Wavelet-Based 3D Compression Scheme for Very Large Volume Data

Insung Ihm and Sanghun Park
Department of Computer Science
Sogang University
Seoul, Korea
{ihm,hun}@graphlab.sogang.ac.kr

Abstract

Visualizing very large volume data has been recognized as a task requiring great effort in a variety of science and engineering fields. In particular, such data usually places considerable demands on run-time memory space. This paper describes an effective 3D compression scheme for very large volume data that exploits the power of wavelet theory. In designing our compression method, we have compromised between two important factors: high compression ratio and fast run-time random access. Our experimental results on the Visual Human data sets show that our method achieves fairly good compression ratios. In addition, it minimizes the overhead caused during run-time reconstruction of voxel values. This 3D compression scheme will be useful in developing many interactive visualization systems for huge volume data, and will make visualization technology accessible to a much wider range of users, as it can be based on personal computers or low-end workstations with limited memory.

Keywords: very large volume data, wavelets, 3D compression, fast random access, visible human data, interactive visualization

Introduction

Volume visualization is one of the most actively researched topics of scientific visualization. It deals with scalar and vector data, called volume data, defined on three- (or higher-) dimensional grids. In various fields such as computational fluid dynamics, earth, space and environmental science, and medical science, volume data are often so huge, ranging from several hundred megabytes to several dozen gigabytes, that they need special treatment for effective manipulation [10, 14, 13].

A few years ago, the National Library of Medicine (NLM) created computed tomography (CT), magnetic resonance imaging (MRI), and color cryosection images of male and female human cadavers in an effort to provide a complete digital atlas of the human body [12]. The "Visible Man" data set consists of axial scans of the entire body taken at 1 mm intervals at a resolution of 512 pixels by 512 pixels, where the whole data set has 1871 cross-sections. The "Visible Woman" data set consists of cross-sectional images taken at one-third the interval of the male. The data sets amount to 15 Gbytes and 40 Gbytes, respectively.

Visualizing such very large volume data needs different approaches to those used in previous work. Most volume rendering techniques, for example, assume, implicitly or explicitly, that the whole volume data is loaded in the main memory during rendering. In spite of the rapid fall of memory costs, it is still not common to equip a general purpose workstation or personal computer with, say, several giga bytes of main memory. Compression of huge volume data is a natural solution to this problem. Our motivation for this research was to develop a 3D compression method that enables users to load a whole compressed Visible Human data set into a main memory of moderate sizes, say 64 to 128 mega bytes, and to visualize them interactively as if the original data were in the memory.

One of the most important requirements in developing such a compression method is that it must allow quick random access to an individual voxel of compressed data. The general concern of most lossy compression techniques is to achieve the best compression rate with minimal distortion in the reconstructed images, and compression techniques often impose some constraints on random access ability [6, 16]. For instance, when data are compressed by variable-bitrate or differential encoding schemes, such as the Huffman or arithmetic coders used in the JPEG (Joint Photographic Experts Group) or MPEG (Moving Pictures Experts Group), or the adaptive differential pulse code modulation coder, it is hard to decode efficiently individual data items that are accessed in a random fashion. When volume data are handled for interactive visualization, the access patterns change in somewhat complicated ways. Hence, those compression schemes are not suitable for our purpose.

In [11], a compression scheme based on vector quantization was proposed. Vectors in this method consist of the density values and the precomputed normal fields of voxels in the partitioning subblocks. They were quantized into a codebook and each subblock was represented by an appropriate index. Ray casting with parallel projection was accelerated by shading only the vectors in the codebook, and composing the partial images in the correct order. Since voxel decoding is just a simple access of the codebook, it provides fast random access to voxel values. In this study, a compression ratio of five, rather moderate, for 128^3 volume data with some blockiness and contouring in the rendered image was reported. The Laplacian pyramid technique for 2D images was extended to volume data in [5]. They constructed a simple hierarchical structure, called the Laplacian pyramid, using a Gaussian low-pass filter, and encoding it by uniform quantization. Voxel values are reconstructed on the fly by traversing the pyramid from bottom to top. To reduce the huge reconstruction overhead, they suggest a cache data structure.

In [20], experimental results, comparing several 2D lossy compression techniques, were described, where the method based on wavelet transform was reported to be best. The wavelet-based method that they applied to each 2D slice was a typical transform coder that had three basic components: a wavelet transform, a vector quantization, and a Huffman or run-length encoder. Their method focused on efficient storage and transmission, rather than on run-time manipulation, and failed to exploit the considerable degree of redundancy that exists between adjacent 2D slices. The idea of using a three-dimensional wavelet to approximate three-dimensional volume data sets was introduced in [8, 9]. The 2D wavelet transform was extended to three dimensions, and was also applied to delete insignificant wavelet coefficients. While he presented the potential of 3D wavelet transforms for volume visualization, the author did not mention whether the encoding technique actually reduced storage space. In [1], a 3D subband transformation on image sequences is performed, then the transformed information is encoded using the zero-tree coding technique, which was originally introduced in [18], and was improved in [15].

In this article, we introduce a new compression scheme that can be used effectively in manipulating and visualizing very large volume data. Our compression scheme was designed in the hope that users on a computer with a limited memory could feel as if they have loaded the whole huge volume data into a large memory. Most of the previously developed compression techniques trade off random access ability for higher compression ratios. In designing our method, we have compromised between these two important goals so that the method achieves fairly good compression ratios as well as minimizing the overhead caused during random access to voxel values. The method is based on the 3D wavelet transform, and hence provides a multi-resolution representation of volume data.

