Java GPU Enhanced Multi-histogram Volume Rendering Framework for Efficient Bio-medical Visualization

Ruida Cheng[1], Alexandra Bokinsky[2], Justin Senseney[1], Matthew J. McAuliffe[1]
1. Center for Information Technology, National Institutes of Health, Bethesda, MD.
2. Geometric Tools, Inc. Chapel Hill, NC

INTRODUCTION

In this work, we present a novel implementation of a Java GPU based multi-histogram volume rendering framework that integrates into the multi-dimensional transfer functions [1]. We implement the framework within MIPAV [2], a Java based bio-medical image processing and visualization tool. The integration approach has been performed in two separate steps: Java OpenGL texture view-aligned volume rendering ported from the original C++ implementation [1], and the WildMagic [3] GPU ray-casting based volume rendering. We also demonstrate several applications integrated into the GPU based multi-histogram framework, including GPU based surface rendering, Nvidia 3D Vision stereo rendering. These flexible features make the interactive multi-histogram volume rendering an efficient framework within the MIPAV visualization.

METHODS

1. Java OpenGL based Multi-histogram Volume Rendering Framework

The implementation entirely relies on top of the Jogl [4] library. We ported the framework as a prototypical implementation from the original C++ implementation. Figure 1 depicts the rendering pipeline.

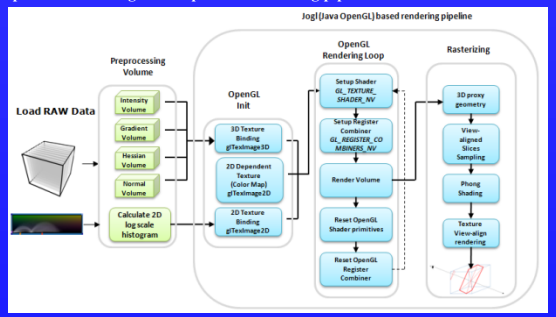


Fig. 1 Java OpenGL based Multi-histogram volume rendering pipeline

The framework provides the high quality rendering with OpenGL 3D texture based compositing. The whole framework is based on the OpenGL primitives. The multi-histogram widgets are drawn with OpenGL geometry shapes. The rendering pipeline uses the register combiners (GL_NV_register_combiners) and texture shader (NV_texture_shader) for rasterizing. The Java OpenGL based framework relies heavily on the OpenGL extension known as render to texture. Rasterization stage implements the 3D texture view aligned rendering.

2. WildMagic GPU (GLSL) based Multi-histogram Volume Rendering Framework

2.1 Paradigm Shift from Java OpenGL Texture based Rendering to GPU Raycasting based Rendering

The WildMagic GPU raycasting approach is multi-pass based rendering. The stream model for volumetric ray-casting that exploits the intrinsic parallelism and efficient communication on modern graphics chips. Our implementation builds upon the functionality that is provided on current programmable graphics hardware. In essence, the following passes are performed on the GPU using GLSL fragment shader:

Pass 1: Render the volume bounding cube, pass in the Depth-Boundary texture with the back face of the bounding volume cube.

Pass 2 : Each uniformly sampled (Figure 2) voxel based multi-pass rendering
1) At each sampling point, render the front face of the volume bounding cube.
2) Current position along the ray is determined by the rendering pass. The position is calculated from the start position (front face) and end position (end face) of the ray and the step size from the rendering pass.
3) Get the color from the 3D volume texture saved on the GPU.
4) Performing intensity lookup on multi-histogram. Go through each multi-histogram control widgets, compute color and opacity for the voxel.
5) Save the color and opacity to the specified fragment output.
6) Step to the next sampling voxel, repeat from step 2. Trace ray from back to front.
7) Swap the rendering OpenGL FrameBuffer (target buffer for each entity, such as volume, translucent and opaque surface, etc.).

