Interactive GPU-based Volume Rendering for Medical Image

Shihao CHEN
Institute of Electronic and Information Engineering
Northwestern Polytechnical University
Xi'an 710072, China
Technical University Munich
Munich, Germany

Chongyang HAO
Institute of Electronic and Information Engineering
Northwestern Polytechnical University
Xi'an 710072, China

***Abstract*—As we know that the 3D medical images are usually in large scalar, the traditional volume rendering method which running on CPU can not achieve interactive rendering speed. Recently, GPU-accelerated direct volume rendering has positioned itself as an efficient tool for the display and visual analysis of volume data. To facilitate the usage of well developed hardware resource, a graphics processing unit (GPU)-based volume ray-casting algorithm is proposed in this paper. The proposed algorithm implement ray casting operation completely in GPU. The algorithm re-samples volume data, represented as a stack of 3D texture, onto a sampling surface. The ray-casting algorithm performs in fragment shaders. Additionally we apply early ray termination. At last we demonstrate the effectiveness of our algorithm with several medical volume data. It proved that the proposed algorithm can perform an interactive rate even for direct volume rendering while keeping the high image quality.**

Keywords-Volume Rendering; Medical Image; Texture; Graphic Processing Unit(GPU)

I. INTRODUCTION

New developments in medical imaging modalities lead to ever increasing sizes in volumetric data. The ability to visualize and manipulate the 3D data interactively is of great importance in the analysis and interpretation of the data. The interactive visualization of such data is a challenge, since the frame rate is heavily depending on the amount of data to be visualized. Due to the large number of trilinear interpolations that must be processed in order to produce image results of high quality, the availability of direct volume rendering has yet been restricted to high-end workstations and special purpose graphics hardware. Although there is a clear trend toward standard PC hardware as visualization platform [1], the application of interactive hardware-accelerated approaches is still limited. Volume rendering techniques that exploit the 2D-texturing hardware of PC graphics boards usually produce images that contain visual artifacts. The basic 2D-texture based approach is to decompose the volume into a set of object aligned slices. The necessary tri-linear interpolation can then be reduced to a bilinear interpolation which can be efficiently computed by standard texturing hardware. However, when zooming closely on a small detail inside the volume data, which is often done in medical applications, the missing tri-linear interpolation is strongly visible. Driven by the mass market of computer games and entertainment software, Graphic Processing Units (GPU) have become more flexible and powerful. Today, texture based approaches have positioned themselves as efficient tools for the direct rendering of volumetric scalar fields on graphics workstations or even consumer class hardware[2]. In this paper we propose a interactive GPU-based volume rendering method that performs volume ray-casting on programmable graphics hardware. The ability to leverage the embedded tri-linear interpolation hardware is at the core of this method. In section 2, we present an overview of related work Section 3 deal with the details of our approach. In section 4, the results are presented and discussed, and in section 5 we summarize our conclusions.

II. RELATED WORKS

There is a variety of different visualization approaches for scalar volumes in multiple application scenarios. Recent approaches are categorized into indirect methods, such as isosurfaces extraction [3, 4], and direct methods, that immediately display the voxel data. We will focus on interactive direct methods. The basic idea of using object-aligned slices to substitute tri-linear by bilinear interpolation was presented by Lacroute and Levoy [5], although the original implementation did not use texturing hardware. For the PC platform, Brady et al. [6] have presented a technique for interactive volume navigation based on 2D-texture mapping. Meißner et al. [7] have provided a method to enable diffuse illumination for semi-transparent volume rendering. Roettger et al. [8] describe a GPU-based pre-integrated texture-slicing including advanced lighting. The authors also describe a GPU-based ray tracing approach with early ray termination. Krüger and Westermann [9] propose a method to accelerate volume rendering based on early ray termination and space-skipping in a GPU-based ray casting approach. In comparison to these techniques, we provide an approach that uses 3D texture hardware interpolation and implement ray casting operation completely in GPU. The approach also provides an implementation of the fast isosurfaces algorithm using 3D-texturing hardware.

