Hardware Accelerated Visualization of Curvilinear Vector Fields

Frank Reck, Christof Rezk-Salama, Roberto Grosso and Günther Greiner.

University of Erlangen, Department of Computer Science
Computer Graphics Group
Am Weichselgarten 9, 91054 Erlangen, Germany
Email: fkreck@immd9.informatik.uni-erlangen.de

Abstract

We present a novel method for hardware-accelerated texture advection of 3D velocity fields defined on curvilinear grids. For uniform rectilinear grids, texture advection can be efficiently performed within the rasterization unit by existing approaches. In a pre-processing step, the vector field is transformed from the curvilinear grid (P-Space) to a uniform rectilinear grid (C-Space). Hardware accelerated texture advection is performed via per-pixel operations in C-Space. The resulting 3D volume data is displayed by direct volume rendering via 3D textures. The decomposition of the volume object into viewport-aligned slices is performed in P-Space and the resulting polygons are textured with the image information calculated in C-Space.

1 Introduction

3D vector data sets, as they arise from measurement or numerical simulation, are frequently used in fluid mechanics, hydrodynamics, electric engineering and computational science. In recent years a number of different techniques for the visualization of 3D-flow phenomena have been developed.

Traditionally, geometric techniques are used, which represent the vector quantities by some kind of geometric primitives, such as arrows, icons [7] or glyphs [3]. More advanced geometric approaches numerically integrate particle paths, which result in stream lines, streak lines and stream surfaces [5]. The main difficulty with such approaches in practice is their restriction to a rather coarse spatial resolution. Displaying a high number of pathlines can easily lead to cluttered images.

As an alternative, texture-based approaches have gained increasing attention. The introduction of texture advection and line integral convolution (LIC) [2] significantly improved the visualization of vector fields for the 2D case. LIC is an efficient technique to depict flow information in an intuitive way by transforming the vector field into a scalar field. In the 3D case however, difficulties arise in the visualization of the intricate structures inside the resulting volume data set. Lisa Forssell [4] presented an extension that allows the mapping of LIC images onto curvilinear surfaces in 3D. Victoria Interrante [6] has developed techniques to visualize 3D LIC by the use of fuzzy clipping objects and sparse noise textures with additional depth cues. Animation techniques for 3D LIC textures have been presented by Rezk-Salama et al [9].

More recently, Weiskopf et al. [1] presented an interesting method to perform texture advection via programmable per-pixel operations. We will have a closer look at their approach in Section 2. A disadvantage of this method however is its restriction to vector fields on uniform rectilinear grids.

In computational science and engineering structured curvilinear grids based on hexahedral cells are often preferred with respect to numerical simulation. Curvilinear hexahedral grids are able to perfectly fill out the space around CAD surfaces while maintaining topological properties which are advantageous for numerical solver algorithms. To these ends we present an extension of the original texture advection algorithm which adapts the method to curvilinear grid structures. The basic idea of our novel approach is a transformation of the vector field from its physical space (P-Space) into a rectilinear grid in computational space (C-Space) [10]. Texture advection is solved within the C-Space and the resulting texture is transformed back to P-Space. In order to display the resulting textures in P-Space, we adapt texture-based volume rendering to curvilinear hexahedral cells.

The remainder of the paper is structured as follows. Section 2 is a short flashback which outlines the original implementation of hardware-accelerated texture advection. In Section 3 we describe how the transformation of the curvilinear grid from P-Space to C-Space is achieved. Section 4 presents our adaptation of the slice decomposition algorithm for 3D texture based volume rendering to allow for curvilinear cell structures. In Section 6 we evaluate the results of our approach and discuss possibilities to further improve the performance. In Section 7 the contributions of our paper are summed up.

2 Texture Advection in Hardware

The path of a mass-less particle in a given vector field $\vec{v}(\vec{x})$ is determined by numerical integration. Starting at given initial position $\vec{x}_0$, the calculation of the particle trace requires the solution of an ordinary differential equation (ODE)

$$\frac{\partial \vec{x}}{\partial t} = \vec{v}(\vec{x}). \tag{1}$$

Traditionally, numerical integration methods such as Euler, Heun or Runge-Kutta methods are used to solved the ODE. These methods mainly differ in their numerical accuracy and in the number of vector interpolations which are required for one integration step.

