High-Performance Volume Rendering for 3D Heart Visualization

Andre Maximo
COPPE-UFRJ
andre@lcg.ufrj.br

Ricardo Marroquim
COPPE-UFRJ
ricardo@lcg.ufrj.br

Claudio Esperança
COPPE-UFRJ
esperanc@cos.ufrj.br

Rodrigo Weber dos Santos
DCC-UFJF
rodrigo@dcc.ufjf.br

Cristiana Bentes
DESC-UERJ
cris@eng.uerj.br

Ricardo Farias
COPPE-UFRJ
rfarias@cos.ufrj.br

Abstract

Cardiovascular diseases are the the leading cause of death and disability in the world. Non-invasive tecniques are required to reduce the number of deaths as well as the patients quality of life. These techniques usually rely on 3D visualization of MRI or CT data. In this work we describe how improved volume rendering techniques, combined with graphics cards programming can provide interactive visualization of the heart internal structures. Our main focus here is to provide doctors with high-performance 3D images for evaluating patient heart anatomy and performance. Our idea is to take full advantage of the triangle-rendering hardware to provide interactive frame rates.

1 Introduction

Cardiovascular diseases are the the leading cause of death and disability in the world. The numbers are increasing as the epidemic of heart disease and stroke continues. In Brazil, about 4 million people suffer from heart failure. The health burden of cardiovascular diseases is matched by its economic burden. In São Paulo, for example, medical expenses for diagnostic tests, surgeries and hospital use about 20% of the public health funds.

Therefore, early diagnosis and treatment are the key to reduce the number of deaths and the economic burden in Brazilian Health System (SUS). Seeing inside the heart and evaluating its health are the first steps in deciding on the best treatment. Today, there are a number of tests that can be performed without any surgery involved [7, 8]. These tests, however, require high quality heart images. Recently, biomedical imaging modalities such as MRI, CT or Ultrasound have become essential in generating high quality heart images in order to identify and localize structures and abnormalities. To understand the data generated by these devices, however, physicians and researchers need to visualize the heart in a lifelike way. To do so, 3D visualization techniques are needed.

Most of traditional tools and techniques for volume visualization allow researchers to explore only the surface of 3D datasets. Direct volume rendering techniques, on the other hand, convey more information than surface rendering methods, enabling the viewer to fully reveal the internal structure of 3D data. For the heart visualization application, direct volume rendering techniques are crucial to gain detailed insight into the heart, in order to examine, explore and precisely pinpoint problem areas before determining final treatment strategies.

Nevertheless, direct volume rendering is notoriously a memory and computationally intensive task. Recently, the increasing capability and performance of commercial graphics hardware made hardware-assisted volume rendering systems a fast alternative for the visualization of medical data. In this paper we present a volume renderer for heart data that is implemented in the programmable Graphics Processor Unit (GPU) of recent graphics cards. Our idea is to show how the use of the graphics hardware can improve the performance of the heart 3D visualization, providing a deeper insight of the data with interactive times.

2 Related Work

Hardware-accelerated volume rendering is nowadays widely used for visualization. There are a number of works in this area. Stein *et al.* [10] developed a new volume rendering approximation using 2D texture mapping. This method attenuated the artifacts produced by linearly approximating the non-linear opacity effects. Röttger *et al.* [6] improved Stein's results by using 3D textures and generalized it for non-linear transfer functions. Lum *et al.* [3] proposed a faster method to compute the pre-integration. This work also introduced lighting effects that produced

more accurate rendering while producing less artifacts. Espinha and Celes [2] propose a new hardware-based ray-casting (HARC) algorithm, introduced by Weiler *et al.* [12]. They implement the HARC algorithm using partial pre-integration.

Our rendering algorithm is based on the Projected Tetrahedra (PT) technique introduced by Shirley and Tuchman [9]. The PT algorithm has grown in importance in recent years due to the advance of programmable graphics hardware. Wylie et al. [14] adapted the Shirley and Tuchman algorithm to programmable graphics hardware using the vertex shader. The use of programmable graphics card was further explored by Weiler et al. [13], as an extension of their earlier View Independent Cell Projection algorithm (VICP) [11].

