Wavelets And Textures With Illumination For Web-based Volume Rendering

Sagar Saladi, Pujita Pinnamaneni
Mississippi State University
Engineering Research Center, 2 Research Blvd.
Starkville, MS 39762-9627, USA
{ss14 | pp4} @ msstate.edu

Joerg Meyer
University of California, Irvine
Department of EECS/BME
644E Engineering Tower, Irvine, CA 92697-2625, USA
jmeyer@uci.edu

Keywords: Volume rendering, texture mapping, Haar wavelets, illumination

Abstract

Advanced medical imaging technologies have enabled biologists and biomedical researchers to gain better insight in complex, large-scale data sets. These data sets, which occupy large amounts of space, can no longer be archived on local hard drives and are also difficult to transmit over currently existing networks. To make the data accessible to researchers at remote locations over the Internet within a reasonable amount of time, we are describing a web-based volume rendering system that incorporates a multi-resolution technique for transforming large-scale data sets into hierarchical volumes. 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 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.

INTRODUCTION

We want 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. Volume rendering is a technique for visualizing, interacting with and interpreting large volumetric data sets by sampling data in three dimensions. It is a powerful tool well suited to a wide range of applications in different scientific disciplines. Enhanced imaging techniques like Computed Tomography (CT), Magnetic Resonance Imaging (MRI), Positron Emission Tomography (PET), Confocal laser-scanning microscopy, etc., allow physicians and biologists to distinguish pathological from healthy tissues or to study microscopic cell structures in greater detail. The common aspect about all of these scanning devices is that they provide comprehensive 2-D cross-sectional images of anatomical structures of the human body at high resolution. The size of these data sets makes them difficult to store on a local hard drive and prohibits efficient transmission over currently existing networks.

To provide a facility for large-scale data storage, San Diego Supercomputer Center (SDSC) maintains a High-Performance Storage System (HPSS), where large structured and unstructured mesh data sets can be stored. The Scalable Visualization Toolkits (*VisTools*), an NPACI initiative, primarily developed at SDSC, The Scripps Research Institute (La Jolla, CA), UC Davis, UC Irvine, U Texas, and Mississippi State University, are used to access these data sets.

The large-scale volumetric data sets provided by HPSS are commonly stored in two typical file formats: the MSH format is used for unstructured grids and supports multiple data sets in a single file, while the VOL format is used for structured, volumetric data. The VOL file format consists of two variants: "V1" (simple format) and "V2" (additional features that can handle large data sets more efficiently). Additionally, both V1 and V2 variants come in three different flavors: *vols*, *volb*, and *volc*. A *vols* file consists of 8-bit scalar values, a *volb* file consists of data stored as 32-bit RGB or RGBA values, and a *volc* file stores data as 64-bit RGB-alpha-beta values. "V2" can store data in both chunked and unchunked formats. The chunked layout is used when the user is interested in rendering a particular region-of-interest (ROI), say a tumor in the brain, or a particular sub-volume in general.

BACKGROUND

Uncompressed data sets require massive storage capacity and transmission bandwidth. Several file compression schemes have been introduced. 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. 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 [2]. The Haar wavelet is one of the simplest wavelet transforms and can be computed efficiently using Integer arithmetics. 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].

In most cases of image transformation, wavelets have been implemented for one-dimensional signal transformation or two-dimensional image decomposition. As volume rendering of images

has become more and more important, the necessity to store volume data in a more compact way and the need for hierarchical, multi-resolution representations and progressive data transmission motivates the implementation of a Haar wavelet transformation in 3-D.

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]. Splatting [5] is also computationally expensive though it is more efficient than ray tracing. The Shear-Warp [6] volume-rendering algorithm is based on the factorization of the viewing matrix, which produces artifacts if the opacity or color attributes of the volume contain high frequencies.

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]. Some authors proposed loading the entire data set onto one's local drive, but these methods could prove tedious due to the limited storage capacity on the client side. Server-based storage can also be inefficient if large data sets need to be transmitted due to the limited bandwidth that current networks typically provide. To overcome these drawbacks, we are developing a web-based volume rendering system, which uses 3-D Haar wavelet transformations to transform large data sets into multi-resolution volumes before being transmitted over the network to the client side. The transformed volumes are then used in texture-based volume rendering on the client side. Previous texture-based volume rendering algorithms have incorporated different lighting methods to bring realism to the rendered volumes [10, 11, 14]. Our texture-based volume rendering application provides an efficient illumination implementation by using a positional light source and pre-calculated normal vectors for all surface voxels of the 3-D volume.

