3-D Volume Modeling, Feature Identification and Rendering of a Human Skull

Joerg Meyer
University of California, Irvine
Department of Electrical Engineering and Computer Science
644E Engineering Tower, Irvine, CA 92697-2625
jmeyer@uci.edu

Abstract

Advanced medical imaging technologies have enabled biologists and biomedical researchers to create accurate models and identify features in complex, large-scale data sets. These models, which are typically based on a CT or MRI scan, occupy large amounts of storage space and can no longer be archived on local hard drives. They are also difficult to transmit over currently existing networks. Large-scale biomedical image data, which is typically stored in a large data repository, must be preprocessed in order to enable real-time data transmission and interactive rendering. To make the model accessible to researchers at remote locations over the Internet within a reasonable amount of time, we are describing a web-based volume modeling and feature identification system that incorporates a multi-resolution rendering technique for transforming large-scale volume models into hierarchical representations. We are using Haar wavelets to decompose the data set into a multi-level-of-detail representation that can be transmitted from the server side to the client side in a progressive fashion. The image is rendered using a texture-based visualization technique in Java3D. A new efficient illumination technique has been implemented to improve the image quality of the rendered volumes and to help with the identification of features by using pre-calculated normal vectors for all the surface voxels. The advantage of this method is that the illumination can be calculated on the client side. It does neither affect the data transmission time nor the interactive behavior of the texture-based rendering algorithm.

Keywords: Volume modeling, feature identification, texture mapping, Haar wavelets, rendering, illumination

Introduction

The purpose of this work is to demonstrate that 3-D texture mapping in combination with wavelet compression is a viable and efficient alternative to conventional ray-based or projection-based volume or surface rendering methods. This is the key step in the volume modeling, feature identification, and rendering pipeline.

The volumetric model is typically obtained as a series of two-dimensional cross-sections from a medical scanner. Enhanced imaging techniques like Computed Tomography (CT), Magnetic Resonance Imaging (MRI), Positron Emission Tomography (PET), Confocal laser-scanning microscopy, etc., allow them to distinguish pathological from healthy tissues or to study microscopic cell structures in greater detail.

The feature identification and rendering are performed on the client side. This method yields maximum performance, responsiveness, and interactive behavior of the client rendering application.

Background

Uncompressed volume models require massive storage capacity and transmission bandwidth. Though there have been advances in mass-storage density, processor speeds, and digital communication system performance, demand on data storage capacity and data-transmission bandwidth continues to pose a challenge to existing applications.

Several file compression schemes have been introduced. JPEG, for instance, is a lossy technique since quantization introduces artifacts due to reduction of detail information. Despite several advantages of JPEG like simplicity, satisfactory compression and decompression performance and availability of special purpose hardware implementations, there are several drawbacks, for instance, loss of color information due to the chosen color model (YIQ), and block artifacts at low bit rates [1].

Wavelets have been proven to eliminate these artifacts as they typically do not use color model transformations, and because their basis functions have local support of variable length. Moreover, wavelet coding is more robust under transmission and decoding errors, because if an error occurs on one level, it will be smoothed out on the next level. Wavelets also facilitate progressive transmission of images (recursive image decomposition). They have an inherent multiresolution nature, which makes them suitable for applications where scalability is required and tolerable degradation can be accepted. A discrete wavelet transform (DWT) can be interpreted as an improved variant or extension of a discrete Fourier transform (DFT). While DFT can provide only frequency information, a DWT can keep track of spatial and frequency information, and therefore preserves location [2]. The Haar wavelet is one of the simplest wavelet transforms and can be computed efficiently using Integer arithmetic. Also, since we are dealing with pixel data, which resemble more a rectangular signal than a continuous analog signal, Haar wavelets are best suited to

represent the original signal, because they use box functions as base functions [3].

Volume rendering has gained lot of importance in recent times due to improvements in rendering hardware and due to its widespread use in various applications. Ray-based algorithms like ray tracing or ray casting produce high quality images but are very time-consuming [4]. Projection-based algorithms like Splatting [5] and Shear-Warp factorization [6] have been proposed as alternatives.

