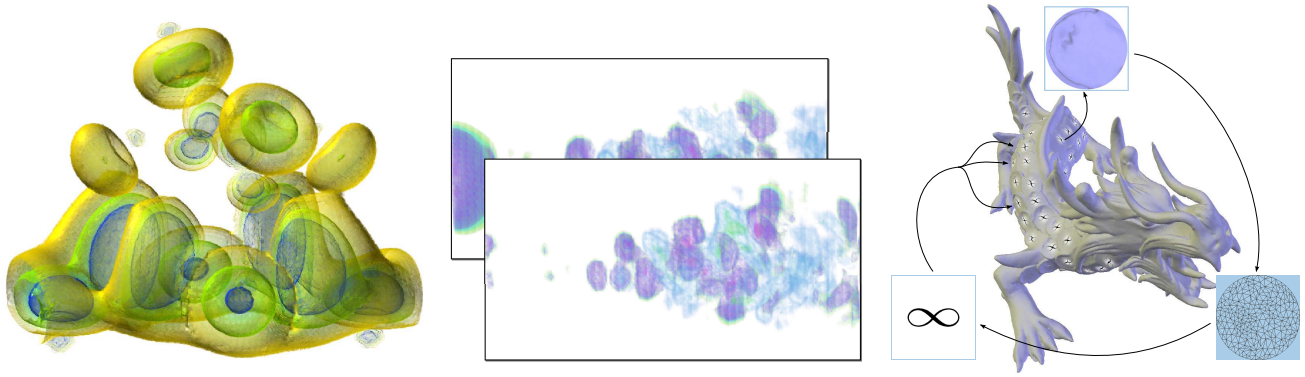
# Improved Algorithms for Volume Rendering and Mesh Processing

André Maximo*, Amitabh Varshney† and Ricardo Farias*
*System Engineering and Computer Science – COPPE/UFRJ
Rio de Janeiro, RJ, Brazil
{andmax,rfarias}@cos.ufrj.br
†Department of Computer Science – University of Maryland
College Park, Maryland, USA
varshney@cs.umd.edu



(a) (b) (c)

Figure 1. Rendering examples of algorithms presented in this thesis: (a) indirect and direct volume rendering using gradients; (b) time-varying volume rendering based on the same algorithm used in (a) called *Projected Tetrahedra*, only two of the 150 frames from the animation is shown; (c) illustration of our mesh processing method applied to mesh parameterization, where one dragon scale is selected (top), parameterized using a standard procedure (bottom right), and used to propagate the same texture (bottom left) to all similar dragon scales seamlessly.

***Abstract*—In this thesis [1], improved algorithms are presented for volume rendering and mesh processing. In the research area of volume rendering, the goal is to improve computational performance and memory consumption, using programmable graphics cards. While in the area of mesh processing, the goal is to introduce a method to augment the usage of shape similarity of a surface. The volume rendering algorithms are based in ray casting and cell projection, handling both regular and irregular data, and employing both direct and indirect volume rendering techniques. The mesh processing method, on the other hand, augments the usage of self-similarity of models to propagate processing done in one part of the mesh to many others similar parts. The presented method is exploited in two distinct applications: detail transfer and parameterization. Finally, the volume rendering algorithms are compared against each other and correlated state-of-art techniques.**

***Keywords*-volume rendering, ray casting, cell projection, mesh processing, similarity measures**

## I. Introduction and Related Work

This thesis presents several improved algorithms for volume rendering and mesh processing. The rendering techniques presented here aim to improve the computational performance and memory consumption, while the processing method introduces a new concept in the usage of self-similarity of models. In rendering, the algorithms presented target the visualization of volumetric datasets, specifically scalar fields defined in regular or irregular grids, using programmable graphics cards (GPUs). In mesh processing, the method introduced uses non-local neighborhood defined by mesh descriptors to propagate mesh processing done in one part of the model to many other parts, which share some desired feature. The algorithms presented of these two research areas are distinct in essence.

