A JAVA-BASED MULTIPLATFORM HIGH PERFORMANCE 2D AND 3D VISUALIZATION ENGINE FOR MEDICAL IMAGE DATA SETS

T. Arola*, M. Hannula*, J. Hyttinen*, P. Dastidar**, S. Soimakallio**, J. Malmivuo*

* Ragnar Granit Institute, Tampere University of Technology, Tampere, Finland
** Department of Diagnostic Radiology, Tampere University Hospital, Tampere, Finland

tuukka.arola@tut.fi

Abstract: We have developed a high performance 2- and 3-dimensional visualization engine for medical image data sets. The software is written entirely in Java and is fully platform independent. Hardware accelerated 3-dimensional modes are implemented via Java3D that allows to use OpenGL or Direct3D as the underlying hardware accelerated application programming interface (API). The visualization engine is part of a modeling framework. As an example of the capabilities of the engine two different data sets have been visualized.

Introduction

Three dimensional modeling and visualization is becoming even more important as the computerized radiology equipment is developed further. Now it is possible to obtain 3D data set from computed tomography (CT) and magnetic resonance imaging (MR) devices in reasonable times. Most of time researchers are interested to see the analysis results being displayed on the computer screen. As user interaction such as segmentation is increasing, visualization becomes a critical feature in medical imaging exploration and analysis.

Thus there is a general interest and demand on high performance and platform independent visualization. In addition there has been a growing demand for a high performance 3D visualization engine for modeling purposes in our institute. A decent visualization engine could benefit researchers but also students in understanding bioelectromagnetic phenomena and human anatomy. Previously toolkits for visualization have been developed [2]. Also other toolkits such as VTK [3] have been tested. A major drawback using these toolkits has been the need for compiled, native binaries, which restrict their applicability on different kinds of platforms.

Java is becoming a major language in software business. A major advantage of Java compared to e.g. C++ is its well designed object oriented programming model as well as the fact that compiled Java-binaries can be run in various environments. *Desktop Java* has developed huge steps under a couple of last years and is becoming an even more suitable language for various other tasks, such as visualization and user interface construction.

Gaming industry is a major driving force in computer technology and thus 3D features have been implemented also in Java. In order to keep to platform independency, an intermediate 3D layer was implemented in Java to provide hardware accelerated 3D graphics. This layer acts as a mediator between Java objects and platform specific implementation layer. Currently Java 3D (version 1.3.2) supports OpenGL on all platforms and also Direct3D on Microsoft Windows operating systems [1].

Development of a software framework for simulation and visualization of Finite Difference (FD) models was started in 2003. Since then, the software has evolved and now is a full featured framework for modeling including various modules. It includes a full 2D and 3D visualization engine among other features such as segmentation, model creation and simulation. While still under development, the results of the graphics engine have been promising and our engine has demonstrated its usability in modeling and segmentation as well as in volumetric analysis. Additionally the framework is tailored to be used in education in our institute as a part of a web based learning environment as an applet.

Material and Methods

Our primary goal has been a powerful imaging tool especially for medical image data sets and for modeling purposes. It should be usable for both researchers and students alike. Thus, the engine as well as the modeling framework was needed to be usable in various environments. Java was chosen for implementation because of group's expertise in commercial Java software, its true cross-platform capability and multiplatform, hardware accelerated Java3D API. The newest versions, Java 1.5.0 and Java3D 1.3.2, have been selected to gain best performance possible.

The emphasis in the design of the graphics engine has been on extensibility. As seen in Figure 1, the 3D engine has interfaces for model extraction, triangulation, mesh reduction and mesh smoothing. These implementations can be changed and new can be developed. Currently there is only one implementation for each.

The graphics engine has both 2D and 3D modes. The engine works in full 32 bit color mode to provide "unlimited" mixture of colors. The engine supports multimodal imaging and thus the color map for each modality or segmentation layer can be selected from pre-defined color maps or a new color map can be created.

Our geographical proximity with Tampere University Hospital (Tampere, Finland) provides us an excellent opportunity to test and develop the software utilizing clinical data from CT, MR and US devices. Thus, to demonstrate the capabilities of our engine we have chosen two different data sets.

- A CT image stack of a patient with abdominal aortic aneurysm (AAA) with 48 slices in 512x512 resolution that was segmented [4].
- A MR stack of a female head with full segmentation with 87 slices in 256x256 resolution.

