Hardware-Assisted Point-Based Volume Rendering of Tetrahedral Meshes

Erik W. Anderson, Steven P. Callahan, Carlos E. Scheidegger, John M. Schreiner, and Cláudio T. Silva
Scientific Computing and Imaging Institute, University of Utah

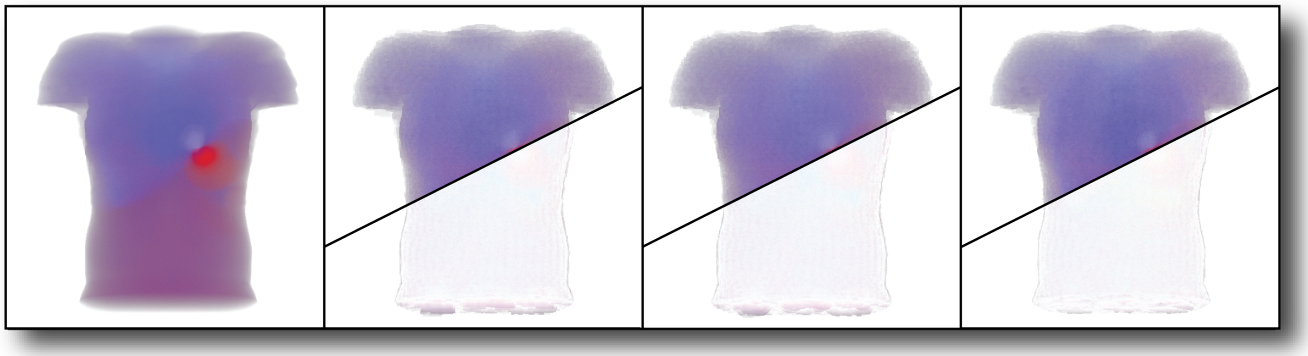


Figure 1. Our novel point-based volume rendering technique is faster than the current state of the art while still generating high quality images. Left to right: baseline image rendered with HAVS; PBVR with unshaped footprints, circular footprints and ellipsoidal footprints, respectively. Lower half (PBVR) show the difference to the baseline exact image. Notice the increasing level of fidelity.

Abstract

Unstructured volume grids are ubiquitous in scientific computing, and have received substantial interest from the scientific visualization community. In this paper, we take a point-based approach to rendering unstructured grids. In particular, we present a novel method of approximating these irregular elements with point-based primitives amenable to existing hardware acceleration techniques. To improve interactivity to large datasets, we have adapted a level-of-detail strategy. We use a well-known quantitative metric to analyze the image quality achieved by the final rendering.

1. Introduction

In this paper, we address the problem of interactively visualizing large, unstructured tetrahedral meshes. Our proposed technique renders each tetrahedron in the mesh as a single point. Approximating the tetrahedra in the mesh in this way is fast, and when combined with point reshaping methods, this technique produces high-quality renderings of large datasets at interactive rates.

There are many advantages to point-based rendering techniques. Many datasets contain tetrahedra that, after projection, represent sub-pixel sized areas. Due to the inclusion of connectivity information associated with rendering triangular or quadrilateral primitives, these tetrahedra are not optimally rendered. By representing these tetrahedra with single points, sub-pixel sized primitives can be rendered accurately using less data. Thus, rendering is often performed faster when using point-based techniques. In addition, difficulties such as storage and indexing additional datastructures associated with dynamic level-of-detail for large datasets are substantially reduced. Our technique is flexible as it is not limited to vertex-centered data, but can be applied easily to large cell-centered data as well. In spite of its advantages, point-based rendering presents several sources of error that are addressed in this paper.

Contributions The results of this work add several unique insights to the point-based rendering community. In this paper, we present the following contributions:

- A novel point-based approach for rendering unstructured tetrahedral meshes with either vertex- or cell-centered data
- Through the use of a compact point representation, this method eliminates the need to process and store connectivity information, while easing the implementation of level-of-detail strategies.
- We introduce point reshaping using a GPU raycasting technique to minimize tetrahedron approximation error.

- Through the direct compositing of the point fragments in hardware shaders, we avoid the need for convolution operations used in existing splatting techniques [6].

This paper is organized as follows. After a brief discussion of related work in Section 2, Section 3 outlines our approach to point-based volume rendering of tetrahedral meshes, including the required preprocessing. Section 4 discusses our level-of-detail methodology, including performance and quality metrics. The results of our method are shown in Section 5 while a discussion of the work and conclusions are described in Sections 6 and 7.

2. Previous Work

Since the first hardware accelerated volume rendering techniques for unstructured grids were presented, many new and innovative methods for volume rendering have produced compelling results. Silva *et al.* and Krüger and Westermann both provide comprehensive surveys of GPU-based techniques [11, 22]. By taking advantage of newly developed graphics processing units (GPUs), these hardware accelerated techniques are able to achieve interactive frame rates for moderately sized datasets.

