Exploring Instationary Fluid Flows by Interactive Volume Movies

Thomas Glau

DaimlerChrysler AG, Research and Technology, Virtual Reality Competence Center, Wilhelm-Runge-Strasse 11, D-89013 Ulm, Germany
Thomas.Glau@DaimlerChrysler.com

Abstract. Volume rendering offers the unique ability to represent inner object data and to realize enclosed structures "at first glance". Unlike software-based methods, the use of more and more available special-purpose hardware allows volume rendering at interactive frame rates - a crucial criterion for acceptance in industrial applications, e.g. CFD analysis. Careful optimizations and the exclusive use of hardware-accelerated data manipulation facilities even enable volume rendered movies supporting real time interactivity. This article presents the most important features and implementation issues of an *OpenInventor*-based stereoscopic, VR-featured volume rendering system for instationary datasets.

1 Introduction

In 3D fluid flow analysis simulations are usually performed by means of the finite element approach using locally adapted, unstructured meshes. As a result, scalar and vector quantities for a large number of cells (varying from 100.000 to >5.000.000) are generated and need to be made accessible to human perception.

Direct volume rendering proved to be an incomparable tool for getting a deeper insight into complex datasets. Because of its ability to show the dataset as a whole, flow structures (*features*) often can be recognized "at first glance". Nevertheless, volume rendering still suffers from computational expense. Hence, it can be hardly found in widespread industrial applications and it is almost exclusively used for stationary data analysis.

This article shows how to implement a volume renderer on a high end graphics workstation, fast enough for playing volume movies with full real time interactivity. Interactivity includes performing geometric transformations as well as feature extraction and data manipulation. By integrating the volume renderer into an existing virtual reality platform, hybrid rendering and stereoscopic viewing are provided as well as several virtual reality features, e.g. tracking.

Note that simulation and visualization are parts of an iterative procedure: parameters are modified until certain optimization criteria are fulfilled. Therefore, fast execution of each process involved in this loop is not only highly desirable, but rather absolutely essential. Considering this practice we can assume that in an early optimization phase interactivity is the major requirement for visualization. However, approaching to the final step fidelity turns to be the more important criterion. The user would neither accept a maximum quality rendering at low frame rates nor a high performance system with too much data loss. Before talking about movies we have to discuss some common details concerning fast volume rendering and feature extraction techniques.

2 Design issues

Even there are many visualization algorithms for direct volume rendering of unstructured grids, real-time interactivity is not achieved [6][16]. To realize a *today*-usable system meeting the requirements discussed above, the use of hardware-accelerated 3D textures seems to be the most promising way. In fact, the availability of this hardware increases permanently while its costs decrease rapidly. Meanwhile, 3D texture hardware is offered by a few vendors producing PC-based graphics systems. The OpenGL standard API enables flexible integration of volume objects into existing surface rendering systems. As three-dimensional texturing must be supported by all OpenGL 1.2 implementations, it actually becomes a standard feature.

The corresponding rendering approach, *slicing*, expects scalar volume data represented as a 3D texture. First, polygonal planes are rendered parallel to the viewport and stacked along the viewing direction. By mapping the 3D texture while rendering the polygons, the volume data are sampled along the planes according to a hardware-accelerated interpolation scheme, usually trilinear interpolation. The textured polygons are blended together in back-to-front order resulting in the final pixel image [4][14]. Here, the physical model for light transport is simplified according to the used blending operator [8][10].

Naturally, 3D textures are limited to Cartesian-grid data. Hence, we have to convert the simulation data from the original unstructured FE-grid by a preprocessing step, expecting some data loss (see chapter 3). After resampling, the voxel representation offers significant advantages for following postprocessing steps. Because it can be regarded as the three-dimensional analogue to the pixel model, a huge number of well-working image processing algorithms can be extended for use in three dimensions - often in a straightforward manner. This includes volume enhancement as well as segmentation and pattern recognition procedures [9].

