Network-based Rendering Techniques for Large-scale Volume Data Sets

Joerg Meyer[1], Ragnar Borg[2], Bernd Hamann[2], Kenneth I. Joy[2], and Arthur J. Olson[3]

[1] NSF-MSU Engineering Research Center (ERC), Department of Computer Science, Mississippi State University, 2 Research Blvd., Starkville, MS 39762-9627, jmeyer@cs.msstate.edu
[2] Center for Image Processing and Integrated Computing (CIPIC), Department of Computer Science, University of California, 1 Shields Avenue, Davis, CA 95616-8562, Ragnar.Borg@proxycom.no, {hamann, joy}@cs.ucdavis.edu
[3] The Scripps Research Institute (TSRI), 10550 North Torrey Pines Road, La Jolla, CA 92037, olson@scripps.edu

Summary. Large biomedical volumetric data sets are usually stored as file sets, where the files represent a family of cross sections. Interactive rendering of large data sets requires fast access to user-defined parts of the data, because it is virtually impossible to render an entire data set of such an enormous size (several gigabytes) at full resolution, and to transfer such data upon request over the Internet in a reasonable amount of time. Therefore, hierarchical rendering techniques have been introduced to render a region of interest at a relatively higher resolution. Regions rendered at coarser resolutions are provided as context information. We present a dynamic subdivision scheme that incorporates space-subdivision and wavelet compression.

1 Introduction

Real-color volume data sets can be obtained by taking photographs or scanning cross sections of objects. These objects are typically in a frozen state (cryosections). These techniques produce high-resolution image data in real color. The resolution is only limited by the camera or the imaging device, and not so much by principal limitations of the scanning device, because there is no complex matrix transformation required to obtain 2D image data, as it is the case for computed tomography (CT) or magnetic resonance imaging (MRI). Therefore, real-color volume data sets tend to be much larger than CT or MRI data.

A typical setup is a client-server architecture, where a large-scale data set is stored on a powerful server, and the rendering is performed on the client side. In order to make a data set available on a visualization server and transmit data progressively to a rendering client, we need to compactify the data set and break it down into smaller "bricks". The order of transmission and the size or resolution of the bricks is determined and driven by the client application. Our system uses a Windows NT-based server system which is

both data repository and content provider for shared rendering applications. The NT system is connected to a Unix file system from where it accesses the data. The client accesses the server via a web-based interface.

The client selects a data set and sends a request to the server. The server analyzes and interprets the request and returns a customized Java applet together with an appropriate representation of the data set. The Java applet is optimized for a specific rendering task. This means that the rendering algorithm is tailored to a particular problem set. This keeps the applet small and avoids additional overhead and considering different cases. The initial data set is also small. It is refined later upon additional requests by the client. Bricks of different sizes and different resolutions may be requested from the server. We present a method that combines dynamic space-subdivision algorithms, such as adaptive octrees for volumes, wavelet-based data representation, and progressive data transmission for hierarchically stored volume data sets [1], [8], [10].

2 Indexing scheme

Previous work on space-subdivision [4], [5], [11] has shown that octrees provide an efficient method to store large volume data sets, as long as the depth of the octree is limited. Otherwise the data structures become so complex that tree traversal causes additional overhead [9]. Wavelet compression has been proven as an efficient method to transform a data set into a multiresolution representation. Unfortunately, it is difficult to extract sub-volumes from a compressed data set. We present a technique which combines both methods in order to optimize performance.

A web-based user interface with local rendering capability on the client side [7] requires hierarchical data representation on the server site. The server transmits a coarse overview representation of the entire data set (context information), which allows the user to select a region of interest. When this region is specified, the client application sends a request to the server, which responds by transmitting sub-volumes of the specified region at increasingly higher resolutions. The client progressively refines the image for the specified region. A prototype implementation is described in [7].

