Efficient Visualization of Large Amounts of Particle Trajectories in Virtual Environments Using Virtual Tubelets

Marc Schirski[1]*, Torsten Kuhlen[1], Martin Hopp[2], Philipp Adomeit[2], Stefan Pischinger[3], and Christian Bischof[1]

[1] Center for Computing and Communication, RWTH Aachen University, Seffenter Weg 23, 52074 Aachen, Germany
[2] FEV Motorentechnik GmbH, Neuenhofstraße 181, 52078 Aachen, Germany
[3] Institute for Internal Combustion Engines (VKA), RWTH Aachen University, Schinkelstr. 8, 52062 Aachen, Germany

Abstract

An effective means for flow visualization is the depiction of particle trajectories. When rendering large amounts of these pathlines, standard visualization techniques suffer from several weaknesses, ranging from ambiguous depth perception to high geometrical complexity and decreased interactivity. This paper addresses these problems by choosing a novel approach to pathline visualization in 3D space, which we call *Virtual Tubelets*. It employs billboarding techniques in combination with suitable textures to create the illusion of three-dimensional tubes, which efficiently depict the particles' trajectories, while still maintaining interactive frame rates. Certain issues concerning virtual environments and immersive displays with multiple projection screens are resolved by choosing an appropriate orientation for the billboards. The use of modern, programmable graphics hardware allows for an additional speed-up of the rendering process and a further improvement of the image quality. This results in a nearly perfect illusion of tubular geometry, including plausible intersections and consistent illumination with the rest of the scene. To prove the efficiency of our approach, rendering speed and visual quality of *Virtual Tubelets* and conventional, polygonal tube renderings are compared.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Virtual Reality; I.3.3 [Computer Graphics]: Picture/Image Generation—Display Algorithms; I.6.6 [Simulation and Modeling]: Simulation Output Analysis; J.2.5 [Physical Sciences and Engineering]: Engineering

Keywords: VR, flow visualization, programmable graphics hardware

1 Introduction

Analyzing the movement of particles through a volume is an intuitive and effective means for understanding the underlying flow. This includes massless particles, which are inserted into the flow as a post-processing measure, as well as the results of dispersed multi-phase flow models [Com 2001], which simulate the behavior of e.g. liquid fuel drops inside a combustion chamber, and which generate additional data as e.g. temperature or size. While displaying the motion of these particles in an animated sequence is a valid method of visualizing their trajectories, it loses expressiveness as soon as the animation is stopped. On the other hand, depicting the whole trajectory of every single particle results in a cluttering of the view space due to the sheer amount of information displayed at any time, especially when visualizing particle movement over a very long period. In this case, occlusion becomes a major problem. Furthermore, the informational connection between a particle's location and the corresponding time is lost. We address these problems by displaying particle traces, i.e. the particle position at a given time, supplemented with a tail of user-definable length to display a particle's recent history. This allows a better understanding of a particle's behavior in time, while minimizing screen clutter.

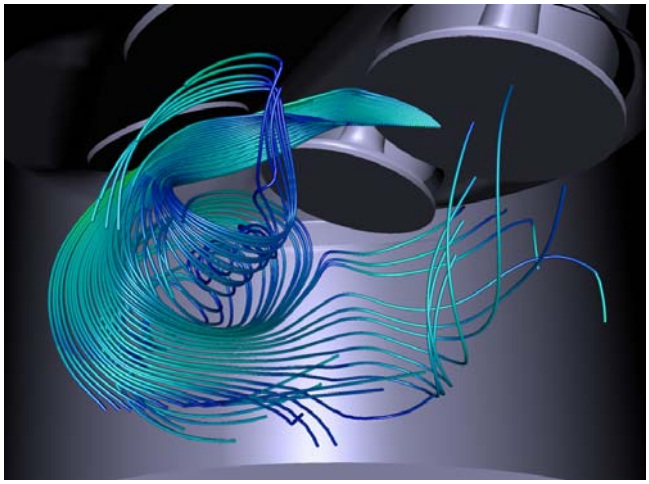

Figure 1: Visualizing pathlines via *Virtual Tubelets*; specular and diffuse illumination are computed in a fragment program.

Common issues of line-based trajectory visualizations include depth perception issues and ambiguities concerning the depth order. A very common workaround is the depiction of particle traces as geometrical tubes, which in turn increase the geometrical complexity of the scene. This results in a significantly higher load on the graphics system and decreased frame rates, thus reducing the responsiveness and interactivity of the visualization. Actually, this is one of the major problems in employing this visualization method in Virtual Reality (VR) applications, because head-tracking and the ubiquitous need for interactivity requires the frame rate to stay above a certain threshold. Furthermore, substituting simple lines by complex geometry reduces the flexibility of the visualization, because every change of visualization parameters requires a recreation of this geometry. This can be reduced somewhat by precomputing tubular geometry for every single time step, which in turn increases memory consumption significantly.