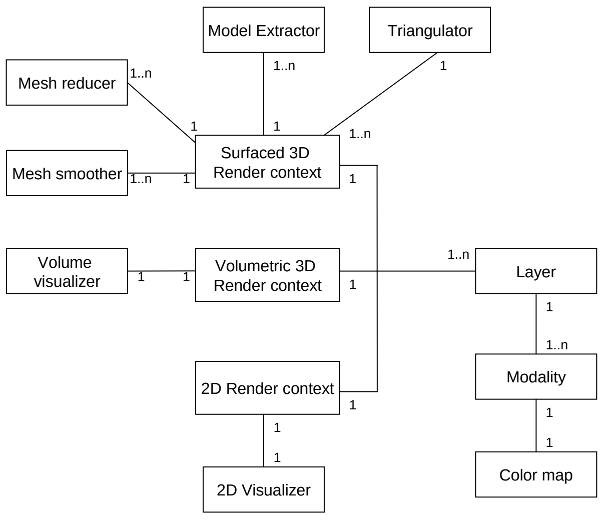

Figure 1: A class diagram of visualization contexts (modes) and their essential interfaces of the graphics engine.

2D mode provides a flexible way of looking data in multiple modalities. The framework has tools for segmentation which require powerful 2D features from the graphics engine such as layering with adjustable transparencies.

2D mode can show cross-cut slices of any data set in axial directions. Imaging and model modalities are layered one on the other. User can adjust transparency and visibility of each layer separately at any time and change the color palette of each layer independently. Generally layering does not distinguish between different types of data; they can render data of any type: pure pixel data or other graphics data such as vector fields.

Performance of the 2D renderer is easily affected by using multiple modalities of data. Implementation of graphics routines and layering has been optimized to cache and flatten the modality and layer structures. For example tree structures of different segmentation layers are flattened before render phase. That ensures only one rendering pass for all segmentation layers on average. Flattening is done again only if the data in layer tree has changed. This ensures fluent operation even when layering multiple segmentations on top of original data.

The engine has two separate modes for 3D visualization. Modes are:

- Hardware accelerated surface rendering mode based on surface triangulation of known tissues
- Volumetric rendering implemented as Ray Casting

Surfacing mode is supported via Java3D. Java3D translates the user (surface) model to OpenGL or Direct3D models and forwards the control to the hardware API in question. In the 3D view the engine is capable of point, triangle and surfaced rendering with gouraud shading and transparency settings in real time. The model can be rotated, zoomed and different tissues can be toggled on and off. Transparency of different tissue types can be adjusted individually.

Triangulation is currently implemented with Marching Cubes (MC) algorithm [5]. The algorithm has been used previously on medical data visualization. MC is a contouring algorithm and is produced by rolling a 2x2x2 window over the 3D data. Surface elements are created according to the elements in the window. Strengths of MC are speed and low memory consumption and the fact that it is really straightforward to implement. However, MC has a tendency of creating excessive amounts of triangles.

Although 3D graphics cards can nowadays render millions of triangles in decent frame rates, reduction and simplification of the 3D mesh is needed. Currently, the engine does offer a simple mesh reduction but a more versatile version will be added in the near future. Also mesh smoothing can be applied to make the generated surfaces appear smoother. Currently we have implemented a simple averaging smoother.

The second 3D mode is volumetric visualization implemented as Ray Casting (RC). RC has been used for a long time in computer graphics. It does not need triangle surfaces constructed but – as the name says – casts rays from user viewport to the volume. Each ray is traced and the hit point in the scene determines the color value and depth of pixel. The method is slow and various ways to optimize it has been introduced [6]. We have used Oct Tree space partition algorithm to find empty areas in the volume. With the help of space partitioning the RC renderer can rapidly decide if a subvolume can be neglected. Also, an adaptive termination method is used to determine the termination point for the ray. Both of these methods are powerful and easily implemented and offer noticeable performance advantages.

Generally, Ray Casting algorithms perform poorly compared to surfacing methods but provide also a more realistic render. Additionally, they can be further extended to include reflections, refractions and dispersion of the rays to produce photorealistic images.

Results

The software produced is capable of high performance visualization both in 2D and in hardware accelerated

3D. The engine itself is not confined to medical image data and can also be used for other visualization purposes.

Figure 2 shows an axial 2D render of the human head with segmentation of gray matter.

