Remote Transformation and Local 3-D Reconstruction and Visualization of Biomedical Data Sets in Java3D

Pujita Pinnamaneni[*], Sagar Saladi[**], and Joerg Meyer[***]

Mississippi State University, NSF Engineering Research Center
2 Research Blvd., Starkville, MS 39759-9627

ABSTRACT

Advanced medical imaging technologies have enabled biologists and other researchers in biomedicine, biochemistry and bio-informatics to gain better insight in complex, large-scale data sets. These datasets, which occupy large amounts of space, can no longer archived on local hard drives. San Diego Supercomputer Center (SDSC) maintains a large data repository, called High Performance Storage System (HPSS), where large-scale biomedical data sets can be stored. These data sets must be transmitted over an open or closed network (Internet or Intranet) within a reasonable amount of time to make them accessible in an interactive fashion to researchers all over the world. Our approach deals with extracting and transforming these data sets using Haar wavelets, and transmitting them over a low- to medium-bandwidth network. These compressed data sets are then reconstructed into a 3-D volume on the client side, and rendered using texture mapping in Java3D. The data sets are handled using the *Scalable Visualization Toolkits* developed within an NPACI (*National Partnership for Advanced Computational Infrastructure*) framework. Sub-volumes of the data sets are extracted to provide a detailed view of a particular region of interest (ROI). This application has also been ported to a C++ platform to obtain higher rendering speed and better performance but lacks platform independency.

Keywords: 3-D Haar Wavelet Transformation, Texture Mapping, Transparency, Scalable Visualization Toolkits, Medical Data Sets

1. INTRODUCTION

Collaborative research in biomedicine requires dealing with large-scale data sets and being able to transmit them over existing Internet-based networks. Large-scale data sets in biomedical imaging are commonly obtained from scanners such as CT, MRI, PET, etc. The common aspect about all these scanners is that they provide detailed cross-sectional views of any part of the human body at a very high resolution. Data sets obtained from these scanners occupy huge amounts of memory space, which may range from several hundred of megabytes to gigabytes. This clearly indicates that these data sets cannot be accommodated on commodity desktop machines. To provide a solution for these types of problems, San Diego Supercomputer Center (SDSC) maintains a large data repository (High Performance Storage System, HPSS). This allows researchers to make their data sets available and accessible to other researchers over the Internet. We are developing applications to visualize biomedical data sets stored in HPSS in 3-D and provide access to researchers involved in studying these data sets.

The *National Partnership for Advanced Computational Infrastructure* (NPACI) has developed *Scalable Visualization Toolkits* (VisTools), to access these data sets stored in HPSS. The *VisTools* project is part of an NPACI Alpha project. These toolkits can be used for analysis, filtering, compositing, rendering, and interacting with large data sets obtained from several different scientific disciplines. These toolkits are available both in a C++ version and in a Java version.

[*] pp2@erc.msstate.edu, Dept. of Computer Engineering, Mississippi State University
[**] saladi@erc.msstate.edu, http://www.cs.msstate.edu/~sagarsv, Dept. of Computer Science, Mississippi State University
[***] jmeyer@erc.msstate.edu, http://www.cs.msstate.edu/~jmeyer, Dept. of Computer Science, Mississippi State University

The *Scalable Visualization Toolkits* store large volumetric data sets in HPSS in a typical file format called the VOL file format. The VOL file format has two variants: "V1" (version 1) and "V2" (version 2). Both of the two variants have three flavors. "V1" includes *vols*, *volb* and *volc* file formats. The *vols* file consists of 8-bit scalar values, while the *volb* file stores data as 32-bit RGB or RGBA values. The *volc* file format represents data as 64-bit RGB-alpha-beta values. The first 32-bit word contains a 10-bit red, 12-bit green, and 10-bit blue value, and the second 32-bit word consists of 16-bit alpha and 16-bit beta values. The second variant "V2" comprises all three formats specified above, but the data in the files are stored in the form of chunks.

We are using the VisTools to access large-scale biomedical data sets stored in HPSS and develop applications to visualize them in 3-D and to provide access to clients all over the world through a web-based interface. Our implementation allows for extracting 2-D cross-sections and sub-volumes, transforming them into lower level-of-detail representations using a 3-D Haar Wavelet Transformation[1], transmitting them over the network to the clients, transforming them back on the client side, and using them in reconstructing a 3-D volume using texture-mapping techniques in Java3D. The methods have been implemented in Java and Java3D, and have also been ported to C++ to improve rendering performance. The following sections describe our system architecture, and the procedures for extracting slices and sub-volumes from the data sets, the 3-D Haar Wavelet algorithm, and the texture-mapping techniques used in our implementation.