Weiskopf et al. [1] have demonstrated how a large number of Euler steps can be computed simultaneously using the capabilities of modern rasterization hardware. Depending on the input texture, which represents the initial state of the algorithm, single particle traces can be computed as well as dense structures of stream lines similar to line integral convolution.

The key features that allow hardware accelerated texture advection are dependent textures or pixel textures. Dependent textures are build upon the multi-texturing capabilities of modern graphics hardware. Multi-texturing refers to a rasterization unit which allows one polygon to be textured with two or more independent texture maps. With dependent textures the texture coordinates for the second texture are extracted from the first texture. This concept allows the specification of texture coordinates on a per-fragment basis.

The idea of texture advection via per-pixel operations is to encode the x-, y- and z-components of the vector field in the RGB color components of a dependent texture. The input texture is rendered into the frame buffer with the texture coordinates specified by the dependent texture. As a result, the frame buffer contains the original color values of the input texture shifted to its new positions after one Euler step. The contents of the frame buffer is copied back into an output texture, which is again used as input texture for the next integration step. The sequence of operations is illustrated in Figure 1.

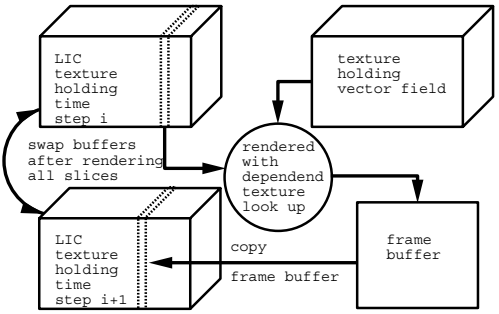


Figure 1: Hardware-accelerated texture advection is performed by rendering the input texture with a dependent texture that stores the shifted positions according to the vector field. The frame buffer is copied to an output texture. For the next integration step input and output textures are swapped.

The main benefit of this approach is the speed-up, which relies on the fast rasterization capabilities of modern graphics cards. This, however, restricts the original application to uniform rectilinear grids.

Our adaptation of the described algorithm to curvilinear grids requires two steps. In the C-Space the hardware-accelerated texture advection algorithm can be used as usually. However, we first have to accurately transform the vector field from its (curvilinear) P-Space to the (rectilinear) C-Space. After the texture advection the results must be displayed in P-Space, which requires the resulting texture to be transformed back. Additionally, the volume rendering algorithm must be adapted to curvilinear structures, which will be described in Section 4.

3 Transformation

As outlined in Figure 2, transformations of a given vector field from P-Space to C-Space and back do

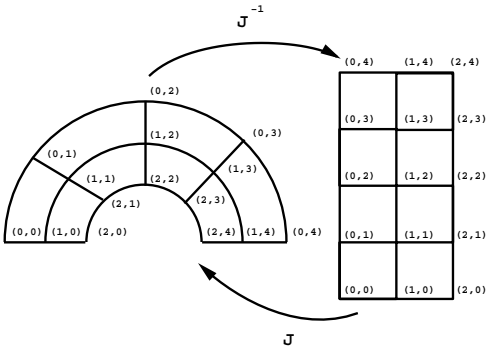


Figure 2: Linear approximations of the transformations of a given data set from the physical space (P-Space) to the computational space (C-Space) and back are represented by the Jacobian matrix and its inverse.

not change the topological structure of the grid. The neighborhood relations of cells and vertices are preserved. Solely the positions of the grid points are modified. After performing the transformation to C-Space the hexahedral cells become axis-aligned cubes.

The exact transformation Φ from P-Space to C-Space is known only at the grid nodes (i, j, k). In order to transform the velocity field, we do not need Φ but its Jacobian J_Φ [10]. For the inverse transformation Φ^{-1}, which maps the results back to P-Space, the Jacobian $J_{\Phi^{-1}}$ is replaced by the inverse matrix $(J_\Phi)^{-1}$.