The maximum resolution of the grid depends on the available texture memory and on further restrictions concerning the maximum texture size of the underlying hardware. Using a SGI *ONYX InfiniteReality* graphics workstation with 16 MB texture memory and four RM6 raster managers enables simultaneous visualization up to four 128^3 x 16 Bit Luminance-Alpha textures without swapping [1][11]. The luminance component is used for scalar value representation while the alpha component contains an optional classification tag. Displaying a dataset in a translucent view is the most important application for volume rendering. But often, the user gets confused with this because of missing depth cues and shape hints within the gel-like volume. Stereoscopic viewing in perspective projection avoids this weakness in a convincing fashion. So we emphasize that stereoscopic viewing is really an essential key feature for serious data analysis using volume rendering.[2][4][5]

3 Data generation

To generate Cartesian-grid data from the unstructured finite element grid input, a 3D *scan-conversion* has to be performed. Here, each FE-cell (hexahedron) is rasterized individually by slicing it into a set of polygons so that a standard 2D rasterization algorithm can be applied. After the voxel set occupied by the cell is determined, trilinear interpolation is used to obtain voxel data from the vertex data of the cells. Linear interpolation schemes are usually preferred because they are fast, first-order continuous and only require data at the vertices of the bounding cell [7]. Compared to image-order resampling techniques this object-order method avoids exhaustive cell searching for point location within the unstructured grid, but the higher the cell-to-voxel ratio becomes, the more the conversion quality reduces. For higher resolution, the conversion procedure can be spatially limited to a user-defined subvolume. To speed up the calculation, the algorithm was parallelized by scheduling chunks of cells to be scan-converted to all available CPUs. The volumetric dataset shown in Fig. 3 consists of 190.080 cells and needs 12.5 secs to be scan-converted into 2.097.152 voxels on a 4x*R10000* 194 MHZ SGI *ONYX*-system.

4 Minimizing the Rendering Costs

While rendering, performing the hardware-accelerated texture interpolation is the most expensive task. To overcome this bottleneck, the polygon stack is usually clipped to the limits of the texture volume instead of drawing simple rectangles (see Fig. 1). Considering the fact that all slicing planes are held parallel to the screen, the clipping algorithm is less complex compared to the general case and can be implemented quite easily with a few lines of code shown below: First, the object transform is applied to both the 3D texture and to each vertex of the bounding box. Then, the parametric representation and some auxiliary parameters of all edges are determined:

```
foreach edge
  if (dz != 0)
      h = dx / dz
      k = dy / dz
      a = p0.y - k * p0.z
      b = p0.x - h * p0.z
      zmin = MIN(p0.z, p1.z)
      zmax = MAX(p0.z, p1.z)
      is_parallel = False
  else
    is_parallel = True
end
```

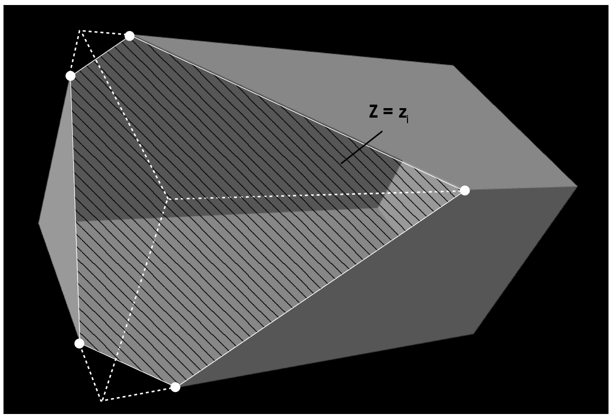

Fig. 1. Slicing a 3D texture (shown as bounding box)

where *p0* and *p1* are the end points of the edge and $dx \ldots dz$ denote the differences between them with respect to the appropriate direction. With these parameters, the intersection points p_i of the edges with each plane of the polygon stack can be easily calculated, provided that the line is not parallel to the xy-plane and the intersection really exists. Finally, to render the polygon properly the convex hull of the intersection points has to be determined using a robust algorithm taken from [17].

```
foreach plane
  z += plane_distance
  foreach edge
    if (!edge.is_parallel && (z >= edge.zmin) && (z <= edge.zmax))
      p[nvertices].x   = z * edge.h + edge.b
      p[nvertices++].y = z * edge.k + edge.a
  end
  convex_hull(p, nvertices)
  render_polygon(p, z, nvertices)
end
```