2. BACKGROUND

Volume rendering has gained much importance in recent times. Researchers around the world have been exploring techniques to render large-scale biomedical data sets at reasonable speeds. Hardware volume rendering techniques based on OpenGL[2] and Open Inventor to enable interactive rendering[3] have been implemented earlier. Many algorithms like ray tracing[4], ray casting[5], and marching cubes[6] have been implemented. But these techniques imposed restrictions on volume size and limited flexibility in terms of rendering and illumination. The need for web applications in recent years has encouraged researchers to explore web-based rendering techniques using Java[3, 7]. Earlier works included methods of extracting slices on the client side[8] or remotely on the server side[9]. Some authors proposed loading the entire data set onto one's local drive, but these methods were not economical in terms of storage and rendering speeds. Moreover, transmitting these large data sets over the network could prove tedious. Hence compression methods have been analyzed. Various image compression schemes such as JPEG, Multiwavelet compression techniques[10] and Multi-resolution trees[11] have been discussed. But most of these techniques were in 2-D. Authors have also been dealing with 3-D wavelets for video coding[12, 13].

In order to overcome some of the problems stated above, we are proposing a platform independent client-based rendering technique[14, 15]. The data sets stored in HPSS are transformed into multi-resolution volumes using 3-D wavelet transformations before being transmitted over the network to the client side where the data are rendered in 3-D using texture-mapping techniques in Java3D.

3. SYSTEM ARCHITECTURE

The scalable visualization system described in this paper is a client-server system (figure 1). On the server side, either a series of 2-D cross-sections, or a sub-volume is extracted from the data repository. The *VisTools* are primarily used to access the data sets from the repository and to decode the file format. The volumes are transformed into lower level-of-detail representations using a Haar wavelet transform. The transformed volumes are then transmitted over the network to the client side where the volumes are reconstructed using the inverse Haar wavelet transform and rendered as a 3-D volume using texture mapping in Java3D. The reconstruction is lossless if all detail coefficients are taken into account and if no quantization is applied to the high-pass and low-pass filtered coefficients. For most applications, it is not necessary to render the entire data set at full resolution. Our system allows the client to specify a region of interest (ROI). This ROI is transmitted first, along with a coarse overview representation of the rest of the data set, and then detail coefficients are transmitted to the client to refine the rendering of the ROI.

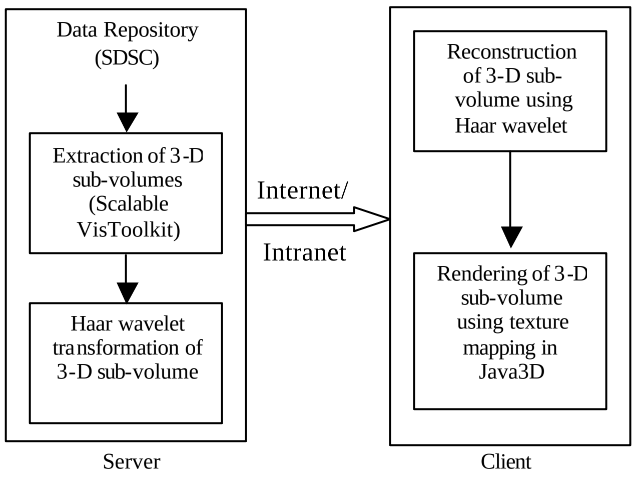

Data Repository (SDSC)
Extraction of 3-D sub-volumes (Scalable VisToolkit)
Haar wavelet transformation of 3-D sub-volume
Internet/ Intranet
Reconstruction of 3-D sub-volume using Haar wavelet
Rendering of 3-D sub-volume using texture mapping in Java3D
Server
Client

Figure 1: Scalable visualization system

4. IMPLEMENTATION

This section describes the extraction of 2-D slices and sub-volumes from the data sets stored in HPSS using the NPACI *VisTools*, the 3-D Haar Wavelet algorithm, and reconstruction of a 3-D volume using 2-D texture mapping in Java3D.

4.1 Extraction of 2-D cross-sections and sub-volumes from the data sets

The 2-D slices and sub-volumes are extracted from the data sets stored in HPSS using the Java version of *VisTools*. In extracting 2-D slices we used a sample data set of a CT scan of a human brain (*ctbrain.vols*). This data was represented in “V1” format. It consists of 512 x 512 x 231 elements, which make up to 60,555,264 bytes or 57.8 MB, comprising 231 2-D cross-sections, each consisting of 512 x 512 pixels. Each file contains additional header information that describes the dimensions of the pixels in each direction. The *Scalable Visualization Toolkit* provides methods for checking the header information to identify the file format, and for reading a range of elements and returning their values in the host’s byte order and word size. In our case we are reading 512 x 512 pixels at each time in order to extract a series of 2-D cross-sections, i.e., slices in *z*-direction. We are also extracting 2-D slices in *x*- and *y*-directions. The purpose of extracting cross-sections in three perpendicular directions will be explained in a later section.

