# VR-based Rendering Techniques for Large-scale Biomedical Data Sets

Joerg Meyer
NSF Engineering Research Center (ERC) - Department of Computer Science
Mississippi State University - Box 9627 - Mississippi State, MS 39762-9627
jmeyer@cs.msstate.edu

Ragnar Borg, Bernd Hamann, Kenneth I. Joy
Center for Image Processing and Integrated Computing (CIPIC)
Department of Computer Science - University of California - One Shields Ave. - Davis, CA 95616-8562
Ragnar.Borg@proxycom.no, hamann@cs.ucdavis.edu, joy@cs.ucdavis.edu

Arthur J. Olson
The Scripps Research Institute - 10550 North Torrey Pines Road - La Jolla, CA 92037
olson@scripps.edu

## Abstract

*VR-based rendering of large-scale data sets is typically limited by timing and complexity constraints of the rendering engine. Decentralized rendering, such as accessing a large data repository over a network and rendering the image on the client side, causes problems due to the limited bandwidth of existing networks. We present a combination of octree space subdivision and wavelet compression techniques to store large volumetric data sets in a hierarchical fashion, and we incorporate a unique numbering scheme, so that subvolumes (regions-of-interest) can be extracted efficiently at different levels of resolution.*

**Keywords:** large-scale visualization, biomedical imaging, remote visualization, multiresolution, octree, wavelet, virtual reality

## 1. Introduction

We present a framework for distributed hierarchical rendering of large-scale data sets that addresses two problems at the same time: (i) limited network bandwidth and (ii) limited rendering resources. Our goal is to compactify the data set and to break it down into smaller bricks, while making effective use of multiresolution techniques. Our system uses a Windows NT-based server system which is both data repository and content provider for shared rendering applications. The client accesses the server via a web-based interface.

The client selects a data set and sends a request for information retrieval to the server. The server analyzes the request and returns a customized Java applet and the appropriate data. The Java applet is optimized for a specific rendering task. This means that the rendering algorithm is customized for a particular problem set, thus keeping the applet small by avoiding additional overhead for different cases. The initial data set is also small and will be refined later upon additional requests by the client. Our hierarchical rendering techniques include adaptive space subdivision algorithms, such as adaptive octrees for volumes, wavelet-based data reduction and storage of large volume data sets, and progressive transmission techniques for hierarchically stored volume data sets. 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 implementation features 2-D/3-D preview capability; interactive cutting planes (in a 3-D rendering, with hierarchical isosurface models to provide context information); a lens paradigm to examine a particular region-of-interest (variable magnification and lens shape, interactively modifiable ROI); etc. Complex scenes can be precomputed on the server side and transmitted as a VRML2 file to the client so that the client can render and the user can interact with it in real time.

## 2. Memory-efficient storage and access

Original data are usually structured as a set of files, which represents a series of 2-D cross-sections.

Putting all those slices together, we obtain a 3-D volume. Unfortunately, when we access the data, we typically don't need the implicit coherency across single slices. This coherency stretches only across one direction. Instead, we need brick-like coherency within subvolumes. We present a new datastructure, which uses a combination of delimited octree space subdivision and wavelet compression techniques to achieve better performance.

We present an efficient indexing scheme, a suitable 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 the standard filesystem to store our derived datastructures, and we use filenames as keys to the database, thus avoiding additional overhead, which is typically caused by adding additional layers between the application and the underlying storage system. We found that this method provides the fastest method to access the data. Our indexing scheme in conjunction with the underlying filesystem provides the database system (repository) for the server application, which reads the data at a low resolution from the repository and sends it to a remote rendering client upon request. After the user has specified a subvolume or region-of-interest (ROI), the client application sends a new request to the server to retrieve a subvolume at a higher level of resolution. This updating procedure typically takes considerably less time, because only a small number of files need to be touched. The initial step, which requires to read the initial section of every file, i.e., all bricks, can be sped up by storing an additional file which contains a reduced version of the entire data set.

Our new data structure uses considerably less memory than the original data set, even if the user chooses lossless compression (see statistics, chapter 6). By choosing appropriate thresholds for wavelet compression, the user can switch between lossless compression and extremely high compression rates. Computing time is balanced by choosing an appropriate filesize (chapter 3).

