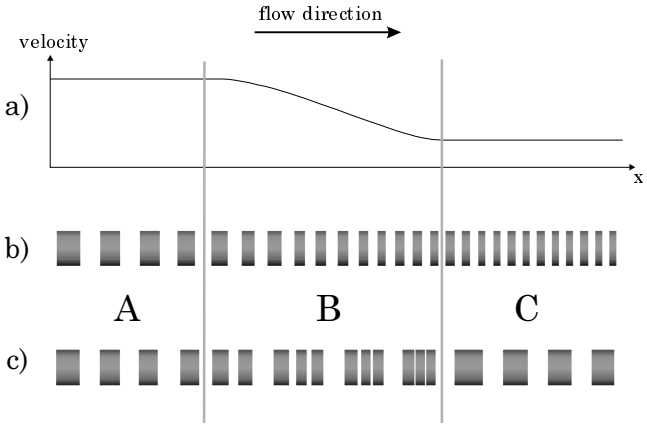

Figure 5: Adaptive texturing with texture-coordinate method

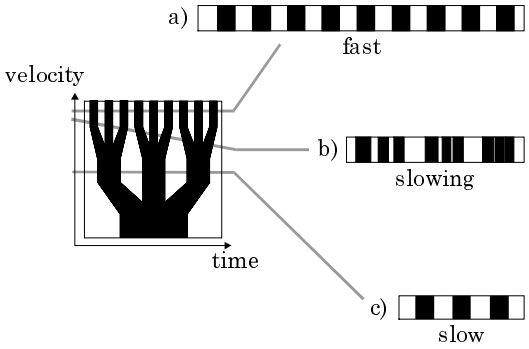

Figure 6: Texture usage of texture-coordinate method

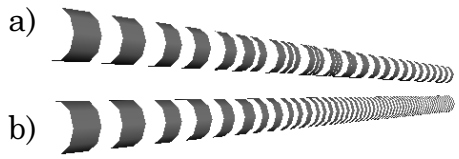

Figure 7: dashtubes with (a) and without (b) adaptive texturing

4.2 Texture-Coordinate Method

If the flow velocity along a dashtube is falling as in Figure 5a, the resulting texturing leads to short dashes in the low-velocity region of the dashtube (Figure 5b, region C). This behavior violates our demands from section 3, where we require the dashes to be distinguishable independently of flow velocity. Since these short dashes can lead to undesirable aliasing artifacts, we would prefer longer dashes in region C, which should still reflect flow velocity. If we want longer dashes moving with the same speed as the shorter ones, we have to somehow reduce the number of dashes travelling from region A to region C. A possible solution, which reduces annoying artifacts shows region B in Figure 5c: every three dashes leaving region A get joined to a single dash while moving through region B. The low-velocity part (region C) of the dashtube contains one long dash in Figure 5c for every three short dashes in Figure 5b. When animated, this shows three dashes leaving the high-velocity region A of the dashtube, which reduce their gaps until they merge into one longer dash in the low-velocity region C.

To implement this behavior using conventional (OpenGL) texture-mapping hardware we have to alter the way we map a texture to the geometry of the dashtube. In ordinary texture-mapping applications two spatial texture coordinates are mapped onto the 2D surface of an object. Since the rotationally-symmetric dashtubes represent essentially 1D objects - the underlying streamlines - we only need one texture coordinate for the spatial mapping. We use the remaining texture coordinate to smoothly vary the dashes appearance.

Figure 6 shows a texture map, which maps velocity and time along the dashtube onto opacity. For constant velocity horizontal rows of the texture are mapped along the tube. Constant low velocity maps the lowest row of the texture onto a short segment of the dashtube (Figure 6c), whereas constant high velocity maps the uppermost row of the map onto a long segment of the tube (Figure 6a). Changing velocity along a segment samples the map along a slanted line (Figure 6b), thereby producing the effects depicted in Figure 5c, region B.

The merging dashes in the transitions exhibit slightly irritating artifacts when the gaps shrink to one pixel width. To reduce this effect, we want to map the transitions only to short segments of the dashtubes. Between this transition segments we allow the dashes to lengthen or shorten a bit, without altering their number. This gives the motion of the dashes a more uniform appearance.