There are situations where the user might be interested in a particular region of the data set and wants it to be rendered at a higher level-of-detail. For example, a researcher might be interested in viewing the tumor in the brain in greater detail, while it might be sufficient to render the rest of the data set at a coarser resolution to provide spatial context and to give the physician a visual aid about the location of the tumor. In these situations, sub-volumes of the data set need to be extracted. For extracting sub-volumes of the data set, we are using another format of the sample data set (*ctbrain_c32.vols)*. This data set uses the “V2” variant of the VOL file format. The *ctbrain_c32.vols* data set consists of data items arranged in the form of chunks. The *Scalable Visualization Toolkit* implements a method which can be used to extract sub-volumes of the data set. This method computes and returns a one-dimensional memory index that can be used to extract cross-sections of a particular chunk using the same method as described before. This allows us to reduce seek times and to extract cross-sections from sub-volumes in a more efficient way. Sample cross-sections of a sub-volume are shown in figure 2.

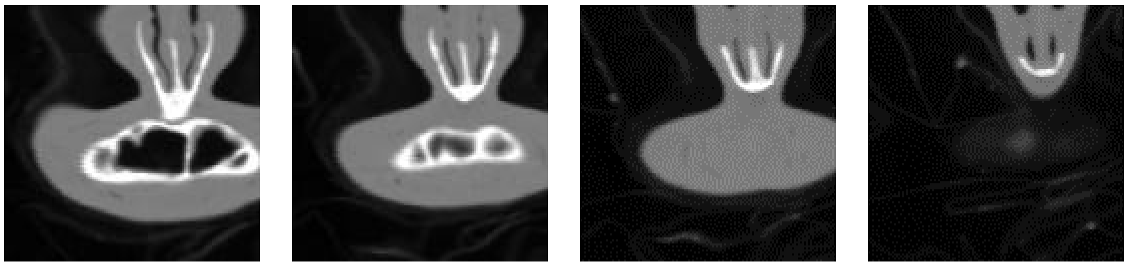


Figure 2: Sample cross-sections of a sub-volume (human skull, nose)

4.2 Haar Wavelet Transformation

To enable faster transmission of the extracted volumes and sub-volumes across existing Internet-based networks, the volumes are transformed into a multi-resolution format on the server. Haar wavelets provide optimal performance for discrete volume data because their base functions and coefficients are easy to compute. Haar wavelets convert a signal from the spatial domain into the frequency domain. This can be implemented using integer arithmetics[16]. To illustrate the simplicity of the Haar wavelets, an example for processing a discrete signal (pixel image) is given below (2-D case). Consider an image with a resolution of 4 x 4 pixels. Each pixel has a scalar value (integer).

6	7	5	4
5	4	9	2
3	8	4	7
6	7	8	4

Figure 3: 2-D image (scalar integer values)

By applying the Haar wavelet transform, the image gets decomposed into a low-resolution image and a set of detail coefficients (see figure 4). The low-resolution image contains the wavelet-transformed image coefficients, i.e., the low-frequency components of the image. These transformed coefficients are obtained by averaging two consecutive pixels in each row. The detail coefficients are obtained by calculating the difference between the average and one of the two consecutive pixels. They are essential for the reconstruction of the image on the client side. Figure 4 (a) shows the low-pass filtered coefficients on the left side while the detail coefficients occupy the right side of the array. For the next run, the algorithm considers the columns instead of the rows, i.e., it takes two adjacent pixels in each column and computes the average and the detail coefficient for each pair of values. After completion of the algorithm, the transformed coefficients occupy the upper left quadrant of the image array, while the other quadrants represent the detail coefficients, i.e., the directional derivatives of the signal.

6	4	0	0
4	5	0	3
5	5	-2	-1
6	6	0	2

(a) First run: x-direction

5	4	4	1
5	5	-1	0
1	0	0	-1
0	0	-1	-1

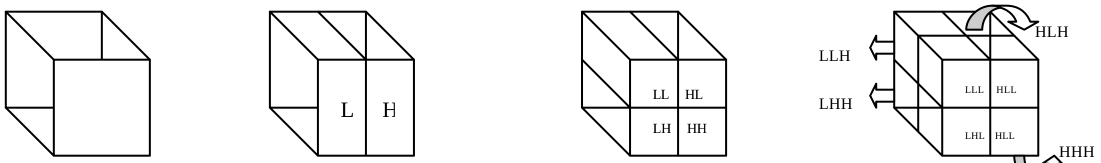
(b) Second run: y-direction

Figure 4: 2-D wavelet transform

Recursively iterating, the image is reduced by a factor of two for each dimension in each cycle. This 2-D scheme can be extended to multidimensional image arrays by adding an additional run for each dimension. A complete cycle comprises of executing the algorithm once for each dimension of the data set. The wavelet coefficients are transmitted sequentially over the network. First, the low-pass filtered coefficients are transmitted, because they represent a reduced version of the original image or sub-volume. It is observed that this part of the image preserves the features of the original image reasonably well even after several cycles (figure 7).