One of the advantages of this approach is the fact that the computing time does not so much depend on the resolution of the subvolume, but merely on the size of the subvolume. This is because 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, which make up the subvolume, usually fits into main memory. Since seek time is much higher than read time for conventional harddisks, the total time for data retrieval mainly depends on the size of the subvolume, i.e., the number of files that need to be accessed, and not so much on the level of detail.

## 3. Filesize considerations

The filesize $f$ for storing the leaves of the octree structure, which is described in chapter 4, should be a multiple $n$ of the minimum page size $p$ of the filesystem. $p$ is typically defined as a system constant in `/usr/include/sys/param.h`). $n$ depends on the wavelet compression. If the lowest resolution of the subvolume 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 $r$ for the wavelet representation. This gives us $8^r \cdot b$ bytes, which must fit in $f$. This means:

$$f = n \cdot p \geq 8^r \cdot b$$

Both $r$ and $b$ are user-defined constants. Typical values are $b = 512$, which corresponds to an $8 \times 8 \times 8$ subvolume, and $r = 3$, which gives us four levels of detail over a range between 512 and $8^3 \cdot 512 = 262144$ data elements, which is more than 2.7 orders of magnitude.

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

$$n \cdot p = 8^r \cdot b,$$

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

## 4. Delimited octree and wavelet structure

The enormous size of the data sets (see chapter 5) requires to subdivide the data into smaller chunks, which can be loaded into core memory within a reasonable amount of time [Hei98, Mey97]. Since we are extracting subvolumes, it seems quite natural to break the data up into smaller bricks. This can be done recursively by using an octree approach [Jac80, Mea80, Red78]. Each octant is subdivided until we reach an empty region which does not need to be subdivided any further, or until we hit the filesize limit *f*, which means that the current leaf fits into a file of the given size.

Each leaf contains the full resolution. The memory reduction occurs by skipping the empty regions. Typically, the size of the data set shrinks to about 20%, i.e., one fifth of the original size (see chapter 6).

Since we want to access the data set in a hierarchical fashion, we have 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. Haar wavelets fulfill these properties. They also have the advantage that they can be easily implemented as integer arithmetic. The lower resolution is stored at the beginning of the file, thus avoiding long seek times within the file.

Another very useful property is the fact that a volume converted into the frequency domain, i.e., the wavelet representation, requires the exact same amount of memory as the original representation. This is also true for all subsequent wavelet recursions.The wavelet recursion terminates when we have reached a predifined minimum subvolume size *b*. The lower bound is the size of a single voxel.

Each octant can be described by a number [Fol96, Hun79]. We use the following numbering scheme (figure 1): A leaf is uniquely characterized by the octree recusion depth and the octree path. We limit the recursion depth to eight, which allows us to encode the depth in 3 bits. In order to store the path, we need 3 bits per recursion step, which gives us 24 bits. 4 bits are spent to encode the depth of the wavelet recursion. The remaining bit is a flag which indicates that the file is empty. This prevents us from opening and attempting to read the file and speeds up the computation. The total number of bits is 32 (double word).

| | | | | | |
|---|---|---|---|---|---|
| 3 | 3 | . . . | 3 | 4 | 1 |
| oct.depth | sub 1 | | sub 8 | wav.depth | empty |

Each bit group can be easily converted into an ASCII character by using binary arithmetic, e.g., `(OCT_DEPTH >> 29) | 0x30)` would encode the octree depth as an ASCII digit. By appending these characters we can generate a unique filename for each leaf.

In order to retrieve a subvolume, we have to find the file(s) in which it is 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 gives us one bit of the subvolume path information. We convert these bits into ASCII characters, using the same macros as above. The first file we are looking for is `7xxxxxxxx??`, 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 filename indicates that the file is empty (last digit), we can skip the file. The filename 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 subvolume, 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 so that the subvolume is complete.

## 5. Applications

Our test applications include molecular biology, medicine, and earthquake simulation. Our prototype for the biomedical field was designed to support three-dimensional visualization of a human brain, which allows us to study details by moving tools, such as an arbitrary cutting plane and variously shaped lenses, across the data set. The various data sets are typically between 20 MB and 76 GB, which makes them impossible 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.