*Corresponding author, e-mail: schirski@rz.rwth-aachen.de

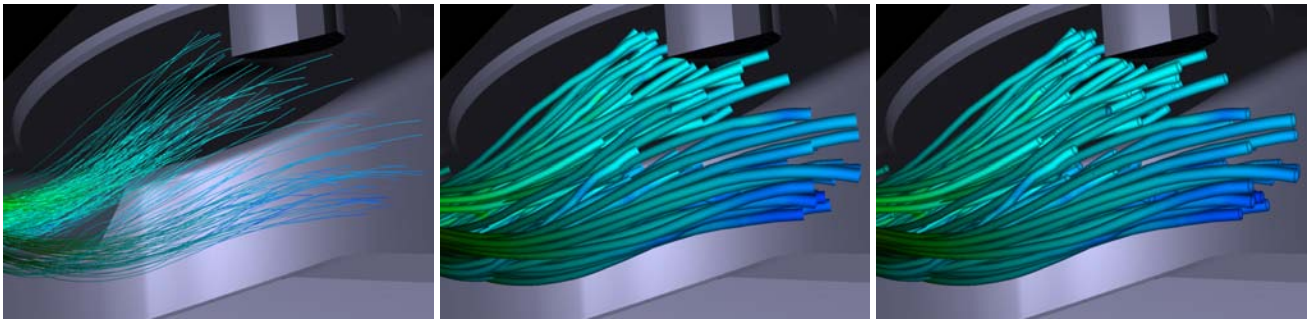

Figure 2: Particle trajectories, depicted as simple lines (left), textured billboards (middle), and textured and capped billboards (right).

In this paper, we show a way of substituting complex tubular geometry by cylindrical billboards in order to depict particle trajectories. In their most basic form, they consists of user-aligned quadrilaterals with a suitable texture map to create the illusion of slightly self-illuminated, rounded geometry, while still maintaining high frame rates (see figure 1). Perception problems caused by abrupt endings of these virtual tubes are avoided by an explicit capping. This approach allows for a high responsiveness and interactivity of the visualization system in combination with a very good visual quality. Using modern graphics hardware, which, through PC clusters, is getting more and more common in VR systems, allows for an even more convincing simulation by incorporating bump mapping and depth replacement, thus creating a nearly perfect illusion. Of course, all principles shown in this paper can be applied to an efficient rendering of conventional streamlines, as well.

The remainder of this paper is structured as follows: Section 2 gives an overview of related work and section 3 deals with a basic form of *Virtual Tubelets* and its implementation. The application of *Virtual Tubelets* in Virtual Environments is discussed in section 4. In section 5, details about improvements to the basic method by the help of modern graphics hardware are given and section 6 discusses the achieved results. The paper closes with a summary and an outlook at future work in section 7.

2 Related Work

A variety of methods for three-dimensional flow visualization are related to this work: Texture-based approaches like Volume LIC [Interrante and Grosch 1997] are mostly extensions to two-dimensional variations of LIC [Cabral and Leedom 1993] and create streamline-like flow representations by filtering noise textures along the underlying vector field.

In contrast to texture-based approaches, geometry-based methods display flow features directly by graphical icons. Examples for geometry-based approaches are given in [Bancroft et al. 1990], where streamlines are depicted as simple, unshaded lines of equal thickness, whose main drawback is the lack of depth cues. Based on streamlines are streamribbons and streamtubes [Schroeder et al. 1998], which represent streamlines as three-dimensional geometrical objects, thus raising the polygon count in the scene to be drawn considerably. [Zöckler et al. 1996] present illuminated streamlines, which improve depth perception by using texture-mapping hardware for a simulation of specular reflection. Nevertheless, the depth order of a group of lines is still ambiguous. Amongst others, this problem is addressed in [Mattausch et al. 2003] by surrounding illuminated streamlines with darker halos, which requires at least two rendering passes. Perception problems at the end of thick streamlines are addressed by tapering these ends in order to create more pleasing images.

[Fuhrmann and Gröller 1998] introduced dash tubes. They depict streamlines as three-dimensional tubes with an animated opacity texture, thus raising the polygon count of the scene. Note, that most of these approaches are limited to the visualization of static flows and display the whole streamline or particle path at once.