*Volume Rendering:* The techniques in this area have been widely used in a variety of fields, such as visualization of geological data, fluid simulation and inspection of medical images. The main goal is to obtain a better insight of the volume data either by rendering iso-surfaces, i.e. *indirect volume rendering*, or by rendering the volume as a semi-transparent material, i.e. *direct volume rendering*. In this thesis, we are interested in both renderings (see an example of a probability distribution dataset in Figure 1(a)).

The main data sources for volume rendering applications are numerical simulations of natural phenomena, e.g. Computational Fluid Dynamics (CFD), and measurement devices

[1] This work summarizes a doctoral thesis completed on 2010.

such as Magnetic Resonance Imaging (MRI). Generally, measurement devices produce regular volume data, while simulations yield both regular and irregular data.

Volume rendering of irregular structures has mainly focused on iso-surface rendering [1], e.g. numerical simulations for fluid dynamics and tensor fields in geosciences [2]. Even so, direct volume rendering is also employed in important applications such as the visualization of geodynamics phenomena of mantle convection [3].

One simple and efficient method widely used for direct volume rendering of irregular meshes is cell projection [4], [5]. Among the cell-projection techniques, the Projected Tetrahedra (PT) algorithm introduced by Shirley and Tuchman [6] is one of the most popular. The PT method subdivides a tetrahedron projection into four classes, where for each class the tetrahedron cell is decomposed into a fixed number of triangles in order to benefit from the graphics card's rasterization pipeline.

Another common approach for volume rendering is ray casting [7], [8]. In this image-order strategy, the final pixel color is computed by casting a ray through the volume, instead of projecting volume cells to the image plane. Since both strategies involve compositing, they depend on a correct traversal order for a given viewpoint. On the one hand, ray-casting algorithms employ an adjacency graph in such way that when a ray leaves a cell it has enough information to find the next one; on the other hand, cell-projection algorithms usually compute an approximate ordering in object space. Even though there are methods for exact ordering of cells [9], they are quite complex and time consuming. Some approaches avoid such problems combining the better of cell-projection and ray-casting techniques [10].

This thesis presents three cell-projection algorithms based on PT and one ray-casting algorithm in Section II. The four volume rendering algorithms are: *VF-Ray-GPU – Visible-Face Driven Ray Casting implemented on the GPU* – a memory efficient GPU-based ray-casting method; *RPTINT* and *IPTINT – Regular and Improved Projected Tetrahedra with Partial Pre-Integration* – two GPU-accelerated cell-projection techniques based on the original PT [6], where the first is specialized to regular data and the second combines indirect and direct volume rendering; *HAPT – Hardware-Assisted Projected Tetrahedra* – a closer adaptation of the PT algorithm to graphics cards, where a flexible and efficient framework is presented, capable of extracting iso-surfaces and handling time-varying data on-the-fly (see an example of two animation frames in Figure 1(b)).

*Mesh Processing:* Surfaces discretized as triangular meshes are ubiquitous in Computer Graphics. The vast majority are generated by artists or scanned from real-world objects. Almost all surfaces, artificial or natural, feature regions with nearly identical properties, such as color, shape, or texture. In this thesis, we are interested in repeated shape patterns and how to use them to propagate mesh processing.

Symmetries are normally detected and described as planar reflections among parts of a mesh [11]. In addition to these mirror-like symmetries, resemblances among details on a surface can also be found [12], [13]. The concept of self-similarity of a meshed surface is defined by such resemblances, and naturally generalizes the concept of reflective symmetries. We say that two or more parts of a mesh are similar when they share local surface features, either reflective or not. Although detection and structural analysis of reflective symmetries may be used to improve mesh processing tasks [14], there have been few works [15], [16] dealing with repeated patterns but without the notion of mesh processing propagation introduced by our method.