Much work has been done to take advantage of the ever-expanding range of capabilities being introduced by modern graphics processors. Many methods have been developed to accelerate the projection of mesh elements for use in both iso-surface and volume rendering [20, 25]. The optimization of the projection, or splatting operator, has since been the intensely studied [16].

The work done by Mao *et al.* and Laur *et al.* [12, 15] forms a basis for the work of Museth and Lombeyda, TetSplat [17]. TetSpalt is representative of methods by which unstructured grids are processed in order to partition volumes for rendering. In these works, methods for splatting volumes are used in conjunction with a hierarchical data representation to achieve high framerates. While these splatting techniques have been able to take advantage of innovations in graphics hardware, raycasting techniques have also made similar adaptations. Weiler *et al.* and Bernardon *et al.* presented methods relying heavily on the GPU to perform raycasting operations used in volume rendering [2, 24]. Additionally, Callahan *et al.* introduced Hardware Assisted Visibility Sorting (HAVS) [5] allowing the correct compositing of fragments generated by rendering tetrahedral faces directly.

Although Georgii and Westernmann use the GPU to reconstruct tetrahedral elements from single vertices [10], our approach differs in several fundamental ways. While Georgii *et al.* reconstruct a triangle strip used to represent a complete tetrahedron, we approximate the tetrahedron with a single point and generate no additional geometry. Furthermore, this method relies on functionality currently available only in Microsoft DirectX 10 and compliant hardware [1].

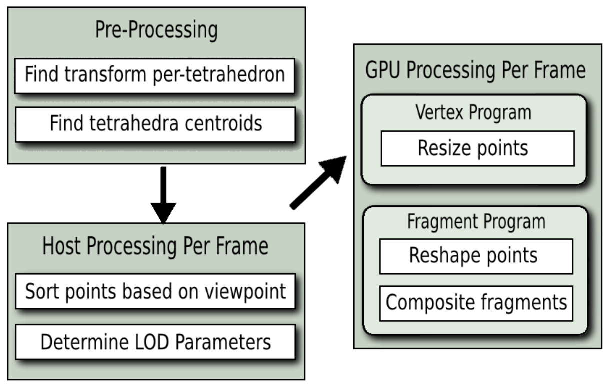


Figure 2. Algorithm overview.

Our method makes no such assumptions regarding the operating system or hardware architecture.

Level-of-detail strategies for use with the volume rendering of unstructured tetrahedral meshes is a difficult problem. Cignoni *et al.* manipulated the underlying geometry of a given mesh in order in response to the time required to render the mesh [7]. Sampling the original mesh to form new representations of it more amenable to level-of-detail strategies is another approach to this problem. Leven *et al.* sample and voxelize the original mesh [13] whereas Callahan et al. present a method by which importance sampling of the underlying geometry [4] is used to achieve interactive framerates.

Point-based representations of underlying meshes has also been well-studied [3, 18, 21]. Point-based rendering techniques have traditionally used convolution kernels to help mitigate errors associated with representing a full tetrahedron with a single point [27] with much research devoted to improving the performance characteristics and approximation elements used in them [6, 26].

3. Algorithm Overview

Our volume rendering approach relies on the representation of complex tetrahedral meshes by simple approximations of the elements comprising the mesh. As illustrated in Figure 2, this algorithm can be broken down into three distinct components: pre-processing of the mesh, per-frame processing on the CPU, and per-frame processing on the GPU. After forming our approximation elements by finding a representative transform and scalar value for each point in a pre-processing step, the mesh is processed on a per-frame basis. The CPU first sorts the points front-to-back by the tetrahedra centroids based on the current viewpoint and then determines the level-of-detail to use in the current frame. The GPU's vertex program then uses information in the transform associated with the point being rendered to re-size it to ensure that the tetrahedron is adequately represented. Then, the fragment program culls fragments based on the shape of the approximating element using a technique reminiscent of Blinn *et al.* [14]. Then, one fragment is kept in a texture and used to composite the ray-gap between the two fragments using a pre-integrated table [9]. The error associated with over-representation of a tetrahedron is thus mitigated by compositing fragment distances instead of applying pre-assigned values for alpha.

3.1. Preprocessing the Tetrahedral Mesh

To produce a final rendering of a dataset at interactive framerates with the best possible visual quality, the input tetrahedral mesh must be preprocessed. As we will show, each tetrahedron can be approximated with exactly one affine transformation from a unit tetrahedron defined on the standard 3-dimensional basis to any given tetrahedron. The reference tetrahedron's barycenter is also transformed by the same affine transformation to a vertex, v, describing the location of the point object representing the associated tetrahedron.