With the advent of programmable graphics hardware, per-pixel shading and bump mapping are becoming more and more widespread. A variety of examples are offered by [NVIDIA]. Enhanced billboards and impostors, which include depth information are discussed by [Akenine-Möller and Haines 2002] and implemented for simple spheres by [NVIDIA].

3 Virtual Tubelets – Basics

As stated before, our goal was to achieve a high-quality visualization of large populations of particle traces, while still maintaining high frame rates and, therefore, a high level of interactivity. As the human eye needs certain landmarks for orientation, especially when employing stereoscopic projection, simple line based visualizations were considered inappropriate (see figure 2, left). In contrast, more seemingly solid methods were to be used - e.g. tubular pathlines, similar to streamtubes. As available graphics systems were unable to cope with the massive amounts of polygons created by this approach, a minimum level of interactivity could not be maintained, which ultimately led to the development of our billboard-based approach – *Virtual Tubelets*.

Virtual Tubelets are based on cylindrical billboards; they are basically quadrilaterals, which are aligned to face the viewer, and which are textured appropriately to create the illusion of self-illuminated, glowing tubes. These tubes are animated to visualize the movement of particles through the flow domain. At any given moment, a tubelet's head is positioned at the current position of the corresponding particle with its tail passing through the particle's recent positions. Cluttering of the view space, especially when dealing with very long trajectories, is avoided by limiting their length to a user-definable time window.

In addition to the illumination texture, a coloration is applied, which allows for additional scalar quantities to be visualized. Examples include the velocity of the particle (i.e. the velocity of the underlying vector field) or, in the case of dispersed multi-phase flow models, the temperature of a droplet. Furthermore, by gradually reducing a tubelets' brightness from its start to its end, additional clues about the behavior of the particle in time are given, which improves the expressiveness of the visualization even more.

A major problem of a simple billboarding approach is a visual inconsistency at the start and the end of the trajectory. As the start and end of the tube do not have the rounded edges, which a viewer expects of geometrical tubes, it seems to point away from the viewer (see figure 2, middle). We resolve this issue by capping the tube explicitly with an additional quadrilateral, which is textured appropriately to, again, create the illusion of rounded geometry. This quadrilateral is positioned orthogonally to the first or last section of the tubelet. This way, when facing the tubelet from the front, it seems to have a well-defined front face; when facing it from the back, the cap visually melts with the tubelet's body and creates a rounded back end (see figure 2, right).

As there is no underlying geometry involved, most tubelet parameters can be changed on the fly, e.g. width, length, coloration, without having to recreate any polygonal representation.

3.1 Implementation

We implemented *Virtual Tubelets* in their basic form using only standard OpenGL. In this case, the orientation of the billboards is computed on the CPU and is recomputed for every change of the view position. The components of a basic *Virtual Tubelet* are depicted in figure 3.

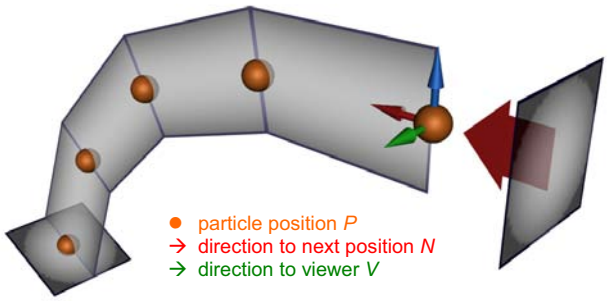

Figure 3: Anatomy of a *Virtual Tubelet*.

If the trajectory data is given as lists of particle positions, the visualization is able to compute triangle strips for the oriented billboards without additional data, thus keeping the memory footprint low. For every particle position within the time window to be displayed, two vertices have to be sent to the graphics system. Their positions P_0, P_1 are determined by

$$P_i = P + r(-1)^i(N \times V),\ i \in \{0,1\}$$

with P being the original particle position, r being the tubelet's radius, and N and V being the normalized vectors from the current particle position to the next particle position and to the viewer[1], respectively. While the computation can be sped up by caching N for every particle position, V is reevaluated for every frame. Caching of V does not seem worthwhile because the cache would have to be rather large and every view point change invalidates it completely. Especially when the viewer's head is tracked, these cache invalidations tend to happen too frequently.

Tubelet caps are created for the start and end of every tubelet to be displayed. They are sent to the graphics system as quadrilaterals with the points $P_{i,j}$ with

$$P_{i,j} = P + r((-1)^i U + (-1)^j R),$$
$$\text{with } (-1)^k N = U \times R$$

and with $i, j \in \{0,1\}$ and $k = 0$ for start positions and $k = 1$ for end positions of the respective tubelet. Thus, they are oriented orthogonally to the first or last segment of the tubelet.