None of these works, however, explore the use of hardware-accelerated volume rendering to generate realistic 3D images of medical MRI data. The work by Zhang *et al* [5] builds a real-time 3D rendering engine based on ray-casting, directly running on the graphics vertex and fragment processors, in which the OpenGL Shading Language (GLSL) is utilized as a GPU programming API. They propose a new dynamic volume texture binding scheme, embedded it into our 3D rendering engine to permit real-time visualize the dynamic beating heart. The work by Rober *et al* [1] describe a flexible framework for GPU-based multi-volume rendering, which provides a correct overlaying of an arbitrary number of volumes and allows the visual output for each volume tobe controlled independently. Their tool is specific for the rendering of functional brain images,

3 Volume Rendering Algorithm

Many algorithms for direct volume rendering have been proposed in the past, e.g. cell projection, ray casting and sweep-plane. Our approach is based on cell projection, where each polyhedral cell of the volumetric data is projected onto the screen [4]. Our idea is to take full advantage of the triangle-rendering hardware to provide interactive frame rates. More specifically, we implemented a algorithm for tetrahedral projection that is based on the Projected Tetrahedra (PT) algorithm [9]. Following we give a brief description of the PT algorithm and after that we explain our approach.

3.1 Projected Tetrahedra Algorithm

In a nutshell, the PT algorithm consists in projecting the tetrahedra onto the screen space and composing them in visibility order. Each projected tetrahedron is decomposed into triangles, and an approximation of the ray integral is used to compute the color and opacity values for the vertices. When rasterizing the primitives, an absorption illumination model is used to compute the color of the pixels by summing the contributions of every semi-transparent triangle in back-to-front order.

The tetrahedra's projected shape is classified into one of four possible cases. Each possible case defines a projection class, and generates a different decomposition into triangles. For each projection, the *thick* vertex is defined as the entry or exit point of the ray segment that traverses the maximum distance through the tetrahedron. All other projected vertices are called *thin* vertices. The scalar values of the ray's entry and exit points (s_f and s_b) are determined by interpolating the scalars of the thin vertices. The distance traversed by the ray segment is the thickness l of the cell.

3.2 Our Approach

Our algorithm is divided in two main parts and, consequently, in two different shaders. In the first step, all data is processed per tetrahedron, that is, the projection class, the thick vertex properties, and the z coordinate of the tetrahedron's centroid are computed. The second step interpolates the vertices' scalar values to compute color and opacity values for each fragment.

To speed up the rasterization process, we make use of vertex arrays and the primitives are drawn as triangle fans ($GL\_TRIANGLE\_FAN$). To draw each fan correctly the order and number of vertices must be determined in the first step which are passed to the second one, as will be described.

3.2.1 Sorting

One of the bottlenecks of the Cell Projection algorithm is the need to sort the cells in visibility order before rendering. The centroids of each tetrahedra are calculated and used by a CPU sorting algorithm, to put them in depth order.

Two sorting algorithms are used in our implementation: a quicker but less precise bucket sort is used whenever the viewing transformation is changing, whereas a standard $O(n\log n)$ merge sort algorithm is employed for still frames.

3.2.2 First Step

The first shader computes the scalar value at the thick vertex, the cell's thickness and determines the vertex order and number of triangles in the fan. All data used in this shader is stored in GPU memory by creating three different *RGBA* textures: the Tetrahedral Texture, the Vertex Texture and the Classification Texture. The Vertex Texture stores the x, y and z coordinates and the scalar s for each vertex. The Tetrahedral Texture stores the four values which are used for retrieving the four vertices of the tetrahedron from the Vertex Texture These two texture lookups eliminate the need

for vertex attributes and reduce the data transfer overhead from CPU to GPU.

The vertices are then projected to screen coordinates and the projection class is determined by means of tests similar to the ones used by Wylie's [14], in his method, except that we expanded the set of test to treat also, all degenerate cases. In addition, our method avoids computational redundancy by performing the tests once per tetrahedron rather than once per vertex.

3.2.3 Second Step

The second step linearly interpolates vertex colors. Each final color is computed by interpolating the values s_f, s_b and l of the triangle's vertices. The color is computed by evaluating the chromaticity and extinction coefficient (τ) of the transfer function at the average interpolated value $(s_f + s_b)/2$. The opacity, on the other hand, is estimated by

$$\alpha = 1 - e^{-\tau l}. \quad (1)$$

Finally, the pixels are composited in back-to-front order, and, for each new color added to the frame buffer, the new final color is computed by:

$$C_{new} = Alpha \times Color + (1 - Alpha) \times C, \quad (2)$$

where C and C_{new} are the previous and new colors stored in the frame buffer, and $Color$ and $Alpha$ are the interpolated scalar and thickness values. Alpha is the transparency factor.

