Progressive Volume Rendering of Unstructured Grids on Modern GPUs

Steven P. Callahan [1] Louis Bavoil [1] Valerio Pascucci [2] Cláudio T. Silva [1]

[1] Scientific Computing and Imaging Institute, University of Utah
[2] Lawrence Livermore National Laboratory

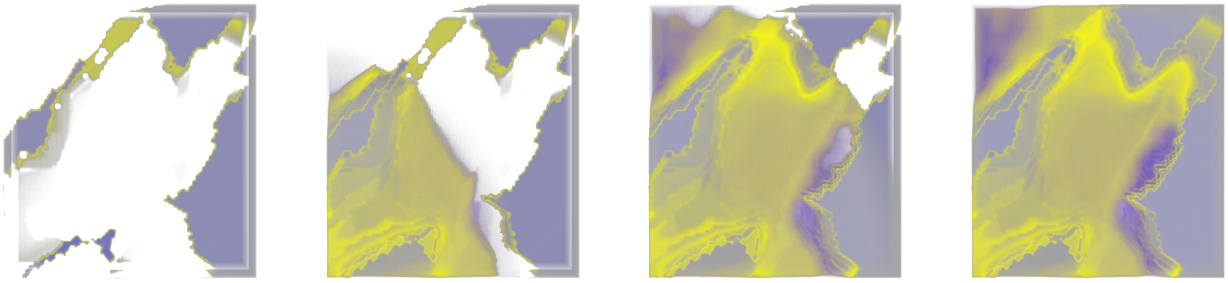

Figure 1: A sequence of progressive volume rendering steps for the SF1 dataset consisting of about 14 million tetrahedra.

Abstract

We have developed a new progressive technique that allows real-time volume rendering of large tetrahedral meshes. Our approach incrementally streams portions of the mesh to a hardware-assisted volume renderer and displays approximations that increase in quality until a full-quality rendering is achieved. The results of previous steps are stored as image data and re-used in each subsequent refinement, thus leading to an efficient rendering. Our interactive representation of the dataset is efficient, light-weight, and high quality. The result is a level-of-detail framework for interactive exploration of tetahedral datasets too large to render using conventional approaches.

1 Introduction

Unstructured grids (*e.g.*, tetrahedral meshes) are common in the simulation domains such as computational fluid dynamics and structural mechanics. The scale of these simulations has continued to grow faster than current visualization techniques can visualize them. Recent techniques such as the Hardware-Assisted Visibility Sorting (HAVS) algorithm [Callahan et al. 2005], have pushed much of the direct volume rendering burden to the GPU. These approaches are capable of rendering datasets consisting of about one million tetrahedra interactively. However, even larger datasets require level-of-detail (LOD) approaches for exploration. Our progressive volume renderer [Callahan et al. 2006] satisfies this requirement by providing a reduced representation during interaction, and incrementally refining the progressive image during idle frames until the full-quality image is rendered. The idea is to provide a rendering system that gives the same effect as the progresive transmission of JPEG images over the web. In this sketch, we focus on our algorithm for performing the progressive volume rendering with programmable GPUs.

2 Overview

The HAVS algorithm works by sorting in both object-space and image-space. In object-space, the triangles that compose the tetrahedral mesh are sorted by centroid and sent to the GPU. Upon rasterization, the fragments are sorted on the GPU in image-space using a fixed-size A-buffer implemented with textures called the k-buffer. Finally, the fragments are composited into the framebuffer using a 3D pre-integrated table that contains the volume integral. The flexibility of this approach is exploited in a recent progressive algorithm we have developed that is capable of rendering small portions of the geometry in each pass.

During interaction, the boundary of the mesh is rendered by inserting front and back fragments of the boundary into the k-buffer. Then, during each iteration of the progression, the next batch of internal geometry is rasterized and accumulated in the k-buffer. Incoming fragments are used along with the stored k-buffer entries to find the two fragments closest to the viewpoint, which are composited into an off-screen framebuffer using their scalar values and the distance between them. The front-most fragment is discarded, and the remaining fragments are stored back in the k-buffer. Next, an approximation for the rasterized portion is created by compositing the off-screen framebuffer with the contribution of the gap between the front and back fragments still in the k-buffer. Finally, the approximation is displayed and the off-screen framebuffer as well as the k-buffer are maintained for subsequent iterations from the same view. Once the user renews the exploration process through interaction, the off-screen framebuffer and the k-buffer are cleared and the process begins again.

Our GPU implementation utilizes recent advancements in programmable hardware to keep incremental data in GPU memory. In particular, Framebuffer Objects (FBOs) are used for offscreen rendering and Multiple Render Targets (MRTs) are used at each pass to read and write from textures. Our progressive renderer avoids the problems associated with using these features without multiple rendering passes over the geometry.

3 Conclusion

We have given a brief overview of the rendering aspect of our progressive volume rendering system for interactively exploring large datasets. An important feature of our algorithm is that it uses programmable GPU features to maintain only a light-weight image representation of the volume between rendering steps. Thus, the results of one increment in the progression are used in the subsequent increment. In addition, our algorithm only renders each triangle in the mesh once per viewpoint. This leads to an efficient LOD algorithm that quickly converges to a full-quality image (see Figure 1).

Acknowledgments Partial support for this work was provided by the NSF, DOE, IBM, and ARO. This work was performed under the auspices of the U.S. DOE by LLNL under contract no. W-7405-Eng-48.

References

CALLAHAN, S. P., IKITS, M., COMBA, J. L., AND SILVA, C. T. 2005. Hardware-assisted visibility ordering for unstructured volume rendering. *IEEE Transactions on Visualization and Computer Graphics 11*, 3, 285–295.

CALLAHAN, S. P., BAVOIL, L., PASCUCCI, V., AND SILVA, C. T. 2006. Progressive volume rendering of large unstructured grids. Technical Report UUSCI-2006-019, Scientific Computing and Imaging Institute, University of Utah.