Since an entire tetrahedron is being represented by a single point, we must assign this point a value based on the scalar field being described by the underlying geometry. For vertex-centered data, the mean of the scalar values at each vertex is assigned to v; otherwise, the cell value is assigned. While this method does not account for the distance of the vertices from the point representing them, we have found that in practice it adequately approximates a general, well-formed tetrahedron.

Because graphics point primitives (GL_POINT) are rasterized as squares they must be shaped to better approximate the tetrahedra they represent in order to improve image quality. The shaping is accomplished by *general representative transformations* for each tetrahdron. This additional information allows the GPU to reshape the point primitive in the fragment shader.

In order to generate this transformation, we define a regular tetrahedron centered at the origin such that it is inscribed in the unit sphere (we call this regular tetrahedron $\hat{\sigma}$). We then find a transformation T such that $T(\hat{\sigma}) = \sigma$. This transformation takes the unit tetrahedron to a given tetrahedron as shown in Figure 5.

3.2. Rendering the Points

As the method we present produces an approximation of the true tetrahedral mesh, we strive to reduce the approximation error in order to create an adequate rendering. Below we discuss the various methods of footprint generation. Each final footprint shows a tradeoff between final image quality and rendering speed. In all of the following footprint generation techniques, point sizes are generated by taking the size of the tetrahedron and the distance from the viewpoint into account.

Circular footprints. As can be seen in Figure 1, generating footprints without regard to the shape of the tetrahedron produces a high degree of error resulting from overdraw. Although the method by which we generate the point size is unchanged from that of the unshaped points, in order to reshape the point into a circle we must inform the fragment program of the radius of the circle we wish to form and its location in screen-space. These data must be sent down the pipeline, as fragments being processed by the fragment shader no longer contain information about the generating primitive. By passing the center of the point and radius to the fragment program via texture coordinates, the program is made aware of all data required to properly rasterize the tetrahedron approximation. Culling fragments that fall outside the radius of the circle is straightfoward.

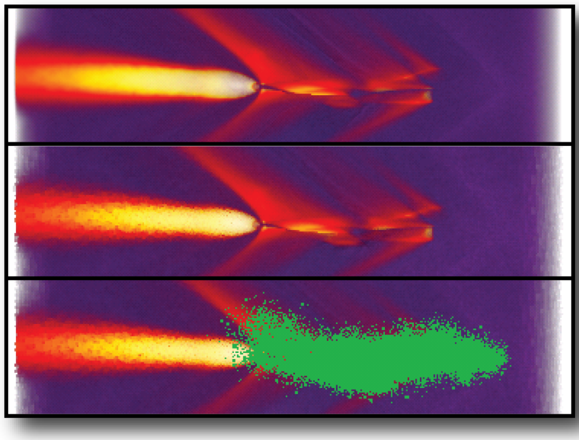


Figure 3. For adaptive meshes, many elements are subpixel sized. The top image shows HAVS, the middle image shows ellipsoidal projection of the tetrahedra, and the bottom image uses green pixels to show subpixel element locations.

Figure 1 shows the image quality improvement by representing tetrahedra as circles compared to simple squares, but even reshaping the points to form circles renders many fragments unnecessarially.

Ellipsoidal footprints. To reduce overdraw and increase image quality, ellipsoidal footprints are useful. Their smooth appearance make them good candidates for point-based level-of-detail (see Section 4). From geometric probability, we know that for a convex volume A contained in another convex volume B, the conditional probability that a random ray that hits B will also hit A is the ratio of surface areas, s_A and s_B: $p(A|B) = s_A/s_B$ [19]. Since s_A is the surface area of the tetrahedron, and hence fixed, we want to minimize the surface area s_B, to maximize the probability. This defines the best approximating ellipsoid over all viewpoints.

Finding the ellipsoid with minimum surface area that en-

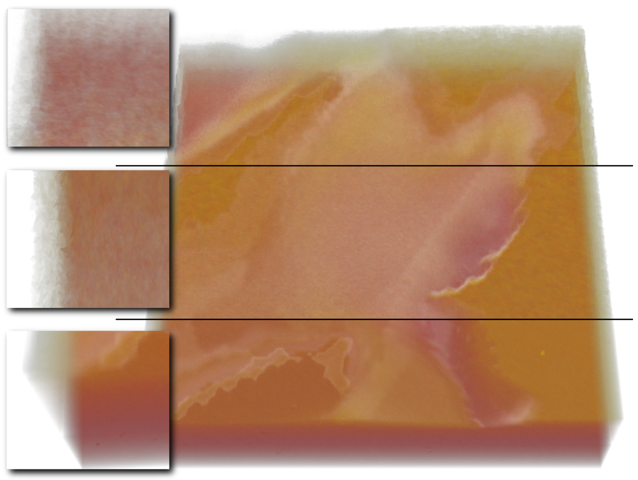
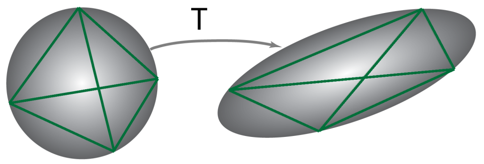


Figure 4. Although interior areas of the volume are rendered with a high degree of accuracy, volume borders are more obviously affected by the point-based technique being used. The sf1 dataset (simplified to approx. 3.5M tetrahedra) displays these errors clearly. Ellipsoidal footprints (top) display more error than exact footprints (middle) when compared with the exact rendering (bottom).