We used two different 1D textures in the second step, one for the transfer function table values and another one for the opacity calculation with values obtained with Equation 1, that is, Tex1D$(u) = e^{-u}$, for u sampled over interval $[0, 1]$, in order avoid on-the-fly computing the exponential function.

4 Experimental Results

Our experimental environment consists of a Intel Pentium IV 3.6 GHz, 2 GB RAM, with a nVidia GeForce 6800, 256 MB graphics card, PCI Express 16x bus interface, from the Computer Graphics Laboratory at COPPE/UFRJ. All rendered image are $512x512$ in size.

4.1 Dataset

We tested our visualization method with three different datasets, acquired by a MRI catscan. The first sequence was taken upwards, from the feet to the head, labeled as down-up. The second goes from front to back, while the third goes from left to the right. Each sequence has dimensions $256x256x20$, or 1.3 million hexahedra. This represents 6.5 million tetrahedra.

For the down-up sequence, Figure 1 shows part of the heart, but only a slice, making it impossible to see the internal structures. Figure 2 shows two images created by the three-dimensional rendering technique of cell projection. Figure 2.a presents the result generated by using a more opaque transfer function, which hides some internal details. Figure 2.b, created with a more transparent transfer function, allows the visualization of internal structures in more details.

Figure 1. Slice of the MRI of the heart taken from the down-up sequence.

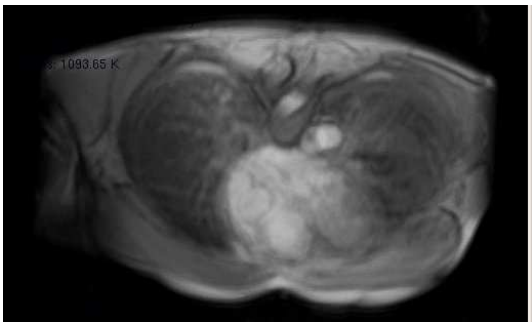
Figure 3 brings five slices of the MRI to compare with the two images shown in Figure 4, created by our cell projection method. It must be noted also, that using volume rendering, it is possible to visualize the dataset from any desired point of view, and yet see the complete structure of the body.

Figure 5 shows one slice of the MRI taken from the left side to the right. The problem with this sequence is that due to the low resolution, twenty slices for all width of the body, only eight slices contain cross sections of the heart. Figure 6 is the rendered image. It can be noted the very low resolution obtained.

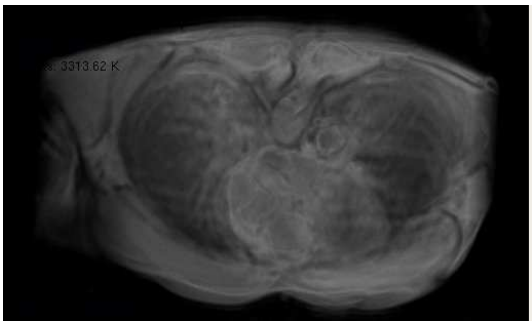
4.2 Performance Evaluation

Table 1 shows more details about the three sequences used in our tests. We trimmed the datasets to eliminate regions of noise, and brought down a little bit the size of the datasets. The table also shows the frame rate, about 0.34 frame per second, and the number of tetrahedra projected per second, about two million cells per second.

The graphics card programming introduced a very powerful resource to improve the cell projection method, even though some modifications had to be made to the implementation to adequate each step of the process to the limitations imposed by the current graphics hardware.



(a) More opaque

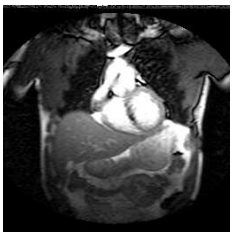


(b) More transparent

Figure 2. a) In this figure, the transfer function was set more opaque. b) With a slightly more transparent transfer function, it is possible to see more internal estructures of the heart.

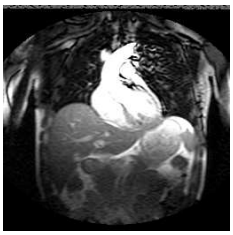
Data	Vertices	Tetrahedra	fps	M tets/s
Down-Up	1.27 M	5.98 M	0.35	2.183
Front-Back	1.27 M	5.98 M	0.34	2.091
Left-Right	1.27 M	5.98 M	0.33	2.067

Table 1. Performance and computed data from original datasets.