Biomedical imaging data are usually structured as sets of files, which represent a series of 2-D cross sections. By arranging all slices in a linear array, we obtain a 3-D volume. Unfortunately, when accessing the data, in most cases we do not make use of the implicit coherency across single slices. This coherency is only useful for extraction of cross sections perpendicular to the scanning direction, i.e., within a single image plane. Instead, in most cases we need brick-like coherency within sub-volumes. Therefore, we present a new data structure, which uses a combination of delimited octree space-subdivision and wavelet compression techniques to achieve better performance.

In this article, we present an efficient indexing scheme, an adaptive data reduction method, and an efficient compression scheme. All techniques are based on integer arithmetic and are optimized for speed. Binary bit operations allow for memory-efficient storage and access.

We use a standard file system (Unix or FAT32) to store our derived data structures, and we use file names as keys to the database. This allows us to avoid additional overhead, which is typically caused by inserting additional access layers between the application and the underlying storage system. We found that this method provides the fastest method to access data. Our indexing scheme in conjunction with the underlying file system provides the database system (repository) for the server application, which reads the data from the repository and sends it to a remote rendering client upon request. Initially, a low-resolution representation is requested from the repository and rendered on the client side. This coarse representation provides context cues and sufficient information for initial navigation. After a user has specified a sub-volume or region of interest, the client application sends a new request to the server to retrieve a sub-volume at a higher level of detail. When using the data structures described below, this updating procedure typically requires considerably less time compared to the single-slice representation, because a smaller number of files needs to be accessed. The initial step, which requires reading the initial section of every file, i.e., all bricks, can be accelerated by storing an additional file that contains a reduced version of the entire data set.

3 Storage scheme

The file size f for storing the leaves of the octree structure should be a multiple n of the minimum page size p of the file system. The value of p is typically defined as a system constant in `/usr/include/sys/param.h` on Unix systems. The value of n depends on the wavelet compression method described below. If the lowest resolution of the sub-volume requires b bytes, the next level requires a total of $8 \cdot b$ bytes (worst case, uncompressed), and so forth.

We assume that we have a recursion depth of r for the wavelet representation, leading to $8^r \cdot b$ bytes that must fit in f. This implies that

$$f = n \cdot p \geq 8^r \cdot b. \tag{1}$$

Both r and b are user-defined constants. Typical values are $b = 512$, which corresponds to an $8 \times 8 \times 8$ sub-volume, and $r = 3$, which provides four levels of detail over a range between $8^0 \cdot 512 = 512$ and $8^3 \cdot 512 = 262{,}144$ data elements, which is a range of more than 2.7 orders of magnitude.

For optimal performance, and in order to avoid gaps in the allocated files, we can assume that

$$n \cdot p = 8^r \cdot b, \tag{2}$$

thus

$$n = 8^r \cdot \frac{b}{p}. \tag{3}$$

The enormous size of the data sets (see Section 4) requires that the data is subdivided into smaller chunks that can be loaded into memory within a reasonable amount of time [2], [6]. Since we are extracting sub-volumes, it seems natural to break the data up into smaller bricks. This can be done recursively using an octree method [4], [5], [11]. Each octant is subdivided until we reach an empty region that does not need to be subdivided any further, or until we hit the file size limit f, which means that the current leaf fits into a file of the given size.

Each leaf contains a part of the original data set at full resolution. Memory space is reduced by skipping empty regions. Typically, the size of the data set shrinks to about 20% of the original size (see Section 5).