The texture map in Figure 6 is designed to restrict the transition segments (Figure 5c, section B) to small velocity ranges. In the areas where the texture contains parallel, vertical stripes the sampled texture is independent of the velocity. Only when the sampling passes into one of the branching areas of the map dashes are joined or split.

5 STREAMLINE PLACEMENT

When using streamlines for flow-visualization, the quality of the result depends heavily on the placement of the streamlines. Even when visualizing two dimensional flow fields a uniform distribution is desirable, but when extending the flow visualization to three dimensions the added complication of occlusions make an even placement essential. The start- and endpoints of streamlines introduce distracting artifacts into the visualization so we want to keep their number small. Therefore we want to populate our flow volume with evenly distributed streamlines of maximum length.

To accomplish this we extend the algorithm of Jobard and Lefer [Joba97] to three dimensions. The streamlines are placed in [Joba97] by using local criteria only. A new streamline can only be placed if the distance to already existing streamlines does not fall below a certain minimum. Since the speed of the algorithm depends mainly on this distance test, certain techniques are applied to accelerate the test. Each streamline is approximated by a set of evenly spaced sample points. The distance between two streamlines is defined as the 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. A regular grid is used to reduce the set of points to be tested to the ones in the immediate neighborhood of a new point. The distribution of seedpoints depends on the desired density of the resulting images. For densely placed streamlines the seedpoints are distributed randomly, while for sparsely placed streamlines the seedpoints are introduced near the sample points of existing streamlines.

The adaptation 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 cells per distance test. Another technique presented in [Joba97], 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 opposed to random startpoint placement.

For this 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 which are short with respect to the dash length can produce irritating ”blinking” artifacts, we reject them as soon as they are introduced, therefore allowing other streamlines to grow.

6 FOCUSSING AND CONTEXT

One of the main problems when visualizing 3D flow fields is finding the correct 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, the amount of information in a given volume is directly related to the number of streamlines through it. To a lesser degree it also depends on the number of sample points along the streamline. As described in section 5 we place our streamlines approximately equidistant to each other. Therefore density of the streamlines in the resulting images 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 a 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, and cycles. Then, when an interesting feature has been identified, we want to single out this feature and investigate it. We want to view it in great detail, without distractions or occlusions from other features.

In many practical cases, these two different goals are difficult to achieve simultaneously. Therefore we tested techniques where we first use the context of a coarse representation to identify interesting regions. Then we use one of the mechanisms described in sections 6.1 and 6.2 to focus our attention on these regions and investigate them with finer detail.

Magic lenses, as presented in [Bier97] are transparent user interface elements 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 the 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. In [Vieg96] magic lenses are used in virtual environments. This work also deals with volumetric lenses, an extension of magic lenses in three dimensions.

We use these interface elements to view a higher resolution representation of the flow field. 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 - lenses and boxes - have specific advantages.

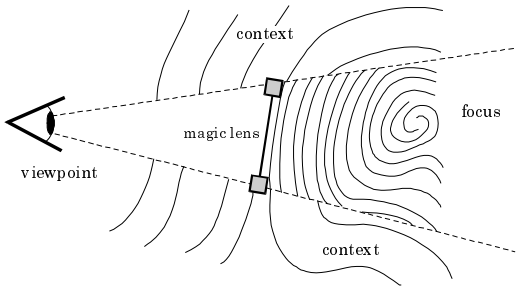

Figure 8: Volume defined by magic lens

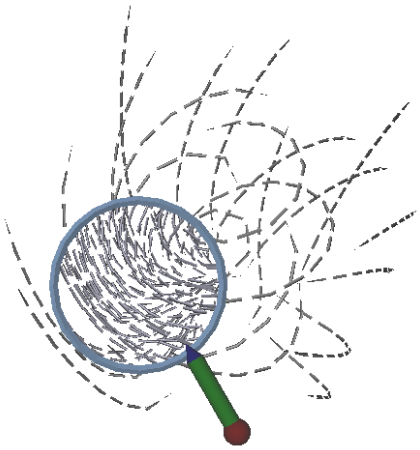

Figure 9: Focussing with a magic lens

6.1 Magic Lenses

A magic lens is a planar polygon 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 focuses on the high-resolution representation.