978-1-4244-4134-1/09/$25.00 ©2009 IEEE

III. OUR APPROACH

A. Texture based visualization

Texture-based visualization is a technique that exploits the texture-based functionality of the graphics hardware. Regular grid data is stored directly in the GPU memory as textures. Early graphics boards only supported 2D textures in hardware, which made volumetric visualization only possible by representing volumetric data as stacks of 2D images. Preliminary approaches sample the volume with parallel planes (proxy geometry) that change according to the the viewing parameters. To avoid artifacts, data is replicated in the three directions, increasing memory usage. Avoiding this extra storage requires on-the-fly slice reconstruction at the expense of performance overhead. The advent of 3D texturing capabilities even further improved the ways that the graphics hardware can be used to perform volumetric visualization. Since the volume data is naturally stored as a 3D texture, it suffices to use a proxy geometry consisting of a set of parallel polygons orthogonal to the viewing direction. It is possible to render slices parallel to the image plane with respect to the current viewing direction (see Figure. 1)

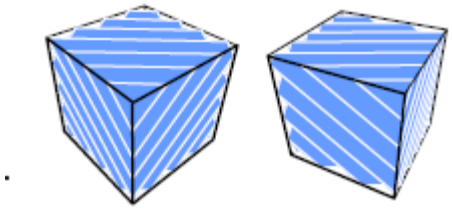

Figure1. Viewport-aligned slices

This means that if the viewing matrix changes, these viewport-aligned slices must be recomputed. Since tri-linear texture interpolation is supported by hardware, this can be done at interactive frame rate. In the final compositing step, the textured polygon slices are blended back-to-front onto the image plane, which results in a semitransparent view of the volume. With this approach it is easy to enhance image quality just by increasing the number of slices. However, in order to obtain equivalent representations of the volume data while changing the number of slices, opacity values must be adapted to the varying slice distance. Although the correct scaling factor is a function of the opacity value, in most cases scaling the values linearly with a constant factor according to the slice distance is a visually adequate approximation. Empty-space skipping techniques allow processing to quickly skip entire regions of the volume that do not contain relevant information, which is often the case in sparse datasets.

B. Ray casting on GPU

In order to take advantage of the built-in tri-linear filtering, the volume data is stored in a 3D-texture. First, a bounding box for this dataset is created where the position inside the dataset, then the following steps will be performed:

- Compute the viewing vector at any given pixel by subtracting the color of the front faces of the color cube at this pixel from he color of the back faces at this pixel.
- Normalizing and storing this vector in a 2D-texture of the exact size of the current viewport (together with its initial length in the alpha channel) yields a 'direction texture' for every screen pixel.
- Casting through the volume : Render the front faces again (the entry points into the dataset) and step along the viewing vector for this pixel (stored at the same position in the direction texture) until the ray has left the bounding box again (i.e. the casted distance is greater than the alpha value of the texture, where the initial length was stored).
- Compositing the final color in a separate texture, which is blended onto the screen.

Since loops and conditionals can be performed in a fragment shader, so we can only with a single call to cast through the volume.

C. Early-ray termination

To be able to perform early ray termination, the volume has to be traversed in a front-to-back order. This can be done by evaluating the volume rendering integral in discrete steps, using the under operator:

$$C_{i+1} = (1 - A_i)a_i c_i + C_i \quad (1)$$

$$A_{i+1} = (1 - A_i)a_i + A_i \quad (2)$$

Whereby C, A denote the color, respectively the opacity value of the current ray, c, α the color and opacity value given by applying the transfer function to the current sample in the volume, and i denotes the sample index. A ray is then saturated when A_i approximates 1. Before a brick is rendered, early ray termination is applied to its destination pixels. This is tested by executing a fragment shader program, while drawing a solid bounding box around the brick with back face culling switched on. The fragment shader program writes the maximal value in the depth buffer for saturated rays. When slicing the brick texture the early z-test will prevent any fragment operations to be executed for those rays, reducing the load on the rasterization and fragment shader. Early ray termination is only performed once per brick, and not more often because the overhead involved would otherwise annihilate the benefits.

