GPU Improvements on the Sorting and Projection of Tetrahedral Meshes for Direct Volume Rendering

Sébastien Barbier[*] Georges-Pierre Bonneau[†]

LJK, Grenoble University, INRIA, CNRS

Abstract

Direct Volume Rendering is one of the most popular visualization techniques. Although approaches such as ray-casting or slicing are fast and well-implemented on graphics hardware for regular and irregular grids, cell projection techniques are still time-consuming for large tetrahedral meshes. We propose improvements to the pipeline of cell projection techniques based on the SXMPVO [4] and the *Projected Tetrahedra* [8] (PT) algorithms. Specifically, we exploit new functionalities of the latest graphics hardware to remove bottlenecks in the sorting and rendering phases.

1 Introduction

Direct Volume Rendering (DVR) is a useful technique in scientific visualization. Contrary to isosurface extraction, it solves occlusion issues and allows internal structures to be percieved in a global context.

DVR evaluates the light transport equation in the volume mesh. A front-to-back or back-to-front sorting of the cells is required to obtain a correct compositing. The contribution of each cell is accumulated to obtain the final rendering. Algorithms have been proposed for regular and irregular grids and many have been recently implemented on the GPU to ensure interactive display.

We focus on a two-step pipeline that ensures a correct rendering for non-convex and non-connected tetrahedral meshes. First, a sorting of the cells according to the viewpoint is performed with the SXMPVO algorithm [4]. Then, the cells are projected to compute the light transport through the GATOR splatting [11].

Nevertheless, their original implementations have some limitations concerning efficiency. For instance, the A-Buffer (phase III, see section 3.1) of the SXMPVO algorithm is computed on CPU and the GATOR splatting transmits extra data to the GPU and computes redundant projections.

We propose improvements of these two algorithms using the latest features of the new graphics hardware:

- the A-Buffer sorting implemented on the CPU is transformed into a depth-peeling algorithm on the GPU;
- the PT rendering is implemented with a geometry shader.

2 Previous Work

Ray-Casting approaches [2] launch rays from the viewpoint and integrate the light along each ray to obtain the correct rendering. Implemented on the GPU [9], this technique is strongly limited by the size of the graphics hardware memory.

In cell projection techniques, the ray integration is performed in object space. The Projected Tetrahedra (PT) rendering of Tuchman and Shirley [8] is the most popular technique and it has several GPU implementations [11, 7] . A prior visibility ordering of the tetrahedra is required and can be implemented, for instance, with the MPVO family of sortings [10, 4].

In contrast, the HAVS system [3] of Callahan et al. is an hybrid approach relying on two consecutive sortings of the faces of the tetrahedra: the former computes on CPU and the latter on GPU using a k-buffer [1]. But current GPU limitations can produce by-pixel incorrect sorting.

3 Tetrahedra Sorting

Cell projection techniques require a two-step pipeline since they are performed in object space. Cells are sorted, then rendered. In the next sections, we discuss our improvements using the new graphics hardware functionalities in comparison to the original algorithms.

3.1 The SXMPVO algorithm

The SXMPVO algorithm of Cook et al. [4] computes a total ordering of occlusions between tetrahedra for non-connected and non-convex meshes on the CPU through four phases, which are briefly recalled hereinafter. For more details, see [4].

Phase I determines a partial ordering of the tetrahedra into connected volumes based on the relations of adjacencies. Phase II sorts the border faces according to depth. Theses faces are transmitted to phase III to compute the relations of occlusions between non-connected tetrahedra using an A-Buffer. At the end of these three phases, we obtain a total ordering. Phase IV realizes a depth first search in order to display the tetrahedra.

The complexity of phase III is strongly correlated with the pixel area of the projected tetrahedra. Indeed, the A-Buffer builds a list of occlusions between sorted border tetrahedra projecting their faces into the image space. This is computed on the CPU. Therefore, the A-Buffer technique in a sense emulates consecutive rasterizations that would be accelerated on the GPU.