The illumination of a tubelets' body and cap is created through a texture map, which contains only luminance and alpha information (see figure 4). The luminance information is used to create the illusion of rounded geometry for both components, and the alpha information is used for the caps to create a rounded appearance by employing a suitable alpha test. For the tubelet bodies, only the u-coordinate of the texture is used, while the v-coordinate is set to 0.5 to use only its middle slice, where the simulated geometry's normal would point directly at the viewer.

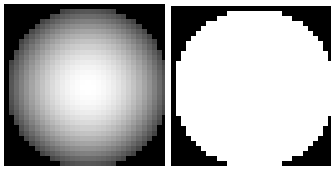

Figure 4: Luminance (left) and alpha (right) channels of the illumination texture for *Virtual Tubelets*.

For maximum rendering efficiency, we use vertex arrays to send the vertex data to the graphics systems. As the data has to be recomputed for every change of the view position, only one vertex array is used, which is refilled for every frame. For unextended OpenGL, this is the most efficient mechanism applicable in this context, because display lists are not an option due to the highly dynamic nature of the billboard geometry.

4 Virtual Environments

For non-immersive display systems like standard computer monitors, billboards are often aligned to the view plane. Due to their small size and the relatively large distance from the viewer to the screen, the projective distortion is kept to a minimum and can often be neglected. However, when using immersive displays, the viewer is relatively close to the projection plane most of the time, due to its sheer size. Thus, projective distortion becomes more apparent and the flat nature of the billboards can be recognized more easily.

In addition, inconsistent depth sorting of billboards, which lie close to each other, across multiple projection screens, can lead to visual artifacts at the borders of these devices, which basically ruin the immersive effect (see figure 5). To avoid this, we align the billboards to the viewer instead of the view plane.

As the alignment and display of the tubelets is considered a part of the rendering process, *Virtual Tubelets* are suited for collaborative multi-user applications, as well. In this case, every user would see the tubelets oriented towards his own view position; thus, the illusion of three-dimensional geometry is preserved.

5 Improvements

Virtual Tubelets achieve a fairly good visual approximation of geometrical tubes. However, there are still some issues, which can be

[1] Note, that we do not align the billboards with the view plane but towards the viewer. Reasons for this are discussed in section 4.

