3-D HAAR WAVELET TRANSFORMATION AND TEXTURE-BASED 3-D RECONSTRUCTION OF BIOMEDICAL DATA SETS

PUJITA PINNAMANENI, SAGAR SALADI, JOERG MEYER
Mississippi State University
NSF Engineering Research Center, 2 Research Blvd., Starkville, MS 39759, USA
{pp2|saladi|jmeyer}@erc.msstate.edu

ABSTRACT

Enhanced biomedical image scanning technology and growing network accessibility have created a need for faster and more efficient data exchange over the Internet and in closed networks. Researchers and biologists are conducting experiments on large biomedical data sets and want to share information with other scientists to progress in their research. But the information they are dealing with are large-scale data sets that occupy gigabytes of space, which makes the storing of these data sets onto one's local hard drive very difficult. The size of these data sets also makes them difficult to transmit over currently existing network links. To overcome these difficulties, those data sets are stored in large-scale data repositories, from which they can be retrieved upon user's request. The challenge is to make the data accessible within a reasonable amount of time at a reasonable quality without losing detail or image resolution. We describe a hierarchical storage scheme based on 3-D Haar wavelets, and a fast 3-D rendering algorithm based on 2-D texture mapping, which has been integrated with the *Scalable Visualization Toolkits*, an alpha project of the National Partnership for Advanced Computational Infrastructure (NPACI).

KEY WORDS

Biomedical Data Sets, Haar Wavelet Transformation, Texture Mapping

1. INTRODUCTION

Recent improvements in CT and MRI scanning technology provide high-resolution imagery of the human body, which needs to be distributed over networks and rendered in real time. The data sets, which are created by these scanning devices, are too large to fit in core memory of current systems. Quadrupling the resolution in each dimension (e.g., from 256 to 1024) increases the size of the volume by a factor of $4^3 = 64$. The size of these data sets ranges from several hundreds of megabytes to about one hundred gigabytes.

San Diego Supercomputer Center (SDSC) maintains a large data repository (High Performance Storage System, HPSS) to store those kinds of data sets. These data sets are made accessible through our web-based Java interface to closed user groups and to researchers all over the world.

The *Scalable Visualization Toolkits* developed at SDSC support a variety of different file formats to store multi-dimensional, multi-modal, time-varying data sets in the data repository. For volumetric data sets, a file format named *vol* has been implemented, which supports regular 3-D grids. A *vol* file can be encoded in three different formats, namely *vols, volb* and *volc*. If the data set contains scalar values, it is written as a *vols* format. This format stores data as 8bit scalar values. The *volb* file format has been implemented to store RGB and RGBA (RGB-alpha, 32 bits). If no alpha value is present, the respective component is set to zero. The *volc* file format stores data as 64 bit RGB-alpha-beta values. The color component values are truncated to 10 bits for red, 12 bits for green, and 10 bits for blue. The alpha and beta components are truncated to a short each. The *vol* file format also supports chunked file format layouts in which data is arranged in chunks, which supports sub-volumes to be extracted. A chunked storage layout makes it easier to access groups of voxels within a user-defined region of interest. Our goal is to extract sub-volumes at different levels of detail and to transmit them from the server (data repository and Scalable Visualization Toolkit) to the rendering client (Java applet). NPACI's *Scalable Visualization Toolkits* are designed to handle SDSC's *vol* file formats. They can handle large-scale data sets that do not fit into a desktop machine or workstation's memory and usually cannot be rendered entirely in real time without reducing the complexity. The internal access strategies are transparent to the user. The Toolkit uses caching schemes and chunked storage layouts to improve performance.

When the data sets are transmitted over the network they are transformed into a smaller representation, which can be transmitted at higher speeds. We use the wavelet transformation to transform the data sets. A wavelet transformation converts data from the spatial domain into a localized frequency domain [1]. Each cycle is associated with a particular scale. The Haar wavelet is one of the simplest wavelet transforms and can be used to create different levels of detail. We have implemented the Haar wavelet transformation in 3-D to explore the spatial coherence between scalar values in the grid nodes in all three dimensions. This results in better compression rates on the server side and faster data transmission for volume rendering [2, 3] on the client side.