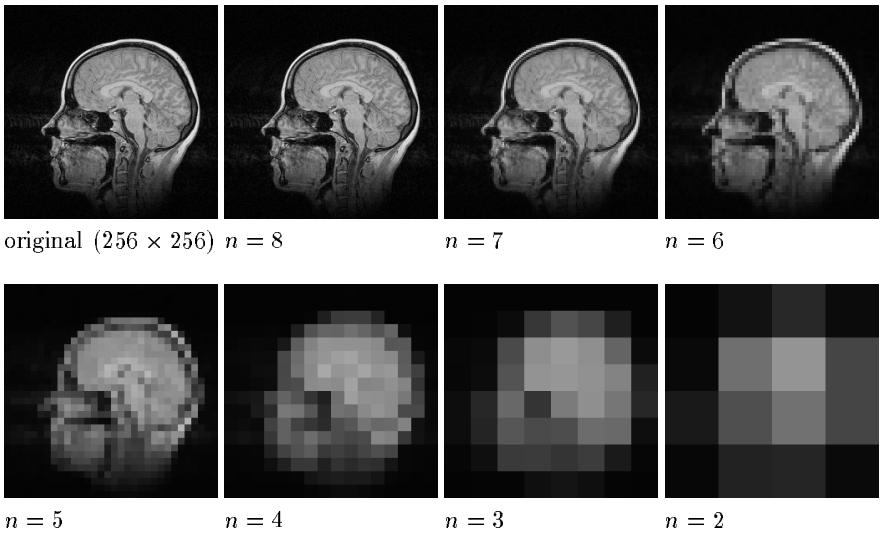

Fig. 1. Haar wavelet compression scheme (2-D case)

Progressive data transmission and extraction of a region of interest requires to access the data set in a hierarchical fashion. Therefore, it is useful to convert the leaves into a multiresolution representation. This representation must be chosen in a way that the reconstruction can be performed most efficiently with minimal computational effort [12]. Haar wavelets satisfy these requirements. They also have the advantage that they can be implemented easily with integer arithmetic [13]. The lowest resolution (Figure 1, lower-right image) is stored at the beginning of the file, thus avoiding long seek times.

For the wavelet representation, we associate each sub-volume with a vector space V that consists of a set of piecewise linear functions. The space V^0 is associated with a constant function that is defined over the domain $[0, 1)$ and describes a single pixel. The space V^i consists of 2^i intervals, with a constant function defined on each of these intervals. All vector spaces are subsets of each other, i.e.,

$$V^i \subset V^{i+1}, i \in \mathbf{N}_0. \tag{4}$$

We choose the following scaling functions as the basis functions for V^i:

$$\begin{aligned} &\phi_{i,j}(x) = \phi(2^i x - j), \;\; j \in \{0, ..., 2^i - 1\}, \text{where} \\ &\phi(x) = \begin{cases} 1, & \text{if } 0 \leq x < 1 \\ 0, & \text{otherwise.} \end{cases} \end{aligned} \tag{5}$$

Figure 2 illustrates the scaling functions.