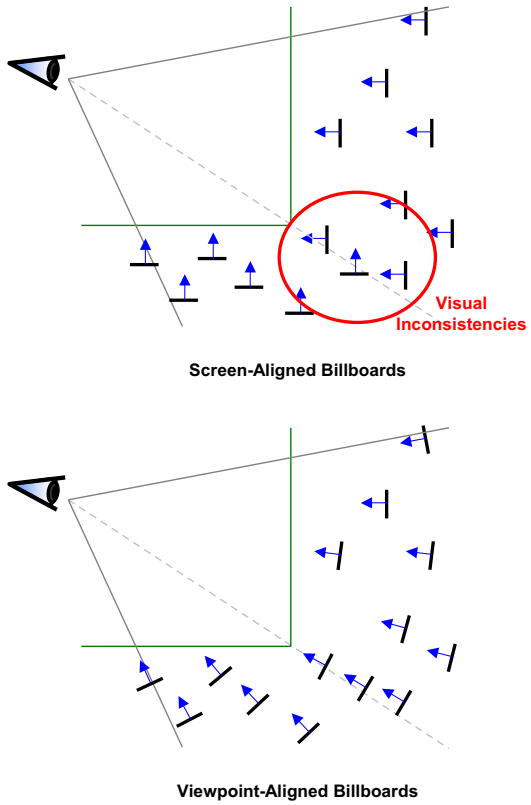

Figure 5: Visual inconsistencies when employing screen-aligned billboards (top) in immersive systems with multiple projection screens. Aligning billboards to the viewer resolves this issue (bottom).

```
!!ARBvp1.0

# constant parameters
PARAM mvp[4] = { state.matrix.mvp };      # model-view-projection matrix

# per-vertex inputs
ATTRIB iPos       = vertex.position;      # particle position
ATTRIB iCol       = vertex.color;         # particle color
ATTRIB iTexCoord  = vertex.texcoord;      # direction to next particle

# local program parameters
PARAM viewPos     = program.local[0];     # view point and tubelet radius

# per-vertex outputs
OUTPUT oPos       = result.position;
OUTPUT oCol       = result.color;
OUTPUT oTexCoord0 = result.texcoord[0];   # texture coordinates

# some constant parameters
PARAM   constants = { 0.5, 0.5, 1, 0 };

# and some temporaries
TEMP  lUp;                    # local up vector
TEMP  temp;

# calculate direction from particle to viewer
SUB temp, viewPos, iPos;

# calculate local up vector
XPD lUp, iTexCoord, temp;

# and normalize it
DP3 temp.w, lUp, lUp;
RSQ temp, temp.w;
MUL lUp, lUp, temp.w;

# set the particle's radius
MUL lUp, lUp, viewPos.w;
MUL temp, temp, viewPos.w;

# determine vertex position
MAD temp, iPos.w, temp, iPos;
MAD temp, iTexCoord.w, lUp, temp;
MOV temp.w, constants.z;

DP4 oPos.x, mvp[0], temp;
DP4 oPos.y, mvp[1], temp;
DP4 oPos.z, mvp[2], temp;
DP4 oPos.w, mvp[3], temp;

MOV oCol, iCol;
MAD oTexCoord0.y, iTexCoord.w, constants.x, constants.x;
MAD oTexCoord0.x, iPos.w, constants.x, constants.x;

END
```

Figure 6: Vertex program for calculating the orientation of a billboard.

addressed by the use of programmable graphics hardware. With the advent of (relatively) low-cost PC clusters being used for operating immersive displays, this becomes an option for more and more VR installations. When using OpenGL, capabilities of modern graphics hardware become accessible through an extension mechanism [OpenGL], which allows hardware vendors or the Architecture Review Board (ARB) to define interfaces to features, which exceed standard OpenGL functionality. In our case, we exclusively used ARB extensions to stay as platform independent as possible.

5.1 Shifting computational load to the GPU

While the basic *Virtual Tubelet* approach allows for a very efficient visualization of large numbers of particle trajectories, there is still quite some workload on the CPU, namely computing the orientation of the billboards. Modern programmable graphics hardware allows for every vertex, which is sent to the graphics system, a small assembler program to be executed, which can e.g. change this vertex' position. By using this mechanism, a major part of the workload can be shifted to the GPU by letting the latter compute the orientation of the billboards.

Figure 6 shows a vertex program according to the specification of the ARB_vertex_program extension [SGI], which receives the position of the viewer and which computes the corresponding positions of the billboard vertices. For every vertex, the vertex position contains the corresponding particle position, and the texture coordinate's x, y, and z components contain the direction to the next position. The w-component of the texture coordinate contains either -1 or +1 to distinguish between the two vertices for a single particle position. Note, that for every particle position we average the direction to the next position with the direction from the previous position to the current one. This way, we minimize visual problems if the particle trajectory contains very sharp bends due to a small number of positional samples.

By shifting workload to the GPU, the CPU is free for other work, which can increase the responsiveness of the visualization system significantly. Especially when dealing with virtual environments, motion tracking, physically-based modeling, and user interfaces in general influence the interactivity of a system considerably.

5.2 Plausible intersection with other geometry

A major drawback of *Virtual Tubelets* in their basic form is the implausible intersection of the billboards with other geometry. Due to the flat nature of the billboards, its depth values do not match its shaded appearance causing visual inconsistencies (see figure 7) when intersecting other polygonal geometry. This issue can be resolved by employing graphics hardware, which features a programmable pixel pipeline. Through a suitable fragment program, a fragment's depth is corrected to approximate the seemingly rounded surface. This results in a nearly perfect illusion of tubular geometry, which is kept up even when mixing *Virtual Tubelets* with other, polygonal visualization methods.

However, as this method requires more complex vertex programs

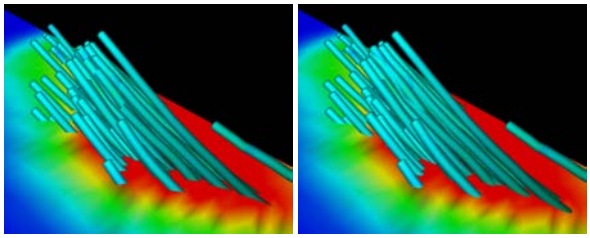

Figure 7: Intersection with other geometry causes visual inconsistencies (left). Replacing the fragments' depth value resolves this issue (right).

and the additional use of fragment programs, the frame rate is reduced considerably.

5.3 More complex lighting models

Furthermore, it may be desirable to shade the *Virtual Tubelets* according to the illumination of the surrounding geometry. This can be achieved through the use of dot3 bump mapping and per-pixel shading. In this case, the illumination texture is replaced by a normal map, which encodes the surface normal in the R, G, and B channels of the texture by scaling and biasing it from [-1;1] to [0;1] (see figure 8).