The goal of the method introduced in this thesis is to take advantage of a similarity descriptor over a mesh in order to propagate processing performed on a region to several other similar regions. Similarity descriptors have been the target of much study in recent years, mainly because of a large growth in the mathematical theory underpinning the discrete geometry of meshes; namely, discrete differential geometry [17], [18]. Mean and Gaussian curvatures, the Laplace-Beltrami operator, and heat-diffusion processes on a triangular mesh constitute the primary tools when devising a signature for a vertex using its differential properties [13], [15]. At the same time, descriptors relying purely on the positions of vertices on the embedded surfaces also exist, such as the ones using Euclidean distances [19]. In this thesis, we are interested in these type of descriptors and present a method to compute a fairly simple similarity descriptor in Section III. We use the heightmap of the surface surrounding a given vertex as the vertex's descriptor. This simple approach naturally encodes the surrounding shape and it is robust with respect to the mesh triangulation. Moreover, we use the Gaussian-weighted Zernike coefficients of these heightmaps in order to make the comparison rotational invariant, more efficient and based on regions rather than vertices. This results in a similarity measurement which allows us to build a non-local neighborhood space based on similar local shapes. We use this neighborhood to propagate mesh processing operations, such as mesh parameterization shown in Figure 1(c).

## II. Volume Rendering

In this section we present briefly the four volume rendering algorithms of this thesis. For more information please refer to the corresponding papers.

*VF-Ray-GPU [20], [21]:* The idea behind VF-Ray is to explore ray coherence, improving caching performance by keeping in memory only the face data of the traversals of a set of nearby rays. In GPU, we divide our algorithm in three steps and, as a result, in three different CUDA kernels (see Figure 2). In the first kernel, the external faces are read and the visible faces are determined by a simple back face culling test. The second kernel computes the projection of each visible face, splitting them in pixels on the screen space.

Finally, the third kernel evaluates the ray casting algorithm over each visible face pixel previously computed. The first kernel is important to reduce the computation done by the next two kernels, while the last kernel is responsible to most of the computation time and to generate the final image.

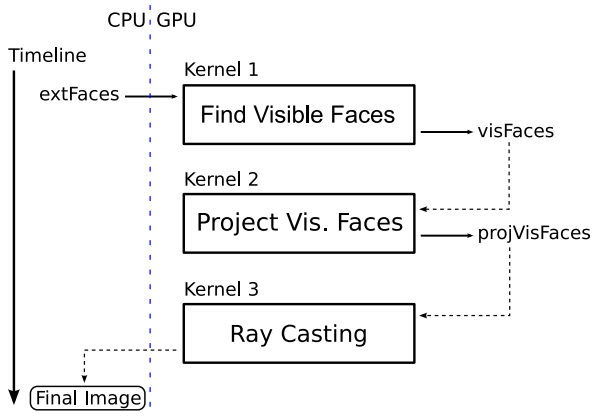



Figure 2. The three CUDA kernels of the VF-Ray-GPU algorithm.

The VF-Ray code, in both CPU and GPU using C/C++ and C for CUDA, is made available with this thesis: http://code.google.com/p/vfray [2].

*RPTINT [22]:* The main idea of this method is to use the PT algorithm taking advantage of the volume data regularity. The method consists of four steps, where the first three steps take place in the CPU, whereas the last is performed by the GPU (see Figure 3). First, the projection of a single hexahedron cell (eight scalar values) is determined by splitting it into five tetrahedra and projecting each tetrahedron using PT. This *basis hexahedron projection* is used as many times as the number of hexahedra inside the volume, since the data is regular and the projection shape is always the same. In the second step, a traversal order for the whole volume is determined in constant time using the implicit ordering of the data regularity. The remaining step on the CPU consists of allocating the volume information (scalar values) in a *Vertex Array* structure. This third step is done only once since it does not depend on the view point. Lastly, triangles corresponding to the projected tetrahedra are rendered and composed in back-to-front order using the GPU. In this last step, the basis hexahedron is iteratively displaced in parallel inside the vertex shader and the projection information from the first step is used repeatedly.