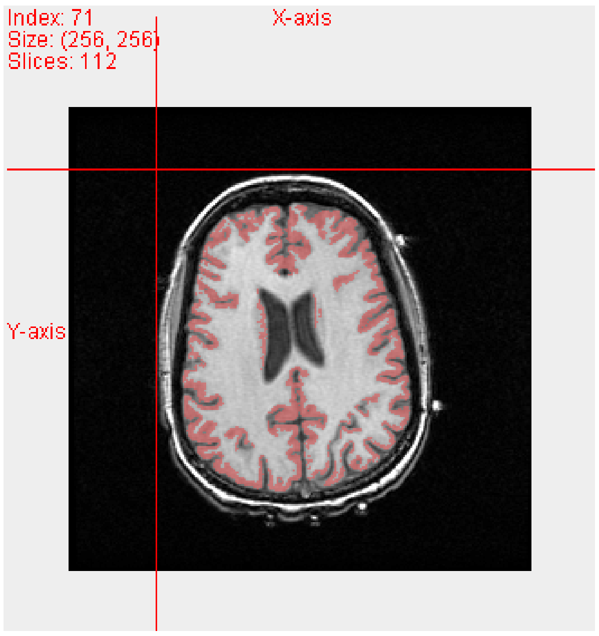

Figure 2: 2D mode view of human brain with segmented gray matter. Transparency (60%) has been applied to the segmentation. Also other modalities can be similarly layered on top each others. The lines show the position of the other cutting planes.

As for the surface 3D mode, the pure performance of the 3D engine is good if compared qualitatively to commercial software such as MATLAB, SEMCAD and FEMLAB. Over 2 million gouraud shaded triangles can be visualized with ease on a standard office desktop computer with 1GB of memory and average 3D graphics card (ATI 9200 series) with frame rates around 5-10. Table 1 below describes some surfacing times for model shown in Figure 3. Generally, the performance of the surfaced 3D mode is determined by the graphics accelerator. Variables having effect on performance are model triangle count, shading and special effects such as transparency.

Table 1: An example of triangle counts, surfacing times, memory consumption and surfaced model frame rates for various tissues in segmented head model with surface smoothing but no mesh simplification (Intel Pentium 4, 2.8GHz, 1GB RAM, ATI 9200 series graphics card, 1024x768 resolution)

	Skull	**White matter**	**Eyes**
Triangle count	493504	569640	8412
Surfacing time (sec)	7.2	9.4	3.2
Memory consumption (MB)	68	97	22
Surfaced model frame rate	13	10	38

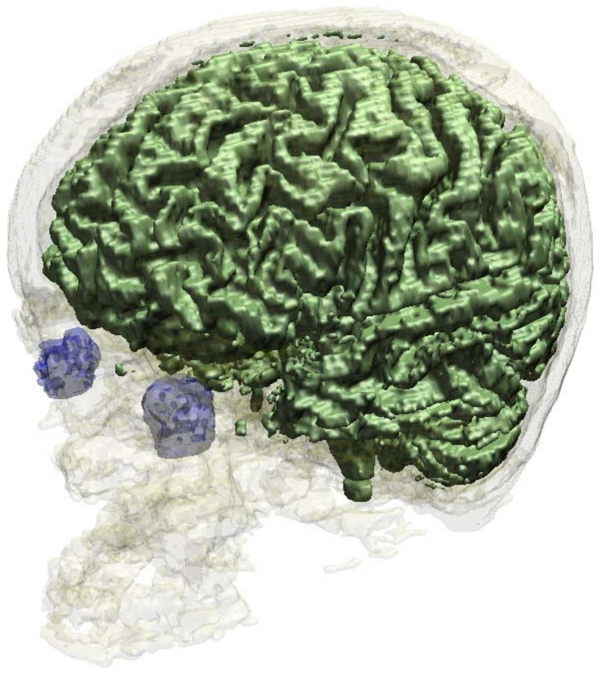

Figure 3: A segmented human brain, skull and the eyes in 3D visualization mode. Table 1 describes the triangle counts, surfacing times, memory consumption and surfaced model frame rates for this model.

In our abdominal aortic aneurysm example, the segmentation produced with the framework was directly forwarded to the 3D engine that creates a point-based non-shaded view of data with gouraud shaded aneurysm region as seen in Figure 4. The stent located inside the aneurysm can also be seen in surfaced mode in Figure 5.

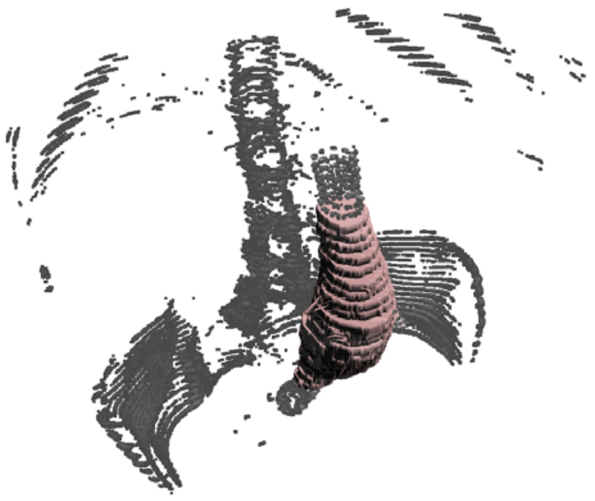

Figure 4: CT image data with a segmented abdominal aortic aneurysm in surfaced 3D visualization mode. Imaging device data and segmentations produced in the framework can be visualized easily.