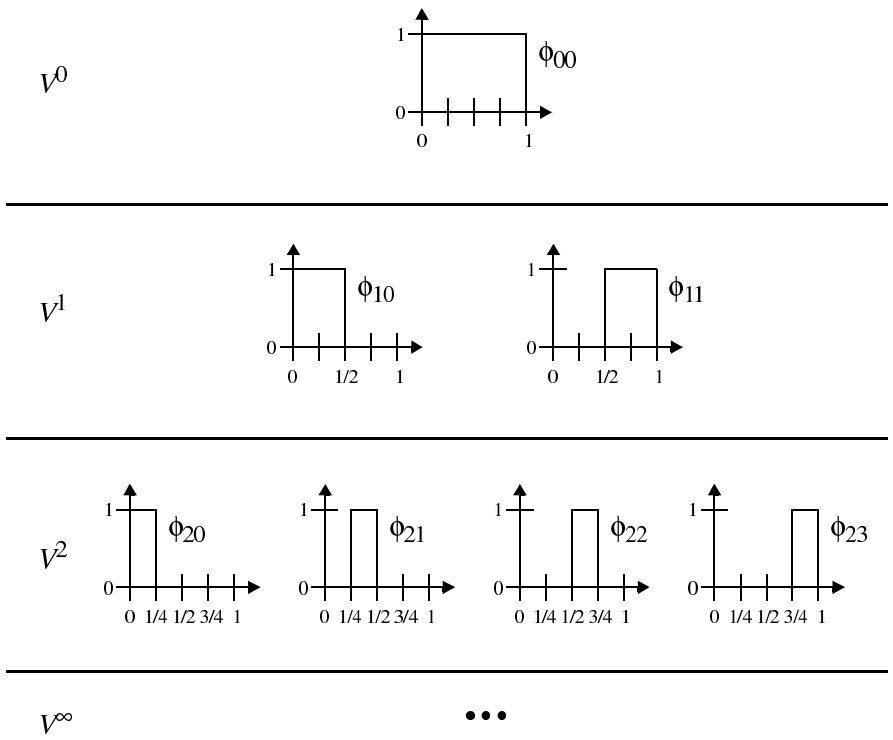

Fig. 2. Wavelet transformation: scaling functions

We define another vector space W^i that comprises all functions of V^{i+1}, and which is orthogonal to all functions in V^i. These basis functions, which span W^i, are the *Haar wavelets*, defined as

$$\psi_{i,j}(x) = \psi(2^i x - j),\ j \in \{0, ..., 2^i - 1\}, \text{where}$$

$$\psi(x) = \begin{cases} 1, & \text{if } 0 \leq x < \frac{1}{2} \\ -1, & \text{if } \frac{1}{2} \leq x < 1 \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

Figure 3 illustrates these basis functions.

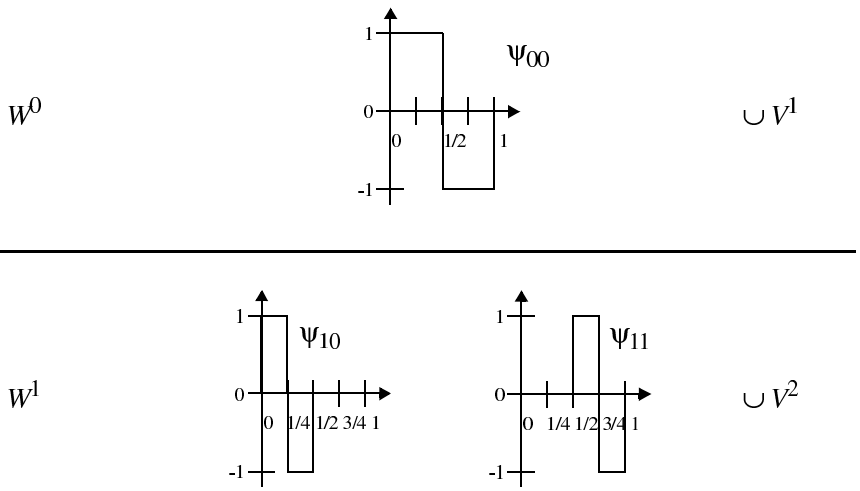

Fig. 3. Haar wavelets

A discrete signal in V^{i+1} can be represented as a linear combination of the basis functions of V^0 and W^0 ... W^i. We can represent an image I as

$$I(x) = \sum_{i=1}^{n} l_i \cdot \phi_{\frac{n}{2}-1,i-1}(x), \tag{7}$$

where $l_i, i \in \mathbf{N}_0$, is the image data. After the first transformation, we obtain

$$I(x) = \sum_{i=1}^{n/2} l'_i \cdot \phi_{\frac{n}{2}-2,i-1}(x) + \sum_{i=1}^{n/2} c'_i \cdot \psi_{\frac{n}{2}-2,i-1}(x), \tag{8}$$

where the first sum represents the averaged image data, which can also be described as a low-pass filtered image, and the second sum comprises the detail coefficients, representing a high-pass filtered image, or the difference between one of two adjacent pixels and their average value. A detail coefficient describes the symmetric error or deviation of the averaged value from one of the two original values.