Texture-based volume rendering can be used as an alternative, as it has been recognized as a very efficient technique, especially after the first SGI Reality Engine™ featuring 3-D texture mapping hardware became available [7]. Hardware accelerated 3-D texture-based volume rendering algorithms let users achieve interactive frame rates and high quality images [8]. Most of the previous texture-based volume rendering algorithms have been primarily based on OpenGL or Open Inventor [9, 10]. The need for web applications in recent years has encouraged researchers to explore web-based rendering techniques using Java and Java3D [11]. Earlier works include methods of extracting slices on the client side [12] or remotely on the server side [13].

Web-based Feature Identification and Volume Rendering System

Our web-based feature identification and volume rendering system implements a client-server model (figure 1) [15, 16]. The 2-D cross-sections are assembled to form a 3-D volume (or 3-D array). This volume is then transformed using a 3-D Haar wavelet transformation and compressed into a more compact representation. The compressed representation of the volume is then transmitted to the client who had requested this data set or a particular sub-volume for rendering. The client-side rendering algorithm renders it as a 3-D volume using 3-D texture mapping in Java3D. The initial data transmission usually consists of a coarse representation of the entire original volume, which serves as an initial preview of the data set for the user. The resolution of the rendered volume is increased later by adding detail coefficients to the initial coefficients that represent the coarse volume. The data reconstruction uses an inverse Haar wavelet transformation. The reconstruction of the original volume becomes lossless if all the detail coefficients are transmitted and taken into account and if no quantization is applied to the high-pass and low-pass filtered coefficients. Though the whole data set can be reconstructed on the client-side and rendered in full resolution, in many cases it is sufficient to render only a particular region-of-interest (ROI) selected by the user in full detail. In this case, a low-level representation of the ROI is transmitted first, along with a coarse overview representation of the rest of the data set (Figure 2). Subsequently, the detail coefficients of the ROI are transmitted to render the ROI in full resolution.

The feature identification method is independent from the resolution of the volume model. This algorithm is therefore applied after the wavelet reconstruction. In a volumetric model, features, such as tumors in brain tissue, contrast-enhanced blood vessels in human organs, etc., are typically characterized by soft material boundaries that cannot be captured by traditional, threshold- or iso-value-based segmentation methods or by traditional iso-surface extraction methods such as Marching Cubes [21]. Since the transition between two materials is often not clear, a segmentation and feature identification method based on a single iso-value or threshold value can potentially lead to falsely identified voxel materials.

In order to capture such gradients, i.e. soft transitions between different materials in the volume model, multi-dimensional transfer functions are employed, where the first dimension represents the original voxel data, and the second dimension represents the gradient of the material. This method has proven to provide results superior to one-dimensional transfer functions, and it can be easily adapted to multi-resolution hierarchies [22].

It should be mentioned here that the gradients computed in this step can also be used to illuminate the volume model.