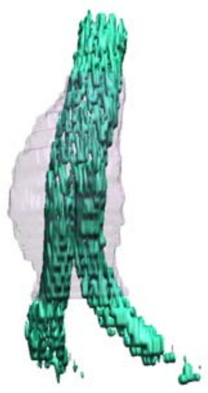

Figure 5: Abdominal aortic aneurysm and the stent implanted in surfaced 3D mode. Visualization is built from CT data set. The structure of the stent is clearly visible.

Discussion

Results provided show that our engine performs well and is suitable for a range of medical image sets. Java has long been though to be too slow for scientific purposes, but newer Java versions 1.4 and 5.0 have been major steps forward in desktop features and especially performance. There are already dozens of commercial software using Java and Java3D including games and scientific software.

A qualitative performance comparison is missing, but as can be observed from Table 1, the performance of the 3D renderer is good. Memory consumption can be optimized further but currently 1GB of memory is enough for complete surfacing a complex 256x256x128 data set.

We have used Marching Cubes algorithm for surface triangulation. There are also other methods for creating surfaces which are not currently supported by our engine. Such methods include Delaunay-based and contour methods. While being harder to implement, these methods could provide smoother surfaces and inherently lower triangle counts. However, it must not be forgotten that surface triangulating volumes produces an approximation of the surface and due to the nature of voxel data, the resulting visualization cannot be totally smooth without softening.

Java has support for hardware accelerated 2D graphics, but currently it is not supported by the engine. However, performance of the 2D renderer is adequate and it can render up to 15 frames per second in 1024x768 resolution with three layers of layered data. One could argue whether hardware acceleration has any impact on performance in this case. After all, rendering itself is done in system memory and there is no need for a double buffer in accelerated memory due to full buffer updates to the screen.

The engine is by no means complete. There are various aspects planned to be improved in the future. Overall performance of the 3D visualization modes need further optimization, especially volumetric model renderer needs attention. Also better reduction of the generated triangle mesh is needed to surface even more complex models. More smoothing operators will be added to provide nicer, smoother surfaces. This can also be achieved by implementing a contour-based surfacing which is under development.

Due to the nature of Java and specific implementation details, the engine can also be used in an applet in web browser through a network. 2D mode has full features in applet mode, but 3D mode could have some limitations. This is due to the memory restrictions web browsers have for applets. The memory consumption of the 3D engine is linear to the model size. The applet mode makes it possible for students to familiarize themselves with modeling concepts and human anatomy without having to download and install the software framework and models needed.

Conclusions

A high performance, multiplatform, Java-based visualization engine for medical image data sets and segmented models has been introduced. We have provided examples to demonstrate the features of our engine. The engine has been utilized in research and teaching purposes in our institute. Our engine currently provides a basic package of utilities for full 2D and 3D medical data visualization and is part of a framework for image segmentation and bioelectric field simulations.

References

[1] Java3D Home page, Internet site address: https://java3d.dev.java.net/

[2] HEINONEN T., DASTIDAR P., FREY H., ESKOLA H. ‘Applications of MR image segmentation’, *Int. J. Bioelectromag, Num 1. Vol 1.* 1999. Internet: http://www.ijbem.org/volume1/number1/html/toc.htm

[3] The Visualization Toolkit home page, Internet adderss: http://public.kitware.com/VTK/

[4] HANNULA M., AROLA T., HYTTINEN J., NARRA N., DASTIDAR P., PIMENOFF G., SOIMAKALLIO S., MALMIVUO J.: ‘Use of segmentation and volumetric estimation in the treatment of abdominal aortic aneurysms using stents ’, Proc. of 3rd European Medical & Biomedical Engineering Conf., Praque, Czech Republ., 2005, *submitted*

[5] LORENSEN W., CLINE H: ‘Marching cubes: A high resolution 3D surface construction algorithm’, Proc. of the 14th annual conf. on Computer graphics and interactive techniques, 1987. p.163-169.

[6] LEVOY M., CLINE H: ‘Efficient ray tracing of volume data’, ACM Transactions on Graphics, 1990