The rest of the paper is organized as follows: In the *Haar Wavelets and Compression* section, we begin by introducing the basic theory of wavelet transforms, and wavelet-based compression. In the following four sections *3D Wavelet Transforms*, *Encoding Wavelets Coefficients*, *Reconstructing Voxel Values*, and *Analysis of Performance*, we provide a detailed description of our compression scheme for very large volume data. Experimental results on the Visible Human data set, are reported in Section *Experiments,Compression Quality*, and *Voxel Reconstruction Time*. Finally, we present conclusions and directions for further research in Section *Conclusions and Future Work.*

Haar Wavelets and Compression

Wavelets are a mathematical tool for representing functions hierarchically, and have recently had a great impact in several areas of computer graphics (For an introduction and discussion of applications, refer to [2], [3], [4], [17], and [19].). They provide a toolkit for decomposing functions in multi-resolution form, which can be very usefully applied to a variety of functional data found in computer graphics applications, such as images, geometric models, global illumination models, animation and volume data.

The simplest example of wavelets is the Haar wavelet. Consider a sequence $X^n = \{x_{n,i}, 0 \leq i < 2^n\}$ of samples of a function, where the size of X^n is assumed to be a power of two, for convenience. When each adjacent pair of samples is averaged, a new sequence $X^{n-1} = \{x_{n-1,i}, 0 \leq i < 2^{n-1}\}$ is obtained, where $x_{n-1,i} = \frac{x_{n,2i}+x_{n,2i+1}}{2}$. This new sequence, which is half the original size, can be regarded as another representation of X^n with a coarser resolution. Since some information has been lost in this down-sampling, we need to keep extra information that is necessary for going back to the original sequence, and that can be expressed in the sequence $Y^{n-1} = \{y_{n-1,i}, 0 \leq i < 2^{n-1}\}$, where $y_{n-1,i} = \frac{x_{n,2i}-x_{n,2i+1}}{2}$. The process of decomposing 2^n samples into 2^{n-1} averages and 2^{n-1} differences, called the detail coefficients, is considered as applying 2-channel subband filters, the smoothing, or scaling filter, and the detail, or wavelet filter, respectively. It is easily seen that the original samples can be reconstructed by reversing the operations: $x_{n,2i} = x_{n-1,i} + y_{n-1,i}$, and $x_{n,2i+1} = x_{n-1,i} - y_{n-1,i}$.

We can apply the same decomposition to the coarser samples X^{n-1} repeatedly, until we get $X^0 = \{x_{0,0}\}$ with one sample. As a result of this wavelet transform, or wavelet decomposition, we obtain a new sequence of 2^n numbers, made of the overall average $x_{0,0}$ and a sequence of the detail coefficients, $Y^0, Y^1, \cdots, Y^{n-1}$. The original data can be reconstructed to any resolution by repeatedly adding and subtracting the detail coefficients from the lower-resolution versions. The new sequence of data is, hence, a multi-resolution representation of the original samples.

The Haar wavelet is simple and computationally cheap because it can be implemented by a few additions, subtractions, and shift operations. Hence it is very effective in applications that require fast decomposition and reconstruction. However, it does not perform as well in terms of quality as other popular wavelets, such as Daubechies' wavelets.

Note that the detail coefficients are sample-to-sample differences. Hence, they tend to be small in magnitude, especially when the sampled data are from smooth signals. Since recursive application of the smoothing filter makes the samples smoother, a large fraction of the detail coefficients will often be very small in magnitude. The basic idea of wavelet compression is that deleting these small coefficients, that is, replacing them by zeros, introduces only small errors in reconstructing the original samples. It is straightforward to prove that when the wavelet basis is orthonormal, the best way to pick some number of wavelet coefficients and make the resulting error as small as possible, measured in the L^2 norm, is simply to select the coefficients with the largest absolute values. By replacing the deleted coefficients by zeros, the original information can be approximated by a smaller set of samples.

3D Wavelet Transforms

The one-dimensional wavelet transforms discussed in the previous section can be extended naturally into a three-dimensional space. Muraki [8, 9] presented the idea of using a three-dimensional wavelet to approximate three-dimensional volume data sets. In his work, he built a 3D orthonormal wavelet basis using all possible tensor products of one-dimensional basis functions. The wavelet transform was applied to volume data to compute wavelet coefficients. The insignificant coefficients were removed, then an approximation of the original data was reconstructed using only the remaining coefficients. This procedure is typical in wavelet compression. However, he did not mention by how much storage space was reduced.

Notice that there are two problems to be addressed: one is how to reduce the number of coefficients needed to approximate volume data, and the other is how to encode and store the necessary information in a smaller

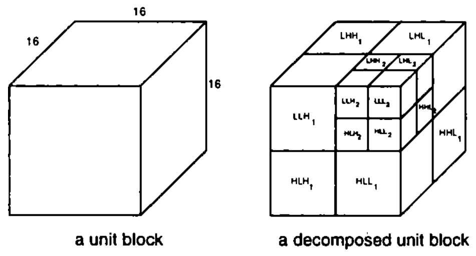


Figure 1: Unit Block and Decomposed Unit Block

number of bits. In our framework, we first partition a given 3D volumetric data set into subblocks, called *unit blocks*, whose size is 16 × 16 × 16. For each unit block, a three-dimensional wavelet transform is repeatedly applied twice. We use a simple transform based on the Haar wavelets whose eight-band filter bank can be expressed as:

