Texture Based 3-D Reconstruction of Biomedical Data Sets

Sagar Saladi, Joerg Meyer

Department of Computer Science, Mississippi State University
Box 9637, Mississippi State, MS 39762-9637

{sagarsv | jmeyer}@cs.msstate.edu

Abstract

Improved medical imaging technology has enabled biologists and other researchers to obtain better insight in the three-dimensional structure of the data they are dealing with. 3-D volume rendering helps to interpret data acquired from biomedical scanning devices. But the data sets obtained from these devices occupy huge amounts of space, which cannot be stored on one's local hard drive. Moreover, transmitting these data sets over the network becomes difficult. To overcome these difficulties, large-scale repositories have been established where these data sets are stored and can be accessed by researchers all over the world. This paper focuses on methods that access the data from these large-scale repositories and enable interactive rendering of the biomedical objects in 3-D on the client side. This paper also illustrates accessing sub-volumes from the data sets.

1. Introduction

Radiologists, physicians, biologists and other researchers are conducting experiments on large-scale biomedical data sets. These data sets occupy several hundreds of megabytes to about one hundred gigabytes of space, which makes them difficult to store on common desktop systems and regular workstations. San Diego Supercomputer Center (SDSC) maintains a large data repository (High Performance Storage System, HPSS) that allows researchers to make their data sets available and accessible to other researchers all over the world.

The *Scalable Visualization Toolkits* (*VisTools*), an NPACI (National Partnership for Computational Infrastructure) initiative, primarily developed at SDSC, The Scripps Institute (La Jolla, CA), UC Davis, U Texas, and Mississippi State University, support a variety of different

file formats to store all kinds of structured and unstructured meshes. The file format used for biomedical 3-D image data is the *vol* file format, which supports regular three-dimensional grids. The *vol* format can be encoded in three different formats, namely *vols*, *volb* and *volc*. If a data set contains scalar values, it is written as a *vols* file. This file format stores data as 8-bit scalar values. If a data set contains RGB or RGBA values, it is written in *volb* format (32 bits). Each byte represents a color channel. If no alpha channel is present, the alpha value is '0'. 64-bit image data is written in *volc* format. A *volc* file stores element 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 16 bits each.

The *vol* file format also supports a chunked layout scheme in which data is arranged in smaller portions, which allows for sub-volumes to be extracted. In a chunked layout, the elements around a selected element have neighboring *x*, *y* and *z* grid elements. In a non-chunked layout, the data is typically arranged in slices (*x*/*y* planes), which makes it difficult and inefficient to access data from adjacent slices, i.e., along the *z* axis.

NPACI's *Scalable Visualization Toolkits* have been developed to support analysis, filtering, compositing and rendering of very large multi-dimensional, multi-modal, time-varying data sets. These *VisTools* have been designed to handle large data sets that do not fit on a regular local hard drive. The goal of NPACI's *Interaction Environments* (*IE*) thrust is to provide infrastructure to make those large data sets available over a network to web-enabled desktop PC users.

The following sections will describe the extraction of individual cross-sections from large data sets, the extraction of sub-volumes, and a texture-based 3-D reconstruction and rendering method for large data sets using Java3D.

2. Background

Volume rendering of biomedical data sets in 3-D is important for physicians and biologists to get a detailed view of a data set from a biomedical scanning device, such as CT (computed tomography), MRI (magnetic resonance imaging), PET (positron emission tomography), and

CLSM (confocal laser-scanning microscopy). A standard method to visualize those data sets is raytracing [8]. This method is very time-consuming and inefficient for large-scale data sets. Most implementations are based on OpenGL and Open Inventor. Java was also used in volume rendering for providing web-based applications [1, 6]. Slicing of large volumetric data sets can be done locally on the client side [3] or remotely on the server side [7]. The problem is to transmit large amounts of data over a low-bandwidth network within a reasonable amount of time to allow for interactive rendering on the client side. Therefore, we favor the second approach (hierarchical data representation and sub-volume extraction on the server side, rendering on the client side). Work has also been done in slicing the biomedical data sets into 2-D cross-sections and using them in texture mapping to reconstruct a 3-D volume [2, 3].

The next section describes our approach. We extract 2-D cross-sections from large data sets that are stored on a *High Performance Storage System* (HPSS), and reconstruct a three-dimensional volume using a set of 2-D texture maps. The rendering is done on the client side. The actual algorithm has been implemented in Java3D.

3. Extraction of 2-D cross-sections from large data sets

NPACI's *Scalable Visualization Toolkits* are available both in a Java and in a C++ version. We have used the Java version to extract 2-D cross-sections from a CT scan of a human brain data set (*ctbrain.vols*). This data set consists of 512 x 231 x 512 elements. Each data element represents a scalar value. The size of this file is 60,555,264 bytes (approx. 57.8GB). This data set is composed of 512 cross-sections, each consisting of 512 x 231 pixels. The *VisToolkit* 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. It automatically recognizes the file type by identifying a magic number at the beginning of the file (*volb, volc* or *vols)*. The resulting byte stream is returned and stored in a buffer. As each cross-section of the *ctbrain.vols* data set consists of 512 x 231 elements, a total of 118,272 elements must be read from the data set in each iteration. Each cross-sections that has been extracted from the data set is written to a *Portable*

Gray Map (PGM) file.