The wavelet representation is uncompressed and decoded for a given level of detail on the client side. Once the original information is reconstructed, it is rendered as a 3-D volume using Java3D. The following sections will describe the 3-D Haar wavelet method and the Java3D rendering algorithm in greater detail. Examples are provided to encourage the reader to reimplement this method and compare the performance of our Java-based implementation to other platforms. NPACI's *Scalable Visualization Toolkits* are also available in C/C++, which might make this step easier.

2. BACKGROUND

Hierarchical volume rendering of biomedical data sets is important to study user-defined regions of interest in greater detail, while the rest of the data set is provided as context information and can be rendered at a coarser level of detail. Previous volume rendering algorithms have been primarily based on OpenGL or Open Inventor. With the advent of more reliable and powerful versions of Java3D, it is now possible to develop web-based rendering clients in Java [4, 5]. Cross-section extraction can be done either on the server side [6] or on the client side [7]. 2-D texture-based methods [8] have been introduced to make use of advanced capabilities of hardware-accelerated graphics pipelines [7, 9, 10]. Our approach uses a Java3D-based texturing method, which is portable and hardware-platform independent.

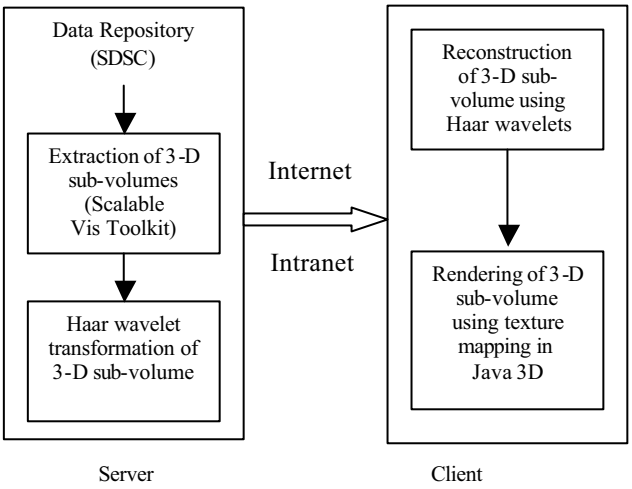

Figure 1: Scalable visualization system

Various image compression methods, e.g., RLE, LZW (lossless), and JPEG (better compression, but lossy), have been discussed in the past. Wavelet compression, which is actually a combination of a transformation in the frequency domain and compression of the detail coefficients using one of the lossless methods, has proven to be a powerful tool for 1-D signal processing and 2-D image processing [11, 12]. In our method, we apply this method to 3-D volume data, which results in better overall compression rates. Multiwavelet image compression techniques [13] and Multi-resolution Trees [14] have been applied to biomedical data sets (CT, MRI), vector data (fMRI, cryosections, oceanographic data) [15], and many other applications, including for example Synthetic Aperture Radar (SAR)[16], a remote sensing technology.

In most cases, wavelet transformations have only been discussed for the two-dimensional case. During the past decade, volume rendering has gained importance because hardware-accelerated graphics cards are readily available on desktop systems. This inspired us to use the Haar wavelet transformation technique for efficient data transformation, and more importantly, faster reconstruction of 3-D volumes on the client side. Wavelets are local in both the frequency and the spatial domain. This helps to preserve features and represent functions with discontinuities and sharp edges in a more compact way and still achieve a reasonable approximation [11]. The detail coefficients usually have very small values. Sometimes they can be neglected or discretized in order to achieve better compression rates. The high compression factor and efficient reconstruction methods for 3-D sub-volumes make wavelets an excellent tool for remote interactive rendering at multiple levels of detail. Transformation techniques using wavelets have also proved advantageous when evaluating the image quality and comparing it to the quality of a 3-D rendering of the original data set. Wavelets have proved to be faster in terms of computational complexity as compared to other transformation schemes such as the Fast Fourier Transform (FFT). The following section provides a comprehensive motivation and a step-by-step introduction for a memory-efficient 3-D wavelet transformation and subsequent level-of-detail rendering, which will be discussed in later sections.