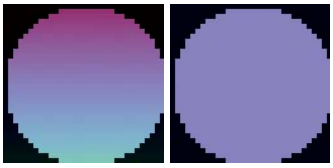

Figure 8: Normal maps for the tubelets' body (left) and cap (right).

In its simplest form, diffuse illumination can be achieved on graphics hardware with just two texture units solely through the use of ARB extensions, but in this case, the additional mapping of scalar values to colors is not possible in a single pass. More sophisticated graphics hardware allows for more complex illumination models to be implemented, including ambient, diffuse, and specular lighting, while still being able to map scalar values to the coloration and luminance of the virtual geometry. A fragment program calculating only diffuse and ambient illumination is shown in figure 9. In this case, the vertex program has to be extended to provide the direction to the light in tangent space for every vertex. In the fragment program, the ambient and diffuse illumination contributions are determined via the dot3 product of this light vertex with the fragment normal and modulated with the particle color.

In order for this to work with our capping approach, the capping algorithm has to be modified slightly. The geometry of the tubelet body is extended at its start and its end with an additional section, which is oriented exactly like the tubelet cap itself. In contrast to simple *Virtual Tubelets*, two separate textures have to be used for the body and the cap. The texture for the body encodes the normals for the rounded geometry and contains an alpha channel for the simulation of rounded back-ends. The texture for the caps contains only normals pointing to the viewer and, again, an alpha channel for its circular shape. This results in a convincing simulation of the illumination of tubular geometry with flat ends, as can be seen in figure 10.

```
!!ARBfp1.0

TEMP lookup;
TEMP temp;

PARAM constants = { 0, 0.1, -1, 2 };

# retrieve tangent space normal and alpha value
TEX lookup, fragment.texcoord[0], texture[0], 2D;
MAD lookup, lookup, constants.w, constants.z;

# calculate diffuse lighting and add some ambient light
# light direction is stored in texture coordinate 1
DP3_SAT lookup.xyz, lookup, fragment.texcoord[1];
ADD lookup, lookup, constants.yyyx;

# modulate lighting with the given color
MUL result.color, lookup, fragment.color.primary;
END
```

Figure 9: Fragment program for the calculation of diffuse illumination.

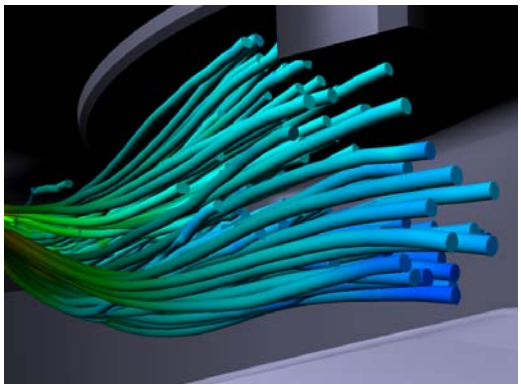

Figure 10: Per-pixel shading creates a convincing illusion of rounded geometry, here illuminated from the front left.

Depending on the complexity of the shading model to be employed, the load on the graphics system increases, which results in a reduced frame rate. Of course, this advanced shading can be combined with the correction of depth values, but in this case the load on the graphics system increases even further.

6 Discussion

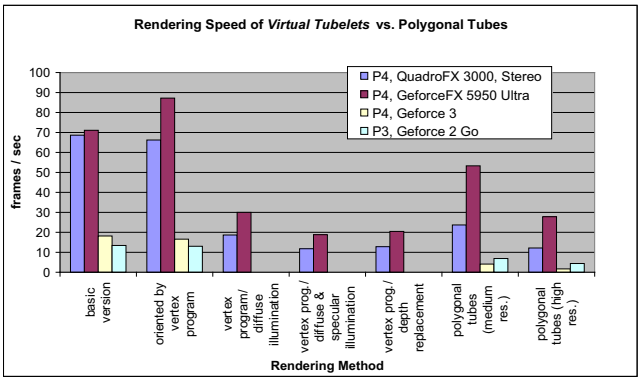

Figure 11: Performance comparison of *Virtual Tubelets* and polygonal tubes on a selection of computing systems.