The Jacobian J_Φ is computed using central differences on the discrete data. Since the transformation from P- to C-Space is performed only once in an initial pre-processing step, the time consumed for estimating the directional derivatives is not relevant.

Every velocity vector of the given flow field is then transformed with the inverse Jacobian, according to

$$\vec{u} = (J_\Phi)^{-1} \cdot \vec{v} \tag{2}$$

The hardware based texture advection requires a texture field with a dimension which is a power of two in each direction. Thus, the velocity field is resampled in C-Space using trilinear interpolation. Aliasing effects can be avoided by oversampling the data, e.g. a data set with dimensions $40 \times 30 \times 30$ is resampled to $64 \times 32 \times 32$. The x-, y- and z-components of the resulting velocity vectors in C-Space are stored in the RGB components of a 3D texture. This texture is used for hardware accelerated texture advection as described in the approach by Weiskopf et al [1]. After the Euler iteration is performed by the rasterization hardware, the original algorithm displays the resulting 3D texture using normal texture-based volume rendering. In our case, however, the user wants to examine the results in the physical space. This requires a modification of the original texture based volume rendering algorithm as it is described in the next section.

4 Volume Rendering on Curvilinear Grids

In traditional volume rendering applications, the scalar data value is mapped to physical quantities, that describe the amount of light which is emitted and absorbed at each point. These physical quantities are then used to synthesize virtual images. Therefore the emitted and absorbed light intensities are integrated along rays of sight. The image generation requires a high number of interpolation operations which result from this resampling of the volume data set at discrete positions along the rays.

The texture units of modern graphics hardware are capable of computing a high number of interpolation operations per second. In order to use the rasterization hardware for volume rendering, the volumetric object must be decomposed into a proxy geometry, which consists of geometric primitives that are *understood* by the underlying hardware.

The volume data set is decomposed into a stack of viewport-aligned slices by calculating the intersection polygons between the volume's bounding box and a stack of planes parallel to the image plane. The numerical integration of the emitted and absorbed light is approximated by blending of successive slices from back to front with respect to the viewing direction. Very efficient procedures for calculating intersection polygons between an axis-aligned box and a set of parallel planes exist [13, 8, 11].

In case of a curvilinear grid, there is no simple bounding box that can be used for the intersection computation. We have experimented with different strategies to compute the intersection poly-

gons. Intersection calculation must be performed at least with every outer boundary edge of the data set. In order to accurately model the non-linear inverse transformation from C-Space to P-Space, which is applied during texture application, it is not sufficient to specify texture coordinates only at the vertices of the intersection polygons.

The first straight-forward idea was to determine the cross sections of every cell with the current slicing plane. This results in a high number of small triangles, but the texture transformation is modeled accurately. Optimizing the intersection calculation e.g. by computing the edge intersection points for successive slice planes iteratively will significantly reduce the computational cost. In the following section we will discuss acceleration techniques to trade accuracy for speed.

5 Acceleration Techniques

There are two different strategies for performing the slice decomposition for direct volume rendering, both of which allow the gradual trading of visual accuracy for rendering speed. The first approach initially calculates a coarse approximation to the original slice polygon which is improved step-wise. The second approach constructs slice polygons for a given level of detail in one step.

In our basic implementation we have computed intersection polygons with every grid cell. In our first approach to remove the high number of intersection calculations, we only intersect every outer boundary edge of the polygon and construct one large polygon from the resulting intersection points. If we render this polygon as it is, the nonlinear texture transformation from C-Space to P-Space will be exact only at the vertices and very inaccurate in the interior of the polygon. More accurate approximations can be generated gradually by inserting additional vertices with correct texture coordinates in the interior of the polygon. The first vertex is inserted in the middle of the polygon and connected to every boundary vertex. Additional vertices are inserted in the middle of every inner edge, which splits every triangle into four new triangles. The refinement process is stopped if a subdivision depth is reached which is defined with respect to a specified accuracy. A similar approach has been applied for non-linear volume deformation [12].