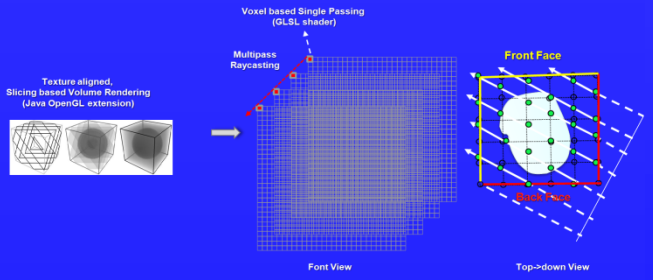


Fig. 2 Paradigm Shift from Java OpenGL texture based rendering to GPU (GLSL) Raycasting

2.2 GPU based Multi-histogram Volume Rendering Pipeline

Figure 3 depicts the rendering pipeline. During initialization, WildMagic library creates the scene graph tree for the different renderers. The OpenGL FrameBuffer Object (FBO) is created for rendering multiple translucent surfaces. The proxy geometry bounding-box is created from the scalar volume, and saved as a scene node. The scalar volume data, gradient map, multi-histogram map information are stored and passed to the shaders as texture images; ShaderEffect contains the shader programs, and shader parameters for handling those texture images. Once the ShaderEffect is created, the corresponding GLSL programs are loaded from disk onto the GPU. The WildMagic setup the rendering context by render the volume bounding box backface first, then it calls the sampling point based GPU raycasting routine as described in Section 2.1.

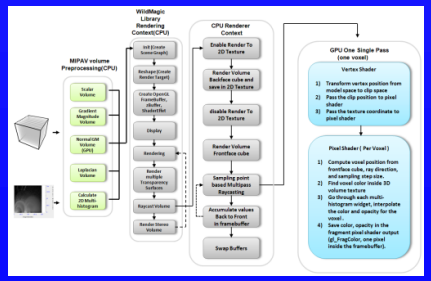
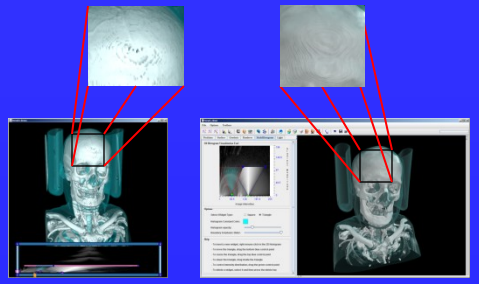


Fig.3 WildMagic GPU (GLSL) based Multi-histogram Volume Rendering Pipeline

Figure 4. shows the comparable results from Java OpenGL extension and GPU raycasting based multi-histogram rendering.



Java OpenGL Extension Texture Aligned — WildMagic GPU Raycasting

Fig.4 Multi-histogram Volume Rendering Results from Java OpenGL texture based and GPU (GLSL) Raycasting based Rendering

3. Nvidia 3D Vision Stereo View for Multi-histogram Volume Rendering

3D stereoscopic pipeline modifies the existing GPU raycast pipeline by splitting it into left and right eye view frustums. The stereo viewer uses two different cameras to build and render distinct scenes to each eye. The rendering pipeline routine is encapsulated into the GPU raycasting render function with OpenGL quad buffer enabled.

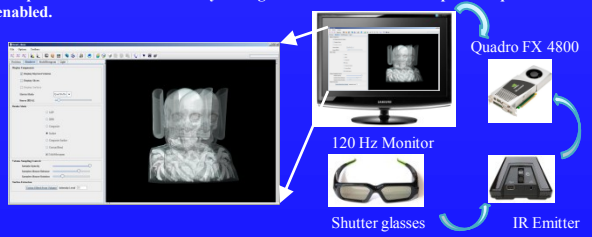


Fig.5 Nvidia 3D Vision multi-histogram volume rendering

CONCLUSIONS & FUTURE WORK

The most important implication of this work is the possibility to exploit the full potential of high quality rendering within MIPAV visualization, which can be useful for clinical studies and surgical planning. Based on the original multi-histogram model, we build up an entirely new pipeline to implement raycasting operation completely on GPU. When preprocessing the raw data volume, GM, normal, Laplacian and multi-histogram computations are performed on CPU, which still costs time. The future work will address those problems by OpenCL operations on both CPU and GPU.

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] MIPAV (http://mipav.cit.nih.gov). From McAuliffe MJ, Lalonde FM, McGarry D, Gandler W, Csaky K, Trus BL. Medical Image Processing, Analysis & Visualization In Clinical Research. IEEE COMPUTER-BASED MEDICAL SYSTEMS (CBMS) 2001, 381-386
[3] WildMagic library, http://www.geometrictools.com
[4] Jogl library, http://jogl.dev.java.net