$$\begin{aligned}
c_{lll} &= (c_1 + c_2 + c_3 + c_4 + c_5 + c_6 + c_7 + c_8)/8 \\
c_{llh} &= (c_1 + c_2 + c_3 + c_4 - c_5 - c_6 - c_7 - c_8)/8 \\
c_{lhl} &= (c_1 + c_2 - c_3 - c_4 + c_5 + c_6 - c_7 - c_8)/8 \\
c_{lhh} &= (c_1 + c_2 - c_3 - c_4 - c_5 - c_6 + c_7 + c_8)/8 \\
c_{hll} &= (c_1 - c_2 + c_3 - c_4 + c_5 - c_6 + c_7 - c_8)/8 \\
c_{hlh} &= (c_1 - c_2 + c_3 - c_4 - c_5 + c_6 - c_7 + c_8)/8 \\
c_{hhl} &= (c_1 - c_2 - c_3 + c_4 + c_5 - c_6 - c_7 + c_8)/8 \\
c_{hhh} &= (c_1 - c_2 - c_3 + c_4 - c_5 + c_6 + c_7 - c_8)/8,
\end{aligned}$$

where $c_i, 1 \le i \le 8$, on the right side are eight coefficients in each 2 × 2 × 2 subregion of a unit block, c_{lll} represents their average, and the remaining coefficients on the left side are the detail values corresponding to the filtering sequences (for example, c_{hlh} is obtained by applying the high-pass filter, the low-pass filter, then the high-pass filter.) This decomposition transform arises from the separable application of filters in three dimensions. Two applications of the 3D wavelet transforms are enough, considering that a smaller number of transforms results in faster reconstruction, and that most of the data $\frac{63}{64}$ $(= 1 - \frac{1}{8^2})$ is already decomposed into wavelet coefficients. The decomposition process converts each unit block into a decomposed version that can be stored in an array with the same number of elements, using a proper ordering of the coefficients (See Figure 1.).

The information in a unit block can be expressed as a weighted sum of wavelet basis functions whose weights are stored in its decomposed unit block. The theory behind wavelet compression, as mentioned, shows that the best way to pick some number of wavelet coefficients, making the resulting error, measured in the L^2 norm, as small as possible, is simply to select the coefficients with the largest absolute values. In other words, we keep only the coefficients greater than some appropriate threshold value, and replace the remaining coefficients by zeros. Then the original information can be approximated by a smaller number of non-zero wavelet coefficients.

Encoding Wavelets Coefficients

Now, we describe how we solve the second problem, that of encoding and storing the necessary information in a smaller number of bits. A typical wavelet compression algorithm has three basic components: transform, quantization, and encoding. The transform stage separates the input data into different bands of frequencies using wavelet filters. The wavelet coefficients are then quantized to restrict the values of the coefficients to a limited number of possibilities. Note that usually all of the information loss occurs in this stage. Then the encoding stage takes the string of symbols coming from the quantizer, and attempts to represent the data stream as efficiently as possible without loss. Popular variable length coders, such as Huffman or arithmetic coders, work well. However, such techniques are not appropriate for the situation where an individual data item must be quickly reconstructed in an arbitrary sequence.

In addition to the quantized wavelet coefficients, information about the positions of the significant coefficients that have survived the truncation of insignificant coefficients, must be encoded. In Shapiro [18], it is shown that determining the positions of the few retained coefficients consumes a significant portion of the bit budget at low rates, and is likely to become an increasing fraction of the total cost as the rate decreases. Run-length encoding is very attractive, considering the fact that most of the coefficients are usually zeros, although the technique is not well suited to random access of individual data items. Another technique, called zerotree encoding [18], greatly improves the performance of a wavelet encoder, but is much slower.

When an encoding scheme is designed, we take into consideration the tradeoff between compression rate, speed, and quality. To design an encoding technique appropriate for our goal, we have compromised between a good compression ratio and fast random access. As emphasized before, when compressed volume data are loaded in the main memory for processing, it is important to be able to quickly reconstruct the value of an individual element, and the access patterns change in somewhat complicated ways.

Figure 2 illustrates how the surviving wavelet coefficients and positional information are encoded. (We explain our scheme in terms of the format of the Visible Human data sets. The basic idea can be applied easily to

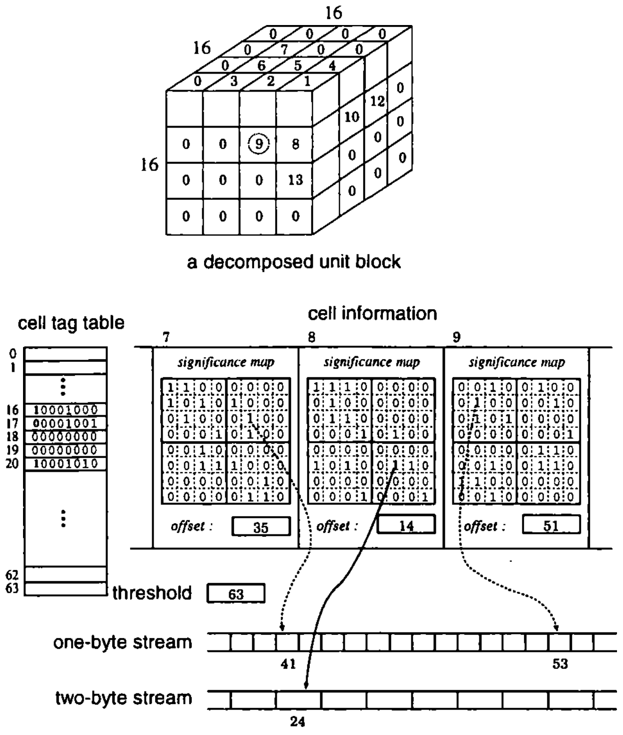


Figure 2: Wavelets Encoding Scheme

other formats.) Consider a decomposed unit block of size $16 \times 16 \times 16$, which is a level-two multi-resolution representation of the corresponding original unit block. Note that a large portion (say, more than 90%) of coefficients, less than a threshold τ, have already been replaced by zeros. Considering the usual spatial coherence in the data, it is quite possible that the zero coefficients exist in thick clusters. We subdivide the decomposed unit block into 4^3 (= 64) subblocks, called *cells*, where each cell represents a $4 \times 4 \times 4$ subregion.