The second alternative to trade accuracy for speed is to merge a specified number of hexahedra together and treat the result as one large hexahedron for intersection calculation. As an example a cluster of eight hexahedra, which share a common vertex are joined together to form one single hexahedron. The approximation error that is introduced by this clustering strongly depends on the structure of the curvilinear grid. In Fig. 5(a) the intersection areas of the viewport-aligned slices with a cluster are shown. In Fig. 5(b) the resulting textured polygons are depicted. In the typical case of a cylindrical data grid, the clustering technique reduced the hexahedra to be rendered from 4096 to only 16 without a noticeable loss in accuracy.

6 Results and Discussion

The methods that we have described in this paper have been applied and tested on several 3D vector fields on curvilinear grids. We remark, that the presented approach is well applicable for 2D flow fields defined on curvilinear grids.

The described algorithms strongly depend on the capabilities of the underlying hardware architecture. Up until now, there are only very few PC graphics boards that support dependent 3D textures, such as the ATI Radeon 8500 graphics boards and NVidia GeForce 4 Ti family boards. Our experiments were performed on an SGI Octane 2 with VPro graphics subsystem. A dependent texturing mechanism is available on this architecture in the form of pixel textures. Since this architecture does not support multi-textures, the first texture must be written to the frame buffer and the dependent texture lookup is performed by a copy step, which reads the pixels from the frame buffer and re-inserts them into the pixel pipeline. We are currently working on an implementation using *native* dependent textures for the above mentioned PC graphics boards.

The first test data set was a synthetically generated vector field of dimensions $16 \times 16 \times 16$ and represents a cylinder. The dimension is a power of two such that the vector advection could be performed without any further processing step. The velocity field is circular, thus a particle trace would be a circle. The second data set is the result of a computational fluid dynamics simulation around the wing of an airplane. The dimensions of this data set are $100 \times 78 \times 38$. In C-Space, the vector field was re-

sampled onto a grid of dimensions $128 \times 64 \times 32$. Finally, we have carried out measurements with the blunt fin data set, which describes an airflow over a flat plate with a blunt fin rising from the plate. The data set has dimensions $40 \times 32 \times 32$, and in C-Space was resampled onto a grid of dimensions $64 \times 32 \times 32$.

In Figure 3(a) and 3(b) the hardware based texture advection for the cylindrical test data set is shown. The texture advection represents a straight line in C-Space and a circle in P-Space which clearly corresponds to the input velocity field. An attenuation or wakening of the intensity in the texture is simulated such that the user can recognize the flow direction, as the dark region is the starting location of the particles. The rendering times for P-Space are given in Table 1. The Table gives the times obtained without and with the clustering strategy. Using clustering an interactive frame rate can be achieved.

The results obtained for the airplane wing data set are shown in Fig. 4. The lower Figure was obtained using the clustering acceleration. The upper Figure was obtained using the high resolution method by slicing all volume cells. The clustering technique improves the performance by a factor of 81, where the size of one cluster is $1 \times 1 \times 38$ of an original cell. Clearly, the accelerated technique produces acceptable results as it can be appreciated in Fig. 4.

The time measurements for the blunt fin data set are given in Table 1. A corresponding flow visualization obtained with the clustering method is given n in Fig. 5(c) which shows a row of injected particles placed near the sharp bend of the tube. For this data set we obtain one frame each two seconds which clearly shows the efficiency of our method.

Data set	Time	Time with clustering
blunt fin	59	2
wing	491	6
synthetic	7	0.2

Table 1: Time in seconds needed for rendering curvilinear grids with and without clustering

7 Conclusion and Future Work

The most time consuming step of the algorithm is the rendering of the textured polygons in P-Space. The usage of bounding hexahedra or bounding spheres could speed up the intersection tests, since the intersection point calculations are considerably reduced. A further possibility is to introduce an edge based data structure, such that each edge intersection is computed only once.

The image quality can be improved by subdividing the polygons into triangles. The texture values at the new vertices are then directly computed from the C-Space texture, thus obtaining a better approximation to the texture in P-Space.