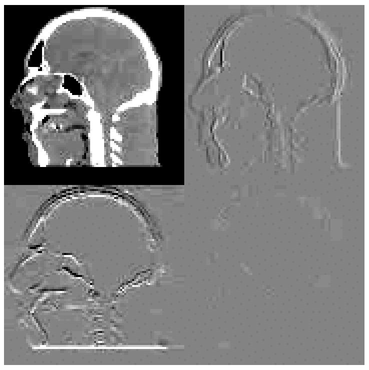
4.2.1 3-D Haar wavelet decomposition

To apply this hierarchical scheme to volume data sets (figure 5(a)), a 3-D Haar wavelet transformation must be implemented. Therefore, the 2-D scheme as described in the example given above is extended into the *z*-direction. The size of the array must be a power of two in each dimension. Unused areas can be filled with zeros. If we apply this algorithm to our sample data set (*ctbrain.vols)*, which consists of 512 x 512 x231 elements, we need to scale the size of the array to the closest powers of two, i.e., 512 x 512 x 256. The algorithm is initially run in *x*-direction, row by row for all 231 slices. The algorithm splits the volume into two halves, the left half representing the low-frequency coefficients while the right half represents the detail coefficients, as shown in figure 5(b). In the second stage of the algorithm, the entire volume is then again transformed in *y*-direction splitting the volume into four quadrants as shown in figure 5(c). For the final run, the volume is transformed in *z*-direction splitting the volume into eight octants. The upper left front octant contains the low-frequency coefficients that are initially transmitted over the network. After completion, the other octants containing the detail coefficients are transmitted to refine the image on the client side.

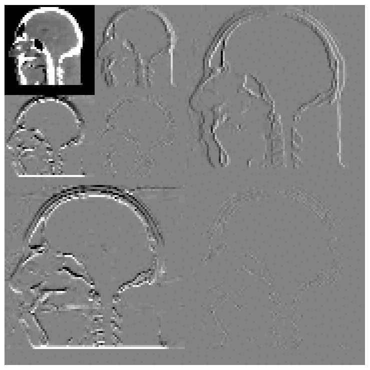


(a) Original volume (b) First run: *x*-direction (c) Second run: *y*-direction (d) Third run: *z*-direction

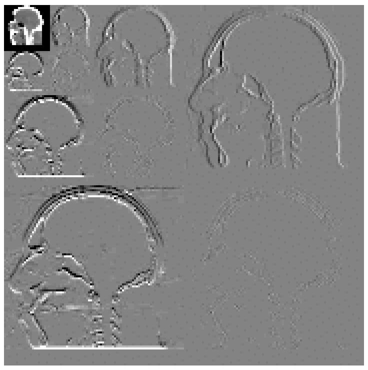
Figure 5: 3-D Haar wavelet transform



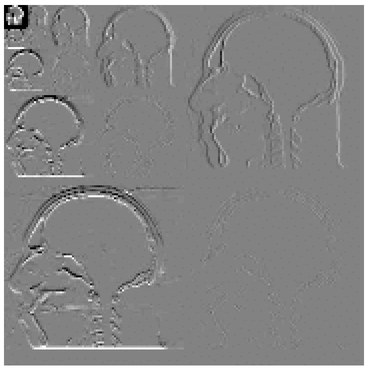
(a) First Cycle



(b) Second Cycle

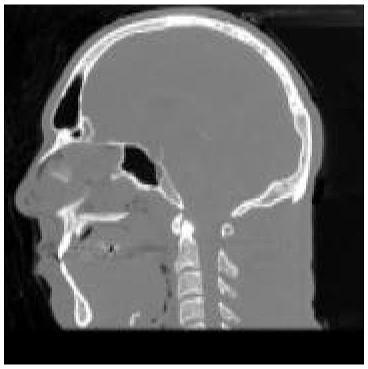


(c) Third Cycle

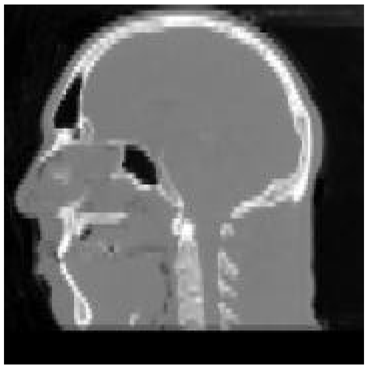


(d) Fourth Cycle

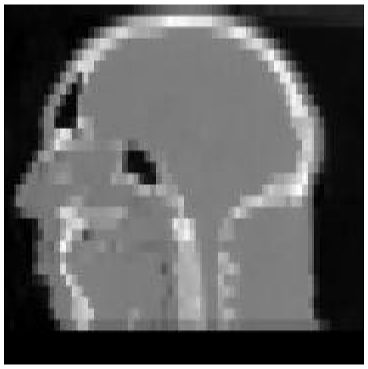
Figure 6: Results showing the low- and high-frequency coefficients for the Haar wavelet transform