The cells in the decomposed unit block are enumerated one-by-one in front-to-back, top-to-bottom, and left-to-right order, tagging with zero the cells whose coefficients are all zero, and with positive integers in increasing order, the cells that contain at least one non-zero coefficient. When each tag is represented in 1 byte, 64 bytes of storage is necessary for the *cell tag table*.

During decomposition, we use enough precision, say four bytes per voxel, to calculate the average and detail coefficients without round-off errors. Since the original voxel values, stored in 12 bits, range from 0 to $2^{12} - 1$, the averages fall between 0 and $2^{12} - 1$, and the details between $-\frac{2^{12}-1}{2}$ and $\frac{2^{12}-1}{2}$. The cells with a non-zero tag, that is, having at least one non-zero coefficient, are classified into two groups: the first group contains the cells all of whose coefficients are in the intervals $[-(\tau+128), -\tau]$ or $[\tau, \tau + 127]$, and the second one contains the remaining cells. We quantize, offset by τ, each non-zero coefficient of the cells in the first group in a signed character (one byte), rounding the fractional part. Note that the maximum round-off error is 0.5. We then represent each non-zero coefficient of the cells in the second group in a signed short integer (two bytes). Since the integral part of the coefficient can be stored in 12 bits, we use the remaining 4 bits for the fractional part, again rounding the less significant portion. In this case, the maximum round-off error is $\frac{1}{2^5}$. The information on which group a cell is included in can be encoded in the most significant bit of its tag. (Notice that this bit is free because positive integers less than or equal to 64 can be represented in the remaining bits.)

These two groups of non-zero coefficients are put in two arrays, called *one-byte stream* and *two-byte stream*, respectively. To store the coefficients, the 64 coefficients in a cell with a non-zero tag are enumerated, putting only the non-zero coefficients in the corresponding stream. For retrieval, the positions of non-zero coefficients in the corresponding stream must be encoded. We allocate an additional chunk of memory, called *cell information*, for each cell that consists of a $4 \times 4 \times 4$ one-bit flag block and offset information. This block of one-bit flags requires 8 bytes and contains a *significance map*, or the binary information as to which of the coefficients in the cell are non-zero. The offset information *offset*, stored in two bytes, contains the address, in the corresponding stream, of the first non-zero coefficients of a cell in the ordering.

Figure 2 illustrates an example of this encoding. Consider the cell numbered 9. It is the 17th one (counting from zero) in the cell ordering. From the tag **00001001** for this cell, we see that its cell information is in the 8th block (counting from zero). In addition, the most significant bit **0** tells that the coefficients are found in the two-byte stream. (In our implementation, **0** means the two-byte stream and **1** means the one-byte stream.) Assume that we are retrieving the coefficient in boldface. There are 10 flags set on before the coefficient, meaning that it is the 11th non-zero coefficient in the cell. Hence the address in the two-byte stream can be obtained by adding 10 to the offset: $10 + 14 = 24$.

Reconstructing Voxel Values

The process of extracting a voxel value from wavelet-compressed data consists of two stages of computation: all the wavelet coefficients necessary for reconstruction are first retrieved from encoded unit blocks, then, the reconstruction formula is applied to the coefficients. Since the wavelet transforms are applied twice for decomposition, each $4 \times 4 \times 4$ subregion of a unit block can be

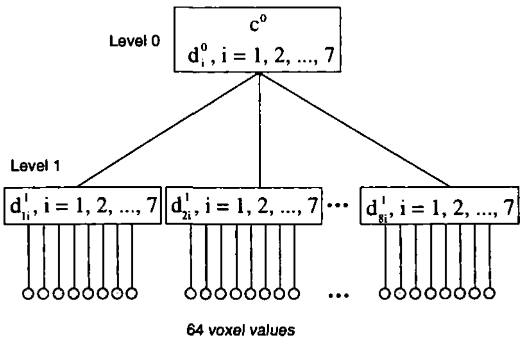

Level 0
c^0
d^0_i, i = 1, 2, ..., 7
Level 1
d^1_{1i}, i = 1, 2, ..., 7
d^1_{2i}, i = 1, 2, ..., 7
d^1_{8i}, i = 1, 2, ..., 7
64 voxel values

Figure 3: Octree Representation for a $4 \times 4 \times 4$ Subregion

considered as represented by an octree in Figure 3. The average coefficient c^0 of the root node is the average of all the voxel values in the subregion, and the seven detail coefficients d^0_i $(i = 1, 2, \cdots, 7)$ provide the necessary information that can, with the average, reconstruct the averages of the eight $2 \times 2 \times 2$ subregions, represented by the nodes on level 1. In turn, each set of the seven detail coefficients d^1_{ji} $(j = 1, 2, \cdots, 8, i = 1, 2, \cdots, 7)$ of the level 1 nodes are used to reconstruct the eight voxel values of the corresponding subregion. To extract the value of a specific voxel, it is necessary to traverse the octree from the root down to the corresponding leaf, applying the reconstruction transforms twice.