The definition of the initial texture in C-Space is a difficult task. It may require some tests before one obtains the desired texture distribution. In order to make the system more user friendly, a correspondence between the initial texture distribution in C-Space and in P-Space have to be defined.

8 Acknowledgments

The research was supported by the German Science Foundation DFG, which funds the Sonderforschungsbereich 603 Modellbasierte Analyse und Visualisierung komplexer Szenen und Sensordaten, Teilprojekt C7 Adaptive Verfahren zur Berechnung und Visualisierung von mechatronischen Sensoren und Aktoren. Many thanks to Matthias Hopf and Daniel Weiskopf from the Visualization and Interactive Systems Group, University of Stuttgart, the group of Stuttgart for supporting me with the code and some advice.

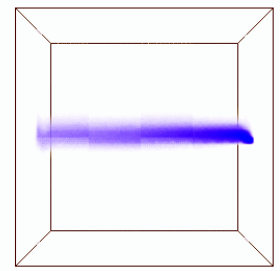
References

[1] Hardware-accelerated visualization of time-varying 2d and 3d vector fields by texture advection via programmable per-pixel operations. In T. Ertl, B. Girod, G. Greiner, H. Niemann, and H.-P. Seidel, editors, *VMV 2001*, Stuttgart.

[2] B. Cabral and L. Leedom. Imaging Vector Fields Using Line Integral Convolution. In *Proc. SIGGRAPH*, 1993.

[3] W. C. de Leeuw and J. J. van Wijk. A Probe for Local Flow Field Visualization. In *Visualization '93*, San Jose, California. IEEE Computer Society.

[4] L. Forssell. Visualizing Flow Over Curvilinear Grid Surfaces Using Line Integral Convo-



(a) Texture advection for the synthetic data set in C-Space. The particle traces are straight lines.

(b) The results of the texture advection in C-Space are rendered in P-Space. Clearly, the flow is circular. The texture attenuation indicates the flow direction.

Figure 3: Hardware based texture advection in C-Space and in P-Space for our test data set.

lution. In *Proc. IEEE Visualization*, Washington, 1994.

[5] J. Hultquist. Interactive numerical flow visualization using stream surfaces. In *Computing Systems in Engineering*, pages 349–353, 1990.

[6] V. Interrante and C. Grosch. Visualizing 3D Flow. *IEEE Computer Graphics and Applications*, 18(4):47–53, 1998.

[7] F. J. Post, Theo van Walsum, F. Post, and D. Silver. Iconic Techniques for Feature Visualization. In G. M. Nielson and D. Silver, editors, *Visualization '95*, pages 288–295, Atlanta, Georgia, 1995. IEEE Computer Society, IEEE Computer Society Press.

[8] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, and T. Ertl. Interactive Volume Rendering on Standard PC Graphics Hardware Using Multi-Textures and Multi-Stage Rasterization. In *Proc. SIGGRAPH/Eurographics Workshop on Graphics Hardware*, 2000.

[9] C. Rezk-Salama, P. Hastreiter, C. Teitzel, and T. Ertl. Interactive exploration of volume line integral convolution based on 3d–texture mapping. In *Proc. IEEE Visualization*, 1999.

[10] A. Sadarjoen, T. van Walsum, A. J. S. Hin, and F. H. Post. Particle Tracing Algorithms for 3D Curvilinear Grids. In *Fifth Eurographics Workshop on Visualization in Scientific Computing*, 1994.

[11] R. Westermann and T. Ertl. Efficiently Using Graphics Hardware in Volume Rendering Applications. In *Proc. SIGGRAPH*, 1998.

[12] R. Westermann and C. Rezk-Salama. Real-Time Volume Deformation. In *Computer Graphics Forum (Eurographics 2001)*, 2001.

[13] O. Wilson, A. Van Gelder, and J. Wilhelms. Direct Volume Rendering via 3D-textures. Technical Report UCSC-CRL-94-19, Univ. of California, Santa Cruz, 1994.