D. Shaded isosurfaces rendering

We exploit rasterization hardware to display shaded isosurfaces. This method evaluates the equation of local illumination:

$$I = I_a + I_d(\vec{n} \bullet \vec{l}) \quad (3)$$

Where $\vec{l}$ is the direction of light and $\vec{n}$ is the normal of the isosurfaces which coincides with the volume gradient. I_a and I_d are the intensities of ambient and diffuse light. At the core of the algorithm the vector components of the voxel gradients are stored in the RGB components of a 3D-texture image. Additionally the intensity values are stored in the alpha-component. The volume is then rendered into the frame buffer using the alpha-test to display the specified isovalues only.. The frame buffer, that contains the voxel gradient coded in RGB components, is reinserted into the rasterization pipeline and the color matrix is used to compute the dot-product with the light vector

.

IV. RESULTS

The presented algorithms were implemented on Windows Vista platform on a standard PC(A single processor Intel(R) Core(TM)2 CPU 6600@ 2.40GHZ 2.39GHZ equipped with GeForce 8800 GTX GPU with 768 MB local video memory).

TABLE I. TIMINGS (FPS) FOR DIRECT VOLUME RENDERING

Data sets	a	b	c	d
View port	1264x 958	1280x 958	1264x 958	1264x 958
Data size	256x256x256	256x256x256	512x512x256	256x256x84
Timings	51.42	64.73	46.99	92.98

The proposed method is not restricted to volumetric data sets. Hence any renderable representation, such as point-based structures, or triangle meshes can be used. In Figure2: a is a skull CT scan data set; b is a human head MRI data set; c is a pet data set for chest; d is a head CT scan data set. From Figure2 we can see that the proposed method can produce high quality image for direct volume rendering.

TABLE I is the timings for volume rendering. The datasets are the same as in Figure2. Even the data set as big as 512x512x256, timings is almost 47 fps for the 1280x958 viewport. This speed is sufficient to interactive rendering.

TABLE II. TIMINGS (FPS) FOR ISO SURFACES VOLUME RENDERING

Data sets	a	b	c
View port	1264x958	1264x958	1264x958
Data size	512x512x1884	256x256x256	512x512x256
Timings	611.52	691.60	691.85

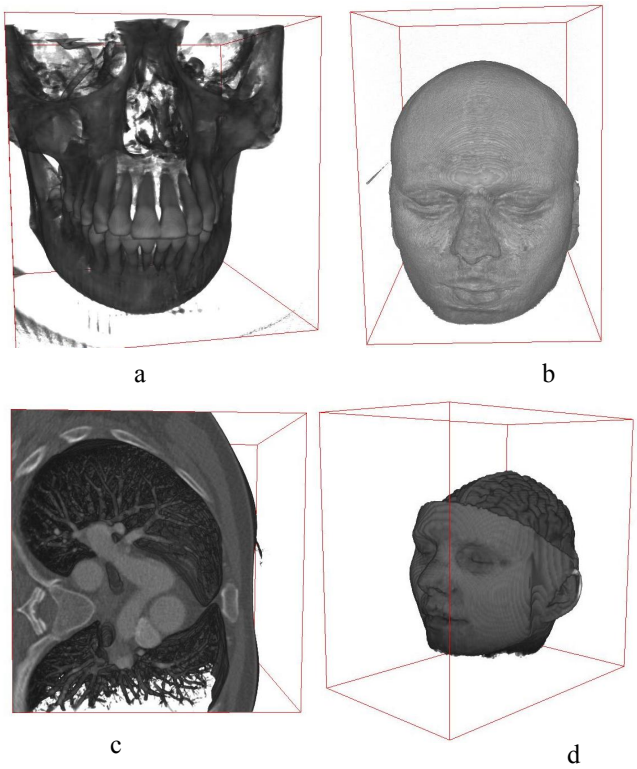

Figure2. Images show the results of direct volume rendering

In Figure3: a is visual human male ct scan data set; b is a human head MRI data set the same as in Figure2 b; c is a pet data set for chest the same as in Figure2 c; From Figure3 we can see that the proposed method also can produce high quality image for iso surfaces rendering. The data set a is as big as 512x512x1884, when rendering the timings can reach almost 611 fps (TABLE II).