The RPTINT code, using C/C++ and GLSL, is made available with this thesis: http://code.google.com/p/rptint.

*IPTINT [23], [24]:* The idea of the PTINT algorithm is to adapt the original PT method to the GPU. The improved version of PTINT, called IPTINT, is similar to the basic version but adding gradient vector interpolation to the original

[2]Source code of applications related to this thesis are open source under the *GNU General Public License version 3* hosted at Google™ code.

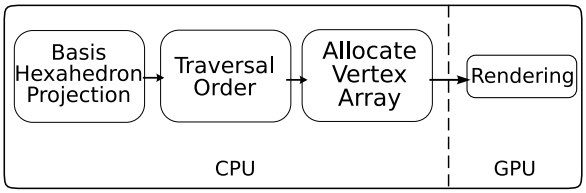



Figure 3. RPTINT algorithm overview.

PT to be able to render illuminated iso-surfaces. The IPTINT algorithm is executed in two GPU passes. In the first step, all relevant information of each tetrahedron is computed, that is, the projection class, the thick vertex properties, and the $z$ coordinate ($c_z$) of the tetrahedron's centroid. The computed data is sent back to the CPU using multiple render targets (MRT) with four framebuffers, depicted in Figure 4. The projection class is encoded in the index to a classification table, called Ternary Truth Table ($id_{TTT}$), and the number of triangles generated by the projection ($count$). The thick vertex properties are responsible to most of the output, consisting of the intersection vertex ($v_i$ used only for class 2), the distance traversed by the ray ($l$), the front and back scalar value ($s_f$ and $s_b$), and the front and back gradient vectors ($g_f$ and $g_b$). During the second step, vertices' scalar and gradient values are interpolated to compute chromaticity and opacity values for each fragment. If an iso-surface is detected, it is rendered with Phong shading using the gradient as its normal vector (see Figure 1(a)), otherwise the volume ray integral is applied.

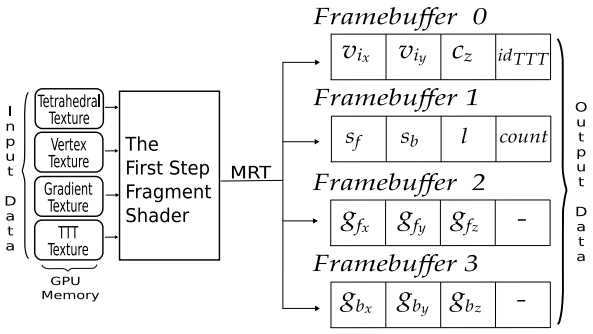



Figure 4. IPTINT input/output scheme of the first step fragment shader. The input data is composed by four textures: *tetrahedral*, containing four vertex indices per tetrahedron; *vertex*, containing the $x$, $y$ and $z$ coordinates and the scalar value $s$ for each vertex; *gradient*, containing the gradient vector for each vertex; *TTT* (Ternary Truth Table), containing all possible projection test results.

The PTINT and IPTINT code (with the TTT table), using C/C++ and GLSL, is made available with this thesis: http://code.google.com/p/ptint.

*HAPT [25]:* The idea of HAPT is to fit the original PT method better than PTINT to the GPU using the geometry shader. HAPT's framework is presented in Figure 5. First the visibility order is computed using any CPU or GPU method.

The ordered tetrahedra are streamed to the graphics pipeline through the vertex shader (VS), where each vertex encodes the information of one tetrahedron. After projecting the cells to screen space in the VS, the tetrahedra are decomposed into triangles in the geometry shader (GS). The triangle primitives are sent down the pipeline with scalar values and traversal length as color attributes for the direct volume rendering (DVR) technique, and face normals for the iso-surface rendering (ISO) technique. This process brings a great benefit highly desirable for a hardware-based approach: fitting a volumetric primitive using vertices and triangles that are well supported by graphics hardware. Finally, we use a fragment shader (FS) to evaluate the volume ray integral and render iso-surfaces. Note that, after sorting, the whole rendering algorithm is done in a single pass, performing the original PT technique entirely on the GPU, while taking full advantage of triangle rasterization dedicated processors. This streaming feature of HAPT allows it to handle time-varying data trivially as a sequence of static volumes per frame, as illustrated in Figure 1(b).