The calculation procedure for a polygon stack composed of 128 slices takes typically 1 msec average calculating time on a *R10000* 194 MHZ CPU and improves rendering speed significantly when applied to non-uniformly scaled textures. Thus, frame rates of >30 Hz in a 1000x750 window are usually achieved. Using a software-clipping algorithm produces no load for the geometry engine and avoids wasting of hardware-supported clipping planes which are mostly limited to a small number even on high end graphics systems.

5 Feature Extraction

Usually, the user wishes to have a volume composed of *features* rather than visualizing the raw data. Preserving real time interactivity, threshold segmentation is performed by manipulating the hardware-accelerated RGBA-Texture-Look-Up-Table (LUT) which maps each luminance component to a RGB pseudocolor and each alpha component to another alpha value, respectively. To gain more flexibility, a virtual LUT (stored in main memory) is introduced for each single feature. The RGB LUT values are derived from an HSV colorbar where H originally moves from 0 . . . 1 while S and V are set to unity to get a color spectrum. The H mapping can be modified by scaling and shifting and is therefore of first order (see Fig. 2). Unlike color mapping, alpha mapping is of zero order, i. e. constant over the whole luminance band delimited by the *value/width*-pair in Fig. 2. Finally, all tables masking and coloring a single feature (scalar band) are merged to yield the resulting transfer function.

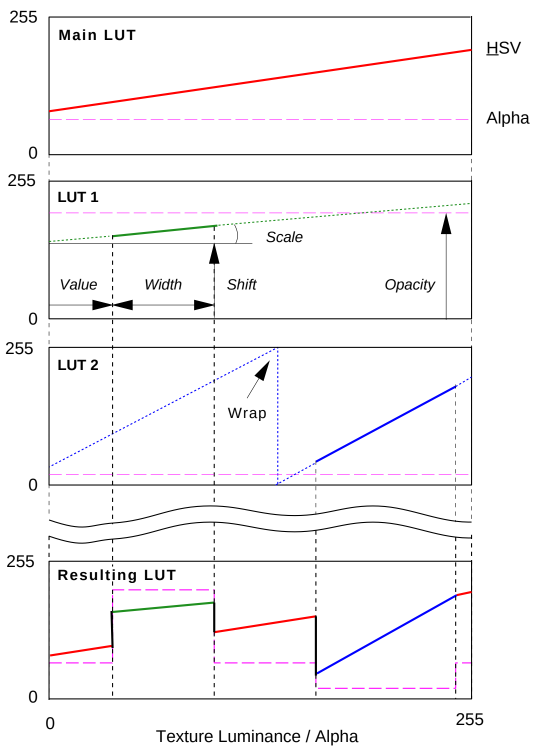

Fig. 2. LUT Merging. Both scalar band size and coloring of each feature can be adjusted very intuitively by five parameters shown in the second diagram.

Sometimes it is desirable to obtain iso-surfaces but to avoid the expensive extraction of a triangular mesh. We can achieve a surface-like impression by choosing a relatively small scalar band around the iso-value which is mapped to a full opaque, bright color (e.g. yellow). After removing dispensable pieces of the projection slices by an alpha-test, the polygon stack is depth shaded with decreasing intensity according to a square function. In addition, specular lighting improves the spatial impression when moving the object, although the surface normals of the polygons are considered for lighting calculation rather than the normals of the real iso-surface.