The reconstruction process is the reverse of decomposition. For the Haar wavelets we use, the reconstruction formulae are:

$$\begin{aligned}
\hat{c}_1 &= c_{lll} + c_{llh} + c_{lhl} + c_{lhh} + c_{hll} + c_{hlh} + c_{hhl} + c_{hhh} \\
\hat{c}_2 &= c_{lll} + c_{llh} + c_{lhl} + c_{lhh} - c_{hll} - c_{hlh} - c_{hhl} - c_{hhh} \\
\hat{c}_3 &= c_{lll} + c_{llh} - c_{lhl} - c_{lhh} + c_{hll} + c_{hlh} - c_{hhl} - c_{hhh} \\
\hat{c}_4 &= c_{lll} + c_{llh} - c_{lhl} - c_{lhh} - c_{hll} - c_{hlh} + c_{hhl} + c_{hhh} \\
\hat{c}_5 &= c_{lll} - c_{llh} + c_{lhl} - c_{lhh} + c_{hll} - c_{hlh} + c_{hhl} - c_{hhh} \\
\hat{c}_6 &= c_{lll} - c_{llh} + c_{lhl} - c_{lhh} - c_{hll} + c_{hlh} - c_{hhl} + c_{hhh} \\
\hat{c}_7 &= c_{lll} - c_{llh} - c_{lhl} + c_{lhh} + c_{hll} - c_{hlh} - c_{hhl} + c_{hhh} \\
\hat{c}_8 &= c_{lll} - c_{llh} - c_{lhl} + c_{lhh} - c_{hll} + c_{hlh} + c_{hhl} - c_{hhh}
\end{aligned}$$

The algorithm in Figure 4 describes how a wavelet coefficient is retrieved from a $16 \times 16 \times 16$ encoded unit block. To decode a coefficient with index (i, j, k), we access the cell tag table for the tag of the cell that contains the coefficient. If it is zero, the coefficient is simply null ([case 1]). Otherwise, we look at its cell information for further processing. Let (i', j', k') be the relative index of the coefficient (i, j, k) in the cell. If the bit-flag for the index (i', j', k') is 0, then the coefficient is zero ([case 2]). If not, the coefficient is non-zero, and it is in the data stream indicated by the most significant bit of the tag ([case 3]). To access the coefficient value, we

```
Input:   a 16 × 16 × 16 encoded unit block and
         an index $(i,j,k)$
Output:  the decoded value of the
         coefficient with index $(i,j,k)$

• Find the cell C that contains the index
$(i,j,k)$.
• If the tag for C is 0, return 0.   [case 1]
• Compute the relative index $(i',j',k')$ in C.
• If the bit-flag for $(i',j',k')$ is 0, return
0.  [case 2]
• Count the number of preceding non-zero
coefficients by table access.
• Add the displacement to the offset to
compute the correct address.
• Access the appropriate data stream, and
return the value.   [case 3]
```

Figure 4: Wavelets Decoding Algorithm

need to compute its address in the data stream. It can be computed by adding its displacement value to the offset value. The displacement is the number of the non-zero coefficients with flag 1 that precede it in the enumeration. To count the number efficiently, we use a precomputed indexing table T(∗) with $2^{16} = 65536$ entries. Given a word made of two bytes, corresponding to 16 bit flags, the table returns the number of 1 bits in the word. Hence, the correct number can be counted by accessing the table only a few times (Note that the correct number of zeros must be padded in the word, from the position (i',j',k') in the last access).

Analysis of Performance

We now analyze the costs that must be paid to access a voxel value in a compressed unit block. To reconstruct the value, an octree is traversed, applying the proper reconstruction formulae twice. Since seven addition/subtraction operations need to be carried out per formula, evaluation of 14 additions/subtractions are necessary. Furthermore, 15 wavelet coefficients (8 on level 0, and 7 on level 1) must be decoded from the encoded unit block. When a wavelet coefficient is decoded, there are three cases (See the decoding algorithm again.). When the tag for the subblock that contains the coefficient is zero, or its one-bit flag is 0, that is, when the coefficient is zero ([case 1] and [case 2]), the cost is trivial. When its flag is 1, indicating that the coefficient is non-zero ([case 3]), a few table accesses, 2.5 on the average, and a few additions are necessary to compute the correct address in the proper stream. In our implementation, we usually have a proportion of non-zero coefficients after wavelet compression of 3 to 10 per cent, implying 90 to 97 per cent of decoding belongs in ([case 1] and [case 2]). From this analysis, we see that retrieving a wavelet coefficient in an encoded unit block involves little cost.

In many applications, voxel access patterns show some degree of locality. (Recall how voxels of volume data are visited in the ray casting or splatting algorithms.) To enhance efficiency, each $4 \times 4 \times 4$ subregion of a unit block can be regarded as a reconstruction unit. In this case, the whole octree is traversed for 64 voxel values, evaluating the 8 reconstruction formulae 9 times. Although there appears to be 56 (= $7 \cdot 8$) addition/subtraction operations in each application of the 8 formulae, a simple optimization technique for removing redundant computations shows that 24 operations are enough. Since the number of necessary additions/subtractions is $9 \cdot 24$, $3\frac{3}{8}$ $(= \frac{9 \cdot 24}{64})$ operations must be evaluated on average per voxel. Also, since each wavelet coefficient in the $4 \times 4 \times 4$ subregion is decoded once, the average number of necessary decoding operations is one per voxel. Hence, the costs for retrieving a voxel value are one decoding operation and $3\frac{3}{8}$ additions/subtractions. Some other operations such as bit-wise operations and address computations must be carried out, but the total cost is quite cheap considering the benefits we get from data compression.

Before turning to the next section, we report a quantitative analysis of the compression ratios. Each unit block of size $16 \times 16 \times 16$ takes $16 \times 16 \times 16 \times 2$ bytes before compression. To store the threshold value and the tag table, 2 + 64 bytes of memory is necessary (See Figure 2 again.). For each cell that contains at least one non-zero coefficient, auxiliary memory is allocated where 8 bytes ($4 \cdot 4 \cdot 4$ bits) are used for the significance map, and 2 bytes for the offset.