The main difference to 2D magic lenses is that our lens 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 detailed representation between lens and viewpoint, resulting in the same occlusion problems as if the entire detailed flow visualization were to be investigated. Together with the current viewpoint a magic lens effectively defines a viewing frustum 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).

Working 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 frame of the lens masks the border between focus and context. While presenting an effective and visually appealing investigation mechanism, magic lenses have one distinctive disadvantage compared to magic boxes (section 6.2): 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: Now, that the detail has been located, it has to be examined from different angles, a procedure

for which magic lenses are not well suited. The lens has to be dragged around the feature together with the changing viewpoint. This is a cumbersome process, which may lead to accidental loss of the focus area and the contained feature.

6.2 Magic Boxes

Magic Boxes - volumetric magic lenses - overcome the above mentioned disadvantages of viewpoint dependency. Instead of only implicitly defining the focussed volume depending on the current viewpoint (Figure 8), they explicitly define a volume of interest (Figure 10). In the interior of a magic box the detailed representation of the flow is displayed.

The user positions the box with a 6DOF input device until it contains the local feature (Figure 11). Then this feature may be viewed from all directions. This is especially important when there are several users viewing the same focus like in our multi-user virtual environment STUDIERSTUBE [Szal98]. When using magic lenses every user has to position his own lens according to his position, or different users have to trade places when looking through a single lens.

The border between focus and context is more noticeable when using boxes instead of lenses, since it consists of the whole surface of the box and cannot be masked by a frame like the image-aligned border of the lens. On the other hand the confined volume of the box allows focussing in all three dimension, whereas magic lenses do not inherently define a far plane of the focus. Additionally the box occludes features of the context behind it, thereby reducing distraction.

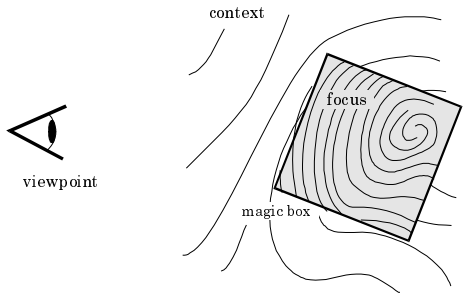

Figure 10: Volume defined by magic box

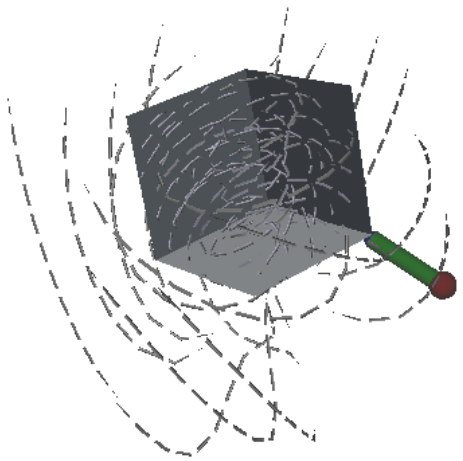

Figure 11: Focussing with a magic box

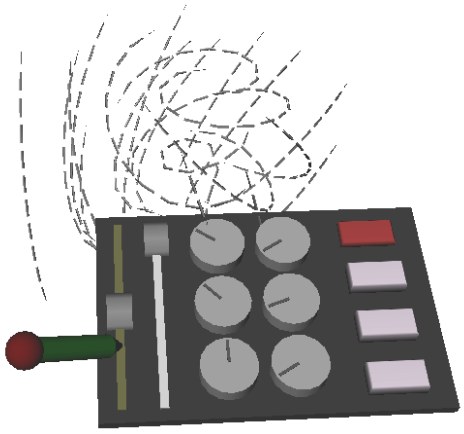

Figure 12: Interaction using the PIP

7 IMPLEMENTATION

The visualization and investigation methods described above were implemented in C++ using Open Inventor [Stra92]. This OpenGL based graphics toolkit enabled us to efficiently realize our methods providing high-level graphics concepts like a scene graph and sophisticated desktop interaction elements, which we used in the early phases of our tests. The main advantage when implementing our methods was Open Inventors ability to supply these high-level concepts while simultaneously enabling direct access to all OpenGL functions. This was essential when manipulating rendering sequences for magic lenses and magic boxes. Since our virtual environment STUDIERSTUBE is also based on Open Inventor, the transfer from a desktop evaluation-implementation to the application in our virtual environment was straightforward. The following sections describe implementation details of the techniques and their integration into our virtual environment.