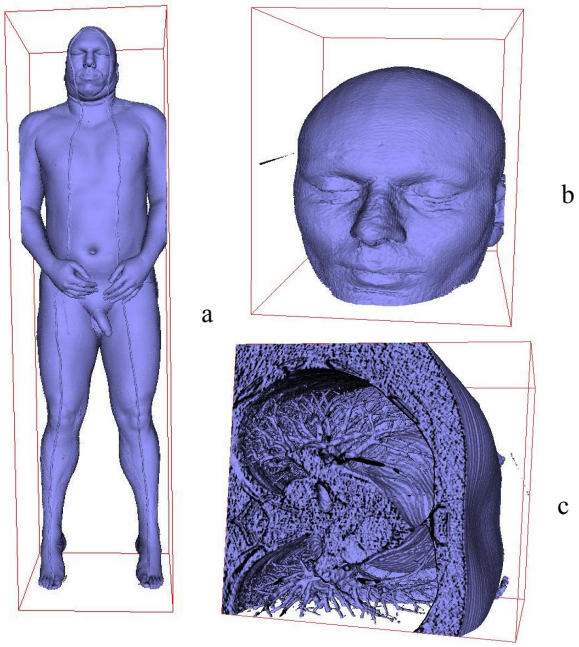

Figure3. Images show the results of iso surfaces rendering

V. CONCLUSIONS

In this paper, we have proposed a interactive visualization method for medical image based on volume ray-casting on DirectX10-level graphics hardware that is programmable via the pixel Shader 4.0 API. Our method makes full use of the strong powder of GPU especially the tri-linear interpolation to accelerate rendering speed. Besides for volume ray-casting back-to-frond order is used when the voxel's opacity is accumulated to a threshold the after calculation will be discard. Furthermore, advanced algorithms like fast isosurface display or shaded volume rendering are efficiently adapted to the PC platform. By means of the GPU's strong ability and the acceleration techniques, the proposed method is capable of efficiently rendering large volumetric data sets in real time. This method is an intuitive and interactive method. Since only low-cost hardware is required, the presented approaches significantly contribute to the availability of interactive direct volume rendering in practice.

Graphics processors have their own strengths, and exploiting these strengths leads to completely different techniques than in CPU based approaches. Fortunately, the ongoing evolution of graphics cards will allow for even more efficient algorithms in the near future, and with the speed of GPUs growing at a much faster pace than that of CPUs, we're looking into a bright future for GPU-based approaches. We have shown that it is already possible to implement a full-fledged ray casting environment

The graphics industry are introducing more powerful hardware at an impressive pace. However developments in medical imaging modalities are equally impressive, resulting in larger volume data sets. It means that in the future the method that was proposed here will preserve their value.

REFERENCES

[1] H. Pfister. Why the PC will be the most pervasive visualization platform in 2001. In Visualization '99, 1999.

[2] A. Van Geldern, and K. Kwansik. Direct Volume Rendering with Shading via Three-Dimensional Textures. In ACM

[3] K. Engel, R. Westermann, and T. Ertl. Isosurface extraction techniques for web-based volume visualization. In Visualization '99, 1999.

[4] W.E. Lorensen and H.E. Cline. Marching Cubes: A High Resolution 3D Surface Reconstruction Algorithm. Comp. Graphics, 21(4), 1996

[5] P. Lacroute and M. Levoy. Fast Volume Rendering Using a Shear–Warp Factorization of the Viewing Transform . Comp. Graphics, 28(4), 1994.

[6] M. Brady, K. Jung, Nguyen HT, and T. Nguyen. Two- Phase Perspective Ray Casting for Interactive Volume Navigation. In Visualization '97, 1997.

[7] M. Meißner, U. Hoffmann, and W Straßer. Enabling Classification and Shading for 3D Texture Based Volume Rendering Using OpenGL and Extensions. In Visualization '04, 2004

[8] S. Roettger, S. Guthe, D. Weiskopf, T. Ertl, and W. Strasser. Smart Hardware-Accelerated Volume Rendering. In VisSym'03: Proc. of the symposium on Data Visualisation 2003, pp. 231-238, 2003.

[9] J. Krüger and R. Westermann. Acceleration Techniques for GPU-based Volume Rendering. In Proc. IEEE Visualization 2003, pp. 287-292, 2003