WEB-BASED VOLUME RENDERING SYSTEM

Our web-based volume rendering system implements a client-server model (figure 1) [15, 16]. The data repository (HPSS), where our input data sets are stored, allows the users to define public and private user groups.

On the server side, depending on the file format either a series of 2-D cross-sections or a sub-volume is extracted from a data set using *VisTools*. 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. 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.

ACCESSING DATA SETS USING VISTOOLS

The *VisTools* are available in both a Java and C++ version. To implement a platform-independent application, the Java version of the *VisTools* is used. The *VisTools* can be used to extract 2-D cross-sections from the data sets.

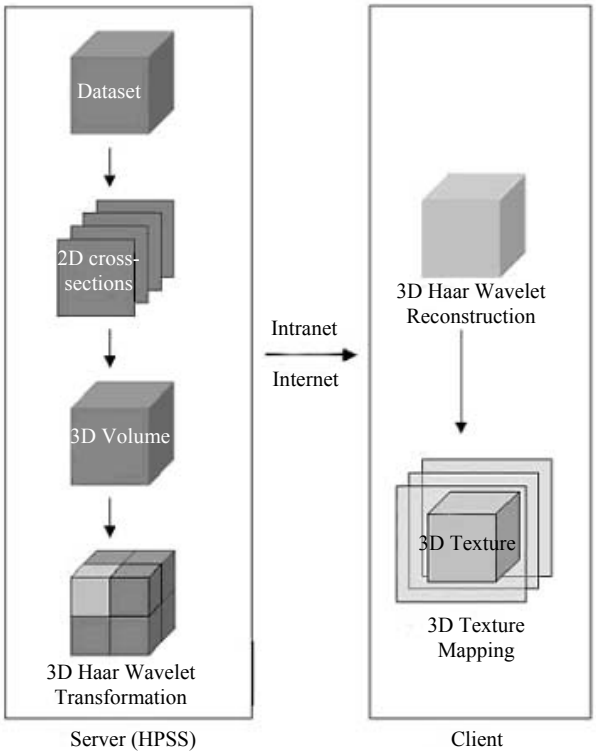


Figure 1. System Architecture

In order to reduce the amount of information that needs to be transmitted from the server to the client, sub-volumes can be extracted from a large data set. Extraction of sub-volumes is helpful when the user is interested in a particular region of the data set and wants it to be rendered at a higher level-of-detail than the rest. A low-

resolution version of the data set is used to provide the necessary context information and to enable orientation and navigation within the data set. For example, a neuroscientist might be interested only in a particular part of the brain for conducting his or her analysis of the tissue. For such cases, *VisTools* support a special format called the *chunked* file format ("V2"), which enables the extraction of sub-volumes from a data set.

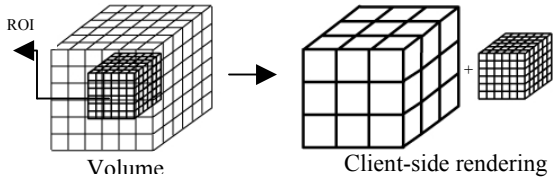


Figure 2. Sub-volume representation

We tested our algorithm by extracting cross-sections from a CT scan of a human head (*ctbrain.vols*), a human brain (*brain_stride_8.volc*), a cancer cell (*cell.volc*) and a series of cryosections of a human brain (*brain_stride_cryo_8.volc*). The CT scan of a human head consists of 512 x 512 x 231 elements, the human brain data set consists of 94 x 113 x 131 elements, the cancer cell data set consists of 251 x 70 x 312 elements, and the cryosections of a human brain consist of 223 x 144 x 184 elements. All the sizes of these data sets clearly show that they occupy large amounts of space and cannot be accommodated on commodity desktop machines.