7.1 Interaction in the Virtual Environment

STUDIERSTUBE [Szal98] is a multi-user augmented environment, which we use for scientific visualization [Fuhr97]. It implements basic interaction methods like positioning objects by dragging them with a 6 DOF pen (Figure 9 and Figure 11) as well as conventional 2D interaction elements like sliders, dials and buttons for parameterization of the visualization methods. These purely virtual interface elements are positioned on the PIP [Szal97], a handheld tablet. Users hold this board in their non-dominant hand while they 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 dynamical system which provided the flow field 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 arbitrary width out of the flow field. Buttons on the PIP were used to switch between magic lens and magic box and to disable the coarse representation on demand.

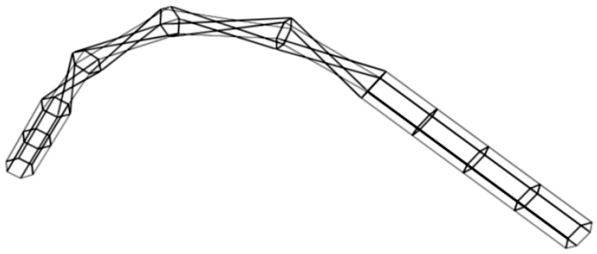

Figure 13: Contraction artifacts due to torsion

7.2 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 to 6 edges depending on the resolution of the display and the required quality of the image. By using Gouraud 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 along the streamline. We use a algorithm similar to [Bloo90], which reduces the torsion by aligning the orientation of the polygonal cross-sections along the segments.

The geometry of the dashtubes was implemented as Open Inventor Shapekit, containing fields for the vertices of the extrusion axis, the texture parameters and the geometric parameters of the cross-section. The Shapekit produces OpenGL trianglestrips, which give a better rendering performance than other OpenGL primitives. Rendering the dashtubes with culled backfaces produces a ”halfpipe” appearance at both ends of the opaque segments as visible in Figure 7. Since this is only evident in extreme close-up, we decided that the rendering speedup justifies this artifact.

7.3 Magic Lenses

Magic lenses act as window from context to focus. Our implementation uses SEAMs [Scha98], a mechanism to connect two virtual worlds by "windows" of arbitrary geometry. Our magic lens has the appearance of a magnifying glass, using a circular SEAM inside a ring geometry providing the frame (Figure 9). According to the nomenclature of [Scha98], the context outside the lens would be the ”primary world” and the focus seen through the lens the ”secondary world”. The geometry of both worlds is given as a directed acyclic graph (scene graph). The scene graph of the context is traversed and rendered. When a SEAM is encountered, the associated polygon - in our case the ”lens” - is passed to the rendering hardware for scan conversion.

To restrict the rendering of the focus to the area covered by the SEAM we use the OpenGL stencil buffer, an additional layer for masking areas of the screen during rendering. 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 the focus is performed inside the stencil mask created in the previous step. This prevents that the focus is drawn outside the SEAM area. A clipping plane coincident with the SEAM polygon prevents the focus from protruding from the SEAM. Finally - before rendering of the context 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 focus. This asserts that no geometric primitive of the context located behind the SEAM will overwrite a pixel generated by rendering the focus.

This gives the desired impression of a "window" behind which only the focus is displayed (Figure 9).

7.4 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 a cube as geometric representation of the focussing volume. 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 focus and context. Six clipping planes coincident with the faces of the cube clip the secondary world (the detailed representation).

Our implementation renders the complete scene (focus and context) only once, while [Vieg96] needs six rendering passes, one for each halfspace derived from a cube face. This leads to a significant improvement in the frame rate. Our method does not display any parts of the context that lie behind the box, which for our application would anyway only be distractive.

8 EVALUATION AND RESULTS

Animated dashtubes produce an intuitive visualization of a 3D flowfield. The main problem when applying our focussing techniques lies in finding the correct density for focus and context. When testing lenses and boxes with different densities of dashtubes in the focus we found that magic boxes work better than lenses with densely placed dashtubes. 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).