3.2 Sorting through Depth-Peeling

A GPU-based *depth-peeling* [5] replaces the CPU-based A-Buffer (phase III). This idea was introduced in [6] for the XMPVO sorting and cited as potential future work in [4].

Depth-Peeling consists of rendering the scene through consecutive passes, “peeling” at each pass the previous visible triangles until the scene becomes empty. The phases can be modified in the following way. The border faces recorded during the phase I are sent to the GPU for an off-screen depth-peeling rendering. During this process, the consecutive multi-pass renderings are fetched and joined by pairs on the CPU. Each output texture records the index of the displayed tetrahedra and each pair models a gap in the volume. Thus, each pair of colored pixels encodes a relation of occlusion in the image space between two border tetrahedra.

To make the depth-peeling algorithm efficient, Frame Buffer Objects are used for off-screen renderings. Communications between the CPU and the GPU are accelerated with Pixel Buffer Objects. As explained in [6], the screen image is split into tiles (4x4 in our implementation) to reduce the number of read-backs. Furthermore, masks of the previous renderings are kept in memory to minimize the access of tiles on the CPU.

[*]e-mail: sebastien.barbier@inrialpes.fr

[†]e-mail:georges-pierre.bonneau@inrialpes.fr

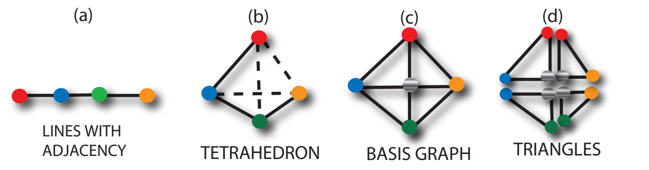

Figure 1: Geometry shader: A line (a) encodes a tetrahedron (b), mapped on the basis graph (c) to deduce the projection (d).

Consequently, the depth-peeling approach eliminates the original phase II and maps phase III on the GPU.

4 Tetrahedra Rendering

Once the exact sorting of the tetrahedra has been computed, the mesh can be displayed using cell projection. We propose a new and efficient single pass techniques that uses a geometry shader implementation of Shirley and Tuchman's original PT algorithm [8].

To improve the efficiency of the rendering pipeline, we transform the vertex shader proposed by Wylie et al. [11] into a geometry shader. As the OpenGL `GL_LINES_ADJACENCY_EXT` extension allows us to send four connected vertices to the GPU, we represent an input tetrahedron by a line with adjacencies. Its faces and edges can be easily reconstructed into the geometry shader (see Figure 1). The mapping to the basis graph and the interpolations are performed once, based on the mapping tables of GATOR, in the geometry shader. Triangle strips resulting from the projection of the tetrahedron are then sent to the fragment shader.

Our geometry shader implementation introduces several improvements in comparison to the two previous GPU-based implementations [11, 7].

A tetrahedron is sent only once instead of five times with the Wylie et al.'s vertex shader, with solely the position of its four vertices and their associated color. Thus, the size of the data sent to the GPU is significantly decreased. Each tetrahedron is projected and mapped to the basis graph only once with minor modifications of the original GATOR vertex shader. Degenerate tetrahedra are discarded and thus, no extra triangles are produced.

This solution avoids the two rendering passes proposed by Marroquim et al. [7]. Furthermore, the tetrahedral mesh is directly sent to the GPU without mapping the data into 2D textures for subsequent access in a fragment shader.

5 Results

We have collected our results on a Windows XP 32 bits PC with 2.8 GHz Xeon processor, 2GB of RAM and a GeForce 8800 GTS with 640 MB of Video RAM.

To validate our improvements to phase III of the SXMPVO algorithm and the PT hardware pipeline, we render the Blunt Fin tetrahedral mesh multiple times. Table 1 illustrates the time improvements obtained using depth-peeling for phase III and a geometry shader during phase IV with enough tetrahedra to provide relevant complexity. Different image resolutions and screen sizes are tested.