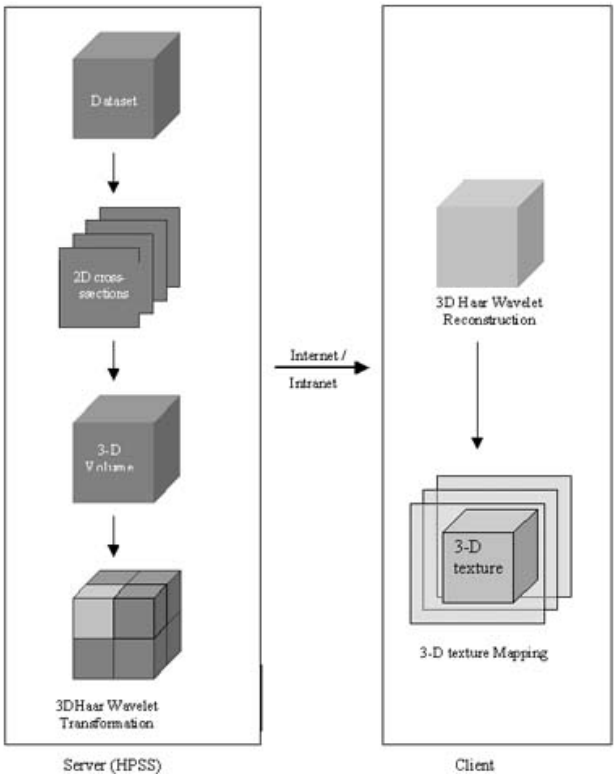


Figure 1. System Architecture

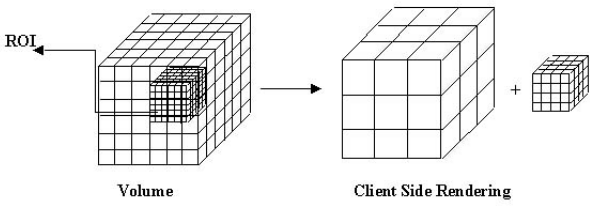
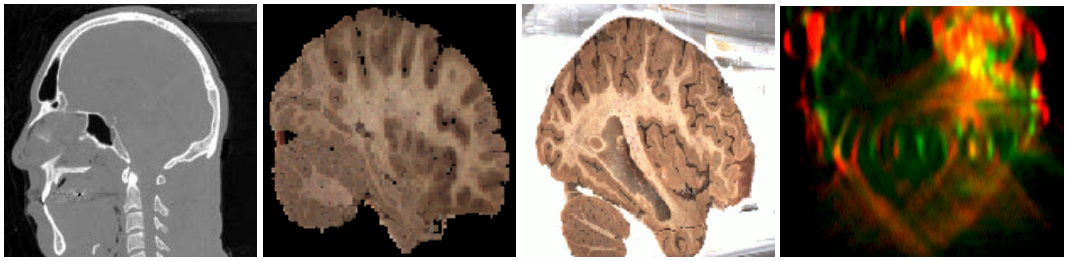


Figure 2. Sub-volume representation

We tested our algorithm by extracting cross-sections from a CT scan of a human head (512 x 512 x 231), a human brain (94 x 113 x 131), a cancer cell (251 x 70 x 312) and a series of cryosections of a human brain (233 x 144 x 184).

Sample 2-D cross-sections extracted from these data sets are shown in figure 3.



CT head Human brain Cryosection of a human brain Cancer cell

Figure 3. Sample 2-D cross-sections

Volume Rendering using Textures

2-D texture mapping can be used to change the appearance of an object by mapping a 2-D image onto the surface of the object while requiring only a little increase in the rendering time [18]. Multi-dimensional texture mapping maps a multi-dimensional image to a multi-dimensional space usually by exploiting hardware-accelerated rendering capabilities. Texture hardware supports texture mapping in 1-D, 2-D and 3-D. A 3-D image can be defined over an (*s, t, r*) parameter space in the range of 0<=s<=1, 0<=t<=1, 0<=r<=1, which can be mapped to a geometric object.

The technique of 3-D texture-based volume rendering involves loading a 3-D texture into the texture buffer and mapping a stack of parallel planes in back-to-front order with suitable textures, perpendicular to the viewing direction. 3-D texture coordinates are then tri-linearly interpolated at the polygon vertices using the 3-D texture mapping hardware in order to map the values from texture space to the object space in which the polygonal surfaces are defined. Alpha blending and compositing is done for each textured plane with the contents of the frame buffer to produce a 3-D representation of the volume. The Java3D API is used for the implementation of 3-D texture-based rendering. Using the Java3D API, the required Java3D objects are created and inserted in a scene graph for rendering [19]. The scene graph of Java3D is an arrangement of objects in a tree like structure that comprises of all information that is needed for rendering (light sources, camera position, etc.) The scene graph that is used in developing our texture-based volume rendering application is shown in the figure 4.

Results obtained by rendering different data sets in 3-D are presented in figure 5.

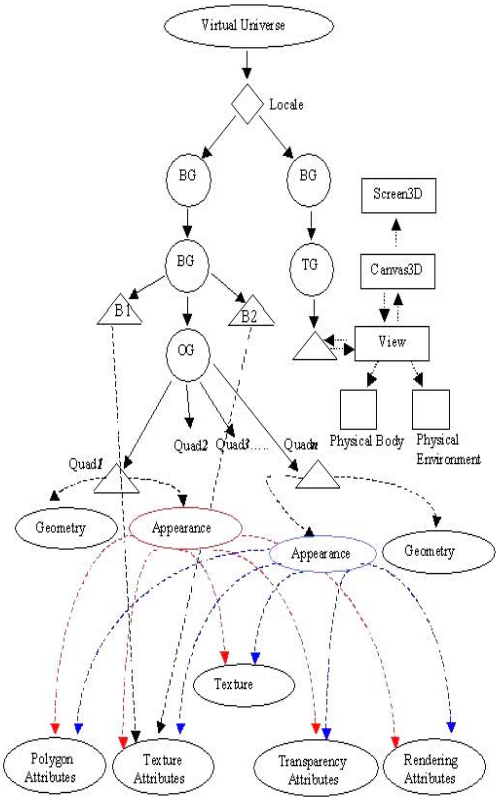


Figure 4. Scene graph

The above-defined texture mapping technique is used in progressively reconstructing the low-resolution volumes that

are transmitted to the client side. The detail coefficients are added to the already rendered low-resolution image to refine the original image.