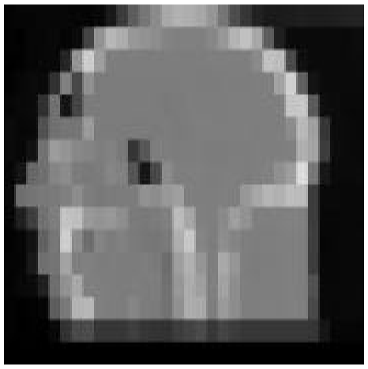
(a) First Cycle



(b) Second Cycle



(c) Third Cycle



(d) Fourth Cycle

Figure 7: Low-frequency components in the image array (scaled to original size)

4.2.2 Reconstruction of the wavelet-transformed data set

On the client side, the detail information is added to the low-frequency components of the volume that have already been transmitted. This means that the texture, which is explained in the next section, is replaced gradually by blending smoothly between the low-frequency image and the more detailed image. To obtain the pixels values at a higher level of detail, we add the detail coefficients to the low-frequency coefficients of each pixel to obtain one pixel value of the original pixel pair, and we subtract them to get the second pixel of each pair.

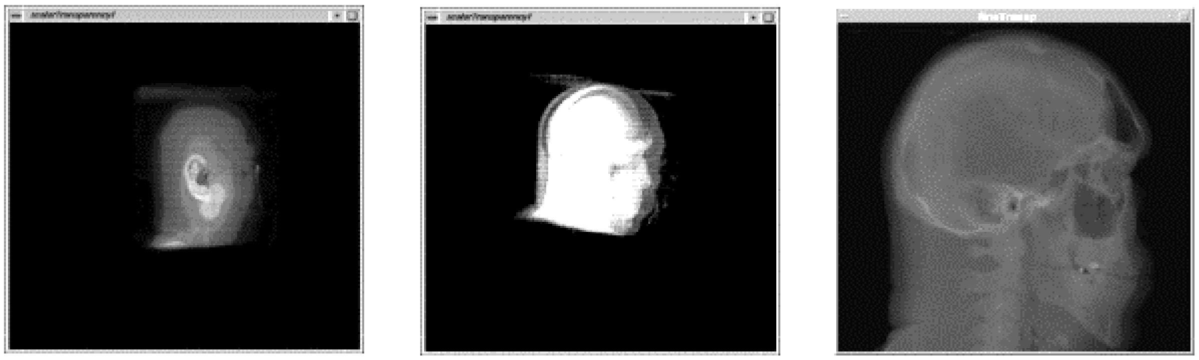
It is not necessary to transmit all the detail coefficients to the client side for reconstruction. We only transmit those that are in the region of interest. This region is gradually refined until it can be rendered at full resolution. The rest of the data set is transmitted at a lower level-of-detail. Transmitting the entire data set at full resolution would require too much memory space on the client side and too much time for transmitting the data over a limited-bandwidth network. During reconstruction, all the detail coefficients that belong to a selected sub-volume are transmitted to the client side and progressively rendered until the highest resolution can be displayed, while the rest of the data set is rendered at a lower resolution providing the spatial context information of the volume.

4.3 2-D Texture mapping

Texture mapping can be used to exploit hardware-accelerated rendering capabilities. Nowadays, texture mapping is often being used in Large-scale Visualization and High Performance Computing[17] . Texture mapping involves mapping a texture image, which is defined over an (*s*, *t*) parameter space in the range 0<=s<=1, 0<=t<=1, onto a geometric object. An affine coordinate transformation maps the texture space into object space.

Texture mapping can be done in 1-D, 2-D, and in 3-D. In our implementation, we are using 2-D texture mapping to utilize the existing texture memory most efficiently (volume depth can be adapted dynamically)[17]. 2-D texture mapping is also very straightforward to implement since we are dealing with a stack of 2-D slices that can be projected in *x*-, *y*- and *z*-directions. The 2-D cross-sections that have been recreated on the client side from the sub-volumes or the entire data set by the wavelet reconstruction algorithm in *x*-, *y*-, and *z*-directions are used to create three sets of perpendicular image planes in 3-D space. These cross-sections are not necessarily the same as the original slices because the data set can be sliced in any arbitrary direction. To load the texture images, a *TextureLoader* class in Java3D is used. After loading, textures are mapped in a back-to-front order onto a series of square-shaped polygons arranged in parallel in *x*-, *y*-, and *z*-directions.