The main goal of this work was an increase in rendering speed to allow for a highly interactive visualization of large amounts of particle trajectories. To determine, whether this goal was reached,

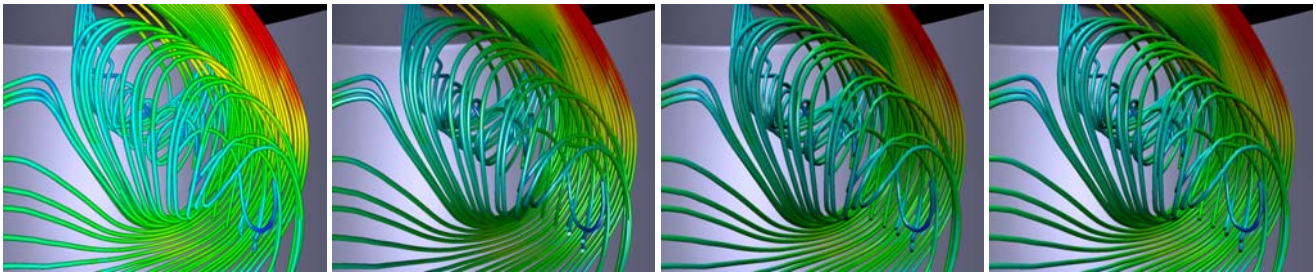

Figure 12: Comparing the visual quality of *Virtual Tubelets* with textured illumination, *Virtual Tubelets* with diffuse and specular illumination, polygonal tubes of medium resolution, and polygonal tubes of high resolution (from left to right).

measurements have been taken on a variety of computing systems, including Intel Pentium IV systems with NVIDIA GeForce FX 5950 graphics and NVIDIA Quadro FX 3000 graphics - the system equipped with NVIDIA Quadro FX graphics cards has been used for monoscopic and stereoscopic rendering. To prove the applicability of our approach, older systems have been used for testing as well, in this case a mobile Intel Pentium III system with GeForce 2 Go graphics and an Intel Pentium III system with GeForce 3 graphics. Furthermore, we tested our strategy on a Sun Microsystems V880z visualization server and an (admittedly outdated) Silicon Graphics Onyx2 IR2. While only the GeForce FX and Quadro FX systems provided the fragment shading capabilities used for depth replacement and more complex lighting, all systems took advantage of the basic *Virtual Tubelet* approach. For brevity reasons, we provide detailed statistics for a selection of Intel-based systems, only - the Sun and SGI systems demonstrated performances comparable to the one equipped with a GeForce 3. Figure 11 shows the results of the performance measurements, which compare the frame rate of a fuel injection visualization for a certain instant in time. For this visualization, we used *Virtual Tubelets* in several variations and geometrical tubes with 6 and 12 sides. For efficiency reasons, the geometrical tubes are rendered via display lists, which require a certain initialization phase, but which allow for a significantly higher frame rate. The measurements have been taken after the initialization of the display lists.

When faking illumination through a simple texture, rendering speed and responsiveness of the visualization are superior to a visualization with polygonal tubes, while even improving on the visual quality. This holds true especially for older systems. When employing programmable fragment shading capabilities of current hardware, this picture changes due to the raw power of these graphics systems. However, in this case, the visual quality of *Virtual Tubelets* is far superior to that of the geometrical tubes.

Figure 12 shows a comparison of the visual quality of various forms of *Virtual Tubelets* and geometrical tubes in different resolutions. While per-pixel shaded *Virtual Tubelets* and geometrical tubes provide a consistent illumination with well-defined specular highlights for the whole scene, simply textured *Virtual Tubelets* provide a clean and crisp visualization of the particle trajectories.

Regardless of rendering speed and visual quality, our strategy provides a much more flexible and efficient approach regarding memory usage and geometry processing than conventional polygonal tube representations. As there is very little geometry involved, the only data to be stored permanently is the trajectory data itself. For the measurements shown above, we added a slight memory overhead by precomputing the normalized vectors from one particle position to the next. Still, memory usage is considerably lower, because there is no need to store a complete tube geometry for every trajectory. In addition, our approach is much more flexible regarding tube parameter changes, like width and length of the time window, because all these parameters can be set on the fly without the need for recomputing any underlying geometry representation. If vertex programs are used, this boils down to a simple input parameter change.

Virtual Tubelets have been successfully used for a variety of particle-based visualization tasks, like simple post-processing pathlines or more complex droplet trajectory visualizations (figure 13). The latter are results of simulations using dispersed multi-phase flow models and include additional data, like droplet size or temperature, which can be encoded in the tubelets color. Besides data sets of an injection of liquid fuel into the combustion chamber of internal combustion engines, we used our approach for an efficient and intuitive visualization of air flow within intake manifolds and catalysts.