Figure 6 shows a progressively reconstructed data set from a low-resolution representation to its original resolution. At each level of reconstruction the detail coefficients are added progressively to the corresponding low-pass filter coefficients. A user interface, VOL<X>REND, has been developed for rendering these reconstructed data sets in 3-D (figure 7). The rendered 3-D volume is displayed in the right panel. VOL<X>REND uses the Java3D API. The number of slices to be mapped onto the planes should be a power of two, since the 3-D texture restricts its dimensions to powers of two. The upper limit of slices and planes a user can select is displayed in a panel of the user interface. A larger number of slices gives a better quality of the rendered image and less number of slices gives more speedup.

The user can also select different transparency transfer functions (*Linear Mapping, Exponential Mapping, Threshold Mapping*, etc.) for rendering and viewing the inner structures of the data set. When the user selects a different transfer function or a different number of slices and planes, a new 3-D texture and the newly specified number of planes are created and the alpha values for the 3-D texture are recomputed.

The user can interact with the 3-D volume with both the keyboard and the mouse. The transformations (rotation, translation and scaling) are applied to the 3-D texture instead of the geometry, keeping the planes always perpendicular to the viewing direction. This method eliminates the redundancy of creating a new 3-D texture for every new coordinate transformation of the 3-D volume.

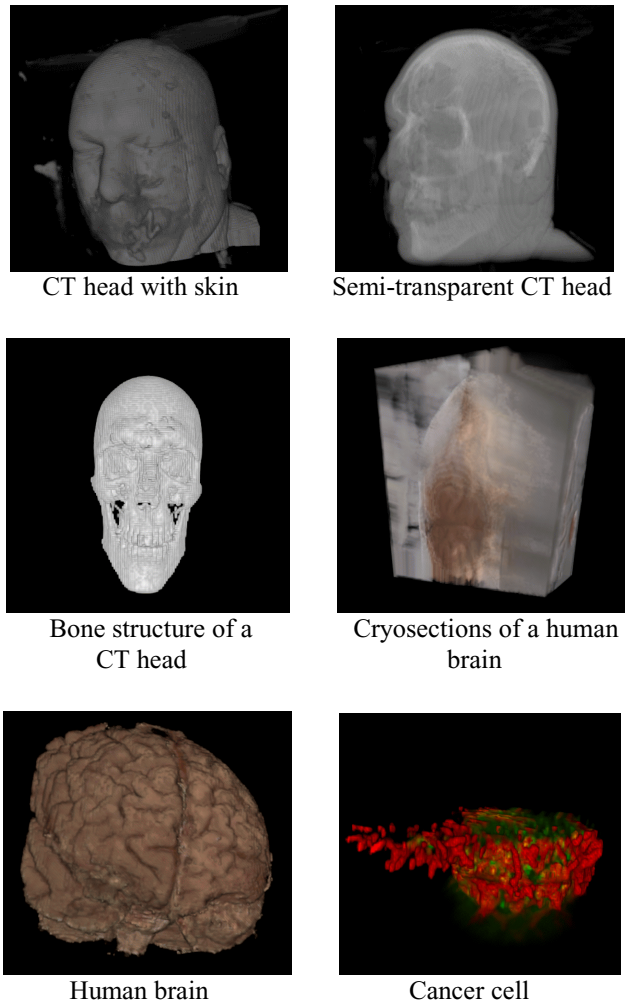


Figure 5. Data sets rendered using 3-D texture mapping

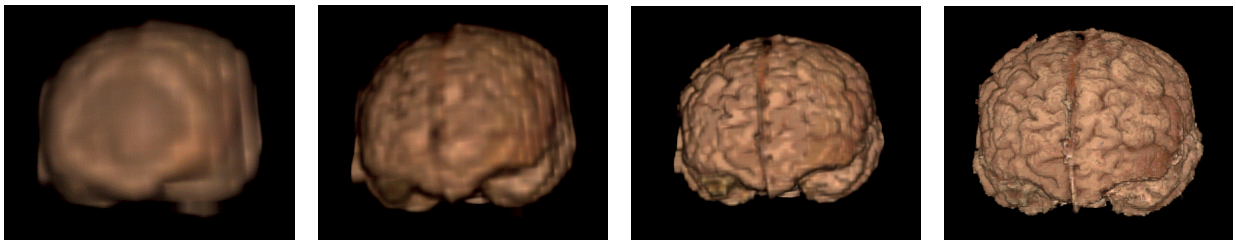


Figure 6. Progressive reconstruction

Illumination

Illumination adds realism to a scene, and it provides a more realistic visual appearance of 3-D models. It also provides an additional depth cue that makes it easier to recognize and distinguish anatomical features. An illumination model determines the color of a surface point by simulating light attributes (intensity, color, position, direction) and material attributes (color, reflectivity, transparency) [20]. A local illumination model considers three types of light: ambient, diffuse, specular (figure 8).