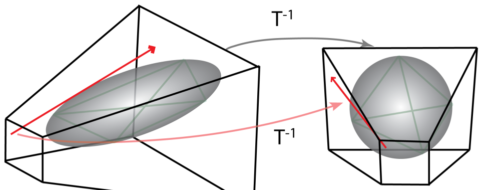
closes the tetrahedron involves a constrained minimization over integrals with no closed form. Instead, we find the enclosing ellipsoid with smallest volume (we will call it the *min-volume ellipsoid*). This is a simpler problem, and in practice, usually generates an ellipsoid with small surface area.

To find the min-volume ellipsoid, we first note that the unit sphere is the min-volume ellipsoid for the unit tetrahedron $\hat{\sigma}$. The transformation associated with the tetrahedron in question moves the min-volume ellipsoid for the unit tetrahedron to an ellipsoid for the represented tetrahedron. The volume of any shape is scaled by the determinant of T. Since this is a constant for all possible shapes, it must be the case that T preserves extrema, and so the ellipsoid is min-volume. The process is illustrated in Figure 5.

The transform, along with the position of the point in $\mathbb{R}^3$ and its associated scalar value must be transferred to the GPU in order to properly cull fragments from the rasterized point. A simple raycaster provides us with a perspective-correct method for determining the projection of an ellipsoid. Since the reference space used during ray-casting is isomorphic to worldspace via the transformation T, transforming the rays allows us to cull fragments on the GPU based on a simple ray-sphere intersection. Figure 5 describes the results of this culling procedure from a rasterized square into the projection of a min-volume ellipsoid.



(a)



(b)

Figure 5. (a) **The transformation taking a unit tetrahedron to a general tetrahedron also takes a unit sphere to the min-volume ellipsoid.** (b) **Ray-casting on the GPU is efficient when taking advantage of the representative transformation.**

Exact Projection Footprints. As with ellipsoidal projection, we note that the transform taking a unit sphere to an enclosing ellipsoid is equivalent to that describing the underlying tetrahedra. Similarly, an appropriate ray-casting test can be used in the fragment program to cull fragments leaving only the exact tetrahedral projection to be rendered.

Several opportunities for optimization are inherent in the underlying algorithm used to describe the geometry being rendered. First, the method by which the transformation is formed allows the tetrahedron being processed to be expressed as a right-angled tetrahedron. When expressed as a right-angled tetrahedron, the ray-caster needs only to test rays against three congruent right triangles in order to properly shape the tetrahedral projection.

3.3. Compositing the points

As our method for footprint generation does not require any kernel-based convolution, it can be implemented independently of the compositing technique. This provides a true point-based volume rendering approach that can be easily implemented for many different compositing algorithms. In this case, we are using a compositing scheme based on HAVS [5], in which the incoming fragment is directly composited with the previous fragment. This is possible as the previous fragment is always stored in a texture so that the ray-gap can be determined. The scalar values of the front and back fragments as well as the distance between them are used to look up into a pre-integrated table [9], then composited directly into an off-screen framebuffer. Since our initial drawing primitives can be thought of as screen-aligned

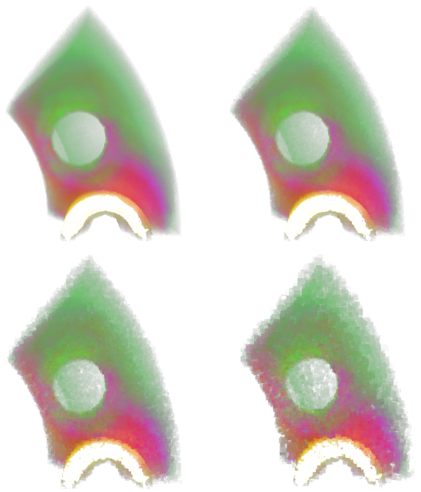


Figure 6. Dynamic level-of-detail of the SPX2 dataset with point resizing. The LOD at each step represents a percentage of the fragment count used: $(top\ left)$ **100% at 5.6 fps,** $(top\ right)$ **50% at 20 fps,** $(bottom\ left)$ **25% at 50 fps, and** $(bottom\ right)$ **15% at 100 fps. For dataset details, please see Table 2.**