3. IMPLEMENTATION

3.1 EXTRACTION OF 2-D CROSS-SECTIONS FROM THE DATA SETS

The *Scalable Visualization Toolkit*s are available both in a Java and in a C++ version. We have used the Java version to read the data as a sequence of 2-D cross-sections from the file (standard storage layout). The data set represents a CT scan of a human brain (*ctbrain.vols*) and consists of 512 x 512 x 231 elements. The size of the data set, which is stored as a single file, is 60,555,264 bytes (approx. 57.8 GB). The data set is composed of 231 cross-sections, each consisting of 512 x 512 pixels. The Scalable Visualization Toolkit provides a method that reads a range of elements from a data set, and returns their values in the host's byte order and word size. This method implements a decoder, which reads the data from the data set and decodes its format. The resulting byte stream is stored in an internal buffer. The data set is too large to be read entirely into

memory. Therefore, we are reading only two slices into memory at each time, which is sufficient to apply our localized 3-D wavelet scheme. As each cross-section of the *ctbrain.vols* data set consists of 512 x 512 elements, a total of 262,144 elements must be read from the data set in each cycle (Figures 2 and 3).

Figure 2: One of the 231 cross-sections of the *ctbrain.vols* data set

Figure 3: A single cross-section of the *ctbrain.vols* data set

3.2 EXTRACTING SUB-VOLUMES FROM THE DATA SETS

There are situations where it is not possible to render the entire data set at full resolution, or where the user might not be interested in the whole data set. For example, a biologist or physician might want to select a particular region of interest for his experiment or analysis. To handle such situations, sub-volumes of the data set need to be extracted [17]. The *Scalable Visualization Toolkits* implement methods to deal with a so-called chunked storage layout scheme.

One of these methods computes a one-dimensional memory index corresponding to the given *n*-dimensional grid coordinates in the data set. This index is used to read the data from the data set by using the method that was described in the previous section. The chunked file format has been used to extract sub-volumes from a human brain (*ctbrain_c32.vols*, Figure 4).

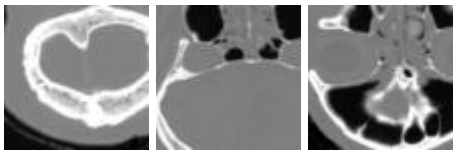

Figure 4: Cross-sections of different sub-volumes

3.3 3-D HAAR WAVELET TRANSFORMATION ALGORITHM

After the cross-sections have been extracted, the data sets are transformed using the 3-D Haar wavelet transformation. This transformation is one of the simplest and basic transformations from the spatial domain into a local frequency domain. In the discrete case, the integrals can be reduced to simple arithmetic operations. This applies to a one-dimensional signal as well as to the 2-D, 3-D, or higher-dimensional case. To give a better idea of the actual implementation of the wavelet transformation, we illustrate the procedure with a simple example [18].

Assume we have a one-dimensional image with an eight-pixel resolution, where the pixels have the following values:

7 5 3 9 3 7 5 3

By applying the Haar wavelet transformation we can represent this image in terms of a low-resolution image and a set of detail coefficients. The transformed data 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. So the above image will be represented as follows after the first cycle:

Low-resolution image:	6	6	5	4
Detail coefficients:	1	–3	–2	1

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 iterating this algorithm leads to an image that is reduced by a factor of two for each cycle.

The detail coefficients are required to reconstruct the image. Reconstruction of the original image involves adding and subtracting the detail coefficients to and from the low-resolution image data. This way the first part of the image (the low-resolution part) can be transmitted first, while the detail coefficients are added later to refine the image.