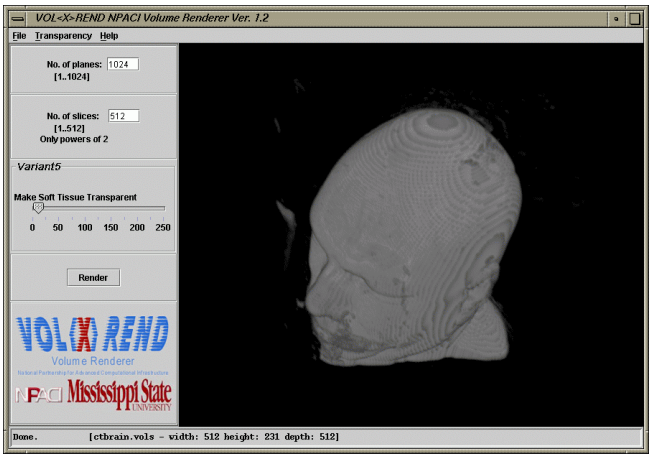


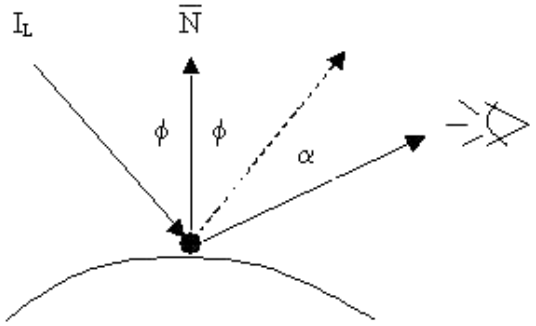
Figure 7. User interface of VOL<X>REND

In order to illuminate a 3-D object, a light source (I_L) must be defined, and normals must be calculated for all surface voxels of the 3-D model. A normal at a point on the surface is defined as a vector that is perpendicular to the surface of the 3-D model. In our texture-based volume rendering algorithm, which uses 3-D texture-mapped 2-D planes as geometric objects, there is no surface information available to calculate normals for surface voxels. Therefore, we estimate normal vectors based on gradients (tri-linear interpolation) for the surface voxels.

The first step in this process is to identify the surface voxels of the 3-D volume. A voxel is a surface voxel if at least one of its neighbor voxels is outside the 3-D object. A voxel is called an outside voxel if it is either completely transparent or close to the background color. Since a data set can be rendered to display different anatomical structures or materials by selecting different transparency mappings, the material boundary keeps changing. For example, when rendering the bone in a CT scan, all the voxels that represent soft tissue are made totally transparent. Therefore, along with testing a voxel's color, its alpha value is also tested for determining whether it is an outside voxel. When a surface voxel is identified, a normal is calculated as the normalized sum of all the gradient-weighted unit vectors of its outside neighbors.

Figure 9 shows a 2-D representation of a normal vector of a surface voxel. A 2-D surface point has 8 neighbors, and a 3-D surface voxel has 26 neighbors. For each voxel in the 3-D texture, all its 26 neighbors must be tested to identify whether it is a surface voxel. A positional light source is chosen and kept fixed at a particular position in the scene.