(a) Image quality

Level of Detail	Unshaped	Circles	Ellipses	Exact
100	6.51	6.12	3.89	3.74
75	6.97	6.46	4.72	3.23
50	8.54	7.77	6.80	3.49
25	12.22	10.62	13.60	14.15

(b) Performance in frames per second

Level of Detail	Unshaped	Circles	Ellipses	Exact
100	1.7	1.7	2.0	2.1
75	3.1	3.1	3.2	3.2
50	6.4	6.6	6.7	6.7
25	25.3	25.2	25.5	25.2

Table 1. (a) Image quality for the fighter dataset at different levels of detail. (b) Respective rendering performance in frames-per-second.

polygons, we need not perform per-fragment depth sorting as HAVS did, and thus require only a single fragment and its associated scalar and depth from the compositing step previously performed. However, as with any volume rendering algorithm, depth ordering is important. Fortunately, as point primitives are rasterized as screen-aligned squares, a sort by centroid method exactly sorts all approximating elements alleviating the need for per-fragment sorting operations.

4. Level Of Detail

One of the advantages of using a point-based approach for rendering is that the lack of connectivity between primitives facilitates dynamically changing data. In particular, level-of-detail (LOD) approaches become much easier. Recently, Callahan et al. [4] introduced a simple, dynamically-adjusting LOD algorithm based on a sample-based simplification of the renderable primitives. Instead of simplifying the geometry, the algorithm performs importance sampling of the primitives, which enables dynamic LOD by adjusting the number of primitives rendered at each frame. To reduce holes in the resulting image, the boundary geometry of the mesh is always rendered.

We have adapted this strategy in our point-based volume renderer by using a purely Monte Carlo approach to importance sampling. As before, the number of points rendered at each frame is adjusted based on the framerate of the previous frame. A key difference is that by using our points as described, holes may appear in the volume with a lower LOD because the points may no longer overlap. To minimize these discontinuities in the resulting image, our ellipsoidal footprint is used. Because this problem will occur even on the boundary, we do not distinguish between boundary and internal points. Instead, we present a technique for resizing the points based on the LOD to reduce the number of gaps created. Cook and Halstead [8] demonstrated this technique using a scale that is linear to the reduction for opaque geometry in a distant scene.

Since our volume renderer relies heavily on fragment processing, we use an LOD algorithm corresponding to the number of fragments, instead of primitives as in [4]. Thus, a 50% LOD rasterizes 50% of the fragments. The problem is to find the correct number of points to render that generates the correct number of fragments with the scaling factor. If the number of fragments rasterized is represented as a function f of the LOD: $f(\lambda) = N(\lambda)S(\lambda)x^2$, where λ is the LOD reduction, N is the a function of the number of points, S is a function of the point scaling, and x is the point size. With no scaling, the number of fragments generated is directly proportional to the number of points rendered, *i.e.* $N(\lambda) = \frac{n}{\lambda}x^2$, where n is the number of points. If we introduce a linear scaling factor the number of points can be computed to obtain the same fragment count: $N(\lambda)\lambda x^2 = \frac{n}{\lambda}x^2$ or $N(\lambda) = \frac{n}{\lambda^2}$. Therefore, at each LOD step, the number of points rendered is adjusted by the square of the LOD, where the LOD $\in [0, 1]$.

The resulting LOD scheme efficiently and dynamically adapts the number of fragments rendered to achieve a target interactivity. In practice, image quality is not dominated by the ellipsoidal representation of tetrahedra as we composite by distance instead of convolution kernels. Figure 6 shows an example of our dynamic LOD with point resizing. It is important to note that although fewer points are