This simple 1-D scheme can be lifted to higher dimensional cases. For a 2-D wavelet transformation, the algorithm is applied in *x*-direction first, and then in *y*-direction. Similarly, in 3-D wavelet transformation the structures are defined in 3-D and the transformation algorithm is applied in *x*-, *y*- and *z*-direction successively. One *cycle* for an *n*-dimensional data set is defined as the completion of the algorithm for all *n* directions. Details about the data structures and our implementation are discussed in the following section.

As mentioned earlier, very large-scale data sets are considered for this transformation technique. Therefore the algorithm must be scalable. In order to obtain scalability, we use the file system to buffer intermediate steps for each cycle, before we proceed to the next level.

The array sizes are expressed in powers of two. More precisely, this means that the original resolution of the images is converted into the next larger power of two, and the array dimensions are adapted accordingly. For the above data set, the 3-D array defined is 512 x 512 x 256. If the buffer would be too large, it can be reduced to two

adjacent slices, which are required to do the last step of the 3-D cycle. While loading the slices into the 3-D array, care is taken to fill in the extra space allotted by zeros.

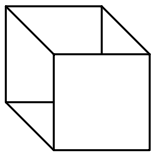

Figure 5: Original Volume

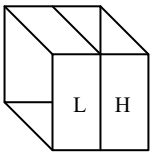

Figure 6: First run: *x*-direction

For the first cycle, the transformation algorithm is first run along the *x*-direction, row by row, for each of the 231 slices. A line buffer would be sufficient to perform this operation, but we prefer to use a 2-D buffer for faster access and more efficient file transfer from the *Scalable Visualization Toolkits*.

As shown in the image above (Figure 6), the image array is split into two halves containing the transformed data and the detail coefficients. The transformed data coefficients are low-pass filtered while the detail coefficients are high-pass filtered. After transforming the data set in *x*-direction, this 3-D array is then transformed along the *y*-direction. The resulting 3-D array is shown in Figure 7.

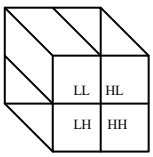

Figure 7 : Second run: *y*-direction

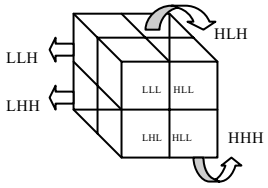

Figure 8: Third run: *z*-direction

Finally, the 3-D array is transformed along the *z*-direction and the resulting 3-D array is represented as shown in Figure 8.

The final array structure gives us a low-resolution version of the 3-D volume (the LLL segment) in one sub-octant, and the detail coefficients, which are used for the reconstruction, in the other seven sub-octants and in the sub-octants of the next higher levels. The indexing of the array is defined such that the transformed coefficients are stored in the upper-left-front corner for each successive cycle. For each cycle, the size of the volume is reduced by a factor of eight. During consecutive cycles, only the upper left portion of the transformed volume is considered, leaving the detail coefficients intact. The number of cycles $m = ld\ (max(x\_size, y\_size, z\_size))$ is limited by the size of the data set. If a cycle cannot be completed, because one dimension is smaller than the others, a null transformation can be used. In our implementation, we simply stop the cycles. Usually the limit l is chosen so that $l < m$.

3.4 3-D RECONSTRUCTION USING TEXTURE MAPPING

The rendering algorithm uses three sets of perpendicular 2-D cross-sections, which are mapped onto polygonal 2-D planes. The algorithm is implemented in Java3D. Rendering on the client side makes it possible to implement interactive features, such as rotating, scaling, and selecting a region of interest. The wavelet reconstruction algorithm is fast enough to decode the textures on the fly, and as new data streams in, it is automatically added to the texture planes. The texture planes always have the same size. If the resolution of the current cross-section is lower than the actual size of the texture plane, pixels are simply duplicated. Interpolation of pixel values would give a slightly better visual impression, but those operations are prohibited due to performance limitations. Artifacts due to limited resolutions are only temporary and are usually compensated very quickly by adding the next level of detail as soon as it has been transmitted by the server.