A more conventional way for volume exploration by clipping planes is also provided. They are realized by spacemouse-driven rectangles sampling the texture along the plane using the OpenGL texture coordinate generating function. Alternatively, OpenGL clipping planes can be applied to cut the polygon stack. Since texture sampling works with arbitrary geometries, data projection on more sophisticated surfaces (e.g. human model) is possible. More comprehensive techniques for hardware-assisted clipping and shading can be found in [15]. Furthermore, a dynamic refinement functionality is provided to improve the rendering quality of motionless volume objects by increasing the number of slicing polygons.

6 Volume Movies

Having implemented a fast 3D texture based volume renderer, generating volume movies is relatively straightforward. Therefore, texture data for each time step are kept into main memory. While playing the movie the current data in texture memory used for rendering are replaced by applying the OpenGL *glTexSubImage3D* extension. Since the download takes about 70 msecs on our hardware, volume exchange rates of approximately 10 Hz are usually achieved. Note that all capabilities described above, like clipping planes and feature extraction, are now available for instationary dataset exploration.

7 Conclusions

This case study describes features and applications of our of hardware-assisted hybrid rendering system. The 3D texture slicing approach is regarded as the most advantageous solution currently available for 3D *signal* analysis, where insight into the dataset is more important than surface inspection as required for volume graphics. Here, simplified light models work well because there is no natural equivalent. Care should be taken when interpreting rendered images from datasets containing discontinuities, like holes, where trilinear interpolation yields misleading results along the interface. Furthermore, artefacts may occur due to perspective distortion if the density of slices doesn't increase with distance from the viewer. A number of remarkable properties makes 3D texture slicing appealing for industrial use: High performance, good-quality rendering, real time surface visualization, immediate low-level feature extraction, arbitrary scan geometries, inherent stereo viewing, standard graphics API, compatibility with existing surface rendering systems and finally - volume movies. For CFD visualization, this technique requires an additional resampling step but offers real 3D data analysis and the application of a wide range of postprocessing algorithms already known from the 2D domain. Today, 3D texture hardware is fairly limited concerning resolution and data depth. Therefore, it is mainly used either to get a coarse overview over the entire dataset or to get a closer look at a relatively small subvolume. But considering the fact that 3D texture mapping becomes a standard feature, advanced hardware support is expected to be available soon.

References

1. K. Akeley, *Reality Engine Graphics*, ACM Computer Graphics, Proc. SIGGRAPH '93, pp. 109-116, July 1993
2. D. S. Ebert, R. Yagel, J. Scott, Y. Kurzion, *Volume Rendering Methods for Computational Fluid Dynamics Visualization,* Visualization '94, Washington, DC, 1994, pp. 232-239
3. T. Elvins, *A Survey of Algorithms for Volume Visualization*, Computer Graphics, Vol. 26, No. 3, 1992
4. R. Fraser, *Interactive Volume Rendering Using Advanced Graphics Architectures,* Silicon Graphics, Inc., Technical Documentation
5. van Gelder, Kim, *Direct Volume Rendering with Shading via Three-Dimensional Textures*, Proc. Symp. on Volume Rendering, San Francisco, CA, ACM 1996, pp. 23-29
6. A. Kaufman*, Volume Visualization : Principles and Advances,* SIGGRAPH '98 Course Notes, Orlando, Florida, 1998
7. D. Kenwright, *Visualization Algorithms for Gigabyte Datasets,* SIGGRAPH '97 Course Notes, Los Angeles, CA, 1997, pp. 4-1 - 4-31
8. W. Krueger, *The Application of Transport Theory to the Visualization of 3-D Scalar Data Fields*, IEEE Visualization '90, pp. 273-280, 1990
9. G. Lohmann, *Volumetric Image Analysis*, Wiley-Teubner, 1998
10. N. Max, *Optical Models for Direct Volume Rendering,* IEEE Trans. on Visualization and Computer Graphics, 1, **2** (1995), pp. 99-108, 1995
11. J. Montrym, D. Baum, D.Dignam, C. Migdal, *Infinite Reality : A Real-Time Graphics System, Computer Graphics*, Proc. SIGGRAPH '97, pp. 293-303, 1997
12. C. Stein, B. Becker, N. Max, *Sorting and hardware assisted rendering for volume visualization,* ACM Symposium on Volume Visualization '94, pp. 83-90, 1994
13. M. Teschner, Ch. Henn*, Texture Mapping in Technical, Scientific and Engineering Visualization,* Silicon Graphics, Inc., Technical Documentation
14. R. Westermann, Th. Ertl, *Efficiently Using Graphics Hardware in Volume Rendering Applications*, Proc. SIGGRAPH '98, Orlando, Florida, 1998
15. R. Yagel, D. M. Reed, A. Law, P.-W. Shih, N. Shareef, *Hardware Assisted Volume Rendering of Unstructured Grids by Incremental Slicing*, ACM Symposium on Volume Visualization '96, pp. 55-63, 1996
16. http://cm.bell-labs.com/who/clarkson/