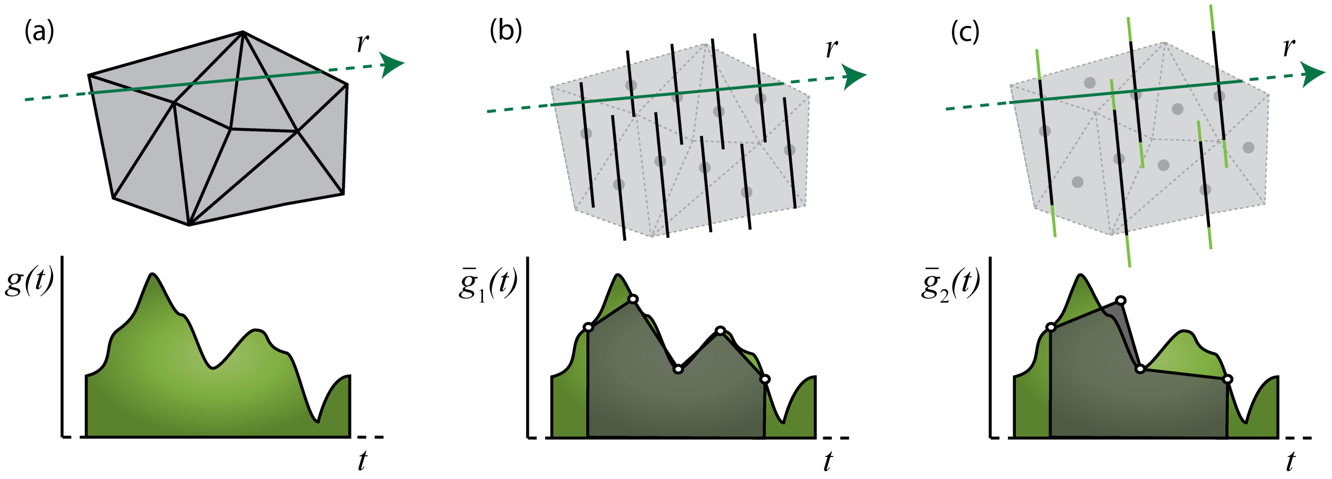


Figure 7. Illustration of LOD Strategy in 2D (a) Original Mesh with viewing ray through the mesh and associated scalar function $g(t)$. (b) Approximating elements intersecting the viewing ray. The approximation $\bar{g}_1(t) \approx g(t)$ is directly influenced by the number of approximating elements intersected by the viewing ray. (c) Resized random sampling of approximation elements (green) and the original approximating elements (black) along with the approximation of the function $g(t)$ under level-of-detail conditions. The approximation $\bar{g}_2(t)$ represents a 50% level-of-detail.

selected for rendering, the sizes of the points being rasterized increases, thus increasing the number of points approximating the scalar field as illustrated by Figure 7. Although the point primitives are reshaped during rendering, the fragments being culled must still be processed. Each fragment, regardless of its final contribution to the framebuffer, is processed by the fragment program. Because of the fragment processing, during application of the LOD scheme, the relative size of the points selected for rendering directly impacts the performance of the selected LOD. This LOD strategy exploits the fact that as the total fragment count decreases, the framerate increases.

5. Results

Here we present the results of our implementation in terms of visual quality and interactivity as they relate to HAVS. To adequately analyze the performance of our volume rendering approach we have implemented HAVS with comparable optimizations. Each rendering method employs Vertex Buffer Objects and parallel sorting and rendering. To generate images with the highest possible quality in HAVS, we use the largest size for the k-buffer in order to remove as many artifacts as possible. All results and images were generated on an Intel Core 2 Duo 2.2 Ghz processor, 2.0 GB RAM and an NVIDIA 7950 GX2 graphics card.

Table 2 describes the performance of our method based on the shape of the footprints being generated compared with the HAVS algorithm [5]. As the table describes, our method's performance is better than the HAVS algorithm for all datasets while yielding acceptable image quality. However, our algorithm approaches the performance of HAVS for large meshes. This is due to the quantity of data being sent to the GPU for further processing. Although each of the increasingly accurate footprint generation methods reduces the amount of fragment overdraw that ocurs at each step, more data must be sent to the GPU, reducing the overall rendering speed of particularly large datasets. As tetrahedra approach sub-pixel accuracy, it becomes unnecessary to reshape the points representing them as unshaped points at that scale represent tetrahedra just as well as shaped points, generating additional speedups. The degree to which these small tetrahedra affect the final visualization depends on the structure found in the original tetrahedral mesh. In Figure 3 the fighter dataset is rendered so that all tetrahedra with sub-pixel sized footprints are colored green. The reshaping of these tetrahedra will do very little to mitigate any rendering error in the final visualization, so they remain represented with squares.

5.1. Image Quality

In order to quantitatively evaluate the quality of a resulting image, a quality metric must be described. The root-mean squared error metric is used here to analyze each rendered image. The implementation of this metric represents the mean of all distances in colorspace between corresponding pixels. This provides a global metric by which we can compare any image generated by our point-based method to an image generated by exact volume rendering. Table 1 presents the results of the application of the above described quality metric. As the error metric implies, lower numbers

Dataset	Num Tets	Unshaped	Circles	Ellipses	HAVS
SPX2	827,904	13.8	14.7	17.2	5.26
Torso	1,082,723	6.7	7.1	6.8	3.70
Fighter	1,403,504	5.3	5.1	4.9	2.78
F-16	6,345,709	0.3	0.28	0.29	0.2

Table 2. Performance summary for full-quality renderings in frames-per-second.

correspond to higher quality images.