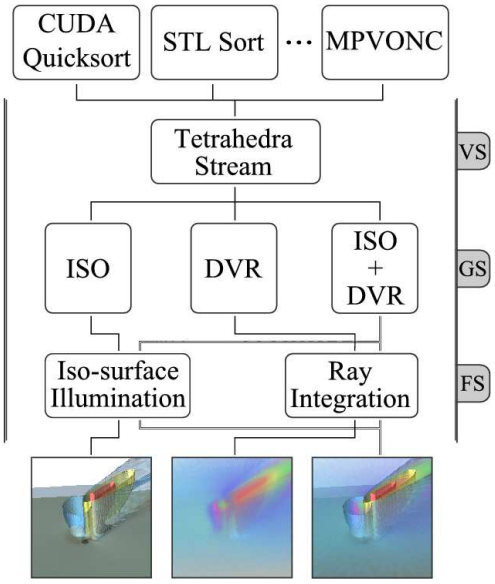



Figure 5. HAPT's framework divided into three GPU shaders. Any sorting method can be used prior to rendering, for instance we tested our method with four different methods: quicksort and bitonic sort on the GPU using CUDA; STL-based introsort and the MPVONC [9] algorithm on the CPU.

The HAPT code (with four different sorting methods), using C/C++, C for CUDA and GLSL, is made available with this thesis: http://code.google.com/p/hapt.

## III. Mesh Processing

In this thesis, we introduce *SAMPLE* – similarity augmented mesh processing using local exemplars – a method to propagate computations from one part of the mesh, the *local exemplar*, to many other similar regions. The concept of similarity in this scenario depends on the mesh properties needed by the mesh processing task. For example, in our test cases (detail transfer and mesh parameterization) we are interested in local shape properties and, thus, define the similarity descriptor as a heightmap for each vertex. This heightmap is computed either by hardware *Z-buffer* or shooting rays from the vertex tangent plane to the mesh surface. The heightmap is discretized as a $16 \times 16$ image containing height values and describing the local surface features. The image is normalized using the cross-correlation idea and converted to a set of Zernike coefficients to make the comparison of two heightmap images rotationally invariant. Finally, we compare mesh regions by applying Gaussian weights to the Zernike coefficients inside the region.

We use two spaces to accomplish similarity augmented mesh processing: *primal* and *dual* spaces. The primal space defines the regular neighborhood of a local surface patch, given by the mesh connectivity and geometry, and the dual space defines the similarity neighborhood, given by the distance between heightmap descriptors. The primal space is widely used by many existing mesh processing tasks, such as Laplacian smoothing [17]. The dual space, on the other hand, is scarcely used in mesh processing, and it is normally employed as a non-local neighborhood for averaging values [19]. We define and use the dual space in a more comprehensive manner. In our case, the dual neighbors define correspondences across regions in addition to the primal neighbors, aiding in the replication of processing done on a region to other close regions in the dual space. Figure 6 shows a number of dual neighbors of one vertex on the scale of the Stanford Asian Dragon model, the vertices in the primal neighborhood are illustrated for comparison.