It is easy to lift this scheme to the 3-D case. A simple solution would be an enumeration of all grid points, e.g., row-by-row and slice-by-slice, in a linear chain, so that one can still apply the 1-D algorithm given above (standard decomposition). This, of course, would reduce the data set only by a factor of two, instead of $2^3 = 8$. Thus, we apply the algorithm alternatingly to each dimension. Figure 4 shows the method for the 2-D case [12].

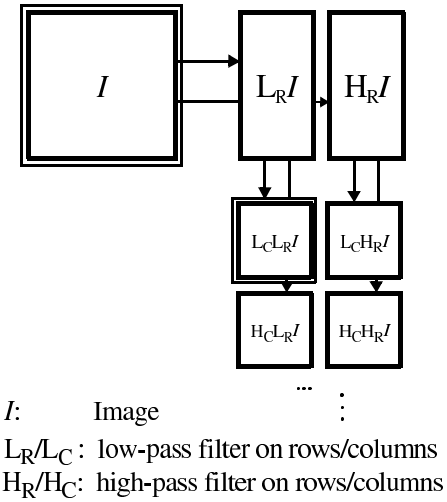

Fig. 4. Wavelet compression scheme

First, all rows of the original image I are decomposed into a low-pass-filtered image $L_R I$ (reduced image) and the high-pass-filtered components $H_R I$ (detail coefficients). For the next transformation, the algorithm is applied to the columns, which results in $L_C L_R I$, $H_C L_R I$, $L_C H_R I$, and $H_C H_R I$ (prefix notation). The same technique can be used in three dimensions.

After each cycle, we end up with a reduced image in the upper-left corner. Subsequently, the algorithm is only applied to this quadrant (see Figure 5). The algorithm terminates when the size of this quadrant is one unit length in each dimension.

This method of alternating between dimensions is known as non-standard decomposition (Figure 1 shows the $L_C L_R$ components for an MRI scan at different levels of detail).

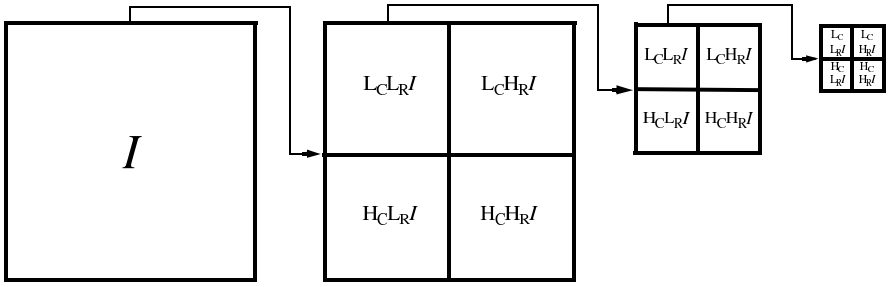

Fig. 5. Wavelet compression: memory management

A very useful property of the wavelet scheme is the fact that even for lossless compression a volume converted into the wavelet representation requires the exact same amount of memory as the original representation. Since many coefficients are relatively small, the number of different discrete values is also small, provided we use integer arithmetic. Extremely small values can be neglected to obtain even better compression rates (lossy compression). We use a simple run-length encoding (RLE) scheme that turns out to be efficient, especially for small brick sizes b. It also allows for easy decoding. The space requirement (lossless compression) is the same for all subsequent wavelet recursions, i.e., for all levels of detail. The wavelet algorithm terminates when it reaches a pre-defined minimum sub-volume size b. The lower bound is the size of a single voxel.