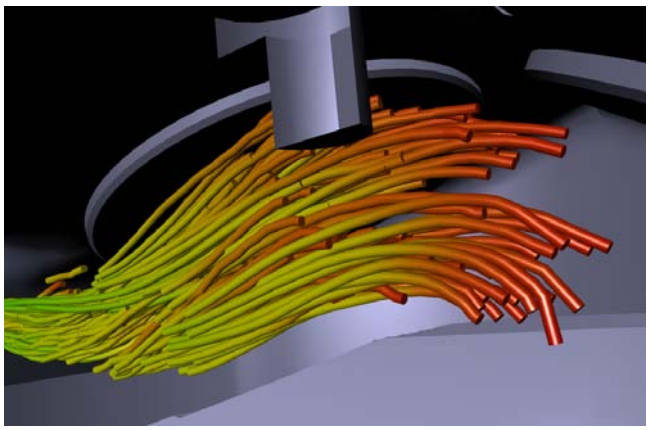

Figure 13: Visualizing the trajectories of liquid fuel droplets; color represents temperature.

7 Summary and Future Work

We have presented a fast and efficient method of rendering large amounts of particle trajectories in virtual environments. Our approach, which we call *Virtual Tubelets*, visually simulates tubular geometry through appropriately textured billboards, which, as we have shown, speeds up the rendering process considerably, thus providing a higher frame rate and an improved responsibility of the visualization system. Furthermore, our approach allows for an interactive change of different parameters, like width, length, and coloration, without a need for expensive recalculation of underlying geometry. Common issues of billboarded tubes, like visual incon-

sistencies at their ends and inconsistent depth sorting across multiple, non-parallel projection screens, are addressed and resolved, as well.

Furthermore, we extended our basic approach to make use of programmable graphics hardware in order to further improve efficiency and rendering speed, resolve intersection issues, and provide a more consistent illumination. These methods increase the visual quality and the illusion of rounded geometry considerably at the expense of maximum achievable frame rate. Nevertheless, compared to a standard, polygonal representation of tubes, *Virtual Tubelets* are still superior in regard to rendering speed, visual quality, and/or flexibility.

Ultimately, we have implemented a system, which can display large amounts of tubular particle trajectories in a Virtual Reality environment. This includes interactive frame rates, stereoscopic rendering, and the use of immersive display systems with a user-centered stereoscopic projection on multiple screens.

Future work includes further refinements of the illumination by incorporating shadowing through shadow maps. In addition, we are going to improve the data transfer to the graphics hardware on the expense of memory usage, by using larger, pre-computed vertex arrays and/or vertex buffer objects [SGI].

References

AKENINE-MÖLLER, T., AND HAINES, E. 2002. *Real-Time Rendering*, second ed. A. K. Peters.

BANCROFT, G. V., MERRIT, F. J., PLESSEL, T. C., KELAITA, P., MCCABE, R. K., AND GLOBUS, A. 1990. FAST: A multiprocessed environment for visualization of computational fluid dynamics. In *Proceedings of IEEE Visualization 1990*, IEEE Computer Society Press, IEEE Computer Society, 14–27.

CABRAL, B., AND LEEDOM, L. C. 1993. Imaging vector fields using line integral convolution. In *Proceedings of ACM SIGGRAPH 1993*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, ACM, 263–270.

COMPUTATIONAL DYNAMICS. 2001. *StarCD 3.15 Methodology*.

FUHRMANN, A., AND GRÖLLER, E. 1998. Real-time techniques for 3D flow visualization. In *Proceedings of IEEE Visualization 1998*, IEEE Computer Society Press, IEEE Computer Society, 305–312.

INTERRANTE, V., AND GROSCH, C. 1997. Strategies for effectively visualizing 3D flow with volume LIC. In *Proceedings of IEEE Visualization 1997*, IEEE Computer Society Press, IEEE Computer Society, 421–424.

MATTAUSCH, O., THEUL, T., HAUSER, H., AND GRÖLLER, M. E. 2003. Strategies for interactive exploration of 3D flow using evenly-spaced illuminated streamlines. In *Proceedings of the Spring Conference on Computer Graphics (SCCG) 2003*.

NVIDIA. NVIDIA Developer Relations. http://developer.nvidia.com.

OPENGL. OpenGL extension documentation. http://www.opengl.org/documentation/extension.html.

SCHROEDER, W., MARTIN, K., AND LORENSEN, B. 1998. *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*, second ed. Prentice Hall.

SGI. OpenGL extension registry. Maintained by SGI and available online at http://oss.sgi.com/projects/ogl-sample/registry.

ZÖCKLER, M., STALLING, D., AND HEGE, H.-C. 1996. Interactive visualization of 3D-vector fields using illuminated streamlines. In *Proceedings of IEEE Visualization 1996*, IEEE Computer Society Press, IEEE Computer Society, 107–113.