(a) Slice 4

(b) Slice 5



(c) Slice 6

(d) Slice 7

(e) Slice 8

Figure 3. Each slice shows part of the heart structure, but does not allow the visualization of the whole structure.

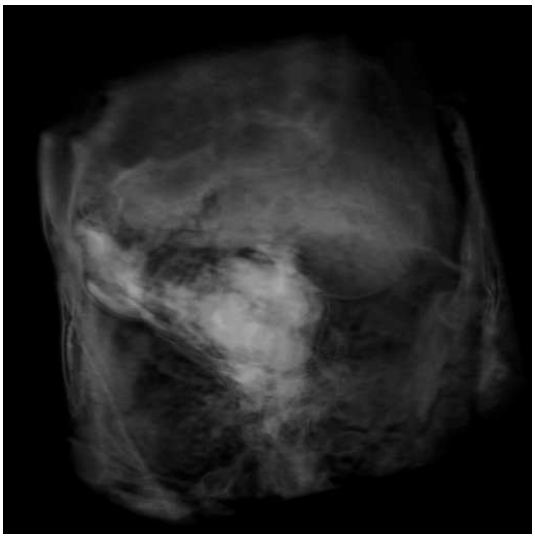
It must be noted that medical data are usually acquired in regular grid format. To represente the geometry of these type of grid, one can make use of eight neighboring vertices of the grid, to define an information unit of the 3D data, called voxel, which is a hexahedral cell. A grid composed by hexahedra cells is said to be a hexahedral grid. By the other hand, the current limitations of the graphics cards, we could only implement the cell projection algorithm to treat tetrahedal cell. To visualize the hexahedral grid, we have to convert, each hexahedral cell into 5 tetrahedral cells, to send them to the graphics hardware. The drawback of this procedure is that, since our method stores the data information, inside the graphics card, using texture memory of the card, the size of the file that can be treated is limited by the available card's memory. In our tests, we could only process up to 6.5 million tretrahedra cells. This limits the size for the treatable regular grids to be at most: $100x100x100$, which leads to $5M$ cells. This would be a safe data size.

5 Conclusions

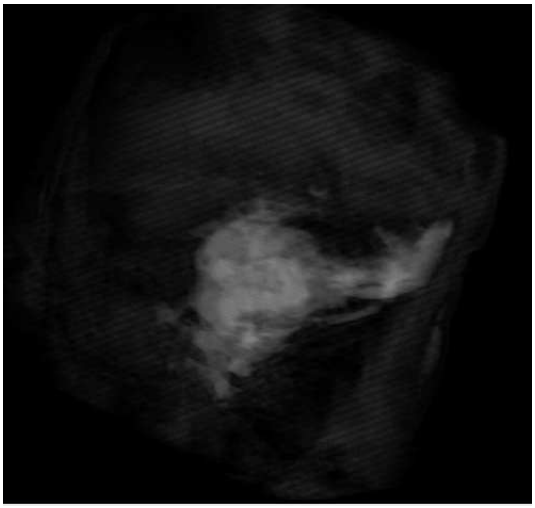
In this work we presented a volume rendering algorithm for the 3D visualization of medical data that takes full advantage of modern graphics hardware. Our approach is based on the implementation of a cell projection algorithm in the Graphics Programming Unit (GPU). Our focus was to develop an interactive volume rendering engine to provide doctors with high-performance 3D images engine to evaluate patient anatomy and performance.

We tested our algorithm with three datasets obtained from MRI scanner and obtained 0.4 frames per seconds, projecting up to 6 millions tetrahedral cells.

As future work we intend to extend our implementation to run on graphical clusters. For this end, we intend to dis-



(a) More opaque



(b) More transparent

Figure 4. Once again, it is possible to investigate internal structures. (a) It is shown more information, including part of the lung, while in (b) using more transparency, only the heart structure appears.

(a) Slice 10

Figure 5. Central slice from MRI taken from left to right.