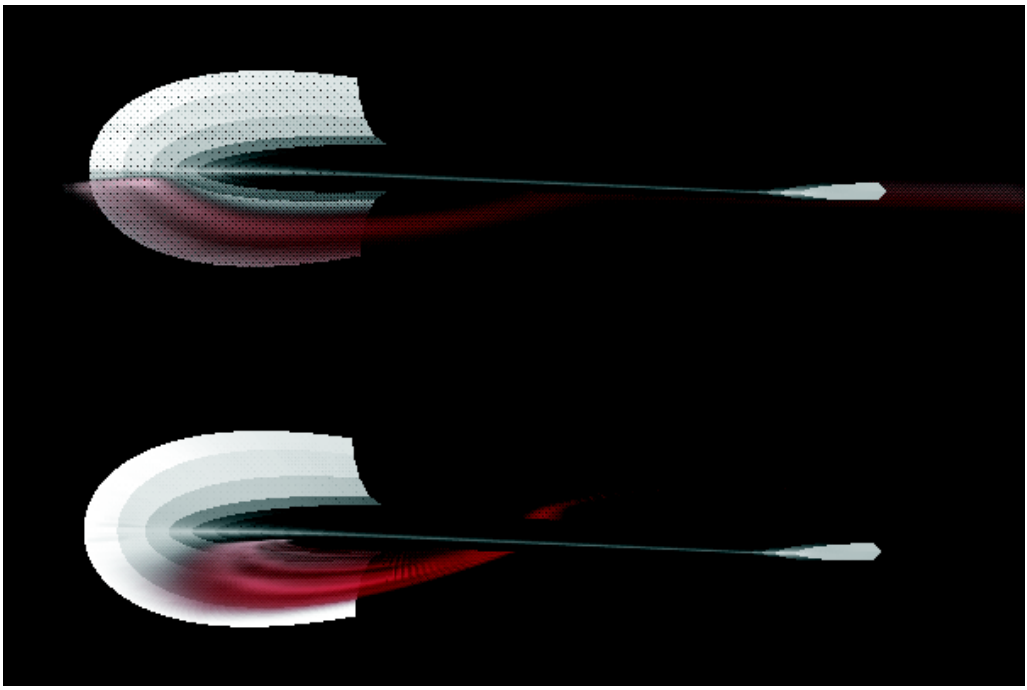
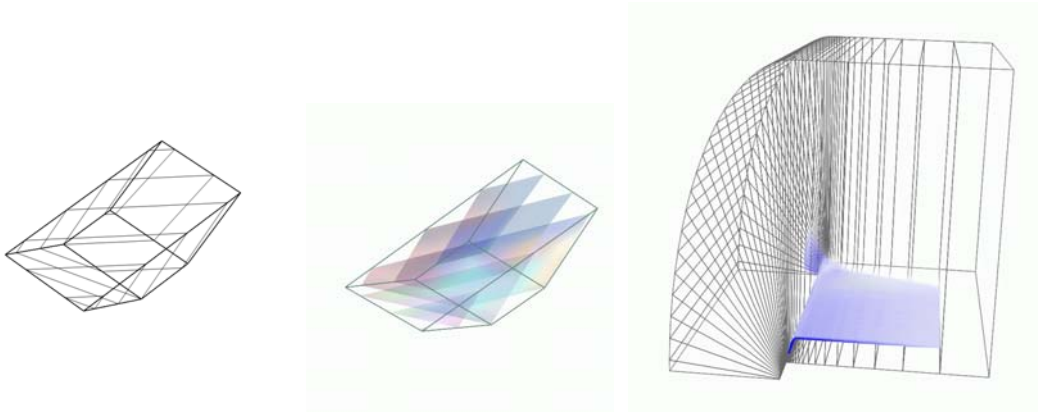


Figure 4: In this picture the flow around the airplane wing is shown. The texture advection with clustering is shown in the lower picture. The upper picture shows the same configuration but without clustering. In both cases the same initial texture was used.



(a) Every cluster has to be sliced separately in P-Space. This picture shows the intersection of the viewport-aligned slices with a cluster.

(b) The intersection areas are filled up using the texture values in C-Space.

(c) This Figure shows a ribbon over the ground for the blunt fin data set.

Figure 5: Hardware accelerated texture advection