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

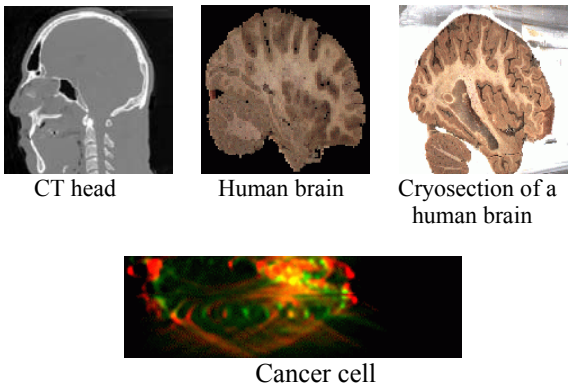


Figure 3. Sample 2-D cross-section

HIERARCHICAL REPRESENTATION OF LARGE DATA SETS USING 3-D HAAR WAVELETS

After the cross-sections are extracted from the data set, the data needs to be transformed into lower-resolution representations to enable faster transmission over the network. The Haar wavelet transformation has been implemented in 3-D to enable transformation of a 3-D volume. The Haar wavelet transformation decomposes an image into a set of low-pass filter coefficients and a set of high-pass filter coefficients (detail coefficients). To give a better idea of the actual implementation of the wavelet transformation, we illustrate the procedure with a simple numeric example.

Assume we have a one-dimensional image with an 8-pixel resolution where the pixels have the following values [17]:

9 7 3 5 9 3 1 9
Original data

The low-pass filtered coefficients are obtained by averaging two consecutive pixels, while the detail coefficients represent the difference between the average and one of the two consecutive pixels. After the transformation cycle, the above image will be represented as follows:

8 4 6 5 — Low-pass filter coefficients
1 –1 3 –4 — Detail coefficients

Now the original image can be represented as a low-resolution image ((a+b)/2), which consists of four pixels, and another four-pixel image, which contains the detail coefficients ((a-b)/2). Recursively repeating this algorithm leads to an image that is reduced by a factor of two for each cycle.

This simple 1-D scheme can be lifted to higher dimensional cases. For a 2-D wavelet transformation, this algorithm is applied in the *x*-direction first, and then in the *y*-direction. Figure 4 shows the low-pass filter coefficients and the detail coefficients obtained after each cycle for a single slice of a human brain. Since most of the detail coefficients are very small, they can be discretized or neglected in order to obtain higher compression rates (lossy compression). One cycle for an *n*-dimensional data set is defined as the completion of the algorithm for all *n* directions [15, 16].

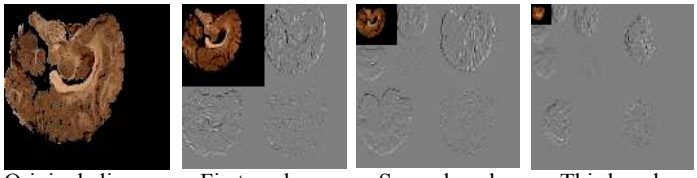


Figure 4. Low-pass filter and detail coefficients of a Haar wavelet transformation

For progressive rendering, the low-pass filtered coefficients are sent to the client first, while the detail coefficients are transmitted at a later time. When the detail coefficients are received on the client side, detail information is added to the volume, which has already been rendered, to refine the image. The reconstruction of the image data uses simple arithmetic operations (integer arithmetic). As the image array received on the client side consists of the low-pass filter coefficients and the details coefficients, the respective pixel values are obtained by adding and subtracting the corresponding detail coefficients to and from the low-resolution image coefficients. The reconstructed pixel values are:

(8+1) (8-1) (4+(-1)) (4-(-1)) (6+3) (6-3) (5+(-4)) (5-(-4))

9 7 3 5 9 3 1 9
Reconstructed data

These values are identical to the original values. After the data set has been reconstructed using the Haar wavelet reconstruction algorithm, the volume is rendered using 3-D texture mapping in Java3D.

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.

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 that is used in developing our texture-based volume rendering application is shown in the figure 5.