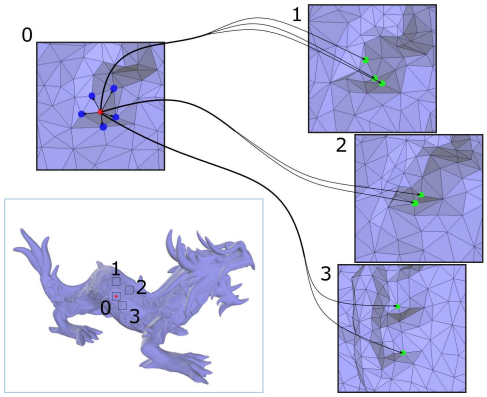



Figure 6. Illustration of primal and dual spaces. The immediate primal neighbors of the selected vertex (in red) are shown in blue (box 0). While several dual neighbors are shown in green (boxes 1, 2 and 3).

The SAMPLE algorithm propagates mesh processing by using the dual space and a *local exemplar*. An exemplar region is any region of the mesh with desired features, which is used to guide the propagation of a target processing task. For example, to paint all the scales of the Stanford Asian Dragon at the same time, one of the scales can be selected

as the exemplar region and, as the texture painting occurs, regions similar to the exemplar are painted with the same pattern (see Figure 1(c)). Another application is to use the dual space to propagate mesh editing throughout meaningful regions, depicted in Figure 7. Both applications use the same local exemplar and similarity distance threshold to affect the identical scales, illustrating that our SAMPLE algorithm can be employed in a similar manner for a broad range of mesh processing tasks.

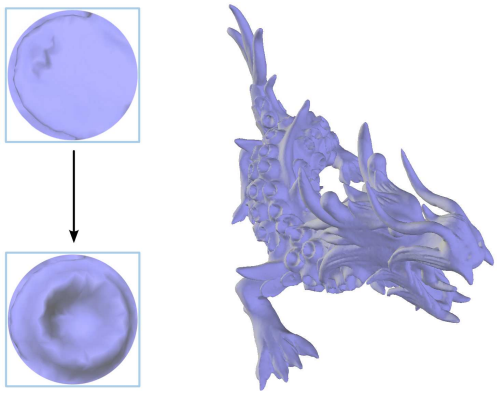


Figure 7. SAMPLE application: detail transfer. One exemplar region (top left) of a dragon scale is edited to resemble a crater (bottom left). Our similarity augmented algorithm propagates the editing result naturally over the similar scales.

The disadvantage of our approach is that it relies upon the size of the exemplar region, which is directly related to the size of the heightmap descriptor. If the local exemplar region is much smaller or larger than the heightmap, then the dual space fails to capture the distinctive features of the local shape, providing the incorrect similar neighbors upon which the processing is propagated. In these cases, the dual space has to be recomputed considering a heightmap area chosen to match the size of the desired region and yield the correct result.

The SAMPLE code is not available yet as the corresponding paper is under revision of the Graphical Models journal.

## IV. Results and Conclusions

We have presented several efficient GPU approaches for volume rendering and introduced a framework for similarity augmented mesh processing. We summarize several rendering results in Figure 1. Each volume rendering approach aims to offer advantages on a specific context: VF-Ray-GPU is a memory-aware algorithm able to render large datasets using ray casting in the GPU; RPTINT specializes the original PT method to regular data using graphics hardware, greatly increasing performance; IPTINT adapts PT using a two-step approach in the GPU, adding indirect volume rendering through iso-surface detection and gradient interpolation; HAPT is efficient by avoiding multiple rendering passes, and flexible allowing easy mixing and matching of other strategies, such as sorting methods and rendering techniques. The former two algorithms were published in the following international conferences: VF-Ray-GPU in Volume Graphics [20]; and RPTINT in GRAPP [22]. The latter two algorithms were published in the following journals: IPTINT in Computer Graphics Forum [23]; and HAPT in the EuroVis conference to appear in a special issue of Computer Graphics Forum [25]. We make available source codes along with each paper in the hope that the research contributions of this thesis are as reproducible as possible.

When compared against each other, VF-Ray-GPU does not present the rendering problems of interpolating scalar values and traversal length at extremities inherited from the PT idea. However, the PT-based approaches tend to be faster and cheaper in terms of memory consumption. RPTINT can handle only regular data, making it faster than the others when dealing with this type of volume. HAPT is the fastest irregular volume rendering algorithm, but it lacks iso-surface Phong shading presented in IPTINT. Further comparisons details are given on each corresponding paper.