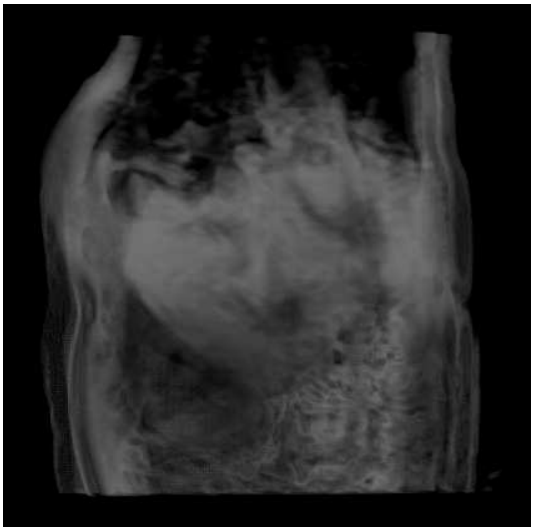


Figure 6. Due to the very low resolution of this sequence, it is very difficult to distinguish any information out of this rendered image.

tribute the data between the nodes of the cluster, diminishing the limitation for the number of voxels that can be processed. Even if a node receives more voxels that it can project, a scheme will be implemented to project the set in layers, which will fit in the graphics card's memory, and perform a final compositing, afterwards. Another scheme that will be implemented is a segmentation over the raw data, to precisely define the region of interest, bringing further down the size of data to be projected, and more important, to generate cleaner rendered images.

References

[1] F. R. E. T. T. F. T. Ertl and M. Knauff. Gpu-based multivolume rendering for the visualization of functional brain images. In *In Proceedings of SimVis*, pages 305–318, 2006.

[2] R. Espinha and W. Celes. High-quality hardware-based raycasting volume rendering using partial pre-integration. In

SIBGRAPI '05: Proceedings of the XVIII Brazilian Symposium on Computer Graphics and Image Processing, page 273. IEEE Computer Society, 2005.

[3] E. Lum, B. Wilson, and K.-L. Ma. High-quality lighting and efficient pre-integration for volume rendering. In *IEEE TVCG Symposium on Visualization 2004*. The Joint Eurographics-IEEE TVCG Symposium on Visualization 2004, 2004.

[4] A. Maximo, R. Marroquim, C. Esperanca, and R. Farias. Gpu-based cell projection for interactive volume rendering. In *to appear in Brazilian Symposium on Computer Graphics and Image Processing*, 2006.

[5] R. P. Qi Zhang; Eagleson and T.M. Real-time visualization of 4d cardiac mr images using graphics processing units biomedical imaging: Macro to nano. In *3rd IEEE International Symposium on Volume*, pages 343–346, April 2006.

[6] S. Rttger, M. Kraus, and T. Ertl. Hardware-accelerated volume and isosurface rendering based on cell-projection. In *VIS '00: Proceedings of the conference on Visualization '00*, pages 109–116, Los Alamitos, CA, USA, 2000. IEEE Computer Society Press.

[7] R. W. d. Santos, F. O. Campos, L. N. Ciuffo, A. Nygren, W. Giles, and H. Koch. Atx-ii effects on the apparent location of m cells in a computational human left ventricular wedge. *Journal of Cardiovascular Electrophysiology*, 17(Suppl 1), 2006.

[8] R. W. d. Santos and H. Koch. Interpreting biomagnetic fields of planar wave fronts in cardiac muscle. *Biophysical Journal*, 88(5), 2005.

[9] 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(5), pages 63–70, 1990.

[10] C. Stein, B. Becker, and N. Max. Sorting and hardware assisted rendering for volume visualization. In A. Kaufman and W. Krueger, editors, *1994 Symposium on Volume Visualization*, pages 83–90, 1994.

[11] M. Weiler, M. Kraus, , and T. Ertl. Hardware-based view-independent cell projection. In *VVS '02: Proceedings of the 2002 IEEE Symposium on Volume visualization and graphics*, pages 13–22, Piscataway, NJ, USA, 2002. IEEE Press.

[12] 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 '93*, pages 333–340, 2003.

[13] M. Weiler, M. Kraus, M. Merz, and T. Ertl. Hardware-based view-independent cell projection. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):163–175, 2003.

[14] B. Wylie, K. Moreland, L. A. Fisk, and P. Crossno. Tetrahedral projection using vertex shaders. In *VVS '02: Proceedings of the 2002 IEEE Symposium on Volume visualization and graphics*, pages 7–12, Piscataway, NJ, USA, 2002. IEEE Press.