As the number of pixels increases, the new phase III becomes much faster than the original one. Zooming into the tetrahedral mesh is no longer time-consuming. The geometry shader is implemented with the mapping table proposed by Wylie et al. [11]. We also tried the mapping table of Marroquim et al. but the rendering times are more expensive (63 ms vs. 55 ms) without noticeable changes in the rendering. Overall rendering times are reduced 2.5 times compared to the original implementation.

Image Size	Original SXMPVO				Improved SXMPVO			
680x840	I+ II	III	IV	R	I	III	IV	R
0.3%	195	24	139	2.75	188	31	54	3.5
2.3%	257	87	138	2.1	185	33	55	3.5
31.1.%	381	476	138	1	185	42	55	3.4
99.9%	242	1629	139	0.5	187	66	56	3.1
945x1210	I+ II	III	IV	R	I	III	IV	R
2.1%	296	122	139	1.9	187	64	55	3.2
29.3%	338	1173	137	0.7	189	78	56	3.0
99.9%	204	2962	134	0.3	187	126	55	2.6

Table 1: Comparison of the rendering framerates for Direct Volume Rendering of two image resolutions. 121259 tetrahedra are displayed. Times are expressed in milliseconds for each phase (I,II,III,IV) of the Cook et al.'s SXMPVO. Global Rendering (R) is in frames per second. The percentage of the screen image filled by the projection of the tetrahedral mesh is indicated.

6 Summary

We have proposed improvements for the SXMPVO and the PT algorithms using the lastest functionalities of new graphics hardware to accelerate Direct Volume Rendering.

The A-Buffer of the SXMPVO algorithm is replaced by a depth-peeling algorithm to exploit the power of the GPU. To obtain a global sorting, each pairwise pass computes the occlusion relations between the non-convex or non-connected parts of the mesh. The PT rendering is done using a geometry shader. Only one pass is necessary with a reduced communication between the CPU and the GPU; each tetrahedron is projected once and only non-degenerated triangles are sent to the fragment shader.

References

[1] L. Bavoil, S. P. Callahan, A. Lefohn, J. L. D. Comba, and C. T. Silva. Multi-fragment effects on the gpu using the k-buffer. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 97–104, 2007.

[2] P. Bunyk, A. Kaufman, and C. T. Silva. Simple, fast, and robust ray casting of irregular grids. *Proceeding of the Dagstuhl, Scientific Visualization Conference*, pages 30–36, 1997.

[3] S. Callahan, M. Ikits, J. L. D. Comba, and C. T. Silva. Hardware-assisted visibility ordering for unstructured volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 11(3):285–295, 2005.

[4] R. Cook, N. Max, C. T. Silva, and P. L. Williams. Image-space visibility ordering for cell projection volume rendering of unstructured data. *IEEE Transactions on Visualization and Computer Graphics*, 10(6):695–707, 2004.

[5] C. Everitt. Interactive order-independent transparency. may 2001.

[6] S. Krishnan, C. Silva, and B. Wei. A Hardware-Assisted Visibility-Ordering algorithm with applications to volume rendering. In *Data Visualization*, pages 233–242, 2001.

[7] R. Marroquim, A. Maximo, R. Farias, and C. Esperanca. Gpu-based cell projection for interactive volume rendering. *SIBGRAPI: Brazilian Symposium on Computer Graphics and Image Processing*, 0:147–154, 2006.

[8] P. Shirley and A. A. Tuchman. Polygonal approximation to direct scalar volume rendering. In *Proceedings San Diego Workshop on Volume Visualization, Computer Graphics*, volume 24, pages 63–70, 1990.

[9] M. Weiler, M. Kraus, M. Merz, and T. Ertl. Hardware-based ray casting for tetrahedral meshes. In *Proc. Visualization '03*, pages 333–340, 2003.

[10] P. L. Williams. Visibility-ordering meshed polyhedra. *ACM Trans. Graph.*, 11(2):103–126, 1992.

[11] B. Wylie, K. Moreland, L. A. Fisk, and P. Crossno. Tetrahedral projection using vertex shaders. In *Proceedings of the 2002 IEEE symposium on Volume visualization and graphics*, pages 7–12, 2002.