The work presented on mesh processing was done during an internship program with Professor Varshney at the University of Maryland. In this work, we have defined a similarity measurement based on a heightmap image per vertex to be our surface descriptor in a new concept of dual space. Figure 8 depicts the Stanford Armadillo model in its dual space with respect to one vertex. The dual space is used to propagate mesh processing done on one region of a mesh to similar regions, using parameterization and editing as two applications example. Our method, called SAMPLE, was recently submitted to the Graphical Models journal.

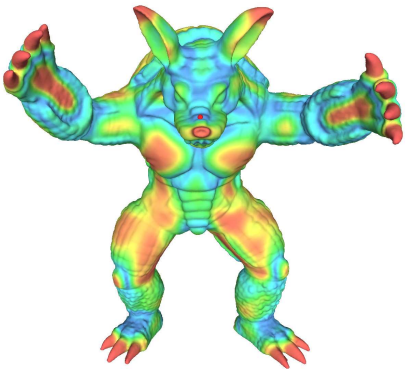


Figure 8. Dual space example for a given vertex (in red). The surrounding vertex region is used to map the entire mesh using the similarity neighborhood. Regions with wrinkles similar to the selected vertex on the nose are painted with blue, such as the ones in the arms and feet, while more dissimilar regions are gradually painted from green to red.

## Acknowledgment

We acknowledge the grant of the first author provided by Brazilian agency CNPq (National Counsel of Technological and Scientific Development).

## REFERENCES


[1] T. Klein, S. Stegmaier, and T. Ertl, "Hardware-Accelerated Reconstruction of Polygonal Isosurface Representations on Unstructured Grids," in *PG '04: Proceedings of the 12th Pacific Conference on Computer Graphics and Applications*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 186–195.

[2] B. Zehner, "Interactive Exploration of Tensor Fields in Geosciences using Volume Rendering," *Computers & Geosciences*, vol. 32, no. 1, pp. 73–84, 2006.

[3] S. Chen, H. Zhang, D. Yuen, S. Zhang, J. Zhang, and Y. Shi, "Volume Rendering Visualization of 3D Spherical Mantle Convection with an Unstructured Mesh," *Visual Geosciences*, vol. 13, no. 1, pp. 97–104, July 2008.

[4] R. Farias, J. S. B. Mitchell, and C. T. Silva, "ZSWEEP: An Efficient and Exact Projection Algorithm for Unstructured Volume Rendering," in *VVS'00: Proceedings of the 2000 IEEE Symposium on Volume Visualization*. New York, NY, USA: ACM Press, 2000, pp. 91–99.

[5] Brian Wylie, Kenneth Moreland, Lee Ann Fisk, and Patricia Crossno, "Tetrahedral Projection Using Vertex Shaders," *Volume Visualization and Graphics, IEEE Symposium on*, vol. 0, pp. 7–12, 2002.

[6] P. Shirley and A. Tuchman, "A Polygonal Approximation to Direct Scalar Volume Rendering," *SIGGRAPH Comput. Graph.*, vol. 24, no. 5, pp. 63–70, 1990.

[7] M. Weiler, M. Kraus, M. Merz, and T. Ertl, "Hardware-Based Ray Casting for Tetrahedral Meshes," in *VIS '03: Proceedings of the 14th IEEE Conference on Visualization*, 2003, pp. 333–340.

[8] R. Espinha and W. Celes, "High-Quality Hardware-Based Ray-Casting Volume Rendering Using Partial Pre-Integration," in *SIBGRAPI '05: Proceedings of the XVIII Brazilian Symposium on Computer Graphics and Image Processing*. IEEE Computer Society, 2005, p. 273.