According to Lambert's law, a dot product must be calculated between the light vector and the normal of a surface voxel to determine the percentage of light a surface voxel reflects. For diffuse surfaces, the reflected light is determined by the cosine between the surface normal N, and the light vector I_L, (i.e., $i_{diff} = N \cdot I_L = \cos\phi$) [20]. The resulting value of the dot product is multiplied with the R, G, and B components of the color of the respective surface voxel. This way we modulate the intensity and calculate the new color of the voxel.



I = ambient (I_a) * k_a + diffuse (I_L, ϕ) * k_d + specular (I_L, α) * k_s

Figure 8. Illumination model

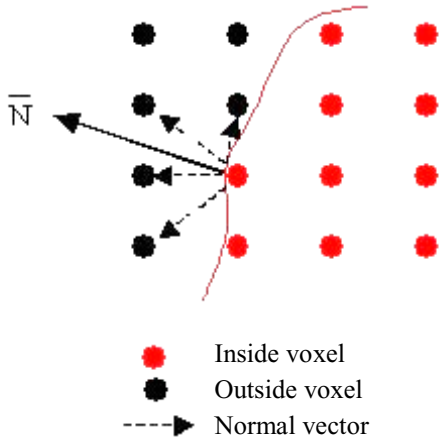


Figure 9. Calculating a normal for a 2-D surface point

In summary, all surface voxels are identified for a 3-D texture, their normals are calculated, and the resulting newly colored pixels replace the existing 3-D texture. This 3-D texture is then used in a 3-D texture-based rendering algorithm as described in the previous section.

Since the light position is fixed to the scene, it is rotated along with the texture, when a camera motion is simulated by rotating the texture. If the light position keeps changing with respect to the texture, recalculation of the 3-D texture is necessary for every new orientation, which decreases the performance of the algorithm. The advantage of this algorithm is that for a fixed light source it is possible to rotate, scale, and translate the object within the texture buffer without any performance loss.

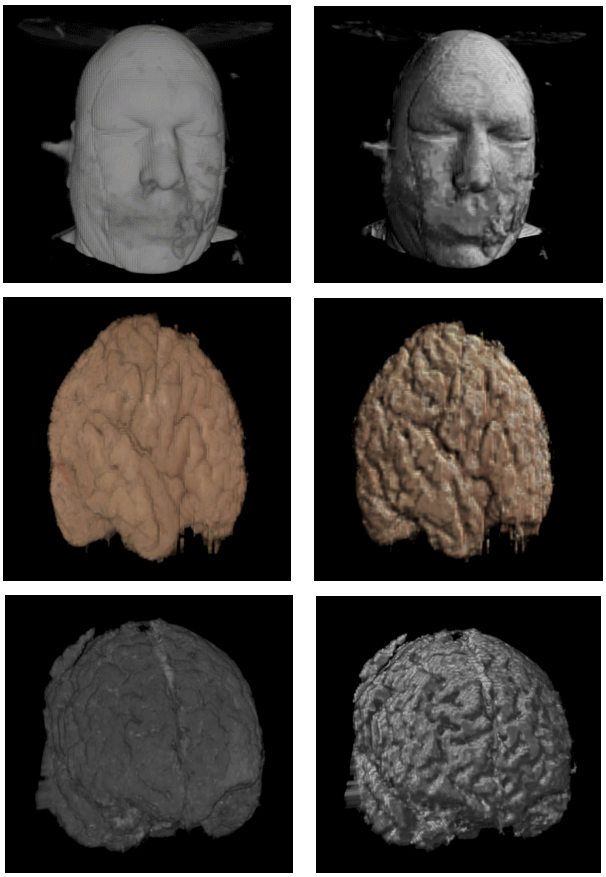


Figure 10. 3-D Rendered volumes (CT, cryosections and MRI) rendered with and without illumination

Figure 10 shows some of the results that were obtained by incorporating a light model in the 3-D texture-based Java3D renderer. The results show a significant improvement in image quality of the 3-D rendered data sets.

Statistics

Timings were taken for sample volume models rendered with VOL<X>REND, and they were taken individually for loading, processing, and rendering. The loading step includes the creation of the data structures and the actual file loading, the processing step includes creating a 3-D texture, inserting alpha values into the 3-D texture, creating planes and finally creating a scene graph, and the rendering step includes the time taken by the Java3D renderer to render the scene graph. The timing values are shown in Tables 1 and 2.