Let α be the proportion of cells with non-zero tags that contain at least one non-zero coefficient, that is, $\frac{\textit{\# of non-null cells}}{64}$, then $(8+2) \cdot 64 \cdot \alpha$ bytes are required to store the cell information.

The non-zero coefficients used after wavelet compression are partitioned into the one-byte stream and the two-byte stream. Let β be the ratio of the number of coefficients in the one-byte stream to the total number of non-zero coefficients used, that is, $\frac{\textit{\# of coefficients in one-byte stream}}{\textit{\# of all non-zero coefficients used}}$. Furthermore, assume that the rate of non-zero wavelet coefficients we use for compression is γ. Then the compression rate ρ is:

$$\begin{aligned}\frac{1}{\rho} &= \{2 + 64 + (8+2) \cdot 64 \cdot \alpha + \\ &\quad 16 \cdot 16 \cdot 16 \cdot \gamma \cdot (\beta + 2(1-\beta)\}/(16 \cdot 16 \cdot 16 \cdot 2) \\ &= \frac{33}{4096} + \frac{5 \cdot \alpha}{64} + \gamma \cdot (1 - \frac{\beta}{2}) \\ &\approx 0.008057 + 0.078125 \cdot \alpha + \gamma \cdot (1 - 0.5 \cdot \beta)\end{aligned}$$

Notice that the first and the second terms in $\frac{1}{\rho}$ are the costs that must be paid to store the necessary cell tag table, threshold, and cell information. In our scheme, this information allows low-cost random access. It could be further compressed using the encoding techniques such as the zerotree, but that only places constraints on random access to the compressed data.

Experiments

Our new compression scheme has been implemented on an SGI Octane workstation with a 175Mhz R10000 CPU. We have generated a test volume data set from the original CT data of the Visible Man. The pixel size and slice spacing of the fresh CT data vary along the vertical axis, where the slices are grouped into nine sections. For a performance test, we took the slices corresponding to the upper body, and rebuilt a $512 \times 512 \times 512$ volume data set. Two bytes are used for each voxel in which 12 bits are actually used, and the whole data set takes up 256 Mbytes. The test data may not be considered very large enough. However, the basic unit in our compression scheme is the $16 \times 16 \times 16$ unit block. Hence, experiments with larger volume data would produce similar results.

Compression Quality

Statistics for the compression of the test volume data are summarized in the table of Figure 5. As explained in Section *Encoding Wavelets Coefficients*, a proper threshold value τ needs to be specified to truncate smaller wavelet coefficients. In our framework, we specify, instead, a desired ratio $\bar{\gamma}$ of nonzero wavelet coefficients to be used, then corresponding threshold values are automatically computed. The ideal way to compute τ for given $\bar{\gamma}$ is to sort first all the wavelet coefficients, and then find the ($\bar{\gamma}\cdot$the total number of voxels)-th largest coefficient. This is not practical when, as in our case, the volume data set is very large.

Recall that we use $16 \times 16 \times 16$ subblocks as unit blocks. When the resolution of the volume data set is, for example, $512 \times 512 \times 512$, the data consist of 32768 (= 32^3) unit blocks. We first apply wavelet transforms to each unit block i, and compute the ratio r_i of nonzero wavelet coefficients to the whole number 4096 (= 16^3) of coefficients. This ratio is a good approximate measure that indicates how rapidly the voxel values change in the unit block. The total number of nonzero coefficients to be used for the whole data is adaptively distributed to unit blocks according to their complexity. It is reasonable that more nonzero coefficients are assigned to unit blocks with higher ratios.

When the data size is $512 \times 512 \times 512$, $512^3 \cdot \bar{\gamma}$ nonzero coefficients are to be distributed to 32768 unit blocks. For the unit block i, we allocate $n_i = \frac{r_i}{\sum_j r_j} \cdot 512^3 \cdot \bar{\gamma}$ coefficients, where the weight $\frac{r_i}{\sum_j r_j}$ is the relative measure of data complexity. Now, the n_i-th largest wavelet coefficient becomes the threshold value of the corresponding unit block, and coefficients smaller than the threshold are replaced by zeros. We find that this adaptive decision on thresholds diminishes the "blockiness" effect that often occurs when a single threshold value is applied to the whole wavelet image. Notice that the actual ratio γ, as shown in the table, is not always the same as the desired ratio $\bar{\gamma}$. This is because unit blocks often contain more than one wavelet coefficient having the same value as the threshold. α and β in the table show the averages of the corresponding values over the whole unit blocks.

We tested with the five desired ratios $\bar{\gamma} = 0.03, 0.05, 0.07, 0.10, 0.15$, and achieved the compression rates 8.5 to 28.2. Figure 7 (a) and (b) show sample slices from the uncompressed data, and one compressed data set with $\bar{\gamma} = 0.03$. We observe that our compression technique reconstructs slices very faithfully. Figure 7 (c), (d), (e) and (f) illustrate ray-cast images for the classification of skin. When the ratio is 0.10 or 0.15, it is hard to distinguish between the volume-rendered images of the uncompressed and compressed volume data. When the ratio is 0.03, the rendered image is visibly different, but most features are still preserved.