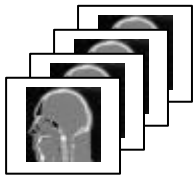

Figure 9: Texture images mapped onto a stack of polygons

A *TextureLoader* utility class in Java3D is used to load the texture images. The 2-D cross-sections are then mapped onto an aligned series of parallel polygons in back-to-front order. Interesting effects can be obtained by making the polygons semi-transparent [19]. Fast blending between an opaque view and a simulated X-ray image is possible (Figures 10 and 11).

Figure 10: Back-to-front: A 3-D view of the ear

Figure 11: Simulated X-ray view

Instead of using per-plane transparency, we can also use per-pixel transparency. Background pixels can be eliminated, keeping the rest of the slice intact, or the brain can be made transparent in order to extract and highlight the bone (see Figures 14 and 15). The transparency transfer function determines the appearance of the 3-D reconstruction.

To access and modify the RGBA pixel values of the 2-D cross-sections, Java2D *BufferedImage* and *ColorModel* classes have been used. The alpha components of the pixe ls constituting the skull are set to zero (opaque), while the alpha component of the remaining pixels is set to 255. Java provides an efficient method for a binary test if a pixel is fully transparent or fully opaque. During the rendering process, the alpha component of each pixel is checked. If the value is zero, the pixel is drawn, otherwise, the pixel is not drawn. Using this method, it was possible to achieve an additional performance gain. To implement this, we used the *RenderingAttributes* class.

4. RESULTS

We applied our 3-D Haar wavelet transformation algorithm to a CT scan of a human brain (512 x 512 x 231). The following images show a cross-section of the 3-D volume at three different levels of detail (Figure 12).

1st cycle 2nd cycle

3rd cycle 4th cycle

Figure 12: Compressed images in successive cycles

Each cycle reduces the size of the original volume by a factor of eight (2^3). By applying a run-length encoding scheme, we were able to obtain much better compression rates for our 3-D algorithm than for a conventional 1-D or 2-D algorithm. The 3-D method makes it possible to transmit a low-pass filtered sub-volume from a Unix-based server system to a Java3D rendering client over standard-bandwidth Internet and Intranet networks, and then successively transmit more data to improve the resolution and quality of the 3-D reconstruction.

1st cycle 2nd cycle

3rd cycle 4th cycle

Figure 13: Compressed image data and detail coefficients after four cycles

Remember that the data set is stored on a server with large storage capacity, while the Java3D-based rendering client runs on a desktop machine. Results of the back-to-front projection technique using Java's binary transparency test method are shown below.

Figure 14: 3-D reconstructed volume

Figure 15: 3-D volume showing the bone

5. CONCLUSIONS

3-D Haar wavelet decomposition, lossless data compression and efficient data reconstruction methods have enabled the implementation of an interactive rendering algorithm in Java3D. The image quality is satisfactory for preview images and web-based rendering. This paper has discussed the implementation of a scalable decomposition algorithm for sub-volumes and a fast 3-D reconstruction and rendering algorithm, which allows us to combine different levels of detail in a 3-D image and successively add detail information to the image without the necessity to recompute the entire volume. The Java3D implementation is platform-independent and fully scalable. Future work will involve better rendering algorithms, which will enhance the visibility of inner sections of the data by using precomputed normals and transparency, which is currently a problem in Java3D due to limitations in 3-D texturing capabilities. The algorithm will also be ported to use the C++ version of the *Scalable Visualization Toolkits* in order to overcome some of these limitations.

6. 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.

REFERENCES

[1] Ingrid Daubechies, Ten Lectures on Wavelets (Pennsylvania: Society of Industrial and Applied Mathematics, 1992).