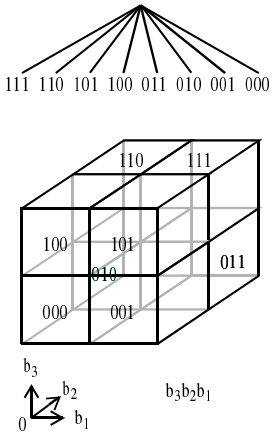

Fig. 6. Numbering scheme

Each octant can be described by a number [3]. We use the following numbering scheme (see Figure 6): A leaf is uniquely characterized by the octree recursion depth and the octree path. We limit the recursion depth to eight, which allows us to encode the depth in three bits. In order to store the path, we need three bits per recursion step, which gives us 24 bits. Four bits are spent to encode the depth of the wavelet recursion. The remaining bit is a flag that indicates whether the file is empty or not. This prevents us from opening and attempting to read the file and accelerates the computation. The total number of bits is 32 (double word).

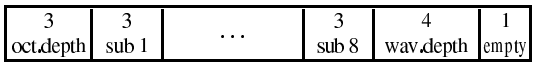

Fig. 7. Tree encoding

Each bit group can be easily converted to an ASCII character by using binary arithmetic, e.g., $(OCT\_DEPTH >> 29) \mid 0x30$ ('C' notation) would encode the octree depth as an ASCII digit. By appending these characters we can generate a unique file name for each leaf.

In order to retrieve a sub-volume, we have to find the file(s) in which the corresponding parts are stored. We start with the lower-left-front corner and identify the subvoxel by recursive binary subdivision of the bounding box for each direction. Each decision yields one bit of the sub-volume path information. We convert these bits to ASCII characters, using the same macros as above. The first file we are looking for is 7xxxxxxx??, where the 'x's describe the path, and '?' is a wildcard. If this file does not exist, we keep looking for 6xxxxxxx???, and so forth, until we find an existing leaf. If the file name indicates that the file is empty (last digit), we can skip the file. The file name also indicates how many levels of detail we have available for a particular leaf. This allows us to scale the rendering algorithm. In order to retrieve the rest of the sub-volume, we must repeat this procedure for the neighboring leaves. The number of iterations depends on the recusion depth and therefore on the size of the leaves found. The algorithm terminates when all files have been retrieved and the sub-volume is complete.

4 Results

Our test application focusses on biomedical imaging. We have designed a prototype to support 3-D visualization of a human brain, which allows us to study details by moving tools, such as an arbitrary cutting plane and different-shaped lenses, across the data set. The various data sets are typically between 20 MB and 76 GB in size, which makes them too large to transfer over the Internet in real time. The rendering client operates independently

from the size of the data set and requests only as much data as can be displayed and handled by the Java applet.

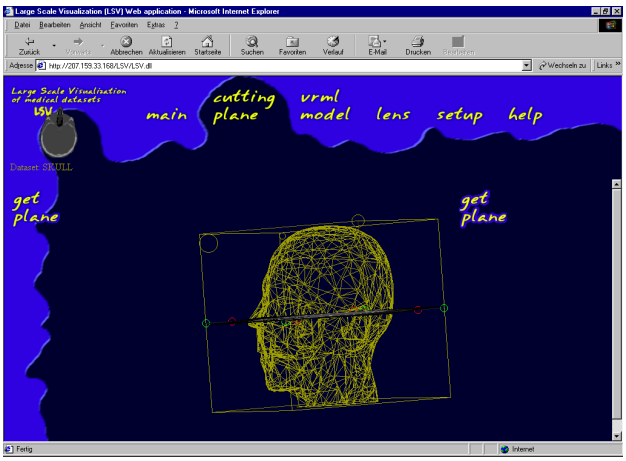

Fig. 8. Prototype application

The web-based user interface combines HTML-form-driven server requests with customized Java applets, which are transmitted by the server to accomplish a particular rendering task.

Our prototype application (Figure 8) features 2-D/3-D preview capability; interactive cutting planes (in a 3-D rendering, with hierarchical isosurface models to provide context information); and a lens paradigm to examine a particular region of interest (variable magnification and lens shape, interactively modifiable region of interest). Complex scenes can be pre-computed on the server side and transmitted as a VRML2 file to the client so that the client can render the scene, and the user can interact with it in real time.