To look at distortion or difference between the original and reconstructed voxel values, we measured two objective fidelity criteria. The mean-square signal-to-noise ratio SNR (dB) is a measure of the size of the error relative to the signal, and the mean-square peak-signal-to-noise ratio PSNR (dB) measures the size of the error relative to the peak value of the signal. We also examined the fidelity of the reconstructed normal vectors. The normal vector plays an important role in volume visualization, and is often approximated at each voxel using the popular central-difference formula. The normal vector is one of the most important factors that determine the quality of volume-rendered images. It is also used in correctly classifying materials in volume data. We measured the average angular deviations from the actual normal for both classifications of bone and skin. The statistics show our compression technique produced very favorable compression performances.

Note that there are larger errors in normal vectors for the bone case. We speculate that the material surrounding the body, that we could not eliminate by classification, introduced additional errors. Considering the fact that the process of approximating differentials using the central difference is ill-conditioned, we think the experimental results are satisfactory.

		$\bar{\gamma}$: Desired Ratio of the Wavelet Coef's Used				
		3%	5%	7%	10%	15%
Compression Performance	Compressed Data Size (Mb)	9.08	13.25	17.01	22.17	29.99
	Compression Ratio	28.2 : 1	19.3 : 1	15.1 : 1	11.6 : 1	8.5 : 1
	α	0.1107	0.1859	0.2474	0.3219	0.4148
	β	0.6488	0.7226	0.7647	0.8052	0.8470
	γ	0.0278	0.0457	0.0632	0.0894	0.1330
Errors in Voxel Values	SNR (dB)	24.9	27.9	30.2	33.4	37.7
	PSNR (dB)	43.0	46.0	48.3	51.6	55.8
Errors in Normals	Skin (deg)	16.7	12.6	9.8	7.0	4.6
	Bone (deg)	27.1	20.8	16.4	11.6	7.0

Figure 5: Experimental Results on Compression Quality

Voxel Reconstruction Time

Two situations were considered to evaluate overheads for reconstructing voxel values from compressed volume data (See Figure 6.). First, the timings for pure random access were taken by repeatedly fetching voxel values with randomly generated indices (i, j, k). To measure the reconstruction overheads, we first accessed randomly selected voxels from the uncompressed volume data in a simple $512 \times 512 \times 512$ array one million times. Then the same measurement was taken for each compressed data. The "Pure Random" timings show that fetching voxel values takes roughly four times as long for the compressed data.

Frequently, voxels of volume data are accessed with some regular pattern. For example, the splatting algorithm traverses voxels slice by slice in front-to-back order along the viewing direction. This kind of access pattern can often be simulated by enumerating the $4 \times 4 \times 4$ cells and then traversing voxels within cells both in front-to-back order. The indices of cells in the volume data were repeatedly generated, and all the 64 ($= 4^3$) voxels were accessed simultaneously. Recall that our encoding scheme offers more efficient reconstruction when the voxels are decoded cell by cell. First, we compared the timing performance for the case in which all the cells in the $512 \times 512 \times 512$ volume data were reconstructed in the front-to-back order. We also measured the timings for reconstructing only the cells in the unit blocks that contain at least one voxel, classified as skin. To do this, a simple spatial partitioning data structure was used in which the min-max pair for each $16 \times 16 \times 16$ unit block is stored. All the unit blocks were scanned in the front-to-back order, reconstructing only the cells in the unit blocks whose min-max intervals intersect with that of the classification for skin. There were 11,000 unit blocks ($16^3 \cdot 11,000$ voxels) reconstructed for this classification. We observe that it took about 1.6 times longer for the compressed data. The timings in "Cell-Wise (Skin)" show that the performance differences are about 3.4 to 6.3 seconds between the uncompressed and compressed data. They are ignorable in many CPU-intensive applications such as, for example, volume rendering which usually takes more than a hundred seconds.

Conclusions and Future Work

We have described an effective 3D compression scheme for very large volume data that exploits the power of wavelet theory. Our experimental results on the Visual Human data sets show that our scheme achieves fairly good compression ratios and also provides fairly fast random access to individual voxels. It can be very useful in many applications where users want to load the whole huge volume data into a main memory and visualize the data interactively. We believe that our compression method makes it more feasible to visualize very large volume data on popular personal computers or low-end workstations, making visualization technology accessible to a much wider range of users.

Some topics deserve further investigation. The Haar wavelet filter is computationally efficient, but it is inferior, as a filter, to other popular wavelet filters used in image compression. We are currently experimenting with several orthogonal Daubechies' wavelets. The trade-offs of accuracy and efficiency between several filters need to be analyzed. When our compression scheme is implemented with various visualization algorithms, it would be desirable to have an efficient cache data structure which temporarily holds decoded voxels. Considering the locality or coherency of voxel access, it could save a great deal of redundant computation. Finally, we are applying our compression scheme to the development of an interactive virtual navigation system for the human body. This sys-

		Uncompressed	$\bar{\gamma}$: Desired Ratio of the Wavelet Coef's Used				
			3%	5%	7%	10%	15%
Pure Random		2.18	7.75	8.20	8.62	9.15	9.84
Cell-Wise	All	19.95	30.31	31.58	32.52	33.62	35.05
	Skin	6.70	10.13	10.75	11.32	12.10	13.02

Figure 6: Experimental Results on Voxel Reconstruction Time

tem, based on direct volume rendering, will complement the polygon-based navigation systems, for example, [7].

Acknowledgements
The authors wish to thank Mr. Gee-Bum Koo for his efforts to prepare our experiments. This work was supported in part by grants from Minstry of Science and Technology of Korea, and Korea Science and Engineering Foundation (KOSEF: 981-0926-138-1).

References

[1] Y. Chen and W. Pearlman. Three-dimensional subband coding of video using the zero-tree method. In *Proceedings of SPIE - Visual Communications and Image Processing '96*, pages 1302–1312, Orlando, March 1996.