Figure 1: A cross-section showing the ear

Figure 2: A cross-section showing a skull without nose

Figure 3: A cross-section showing a part of the nose

Figure 4: A cross-section showing the center of the nose

4. CPU time analysis for extracting 2-D cross-sections from a large data set

Reading the elements from a large data set stored in a data repository is one of the bottlenecks for interactive rendering, because it strongly affects the running time of the algorithm. The *Scalable Visualization Toolkit* uses an internal caching scheme, which is transparent to the user. The number of bytes read at a time does not necessarily coincide with the page or cache size, or the interleave factor of the hard drive. By changing the number of bytes read at one time

from the *vols* file, we tried to find an optimal run length for the *VisToolkit* and the given hardware (Sun Ultra Sparc 10). If those numbers match, we can expect a performance gain. Therefore, the total CPU time taken to read the entire data set from the data repository has been analyzed. The results are shown in the table below.

Total number of bytes in the *ctbrain.vols* data set = 60,555,264

"*x*" = 512 elements

"*y*" = 231 elements

"*z*" = 512 elements

Sno	Number of bytes in an element	Total no. of cross-sections to be read	No. of bytes per each cross-section (bytes)	CPU time (ms)
1	1	512	118,272	8633.37
2	2	256	236,544	9156.38
3	4	128	473,088	7685.44
4	8	64	946,176	7799.72
5	16	32	1,892,352	7346.91
6	32	16	3,784,704	7039.42
7	64	8	7,569,408	7093.85
8	128	4	15,138,816	7476.08
9	256	2	30,277,632	9287.81

Table 1: Results obtained with the application of our algorithm on the CT scan of a human brain data set. This table shows the CPU time required to access the entire data set for nine different executions of the algorithm.

The results indicate that a run length of 32 bytes, which is a power of two, yields optimal performance for the given architecture. Similar experiments have been conducted on other hardware platforms with comparable results.

5. Extracting sub volumes from a data set

There are situations where the user might not be interested in the whole data set. For example, a biologist might be interested in viewing only a certain part of the data set in greater detail for his experiment. To handle such situations, sub-volumes of the data set need to be extracted [4]. The *Scalable Visualization Toolkit* implements a method for a chunked storage layout, which can be used to extract sub-volumes of the dataset. This method computes and returns a one-dimensional memory index corresponding to the given *n*-dimensional grid coordinates in the chunked file format data set. By specifying the required sub-volume's *n*-dimensional grid coordinates, a corresponding one-dimensional memory index is obtained. These one-dimensional memory indices are used to read the data from the file by using the same method that was used to extract a cross-section of the data set. This allows us to extract cross-sections from sub-volumes in a more efficient way. We used a file that has been stored in chunked format to extract sub-volumes. Our sample file is again a CT scan of a human brain (*ctbrain_c32.vols*).

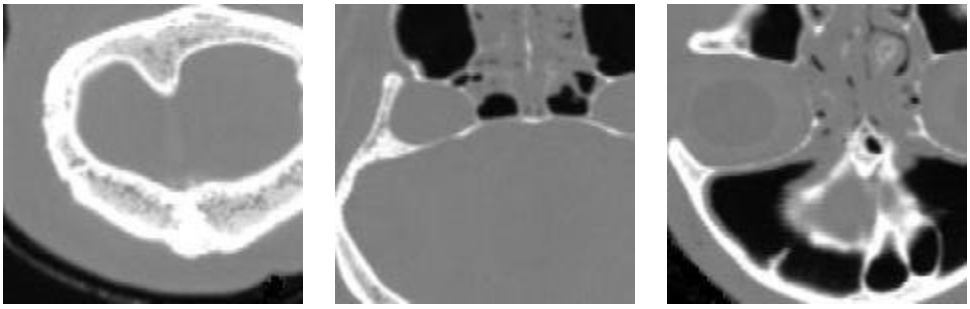

Figure 5: Cross-sections of different sub-volumes

6. 3-D Reconstruction using Texture mapping

Texture mapping is a technique to add visual richness to a scene. Creating a polygon and mapping a texture image onto it by specifying the texture image location and setting texturing attributes achieve texturing in Java3D. The 2-D cross-sections that have been extracted from the data set using the *Scalable Visualization Toolkit* are used to reconstruct a 3-D volume using texture mapping in Java3D. These cross-sections are not the same as the original slices,

because the data set can be sliced in arbitrary direction. A *TextureLoader* utility class in Java3D is used to load the texture images. Since the Java3D TextureLoader class supports the GIF file format, all 2-D cross-sections that were extracted from the data set are converted from PGM to GIF format. Once the 2-D cross-sections are extracted, they are then mapped onto an aligned series of parallel polygons in back-to-front order. 2-D texture mapping in Java3D has been used to achieve this. All the polygons are drawn as parallel planes, and the 3-D texture coordinates are chosen accordingly.