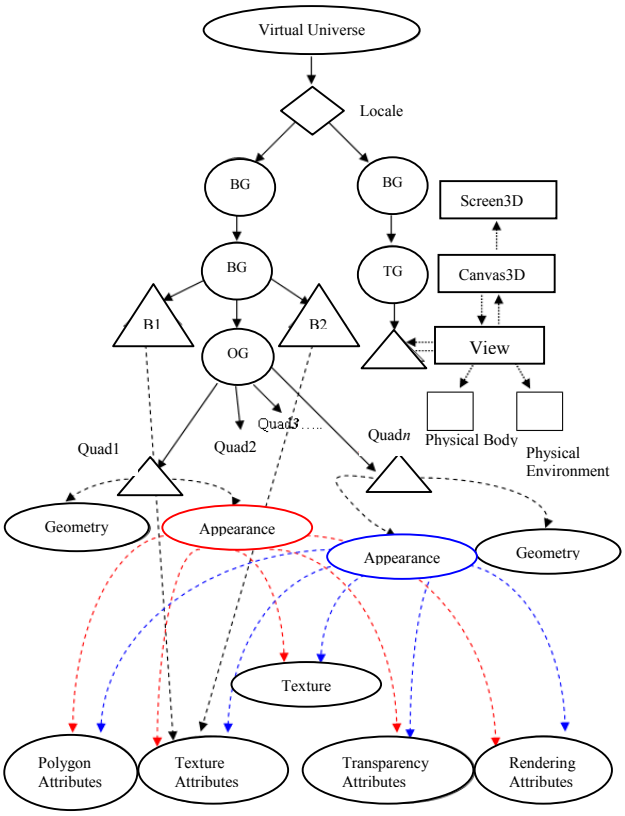


Figure 5. Scene graph

For loading a data set into memory, a mesh is created using the *VisTools*. Once the mesh is loaded into the memory successfully, pre-processing is performed to create a scene graph with the 3-D texture and its attributes for rendering. Alpha values are assigned to the extracted 2-D cross-sections for enabling transparency and alpha blending, through which the interior structures of the volumetric data set can be rendered. All the planes that are in the scene graph are rendered in a back to front order using the 3-D texture mapping hardware in order to display a 3-D reconstructed volume. Results obtained by rendering different data sets in 3-D are presented in figure 6.

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 7 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, that is capable of reading *volb*, *volc* and *vols* (*vol*x) files, has been developed for rendering these reconstructed data sets in 3-D (figure 8). 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.

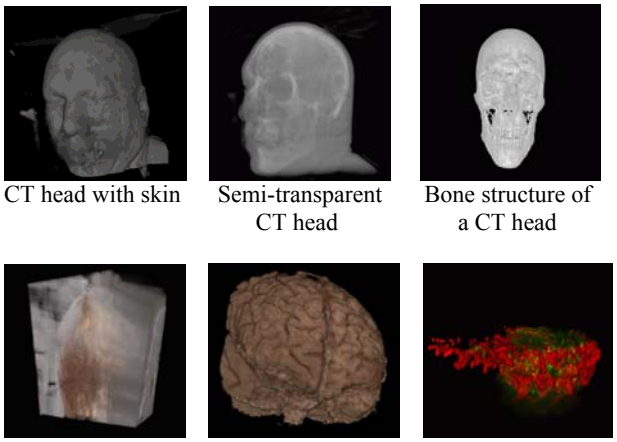


Figure 6. Data sets rendered using 3-D texture mapping

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. For each transparency function, the alpha value of every pixel in the 3-D buffer is modified. When the user selects a different transfer function or different number of slices and planes, a new 3-D texture and the newly specified number of planes are created with the new alpha values inserted into the 3-D texture buffer.

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.