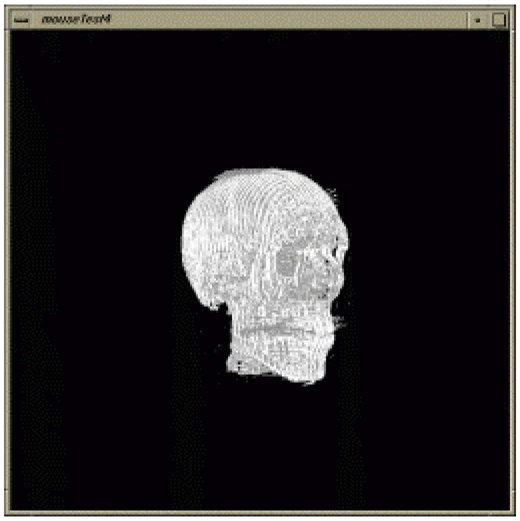
When mapping the cross-sections onto the polygons in all three directions, we need to make them partially transparent. Otherwise we could see only the front-most texture image. To make the remaining slices visible, we need to apply transparency[19, 20] to all the cross-sections that are used in texture mapping. We have implemented two types of transparency: scalar transparency and binary transparency, which are the only two modes currently supported in Java3D. For scalar transparency, a single transparency value is selected and applied to the entire geometric object, i.e., to the textured plane as a whole. The resulting images reveal the interior structures of the data set (figure 8).



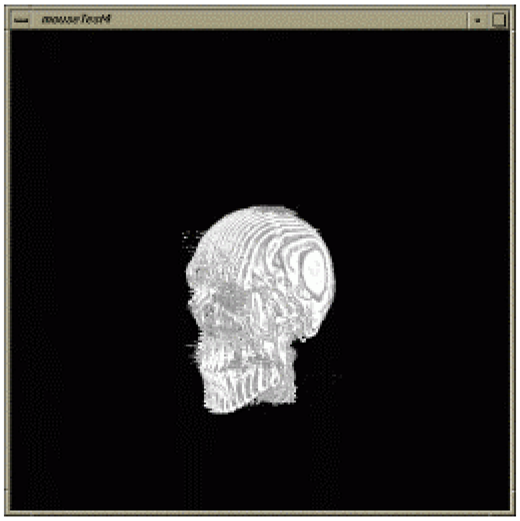
(a) 3-D view of the ear (b) 3-D view of the brain (c) Simulated X-ray view

Figure 8: Results of the Scalar Transparency method in Java3D

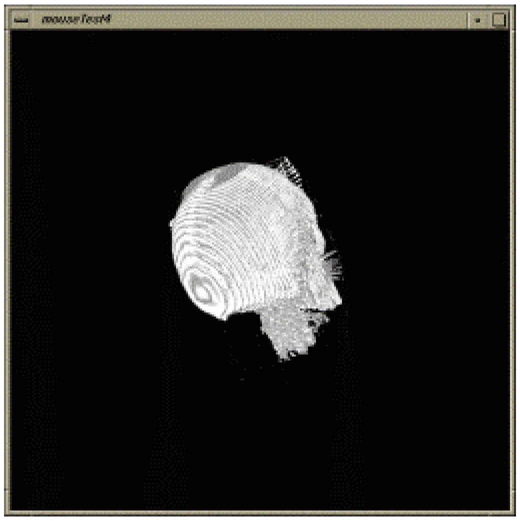
Instead of assigning a single transparency value to an entire cross-section, we want to be able to eliminate background pixels and to render the rest of the slice more opaque. Removing all the background pixels from the 2-D cross-sections and rendering the pixels that make up a slice creates a surface representation of the 3-D volume. This can be achieved using binary transparency. In this case, two transparency values are chosen. One is applied to the background pixels, and the other to the pixels that make up the object. A threshold defines the cut-off value for the background pixels, allowing to select different materials, such as skin or bone. This can be implemented using the alpha channel of each pixel. The current implementation of Java3D does not allow for per-pixel transparency, but it supports alpha testing. The alpha component of the pixels constituting the object is set to zero, while the alpha component of the remaining pixels is set to 255. During the rendering process, we render only those pixels which have alpha values different from zero. Interactive handling methods like rotation, translation, zoom, and spinning (continuous rotation) have been incorporated using Java3D classes.



(a) Left view of the 3-D reconstructed skull



(b) Right view of the 3-D reconstructed skull



(c) Rear view of the 3-D reconstructed skull

Figure 9: Results of the Binary Transparency method in Java3D

5. PORTING TO C++

Despite the fact that the C++ version of the *VisTools* is not platform-independent and that the *VisTools* need to be recompiled for each Unix platform, the anticipation of higher rendering speed in OpenGL and better image quality due to faster update of the high-frequency components of the image motivated us to port both the server and the client to C++. The software interfaces directly with the C++ version of *VisTools*. The wavelet algorithm was ported to C++ using the same algorithm as used by the Java version. Similar results were obtained for both Java and C++ implementations. A timing comparison resulted in a considerable speed-up factor for the C++ version[14]. For the C++ version, we used the same texture-based rendering methods as in Java3D (figure 10).