The volume renderer introduced here is part of the virtual reality platform DB*View, developed at the DaimlerChrysler Virtual Reality Competence Center. Simulations were performed by DaimlerChrysler Research and Technology, Dep. FT 1/AK.*

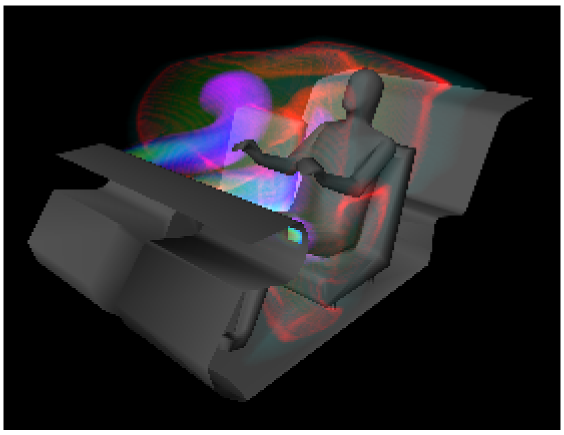

Fig. 3. Volume rendered temperature dataset inside a car cabin. LUT merging was applied for feature highlighting.

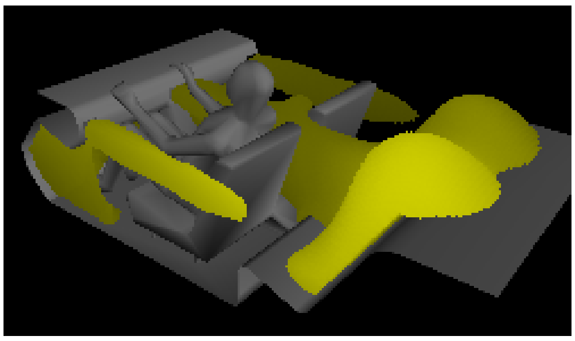

Fig. 4. Iso-Surface generated using a depth shading approach.

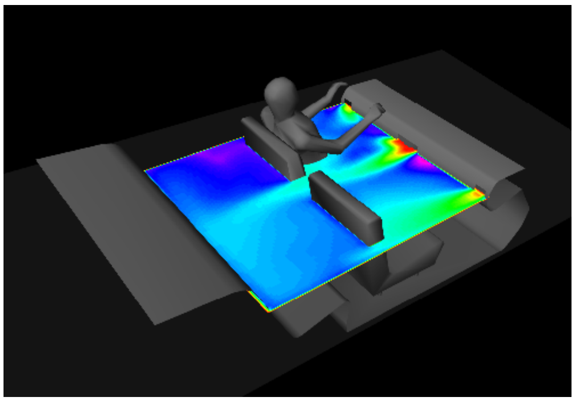

Fig. 5. Clipping plane.