Java Based Volume Rendering

Ruida Cheng[1], Alexandra Bokinsky[3], Paul F. Hemler[2], McCreedy Evan[1], Matthew J. McAuliffe[1]

1. Center for Information Technology, National Institutes of Health, Bethesda, MD.
2. Hampden-Sydney College, Hampden-Sydney, VA.
3. Geometric Tools, Inc. Chapel Hill, NC

INTRODUCTION

In recent years the number and utility of 3D rendering frameworks has grown substantially. A quantitative and qualitative evaluation of the capabilities of a subset of these systems is important to determine the applicability of these methods to specific medical visualization tasks. Libraries evaluated in this paper include the Java3D API, Jogl (Java OpenGL API), a multi-histogram based rendering method, and the WildMagic visualization libraries. Volume renderer implementations using each of these frameworks were developed using the platform-independent Java programming language. In addition, all four frameworks were implemented or ported to the Java language thereby allowing a complete Java based solution. Quantitative performance measurements (frames per second, memory usage) were used to evaluate the strengths and weaknesses of each renderer implementation.

METHODS

Java3D based Volume Rendering

Java3D renders 3D volumes with an image scene graph approach (Fig.1) . Texture and raycast volume rendering have been implemented. 3D texture volume rendering uses view texture aligned slicing with tri-linear interpolation. Raycast volume rendering uses the classical ray cast sampling and compositing. High quality rendering results are shown in Fig.2.

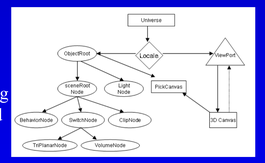

Fig.1 Java3D Image SceneGraph

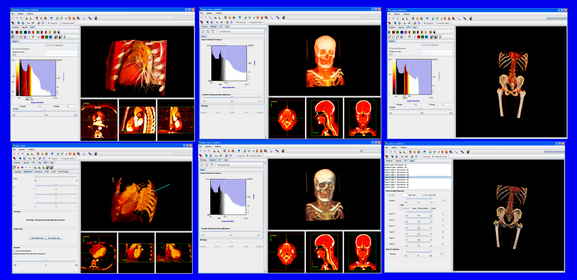

Fig.2 Java3D texture-based volume rendering

Jogl Multi-histogram based Volume Rendering

Using the Jogl (Java binding for OpenGL) library, we implement a multi-histogram volume rendering method [1]. The multi-histogram method is based on data values, gradient magnitude, and second directional derivatives that enable better discrimination between different material properties and boundaries.

Fig.3 illustrates the relationship between data value and it's derivatives. At the center of the boundary, the gradient magnitude (f') is high and the second derivatives (f'') is zero. Thus, boundary is enhanced from the materials. The Multi-histogram method allows interactive volumetric shading and shadowing to the traditional 3D texture based volume rendering pipeline. Fig.4 shows some multi-histogram volume rendering results.

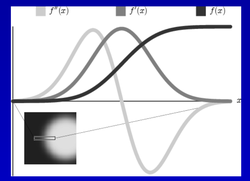

Fig.3 Multi-dimensional Transfer Function.

Fig.4 Multi-histogram Volume Rendering

Jogl GPU (GLSL) based Volume Rendering

We used the Stegmaier[2] method of single pass volume raycasting to implement Jogl GPU (GLSL,NV_fragemnt_program2) based volume rendering in Java. 3D texture data values are transferred to the graphics card, and single pass volume rendering on per pixel basis is performed. For each pixel of the final image a single ray is traced independently through the volume. On each fragment, sampling along the ray path and accumulating the intensity and opacity are performed.

Pseudo code of a fragment based volume raycaster is shown in Fig.5. Fragments are processed independently of each other. The Jogl GPU based method improves performance using the parallel processing power of the graphics card. Fig.6 shows the Jogl GPU based volume rendering results.

```
Compute volume entry position
Compute ray of sight direction
While in volume
    Lookup data value at ray position
    Accumulate color and opacity
    Advance along ray
```

Fig.5 Fragment Raycasting

Fig.6 Jogl (GLSL shader) based Volume Rendering

WildMagic GPU (Cg) based Volume Rendering

Volume rendering in MIPAV is currently implemented using the WildMagic Shader library. The WildMagic Shader (Jogl based) library dynamically loads and compiles shaders written in the Cg shading language. The WildMagic Shader library implements single pass or multipass rendering with one or more vertex and pixel shaders. Blending modes can be specified for multi-pass shaders which accumulate color results between passes. Users can change the shader input parameters (material properties, lighting, and viewing transform) on the fly while the application is running. Fig.7 shows the WildMagic GPU based volume rendering results.

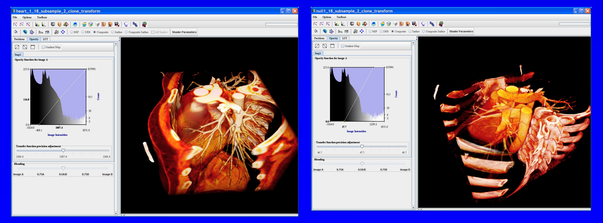

Fig.7 WildMagic GPU based Volume Rendering

PERFORMANCE MEASURE

All performance measurements were conducted on a PC with the NVIDIA GeForce 6800 GS based graphics card. For each rendering method, we capture the average frame rate and memory usage.

Frame Per Second (FPS)	Multi-histo	Java3D	Jogl(GLSL)	WildMagic
Heart(128^2x128)	8.92	30.123	21.21	7.02
Abdomen(256^2 x 256)	1.67	15.039	16.68	8.98
Stent(512^2x128)	1.57	N/A	12.81	N/A
Cardiac(512^2x256)	N/A	N/A	N/A	N/A

Memory (MB)	Multi-histo	Java3D	Jogl(GLSL)	WildMagic
Heart(128^2x128)	15	100	2	112
Abdomen(256^2 x 256)	170	740	14	870
Stent(512^2x128)	347	N/A	32	N/A
Cardiac(512^2x256)	N/A	N/A	N/A	N/A

DISCUSSION AND CONCLUSION

Pros:

- Java3D can render with high quality.
- Multi-histogram method has an intuitive user-interface for transfer function manipulation and has good rendering quality.
- Jogl GPU (GLSL) renderers the volume quickly with low memory usage.
- WildMagic has the best rendering quality and also allows the users greater freedom to control the shader parameters.

Cons:

- Java3D library has internal memory management issues.
- Multi-histogram method is CPU based, performance can be slow when exposing more detailed volume information.
- Jogl GPU (GLSL) shader assembly module is difficult to extend.
- WildMagic memory management requires improvement.

REFERENCES

- [1]. J. Kniss, G. Kindlmann, and C. Hansen, "Interactive Volume Rendering Using Multi-Dimensional Transfer Functions and Direct Manipulation Widgets," *Proc. IEEE Visualization Conf. '01*, pp. 255-562, 2001.
- [2]. S. Stegmaier, M. Strengert, T. Klein, and T. Ertl, "A Simple and Flexible Volume Rendering Framework for Graphics Hardware based Raycasting," Proceedings of Volume Graphics 2005, Stony Brook, New York, USA, pp.187-195, 2005
- [3]. L. Mroz and H. Hauser. RTVR - a flexible java library for interactive volume rendering. In IEEE Visualization 2001. http://citeseer.ist.psu.edu/mroz01rtvr.html