[9] P. L. Williams, "Visibility-Ordering Meshed Polyhedra," *ACM Trans. Graph.*, vol. 11, no. 2, pp. 103–126, 1992.

[10] S. Callahan, M. Ikits, J. Comba, and C. Silva, "Hardware-Assisted Visibility Ordering for Unstructured Volume Rendering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 11, no. 3, pp. 285–295, 2005.

[11] J. Podolak, P. Shilane, A. Golovinskiy, S. Rusinkiewicz, and T. Funkhouser, "A Planar-Reflective Symmetry Transform for 3D Shapes," in *SIGGRAPH '06: Proceedings of the 33th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 2006, pp. 549–559.

[12] R. M. Rustamov, "Laplace-Beltrami eigenfunctions for deformation invariant shape representation," in *SGP '07: Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2007, pp. 225–233.

[13] J. Sun, M. Ovsjanikov, and L. J. Guibas, "A Concise and Provably Informative Multi-Scale Signature Based on Heat Diffusion," *Comput. Graph. Forum*, vol. 28, no. 5, pp. 1383–1392, 2009.

[14] A. Golovinskiy, J. Podolak, and T. Funkhouser, "Symmetry-aware mesh processing," in *Proceedings of the 13th IMA International Conference on Mathematics of Surfaces*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 170–188.

[15] S. Zelinka and M. Garland, "Similarity-Based Surface Modelling using Geodesic Fans," in *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. New York, NY, USA: ACM, 2004, pp. 204–213.

[16] R. Gal and D. Cohen-Or, "Salient Geometric Features for Partial Shape Matching and Similarity," *ACM Trans. Graph.*, vol. 25, no. 1, pp. 130–150, 2006.

[17] G. Taubin, "A Signal Processing Approach to Fair Surface Design," in *SIGGRAPH '95: Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: ACM, 1995, pp. 351–358.

[18] B. Levy, "Laplace-Beltrami Eigenfunctions Towards an algorithm that "understands" geometry," *International Conference on Shape Modeling and Applications*, 2006.

[19] O. Schall, A. Belyaev, and H.-P. Seidel, "Feature-preserving non-local denoising of static and time-varying range data," in *SPM '07: Proceedings of the ACM Symposium on Solid and Physical Modeling*. New York, NY, USA: ACM, 2007, pp. 217–222.

[20] André Maximo, Saulo Ribeiro, Cristiana Bentes, Antônio Oliveira, and Ricardo Farias, "Memory Efficient GPU-Based Ray Casting for Unstructured Volume Rendering," in *VG-PBG*. Los Angeles, California, USA: Eurographics Association, 2008, pp. 155–162.

[21] Saulo Ribeiro, André Maximo, Cristiana Bentes, Antonio Oliveira, and Ricardo Farias, "Memory-Aware and Efficient Ray-Casting Algorithm," *Computer Graphics and Image Processing, Brazilian Symposium on*, vol. 0, pp. 147–154, 2007.

[22] André Maximo, Ricardo Marroquim, Ricardo Farias, and Claudio Esperança, "GPU-Based Cell Projection for Large Structured Data Sets," in *GRAPP (GM/R)*. INSTICC – Institute for Systems and Technologies of Information, Control and Communication, 2007, pp. 312–322.

[23] Ricardo Marroquim, André Maximo, Ricardo Farias, and Claudio Esperança, "Volume and Isosurface Rendering with GPU-Accelerated Cell Projection," *Computer Graphics Forum*, vol. 27, pp. 24–35, 2008.

[24] Ricardo Marroquim, André Maximo, Ricardo Farias, and Claudio Esperanca, "GPU-Based Cell Projection for Interactive Volume Rendering," *Computer Graphics and Image Processing, Brazilian Symposium on*, vol. 0, pp. 147–154, 2006.

[25] André Maximo, Ricardo Marroquim, and Ricardo Farias, "Hardware-Assisted Projected Tetrahedra," *Computer Graphics Forum (to appear)*, 2010.