[2] H. Mohammad Ghavamnia and D. Yang Xue, Direct Rendering of Laplacian Pyramid Compressed Volume Data, *IEEE Visualization '95 Proceedings*, Atlanta, Georgia, 1995.

[3] Günter Knittel, High-Speed Volume Rendering Using Redundant Block Compression, *IEEE Visualization '95 Proceedings*, Atlanta, Georgia, 1995.

[4] M. Bailey, Interacting with Direct Volume Rendering, *IEEE Computer Graphics and Applications*, Vol. 21, Issue 1, 2001, 10-12.

[5] M. Meissner, U. Hoffmann, and W. Strasser, Enabling Classification and Shading for 3D Texture Mapping based Volume Rendering Using OpenGL and Extensions, *IEEE Visualization '99 Proceedings*, 1999, 207-526.

[6] Jörg 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*, Granlibakken Conference Center, Tahoe City, CA, 2000, 73-76.

[7] K. Engel, P. Hastreiter, B. Tomandl, K. Eberhardt and T. Ertl, Combining Local and Remote Visualization Techniques for Interactive Volume Rendering in Medical Applications, *IEEE Visualization 2000 Proceedings*, 2000, 449-452, 587.

[8] H. Helminen, J. Alakuijala, J. Laitinen and S. Sallinen, Constant Z Line Texture Mapping in Interactive Visualization of Medical Images, *IEEE 17th Annual Conference*, Vol. 2, 1995, 1039-1040.

[9] B. Cabral, N. Cam and J. Foran, Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware, *ACM Symposium on Volume Visualization*, 1994, 91-98.

[10] James E. Fowler, John van der Zwaag, Shivaraj Tenginakai, Raghu Machiraju, and Robert J. Moorhead, Decoding of Large Terrains Using a Hardware Rendering Pipeline, *Tech. Rep., MSSU-COE-ERC-00-13*, Engineering Research Center, Mississippi State University, 2000.

[11] Brani Vidakovic and Peter Müller, Wavelets for Kids - A Tutorial Introduction, Institute of Statistics and Decision Science, Duke University, Durham, NC, 1991.

[12] Hongyang Chao and Paul Fisher, An Approach to Fast Integer Reversible Wavelet Transforms for Image Compression, *The proceedings of Guangzhou International Symposium on Computational Mathematics*, Guangzhou, P. R. China, 1997.

[13] Michael B. Martin and Amy E. Bell, New Image Compression Techniques Using Multiwavelets and Multiwavelets Packets, *IEEE Trans. Image Processing*, Vol. 10, 2001, 500-510.

[14] Jane Wilhelms and Allen Van Gelder, Multi-dimensional Trees for Controlled Volume Rendering and Compression, *ACM Symposium on Volume Visualization*, 1994, 27-34.

[15] Aaron Trott, Robert Moorhead II, and John McGinley, Wavelets Applied to Lossless Compression and Progressive Transmission of Floating Point Data in 3-D Curvilinear Grids, *IEEE Visualization '96 Proceedings*, 1996.

[16] Zhaohui Zeng and Ian G. Cumming, SAR Image data Compression using Tree–Structured Wavelet Transform, *IEEE Trans. Geosciences and Remote Sensing*, Vol. 39, No. 3, 2001, 546.

[17] P. Hastreiter, B.Tomandl, K. Eberhardt, and T. Ertl, Interactive and Intuitive Visualization of Small and Complex Vascular Structures in MR and CT, *Proceedings of the 2nd Annual International Conference of the IEEE*, Vol. 2, 1998,532-535.

[18] Eric J. Stollnitz, Tony D. DeRose, and David H. Salesin, Wavelets for Computer Graphics: A Primer Part 1, *IEEE Computer Graphics and Applications*, 1995.

[19] K. Kreeger and A. Kaufmann, Mixing Translucent Polygons with Volumes, *IEEE Visualization '99 Proceedings*, 1999, 191-525.