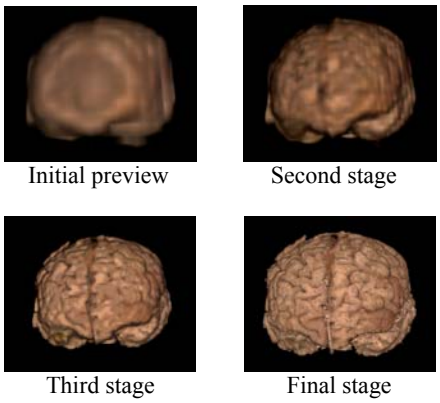


Figure 7. Progressive reconstruction

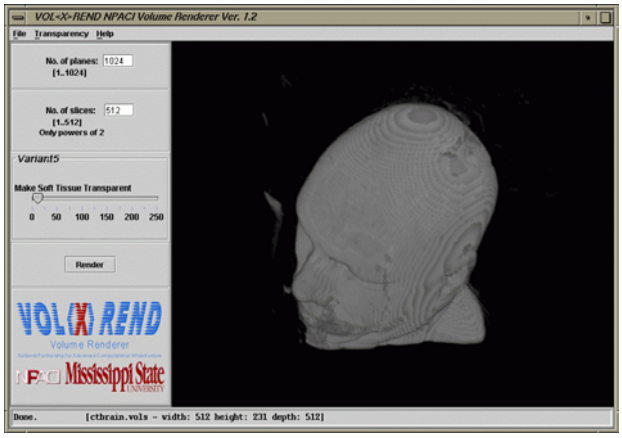


Figure 8. User interface of VOL<X>REND

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].

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. 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.

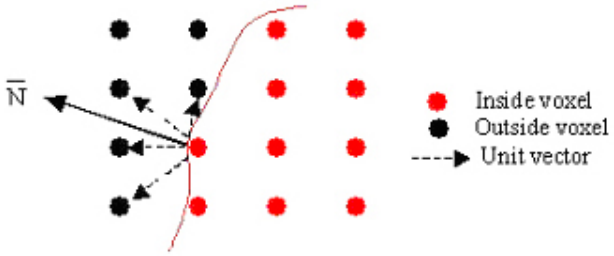


Figure 9. Calculating a normal for a 2-D surface point

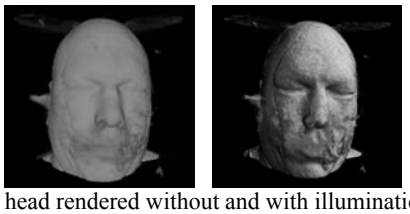
According to Lambert's law, a dot product must be calculated between the light vector and the normal of a surface voxel to determine the percentage of light a surface voxel reflects. For diffuse surfaces, the reflected light is determined by the cosine between the surface normal N, and the light vector 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.

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.

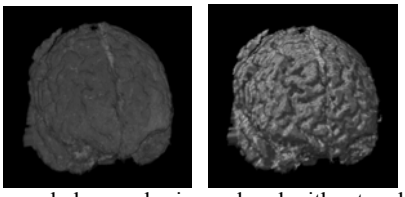
The advantage of this algorithm is that for a fixed light source it is possible to rotate, scale, and translate the object within the texture buffer without any performance loss.

Figure 10 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.



CT head rendered without and with illumination

Human brain rendered without and with illumination



A gray scale human brain rendered without and with illumination

Figure 10. 3-D Rendered volumes showing the effects of illumination

STATISTICS

Timings were taken for sample data sets 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+2.

These timings were taken for rendering the data sets 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™ with 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 and planes, 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.

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

CONCLUSIONS AND FUTURE WORK

We have presented a 3-D Haar wavelet algorithm for hierarchical representation of large data sets that can be used to transmit low-resolution representations followed by detail information across the network. Progressive data transmission enables the client to render 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 volume rendering system for large biomedical data sets that are encoded in the VOL file format. It can be easily extended to read other volumetric formats by replacing the read method, and it can also be used for other application domains.

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 lighting, which affects the overall performance of VOL<X>REND, but the rendering time remains constant, so that the frame rate and the interactive response time of client application remain unchanged. This is an important feature and one of the major contributions of this work.

Future work will focus on calculating shadows for volumes, which will provide even more realism. New techniques need to be explored for re-rendering a data set without much decrease in performance when the position of the light source is moved. Since VOL<X>REND is implemented in Java, it will be integrated into a web-based volume rendering system, such that it supports multiple users and allows to render large-scale biomedical datasets or sub-volumes of such data sets hierarchically (low-resolution followed by detail information) over the web.

ACKNOWLEDGEMENTS

The authors 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.* 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. and 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. and 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, 587.

[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. and K. Kim. 1996. "Direct Volume Rendering with Shading via Three-Dimensional Textures.” *Proceedings 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, 2002). 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 1995: 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] Moller, T. A.; E. Haines. 2002. *Real-Time Rendering*. A K Peters, Natick, MA.