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

The method of slicing the flow field with ”slabs” - magic boxes of small height - as mentioned in section 7.1 was implemented after users started to experiment with the distances of near and far clipping plane of the view volume to achieve this "slicing" effect in a view dependent manner.

When using magic boxes, most users applied the following technique: position coarse representation conveniently; position box until interesting features visible; switch off 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 magic lenses and magic boxes using the 2D desktop mouse is cumbersome and non-intuitive compared to the interaction in the virtual environment.

9 CONCLUSION

In this paper we discussed several techniques which facilitate 3D-flow visualization within a virtual environment. The newly introduced adaptive texture mapping method 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 automatically positioned in phase space to produce an even representation of the underlying 3D flow.

Interactive tools like magic lenses and magic boxes proved to be valuable in the investigation of local features. They enable the user to interactively select finely detailed features and reduce distraction by context. 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 quite helpful when inspecting these structures.

ACKNOWLEDGEMENT

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

This work has been supported by the Austrian Science Foundation (FWF) under project no. P-12074-MAT.

REFERENCES

[Bier97] E. Bier, M. Stone, and K. Pier. Enhanced illustration using magic lens filters. IEEE Computer Graphics and Applications, 17(6), pages 62–70, November/December 1997.

[Bloo90] J. Bloomenthal. Calculation of Reference Frames Along a Space Curve. Graphic Gems, pages 567-571. Academic Press, Cambridge, MA, 1990.

[Bry97] 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.

[Cabr93] B. Cabral and L. C. Leedom. Imaging vector fields using line integral convolution. In Proceedings SIGGRAPH ’93, pages 263–272, 1993.

[Craw93] 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.

[Fuhr97] A. Fuhrmann, H. Löffelmann, and D. Schmalstieg. Collaborative augmented reality: Exploring dynamical systems. IEEE Visualization ’97 Proceedings, pages 459–462. IEEE Computer Society, October 1997.

[Inte97] V. Interrante and Ch. Grosch. Strategies for effectively visualizing 3D flow with volume LIC. In Proceedings of Visualization ’97, pages 421–424, 1997.

[Joba97] 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-NewYork, 1997.

[Löff97] H. Löffelmann, L. Mroz, E. Gröller, W. Purgathofer. Stream Arrows: Enhancing the Use of Stream Surfaces for the Visualization of Dynamical Systems. Visual Computer, Springer, Vol. 13(8), pages. 359-369, 1997.

[Max94] 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.

[Post93] 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.

[Scha98] G. Schaufler and D. Schmalstieg. Sewing worlds together with seams. Technical Report TR-186-2-98-11, Institute of Computer Graphics 1862, Technical University of Vienna, Vienna, Austria, August 1998.

[Stal95] D. Stalling and H.Ch. Hege. Fast and resolution independent line integral convolution. In Robert Cook, editor, Computer Graphics (SIGGRAPH 1995 Proceedings), pages 249–256, August 1995.

[Stra92] P. Strauss and R. Carey. An object oriented 3D graphics toolkit. In Proceedings SIGGRAPH 1992, pages 341–347, 1992.

[Szal97] 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.

[Szal98] Z. Szalavari, D. Schmalstieg, A. Fuhrmann, and M. Gervautz. Studierstube - An Environment for Collaboration in Augmented Reality. Virtual Reality: Research, Development & Applications, 1998.

[Turk96] G. Turk and D. Banks. Image-guided streamline placement. In Proceedings SIGGRAPH 1996, pages 453–459, 1996.

[Vieg96] J. Viega, M.J. Conway, G. Williams, and R. Pausch. 3D magic lenses. In ACM UIST’96 Proceedings, pages 51–58. ACM, 1996.

[Wege97a] 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.

[Wege97b] R. Wegenkittl, E. Gröller, W. Purgathofer, Animating Flowfields: Rendering of Oriented Line Integral Convolution, Computer Animation '97, pages 15-21, IEEE Computer Society, June 1997.

[Will83] L. Williams. Pyramidal parametrics, Computer Graphics (SIGGRAPH 1983 Proceedings), 17(3), pages 1-11, July 1983.

[Wijk93] J. J. van Wijk. Flow visualization with surface particles. IEEE Computer Graphics & Applications, 13(4):18–24, July 1993.