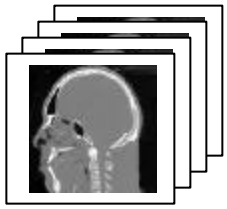

Figure 6: Texture images mapped onto a stack of polygons

Since the cross-sections are mapped one in front of the other, the first cross-section would be visible and the remaining cross-sections would be occluded. To make the remaining cross-sections visible, transparency values are assigned to all cross-sections that are used in texture mapping. Java3D supports two modes of transparency: a scalar transparency value can be applied to the texture as a whole, or a binary transparency value can be assigned to each pixel. The resulting semi-transparent [5] images of the data set give insight to interior structures. Following are the results that are obtained by using the per-plane transparency (Figures 7 and 8).

Figure 7: Back-to-front: A 3-D view of the ear

Figure 8: Simulated X-ray view

Instead of setting transparency to an entire cross-section, we can eliminate the background pixels and keep the rest of the slice intact. All the 2-D cross-sections that are extracted from the data set contain background pixels. Removing all the background pixels from the 2-D cross-sections and using them in texture mapping creates a 3-D volume. To access and modify the RGBA pixel values of the 2-D cross-sections, Java2D *BufferedImage* and *ColorModel* classes have been used. The alpha component of the pixels constituting the skull is set to zero, while the alpha component of the remaining pixels is set to 255. The regions are distinguished by thresholding. During the rendering process, the alpha component of each pixel is checked. If the value is zero, then the pixel is drawn, otherwise the pixel is not drawn. To implement this, we used the *RenderingAttributes* class.

7. Results

The data set we worked on is a CT scan of a human brain. Our 3-D reconstruction algorithm was applied to different sets of 2-D cross sections of the data set. Results obtained when we applied our algorithm to the data set are shown below. The artifacts on the 3-D reconstructed image near the mouth are a part of the data set (Figures 9 and 10).

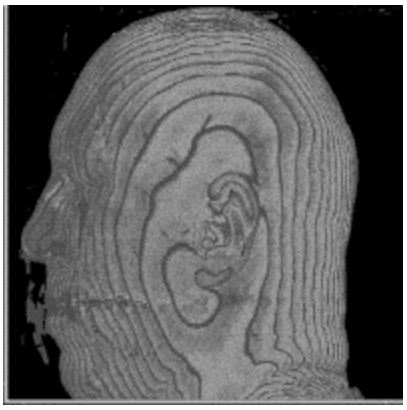

Figure 9: 3-D reconstructed volume

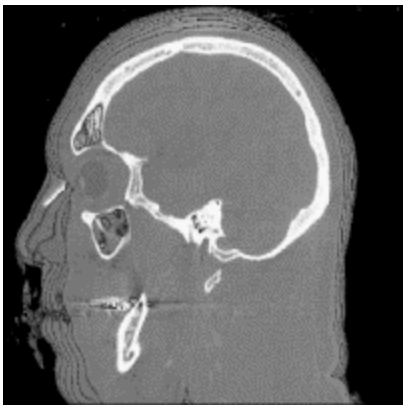

Figure 10: 3-D volume showing the bone

8. Conclusions

This paper has presented methods to access large data sets, extract sub-volumes from large data sets, and to reconstruct a 3-D volume from a series of cross-sections. The 3-D volume gives a clear view of the data set, and the extracted sub-volumes provide a detailed view of a selected region of the data set. Future work will involve enhancing the visibility of the inner sections of the data using transparency, which is currently a problem in Java3D due to limitations in texture mapping.

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] M. Bailey, "Interacting with Direct Volume Rendering," IEEE Computer Graphics and Applications, Vol. 21, Issue 1, pp. 10-12, February 2001.

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

[3] 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, pp. 449-452, 587, October 2000.

[4] P. Hastreiter, B.Tomandl, K. Eberhardt, and T. Ertl, "Interactive and Intuitive Visualization of Small and Complex Vascular Structures in MR and CT," Engineering in Medicine and Biology Society, Proceedings of the 2nd Annual International Conference of the IEEE, Vol. 2, pp. 532-535, November 1998.

[5] K. Kreeger and A. Kaufmann, “Mixing Translucent Polygons with Volumes,” IEEE Visualization ’99, Proceedings, pp. 191-525, October 1999.

[6] 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, pp. 207-526, October 1999.

[7] Meyer, Joerg, 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, pp. 73-76, October 15 – 17, 2000.

[8] Marc Levoy, “Efficient Ray Tracing of Volume Data,” ACM Transactions on Graphics, Vol. 9, No. 3, pp. 245-261, July 1990.