Texture Mapping in OpenGL can be dealt with in two ways. One way is 2-D texture mapping, where a sequence of 2-D slices are mapped onto polygons located behind one another so that they form a texture volume. Again, three sets of perpendicular slices can be created to avoid gaps when rotating the object, thus avoiding that one can see through between the gaps. The other method is 3-D texture mapping, where the texture is stored in 3-D space and mapped onto a quadrilateral or onto a 3-D object. In both cases, alpha blending results in a much better image quality than alpha testing. In most cases, there is not enough texture memory available on the graphics card to hold the entire data set. In the example given above, the size of the data set is 512 x 512 x 231, which exceeds the size of contiguous texture buffer on most machines. A simple way to work around this problem is to break down the texture of each face into rectangular regions of different resolution. Most of the face is covered by coarser textures (spatial context information), while the area that intersects the ROI is textured using a higher level-of-detail. This results in saving a considerable amount of texture memory. The amount depends on the size of the ROI, and the reduction is typically cubic due to the three-dimensional nature of the data set.

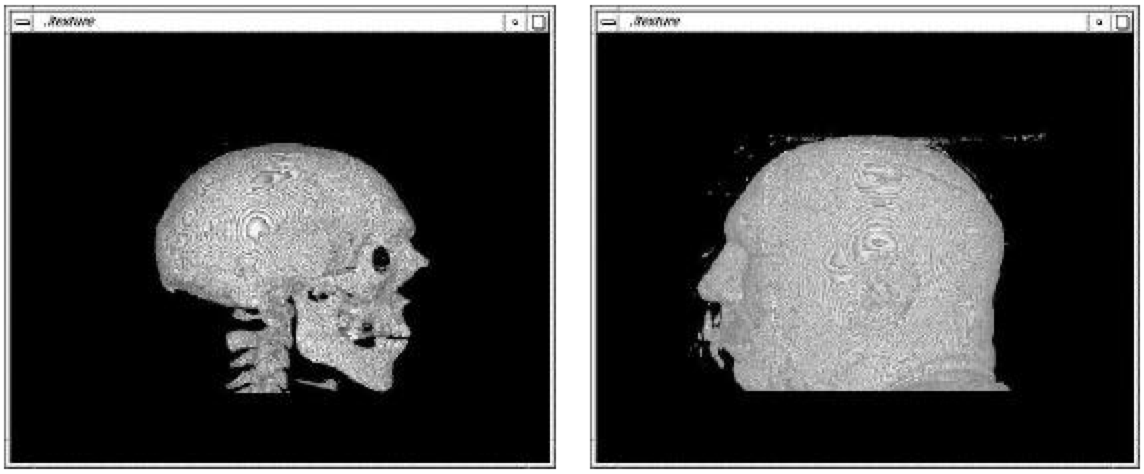


Figure 10: Results of the Binary Transparency method in C++

6. CONCLUSIONS

Using the 3-D wavelet algorithm, the original volume was reduced to 1/8th of its original size after each cycle. The image quality of the transformed coefficients was satisfactory even after several iterations. To obtain a clear view of the inner structures of the data set, a pixel-based transparency method needs to be implemented, which is currently not supported in Java3D. Pixel transparency involves setting a range of transparency values to individual pixel values. To enhance the performance of the rendering method, this application was ported to C++ and OpenGL. The image quality is sufficient to give a quick overview of the data set. The algorithm interfaces directly with the Java and C++ versions of the *Scalable Visualization Toolkits*. The 3-D wavelet transformation algorithm allows to transmit a volumetric data set progressively at different levels of detail over the Internet. Future work involves exploring techniques to enable lighting to the final reconstructed 3-D volume to produce more realistic images.

7. ACKNOWLEDGEMENTS

We thank Arthur J. Olson (The Scripps Institute, La Jolla, CA) for providing the sample data sets. We also thank David Nadeau and Jon Genetti (San Diego Supercomputer Center, SDSC) for providing additional information and the source codes of the Scalable Visualization Toolkits. This project was funded by the National Partnership for Advanced Computational Infrastructure (NPACI) under award no. 10195430 00120410. We would like to acknowledge our collaborators at the Center for Image Processing and Integrated Computing (Bernd Hamann et al., CIPIC) at UC Davis. We also like to thank the Center for Neuroscience at UC Davis for providing additional funding.