## 6. Statistics

Table 1 shows the reduction of memory which is required to store a large data set, if 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. The recon-

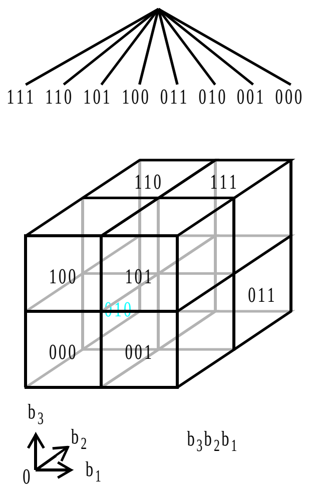
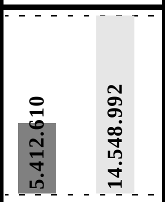
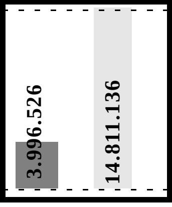
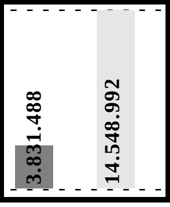
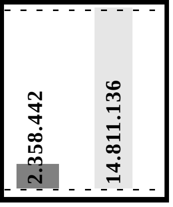


*Fig. 1:* Numbering scheme

| Algorithm | Octree | | | |
|---|---|---|---|---|
| | level 1 | | level 2 | |
| Data type | MRI | CT | MRI | CT |
| Pre-processing | 56 | 63 | 98 | 97 |
| Depth | 4 | 4 | 5 | 5 |
| 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 |

*Tab. 1:* Space subdivision algorithm

struction can be done more efficiently and usually takes 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 and will be implemented in a later version. We will choose algorithms with asymmetric behavior, i.e., compression time is higher than decompression time.

## 7. Conclusions

We have presented an efficient numbering scheme and access method for hierarchical storage of subvolumes on a regular filesystem. This method allows us to access a region-of-interest as a set of bricks at various resolutions. The simplicity of the method makes it easy to implement. The algorithm easily scales by increasing word length and filename length. Future work includes better wavelet compression schemes and time-variant 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. Integration of San Diego Supercomputer Center's High-performance Storage System (HPSS) as a data repository to retrieve large-scale data sets, accessing the data via NPACI's Scalable Visualization Toolkits (also known as VisTools), and evaluation of particular applets with NPACI partners, are main goals for future research efforts.

## Acknowledgements

NSF (CAREER Awards: ACI 9624034, ACI 9983641, LSSDSV: ACI 9982251, NPACI); Office of Naval Research (ONR, N00014-97-1-0222); Army Research Office (ARO 36598-MA-RIP); NASA Ames Research Center (NAG2-1216); LLNL (ASCI ASAP Level-2, B347878, B503159); NATO (CRG.971628); ALSTOM Schilling Robotics; Chevron; Silicon Graphics, Inc.; ST Microelectronics, Inc.; Data sets courtesy of Arthur W. Toga, UCLA School of Medicine, Arthur J. Olson, The Scripps Institute, and Edward G. Jones, Neuroscience Center, UC Davis.

## References

[Hei98] Heiming, Carsten, "Raumunterteilung von Volumendaten," *thesis*, Department of Computer Science, University of Kaiserslautern, Germany, January 1998.

[Hun79] Hunter, G. M.; Steiglitz, K., "Operations on Images Using Quad Trees," *IEEE Trans. Pattern Anal. Mach. Intell.*, 1(2), April 1979, 145–154.

[Jac80] Jackins, C.; Tanimoto, S. L., "Oct-Trees and Their Use in Representing Three-Dimensional Objects," *CGIP*, 14(3), November 1980, 249–270.

[Mea80] Meagher, D., "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, October 1980.

[Mey97] Meyer, Jörg; Gelder, Steffen; Heiming, Carsten; Hagen, Hans, "Interactive Rendering—A Time-Based Approach," *SIAM Conference on Geometric Design '97*, Nashville, TN, November 3–6, 1997, 23.

[Red78] Reddy, D.; Rubin, S., "Representation of Three-Dimensional Objects," *CMU-CS-78-113*, Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA, 1978.

[Sch97] Schneider, Timna Esther, "Multiresolution-Darstellung von 2D-Schichtdaten in der medizinischen Bildverarbeitung," *thesis*; Department of Computer Science, University of Kaiserslautern, Germany, December 1997.