6. Discussion

The method presented here is generally faster than current direct volume rendering approaches; however, it is clear that some meshes are better represented by certain footprint shapes than others. Due to the number of fragments generated by the GL_POINT primitive, directly rendering the tetrahedral mesh may be a better approach than approximating the projections of the tetrahedra. This is true for datasets in which the tetrahedra sizes vary widely as the size of points representing large tetrahedra increase more rapidly than the sizes for small tetrahedra thereby increasing the overall number of fragments unnecessarily generated. Although many of these fragments may be culled in the fragment program, the testing and branching in the fragment program is an expensive operation. Additionally, only the exact projection approach does not render a large number of fragments that lay outside of the true projection of the tetrahedron that must be composited to form the final rendering. These extra fragments are culled in the point reshaping process.

As Figure 4 suggests, the point based rendering method presented here may yield a poorer quality rendering of a dataset when compared with a direct rendering approach. This is particularly noticeable at the dataset boundaries as highlighted by the figure. This can be explained by the fundamental representation techniques of the underlying geometry involved in the rendering. While direct methods often represent tetrahedra as a collection of faces, this point-based approach uses only a single rasterized square. As each tetrahedron is represented by a single point, and thus a single scalar, the final rendering also lacks scalar interpolation throughout the tetrahedron. This limits the algorithms ability to produce proper pixel values when the depth complexity at that location is low.

Although this approach to point-based rendering suffers from problems related to fragment generation and approximation of the tetrahedral mesh, it poses several unique advantages over previous methods. Since the footprint generation and reshaping are completely independent of the compositing and rendering stage, it is trivial to implement various compositing mechanisms. Also, as we have shown, a global level-of-detail strategy based on importance sampling generated as a pre-process combined with point resizing enables this method to excel during interaction. Furthermore, if lighting and shading are desirable for the final rendering, passing normal information gathered from gradient calculation in a pre-processing step can be easily be used to implement a variety of shading models.

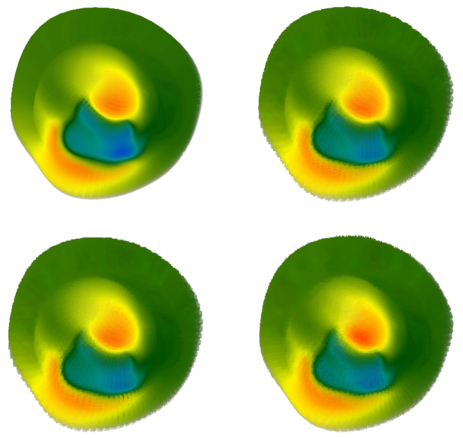


Figure 8. Rendering of the Heart dataset: $(top\ left)$ HAVS; $(top\ right)$ circular footprints, $(bottom\ left)$ ellipsoidal footprints; and $(bottom\ right)$ exact projections. Note: This dataset has malformed tetrahedra that cause HAVS to produces erroneous renderings; PBVR rendering does not fail in these situations.

7. Conclusion

The method we present for the point-based volume rendering of unstructured tetrahedral meshes has proven to be a fast an effective method for visualizing large datasets. As with any visualization method, there is a tradeoff between speed and accuracy. As the complexity of the approximations increases, so does the accuracy of the final rendering. However, this increased accuracy necessarilly comes with a decrease in performance of the technique. Our method acheives interactive framerates through a combination of point-based footprint generation and a natural level-of-detail strategy that compromises accuracy for speed during periods of interaction while rendering at full-quality after interaction stops.

Future Work The next generation of GPUs will include not only programmable vertex and fragment shaders, but programmable geometry shaders allowing better approximations to tetrahedra while further minimizing excessive fragment generation and overdraw-based error. We hope to be able to optimize this algorithm for use with this new technology while adding features such as lighting and shading. Furthermore, this approach lends itself well to distributed rendering as each rendering primitive exerts only a local change and, with care, can be partitioned in such a way as to take advantage of systems capable of massive parallel visualization of large datasets [23].

While we have focused on datasets containing exclusively vertex-centered data, this technique is easily extended to produce high-quality visualizations of cell-centered data as well. As cell-centered data assigns a single scalar value per tetrahedron, the approximation of the scalar value of each point primitive exactly represents the scalar content of the underlying tetrahedron.

Acknowledgments Our work is funded by the Department of Energy under the ASCI VIEWS program and the MICS office, the National Science Foundation (grants CCF-0401498, EIA-0323604, OISE-0405402, IIS-0513692, CCF-0528201), Sandia National Laboratories, Lawrence Livermore National Laboratory, an IBM Faculty Award and a University of Utah Seed Grant.

The authors thank Bruce Hopenfeld and Robert MacLeod (University of Utah) for the heart dataset, Bruno Notrosso (electricite de France) for the SPX dataset, Neely and Batina for the fighter dataset, and Udo Tremel (EADS-Military) for the F16 dataset.