8. REFERENCES

1. Brani Vidakovic, and Peter Mueller, "Wavelets for kids", *A Tutorial Introduction*, Durham, NC, 1991.
2. Z. Lee, P. Diaz, and E. Bellon, "Manipulation of volume and graphics objects for the visualization of medical data under OpenGL", *Proceedings of the Eighth IEEE Symposium on Computer-Based Medical Systems*, pp. 89-93, Lubbock, TX, 1995.
3. M. Bailey, "Interacting with direct volume rendering", *IEEE Computer Graphics and Applications*, vol. 21, pp. 10-12, 2001.
4. Marc Levoy, "Efficient Ray Tracing of Volume Data," *ACM Transactions on Graphics*, vol. 9, pp. 245-261, ACM Press, 1990.
5. L. Ibanez, C. Hamitouche, and C. Roux, "Ray casting in the BCC grid applied to 3D medical image visualization", *Engineering in Medicine and Biology Society, Proceedings of the 2nd Annual International Conference of the IEEE,* vol. 2, pp. 548-551, Hong Kong, China, 1998.
6. William E. Lorensen, and Harvey E. Cline, "Marching Cubes: A High Resolution 3D Surface Reconstruction Algorithm", *ACM Transactions on Graphics*, vol. 21, no. 4, pp. 163-169, ACM Press, 1987.
7. M. Meissner, U. Hoffmann, and W. Strasser, "Enabling Classification and Shading for 3D Texture Mapping based Volume Rendering Using OpenGL and Extensions", *Visualization '99. Proceedings*, pp. 207-526, San Francisco, CA, 1999.
8. K. Engel, P. Hastrieter, B. Tomandl, K. Eberhardt, and T. Ertl, "Combined local and remote visualization techniques for interactive volume rendering in biomedical applications", T. Ertl, B. Hamann, and A. Varshney, *Visualization 2000. Proceedings,* pp. 447-452, 587, Salt Lake City, UT, 2000.
9. Joerg Meyer, Ragnar Borg, Bernd Hamann, Kenneth I. Joy, and Arthur J. Olson, "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*, pp. 73-76, Granlibakken Conference Center, Tahoe City, CA, 2000.
10. Michael B. Martin, and Amy E. Bell, "New Image Compression Techniques Using Multi-wavelets and Multiwavelets Packets", *IEEE Transaction on Image Processing*, vol. 10, pp. 500-510, Herdon, VA, 2001.
11. Jane Wilhelms, and Allen Van Gelder, "Multi-dimensional Trees for Controlled Volume Rendering and Compression", *Proceedings of the 1994 symposium on Volume visualization*, pp. 27-34, ACM Press, Tysons Corner, VA, 1994.
12. A. S. Lewis, and G. Knowles, "Video compression using 3D wavelet transforms", *Electronic Letters,* vol. 26, pp. 396-398, 1990.
13. I. K. Levy, and R. Wilson, "Three dimensional wavelet transform video compression", *IEEE International Conference on Multimedia Computing and Systems,* vol. 2, pp. 924-928, Florence, Italy, 1999.
14. Sagar Saladi, Pujita Pinnamaneni, and Joerg Meyer, "Texture-based 3-D Brain Imaging", *2nd IEEE International Symposium on Bioinformatics & Bioengineering*, Bethesda, MD, 2001.
15. Pujita Pinnamaneni, Sagar Saladi, and Joerg Meyer, "3-D Haar Wavelet Transformation and Texture-Based 3-D Reconstruction of Biomedical Data Sets", *Visualization, Imaging and Image Processing (VIIP 2001), The International Association of Science and Technology for Development (IASTED)*, pp. 389-394, ACTA Press, Marbella, Spain, 2001.
16. Eric J. Stollnitz, Tony D. DeRose, and David H. Salesin, "Wavelets for Computer Graphics: A Primer Part 1", *IEEE Computer Graphics and Applications*, 1995.
17. B. Cabral, N. Cam, and J. Foram, "Accelerated Volume Rendering and Tomographic Reconstruction using texture

mapping hardware", *ACM Symposium on Volume Visualization*, pp. 91-98, ACM Press, Tysons Corner, VA, 1994.

18. Sagar Saladi, and Joerg Meyer, "Texture-based 3-D Reconstruction of Biomedical Data Sets", *Tri-State Engineering Society Meeting*, Destin, FL, 2001.
19. J. D. Mulder, F. C. A. Groen, J. J. van Wijk, "Pixel masks for screen-door transparency", *Visualization '98. Proceedings*, D. Ebert, H. Rushmeier, and H. Hagen, pp. 351 – 358, 550, Research Triangle Park, NC, 1998.
20. V. Interrante, H. Fuchs, and S. M. Pizer, "Conveying the 3D shape of smoothly curving transparent surfaces via texture", *IEEE Transactions on Visualization and Computer Graphics*, vol. 3, pp. 98-117, 1997.