These timings were taken for rendering the volume models in 3-D with and without illumination both on an SGI™ workstation (Machine A), and on a Sun workstation (Machine B). Machine A is a SGI™ which four 400Mhz IP27 R12000 processors and 4096 MB of main memory. It has an InfinityReality3™ graphics engine that supports a texture of size 2048 x 2048 x 1. Machine B is a sun4u 8-slot Sun™ Enterprise 4000/5000 machine with a system clock frequency of 82 MHz and a main memory size of 2048 MB.

Please note that when the user selects a different set of slices (2-D cross-sections, stacked and used as textures) and planes (geometric objects perpendicular to the viewing direction), the data set is not loaded again, since it is already present in the memory. Moreover, rendering time also decreases for subsequent combinations of slices and planes, since the scene graph is not constructed again from scratch, instead new nodes are added to the already constructed scene graph.

The average total time for rendering eight slices and sixteen planes of any data set is less than 15 seconds. Therefore, in less than 15 seconds the user can get a preview of the data set, and the image will be continuously refined as new detail coefficients are received by the client. There is a 2.51 factor of increase in the total time to render the CT head data set with 256 slices and 512 planes on Machine A, and a 1.73 factor of increase on Machine B, when compared with the times taken to render the same data set without illumination. Similarly, for a human brain data set the time increases by a factor of 4.93 on Machine A and 3.74 on Machine B (rendered with 128 slices and 220 planes). This factor of increase in time is in the processing step where all the voxels in the 3-D texture are traversed to define normals for surface voxels and calculate new shaded pixel colors, which are again saved in the 3-D texture.

There is no significant change in the rendering time with illumination and without illumination, since in both cases always the same size of the 3-D texture is rendered irrespective of its pixel colors.

Conclusions and Future Work

We have presented a 3-D Haar wavelet algorithm for hierarchical representation of large volume models that can be used to transmit low-resolution representations followed by detail information across the network. Progressive data transmission enables the client to identify features in an initial preview image instantly while the rest of the data set is added as detail information as soon as it becomes available. VOL<X>REND is a fast, interactive, Java-based, platform-independent, texture-based, direct feature identification and volume rendering system for large biomedical volume models.

Defining gradient-based normals for surface voxels in 3-D volumes and incorporating an illumination model have significantly improved the image quality of the rendered images. Processing time is slightly increased when using illumination, but the rendering time remains constant, so that the frame rate and the interactive response time of client application remain unchanged. Future work will include

shadow calculation and a C/C++ implementation of the rendering engine.

Table 1. Timing results of a CT scan of a human head without and with illumination

Slices / Planes	Loading (ms)		Processing (ms) (Without Illumination)		Processing (ms) (With Illumination)		Rendering (ms)	
	A	B	A	B	A	B	A	B
8/16	1435	3108	5512	10547	7671	13834	158	92
64/128	0	0	26921	42049	48505	72488	103	227
128/256	0	0	53286	84160	113118	144540	132	538
256/512	0	0	105541	167784	266114	291258	196	756

Table 2. Timing results of a human brain without and with illumination

Slices / Planes	Loading (ms)		Processing (ms) (Without Illumination)		Processing (ms) (With Illumination)		Rendering (ms)	
	A	B	A	B	A	B	A	B
8/16	2864	3388	2952	5753	5215	9723	364	85
16/32	0	0	2499	2977	9616	16031	49	185
32/64	0	0	4686	5980	17961	26431	31	199
64/128	0	0	9493	11881	40118	49629	51	235
128/220	0	0	18772	25324	92749	96182	83	455

Acknowledgements

The author would like to thank Arthur J. Olson (The Scripps Research Institute, La Jolla, CA) for providing the sample data sets, David Nadeau and Jon Genetti (San Diego Supercomputer Center, SDSC) for providing the source codes for the *Scalable Visualization Toolkits*, and Sagar Saladi and Pujita Pinnamaneni for the implementation of the Java3D renderer. This project was funded in part by the National Partnership for Advanced Computational Infrastructure (NPACI) under award no. 10195430 00120410.

References

[1] Saha, S. 2000. “Image Compression-from DCT to wavelets: A Review.” http://www.acm.org/crossroads/xrds6-/shahimgcoding.html

[2] Boggess, A.; F. J. Narcowich. 2001. A First Course in Wavelets with Fourier Analysis. Prentice-Hall.