References

[1] R. Balaz and S. Glassenberg. DirectX and Windows Vista presentations
. *http://msdn.microsoft.com/directx/archives/pdc2005*, 2005.

[2] F. Bernardon, C. Pagot, J. ao Comba, and C. Silva. GPU-based tiled ray casting using depth peeling. *Journal of Graphics tools*, 11(4):1–16, 2006.

[3] M. Botsch, A. Hornung, M. Zwicker, and L. Kobbelt. High-quality surface splatting on today's GPUs. *Eurographics Symposium on Point-Based Graphics*, pages 17–24, 2005.

[4] S. P. Callahan, J. L. D. Comba, P. Shirley, and C. T. Silva. Interactive rendering of large unstructured grids using dynamic level-of-detail. *IEEE Visualization*, pages 199–206, 2005.

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

[6] W. Chen, L. Ren, M. Zwicker, and H. Pfister. Hardware-accelerated adaptive EWA volume splatting. *IEEE Visualization*, pages 67–74, 2004.

[7] P. Cignoni, L. D. Floriani, P. Magillo, E. Puppo, and R. Scopigno. Selective refinement queries for volume visualization of unstructured tetrahedral meshes. *IEEE Transactions on Visualization and Computer Graphics*, 10(1):29–45, 2004.

[8] R. Cook and J. Halstead. Stochastic pruning. Technical Report #06-05, Pixar, 2006.

[9] K. Engel, M. Kraus, and T. Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. *ACM SIGGRAPH/EUROGRAPHICS workshop on graphics hardware*, pages 9–16, 2001.

[10] J. Georgii and R. Westermann. A generic and scalable pipeline for GPU tetrahedral grid rendering. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1345–1352, 2006.

[11] J. Krüger and R. Westermann. Acceleration techniques for GPU-based volume rendering. *IEEE Visualization*, pages 38–45, 2003.

[12] D. Laur and P. Hanrahan. Hierarchical splatting: a progressive refinement algorithm for volume rendering. *ACM SIGGRAPH*, pages 285–288, 1991.

[13] J. Leven, J. Corso, J. Cohen, and S. Kumar. Interactive visualization of unstructured grids using hierarchical 3D textures. *IEEE Symposium on Volume Visualization and Graphics*, pages 37–44, 2002.

[14] C. Loop and J. Blinn. Resolution independent curve rendering using programmable graphics hardware. *ACM Transactions on Graphics*, 24(3):1000–1009, 2005.

[15] X. Mao, L. Hong, and A. Kaufman. Splatting of curvilinear volumes. *IEEE Visualization*, pages 61–68, 1995.

[16] R. Marroquim, A. Maximo, R. Farias, and C. Esperanca. GPU-based cell projection for interactive volume rendering. *SIBGRAPI*, pages 147–154, 2006.

[17] K. Museth and S. Lombeyda. TetSplat: Real-time rendering and volume clipping of large unstructured tetrahedral meshes. *IEEE Visualization*, pages 433–440, 2004.

[18] H. Pfister, M. Zwicker, J. van Baar, and M. Gross. Surfels: surface elements as rendering primitives. *ACM SIGGRAPH*, pages 335–342, 2000.

[19] M. Pharr and G. Humphreys. *Physically Based Rendering*, chapter 4.4. Morgan Kaufmann, 2004.

[20] S. Röettger, M. Kraus, and T. Ertl. Hardware-accelerated volume and isosurface rendering based on cell-projection. *IEEE Visualization*, pages 109–116, 2000.

[21] S. Rusinkiewicz and M. Levoy. QSplat: a multiresolution point rendering system for large meshes. *ACM SIGGRAPH*, pages 343–352, 2000.

[22] C. T. Silva, J. L. D. Comba, S. P. Callahan, and F. F. Bernardon. A survey of GPU-based volume rendering of unstructured grids. *Brazilian Journal of Theoretic and Applied Computing(RITA)*, 12(2):9–29, 2005.

[23] H. T. Vo, S. P. Callahan, C. T. Silva, W. Martin, D. Owen, and D. Weinstein. iRun: Interactive rendering of large unstructured grids. *Eurographics Symposium on Parallel Graphics and Visualization*, pages 93–100, 2007.

[24] M. Weiler, M. Kraus, M. Merz, and T. Ertl. Hardware-based ray casting for tetrahedral meshes. *IEEE Visualization*, pages 44–52, 2003.

[25] B. Wylie, K. Moreland, L. A. Fisk, and P. Crossno. Tetrahedral projection using vertex shaders. pages 7–12, 2002.

[26] D. Xue and R. Crawfis. Efficient splatting using modern graphics hardware. *Journal of Graphics Tools*, 8(3):1–21, 2004.

[27] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. EWA volume splatting. *IEEE Visualization*, pages 29–36, 2001.