5 Comparison

Our data structure uses considerably less memory than the original data set, even if we choose lossless compression. In our experiments we found that we can save between 62% and 85% of memory for storing a typical biomedical data set (about 14 MB). By selecting appropriate thresholds for the wavelet compression algorithm, we can switch between lossless compression and extremely high compression rates. Computing time is balanced by choosing an appropriate file size (see Section 3).

One of the advantages of our approach is the fact that the computing time does not depend so much on the resolution of the sub-volume, but merely

on the size of the sub-volume, i.e., the region of interest. This is due to the fact that the higher-resolution versions (detail coefficients in conjunction with the lower-resolution versions) can be retrieved in almost the same time from disk as the lower-resolution version alone. All levels of detail are stored in the same file, and the content of several files ("bricks"), which make up the sub-volume, usually fits into main memory. Since seek time is much higher than read time for conventional harddisks, the total time for data retrieval primarily depends on the size of the sub-volume, i.e., the number of files that need to be accessed, and not so much on the level of detail.

Algorithm	Octree				BSP tree	
	level 1		level 2			
Data type	MRI	CT	MRI	CT	MRI	CT
Pre-processing	56	63	98	97	156	159
Depth	4	4	5	5	12	12
Memory	5.412.610 / 14.548.992	3.996.526 / 14.811.136	3.831.488 / 14.548.992	2.358.442 / 14.811.136	3.299.516 / 14.548.992	2.137.326 / 14.811.136

Fig. 9. Space-subdivision algorithm

Figure 9 shows the reduction in the amount of memory required to store a large data set when we use an octree at two different levels. The column on the right represents the original data set. The wavelet decomposition takes about 0.07 sec for a 64^3 data set, and 68 sec for a 1024^3 data set. Reconstruction can be done more efficiently and usually requires about 30% of the time (measurements based on an R12000 processor). For the above data we assume lossless wavelet decomposition. RLE or other (lossy) compression/decompression algorithms take an additional amount of time, but this is negligible compared to data transmission time.

6 Conclusions

We have presented an efficient numbering scheme and access method for hierarchical storage of sub-volumes on a regular file system. This method allows us to access a region of interest as a set of bricks at various levels of detail. The simplicity of the method makes it easy to implement. The algorithm is scalable by increasing word length and file name length. Future work will address better wavelet compression schemes, lossy compression techniques, and the integration of time-varying data sets.

We are currently working on the integration, adaptation and evaluation of these tools in the National Partnership for Advanced Computational Infrastructure (NPACI) framework. Future research efforts will include integration of San Diego Supercomputer Center's High-performance Storage System (HPSS) as a data repository to retrieve large-scale data sets, and accessing the data via NPACI's Scalable Visualization Toolkits (also known as VisTools).

7 Acknowledgements

Data sets are courtesy of Arthur W. Toga, UCLA School of Medicine. We thank Kwan-Liu Ma, Ikuko Takanashi, Eric B. Lum, Eric C. LaMar, UC Davis; Elke Moritz, Mississippi State University, and Edward G. Jones, Neuroscience Center, UC Davis.

This work was supported by the U.S. Army Engineer Research and Development Center (DACA42-00-C-0039); the Army Research Office under contract ARO 36598-MA-RIP; the National Science Foundation under contracts ACI 9624034 (CAREER Award), through the Large Scientific and Software Data Set Visualization (LSSDSV) program under contract ACI 9982251, and through the National Partnership for Advanced Computational Infrastructure (NPACI, NSF SPA#00120410), the Office of Naval Research under contract N00014-97-1-0222; the NASA Ames Research Center through an NRA award under contract NAG2-1216; the Lawrence Livermore National Laboratory under ASCI ASAP Level-2 Memorandum Agreement B347878 and under Memorandum Agreement B503159; the Lawrence Berkeley National Laboratory; the Los Alamos National Laboratory; and the North Atlantic Treaty Organization (NATO) under contract CRG.971628. We also acknowledge the support of ALSTOM Schilling Robotics and SGI. We thank the members of the Visualization and Graphics Research Group at the Center for Image Processing and Integrated Computing (CIPIC) at the University of California, Davis.