[3] Meyer, J. 1999. “Interactive Visualization of Medical and Biological Data Sets." Ph.D. Dissertation, University of Kaiserslautern, Germany.

[4] Levoy, M. 1990. "Efficient Ray Tracing of Volume Data." ACM Transactions on Graphics, vol. 9, no. 3, July: 245-261.

[5] Westover, L. A. 1990. "Footprint Evaluation for Volume Rendering.” Computer Graphics, vol. 24, no. 4, August: 367-376.

[6] Lacroute, P.; and M. Levoy. 1994. “Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation.” Proceedings of the 21st conference on Computer graphics and interactive techniques. ACM Press, NY, 451-458.

[7] Akeley, K. 1993. “Reality Engine Graphics.” In Computer Graphics, Proceedings of SIGGRAPH 93, 109-116.

[8] Cabral, B; N. Cam; J. Foran. 1994. “Accelerated Volume Reconstruction and Tomographic Reconstruction Using Texture Mapping Hardware.” In 1994 workshop on volume visualization (Washington, DC, October). 91-98.

[9] Meissner, M; U. Hoffmann; W. Strasser. 1999. "Enabling Classification and Shading for 3-D Texture Mapping based Volume Rendering Using OpenGL and Extensions.” Proceedings of IEEE Visualization (Oct 24-29). IEEE, 207-526.

[10] Westermann, R; T. Ertl. 1998. “Efficiently Using Graphics Hardware in Volume Rendering Applications.” Proceedings of the 25th annual conference on Computer Graphics and interactive techniques (July). ACM Press, NY, 169-177.

[11] Bailey, M. 2001. "Interacting with Direct Volume Rendering.” IEEE Computer Graphics and Applications, vol. 21, issue 1, February: 10-12.

[12] Engel, K; P. Hastrieter; B. Tomandl; K. Eberhardt; T. Ertl. 2000. "Combined local and remote visualization techniques for interactive volume rendering in biomedical applications.” Visualization 2000. Proceedings (Salt Lake City, UT). 447-452.

[13] Meyer, J.; R. Borg; B. Hamann; K. I. Joy; A. J. Olson. 2000. "VR based Rendering Techniques for Large-scale Biomedical Data Sets.” Online Proceedings of NSF/DoE Lake Tahoe Workshop on Hierarchical Approximation and Geometrical Methods for Scientific Visualization (Granlibakken Conference Center, Tahoe City, CA). 73-76.

[14] Gelder, A. V.; K. Kim. 1996. "Direct Volume Rendering with Shading via Three-Dimensional Textures.” Proceeding of 1996 Symposium on Volume Visualization (October). IEEE Computer Society, 23-30, 98.

[15] Pinnamaneni, P.; S. Saladi; J. Meyer. 2002. "Remote Transformation and Local 3-D Reconstruction and Visualization of Biomedical Data Sets in Java3D.” Visualization and Data Analysis 2002 Conference, Photonics West - Electronic Imaging 2002 (San Jose, CA, Jan 19 – 25). 44-54.

[16] Saladi, S.; P. Pinnamaneni; J. Meyer. 2001. “Texture-Based 3-D Brain Imaging.” 2nd IEEE International Symposium on Bioinformatics & Bioengineering (Bethesda, MD).

[17] Eric, T. D. D.; J. Stollnitz; D. H. Salesin. 1995. “Wavelets for Computer Graphics: A Primer Part I.” IEEE Computer Graphics and Applications, vol. 15, no. 3, May: 76-84.

[18] Hackbert, P. S. 1986. “Survey of texture mapping.” IEEE Computer Graphics and Applications, Nov: 56-57.

[19] Bouvier, D. J. 2002. “Getting started with the Java3D API.” http://developer.java.sun.com/developer/onlineTraining/java3d.

[20] Moeller, T. A.; E. Haines. 2002. Real-Time Rendering. ISBN 1568811829.

[21] Lorensen, W.E.; H. Cline. 1987. “Marching Cubes: A High Resolution 3D Surface Construction Algorithm.” ACM Proceedings, Computer Graphics, vol. 21, no. 4, July: 163-169.

[22] Nguyen, H. T.; J. Meyer. 2005. “Texture-Based Volume Rendering of Large-Scale Biomedical Data.” UC System-wide Bioengineering Symposium “Envisioning the Biomedical Future,” UC Santa Cruz, CA, June 25-27, 74.