References

1. Bertram, M., Duchaineau, M. A., Hamann, B. and Joy, K. I. (2000), Bicubic subdivision-surface wavelets for large-scale isosurface representation and visualization, in: Ertl, T., Hamann, B. and Varshney, A., eds., *Visualization 2000*, IEEE Computer Society Press, Los Alamitos, California, 389–396 (presented at: "Visualization 2000," Salt Lake City, Utah).
2. Heiming, Carsten (1998) Raumunterteilung von Volumendaten. Thesis, Department of Computer Science, University of Kaiserslautern, Germany.
3. Hunter, G. M.; Steiglitz, K. (1979) Operations on Images Using Quad Trees. IEEE Trans. Pattern Anal. Mach. Intell., 1(2), 145–154.
4. Jackins, C.; Tanimoto, S. L. (1980) Oct–Trees and Their Use in Representing Three–Dimensional Objects. CGIP, 14(3), 249–270.

5. Meagher, D. (1980) Octree Encoding: A New Technique for the Representation, Manipulation, and Display of Arbitrary 3–D Objects by Computer. Technical Report IPL-TR-80-111, Image Processing Laboratory, Rensselaer Polytechnic Institute, Troy, NY.
6. Meyer, Joerg (1999) Interactive Visualization of Medical and Biological Data Sets. Ph. D. thesis; Shaker Verlag, Germany.
7. Meyer, Joerg, Borg, Ragnar, Hamann, Bernd (2000) VR–based Rendering Techniques for Time–Critical Applications. Proceedings of Scientific Visualization 2000, Schloss Dagstuhl, Germany.
8. Pinnamaneni, Pujita, Saladi, Sagar, Meyer, Joerg (2001) 3–D Haar Wavelet Transformation and Texture–Based 3–D Reconstruction of Biomedical Data Sets, in: Hamza, M. H., ed., Proceedings of *Visualization, Imaging and Image Processing (VIIP 2001)*, ACTA Press, 389–394 (presented at: "Visualization, Imaging and Image Processing," The International Association of Science and Technology for Development (IASTED), Marbella, Spain).
9. Pinskiy, Dmitriy V., Meyer, Joerg, Hamann, Bernd, Joy, Kenneth I., Brugger, Eric, Duchaineau, Mark A. (1999) A Hierarchical Error–controlled Octree Data Structure for Large–scale Visualization. ACM Crossroads, Data Compression, Spring 2000 – 6.3, 26–31.
10. Pinskiy, D. V., Brugger, E. S., Childs, H. R. and Hamann, B. (2001) An octree-based multiresolution approach supporting interactive rendering of very large volume data sets, in: Arabnia, H. R., Erbacher, R. F., He, X., Knight, C., Kovalerchuk, B., Lee, M. M.-O., Mun, Y., Sarfraz, M., Schwing, J. and Tabrizi, M. H. N., eds., Proceedings of *The 2001 International Conference on Imaging Science, Systems, and Technology (CISST 2001)*, Volume 1, Computer Science Research, Education, and Applications Press (CSREA), Athens, Georgia, 16–22 (presented at: "The 2001 International Conference on Imaging Science, Systems, and Technology," Las Vegas, Nevada).
11. Reddy, D.; Rubin, S. (1978) Representation of Three–Dimensional Objects. CMU-CS-78-113, Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA.
12. Schneider, Timna Esther (1997) Multiresolution–Darstellung von 2D–Schichtdaten in der medizinischen Bildverarbeitung. Thesis; Department of Computer Science, University of Kaiserslautern, Germany.
13. Stollnitz, Eric J., DeRose, Anthony D., Salesin, David H. (1996) Wavelets for Computer Graphics: Theory and Applications. Morgan Kaufmann Publishers.