[2] C. K. Chui. *An Introduction to Wavelets*. Academic Press Inc., 1992.

[3] I. Daubechies. *Ten Lectures on Wavelets*. SIAM, 1992.

[4] A. Fournier, editor. *Wavelets and Their Applications in Computer Graphics*. ACM SIGGRAPH, 1995. ACM SIGGRAPH '95 Course Notes.

[5] M. H. Ghavamnia and X. D. Yang. Direct rendering of Laplacian pyramid compressed volume data. In *Proceedings of Visualization '95*, pages 192–199, Atlanta, October 1995.

[6] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Addison-Wesley Pub. Comp., 1993.

[7] L. Hong, S. Muraki, A. Kaufman, D. Bartz, and Taosong He. Virtual voyage: interactive navigation in the human colon. In *Proceedings of ACM SIGGRAPH '97*, pages 27–34, Los Angeles, August 1997.

[8] S. Muraki. Approximation and rendering of volume data using wavelet transforms. In *Proceedings of Visualization '92*, pages 21–28, Boston, October 1992.

[9] S. Muraki. Volume data and wavelet transforms. *IEEE Computer Graphics and Applications*, 13(4):50–56, 1993.

[10] G. M. Nielson and B. Shriver, editors. *Visualization in Scientific Computing*. IEEE Computer Society Press, 1990.

[11] P. Ning and L. Hesselink. Fast volume rendering of compressed data. In *Proceedings of Visualization '93*, pages 11–18, San Jose, October 1993.

[12] NLM. *http://www.nlm.nih.gov/research/visible/visible_human.html*, 1997.

[13] T. M. Rhyne, editor. *Visualizing and examining large scientific data sets: a focus on the physical and natural sciences*. ACM SIGGRAPH, 1994. ACM SIGGRAPH '94 Course Notes.

[14] L. Rosenblum et al., editor. *Scientific Visualization: Advances and Challenges*. IEEE Computer Society Press, 1994.

[15] A. Said and W. Pearlman. Image compression using the spatial-orientation tree. In *Proceedings of IEEE Intl. Symp. on Circuits and Systems*, pages 279–282, Chicago, May 1993.

[16] K. Sayood. *Introduction to Data Compression*. Morgan Kaufmann Publishers, 1996.

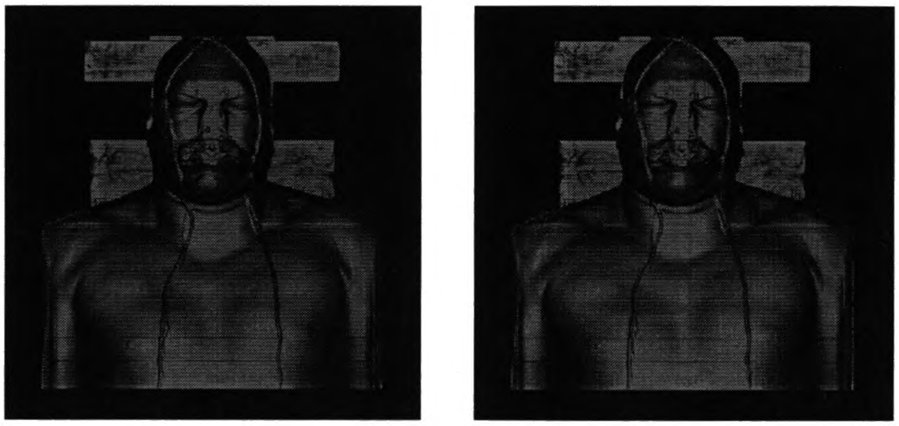
[17] P. Schröder and W. Sweldens, editors. *Wavelets in Computer Graphics*. ACM SIGGRAPH, 1996. ACM SIGGRAPH '96 Course Notes.

[18] J. M. Shapiro. Embedded image coding using zerotrees of wavelet coefficients. *IEEE Transactions on Signal Processing*, 41(12):3445–3462, December 1993.

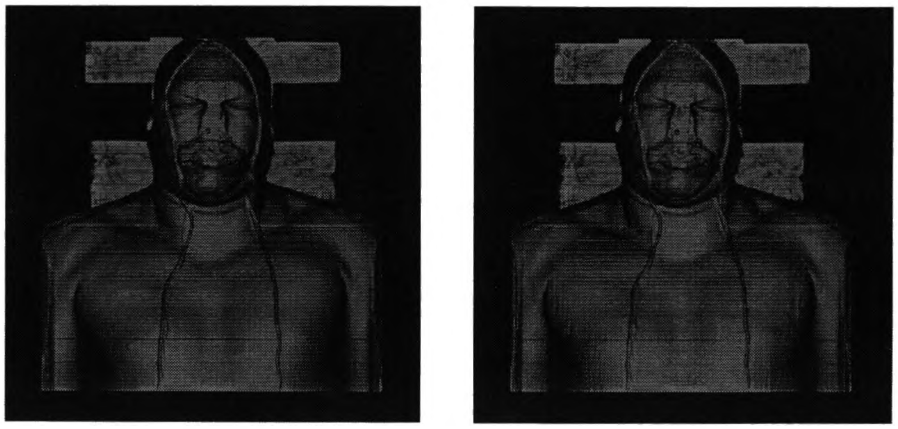
[19] E. Stollnitz, T. DeRose, and D. Salesin. *Wavelets for Computer Graphics: Theory and Applications*. Morgan Kaufmann Publishers, 1996.

[20] G. R. Thoma and L. R. Long. Compressing and transmitting Visible Human images. *IEEE Multimedia*, 4(2):36–45, 1997.

(a) Original (b) $\bar{\gamma} = 0.03$



(c) Original (d) $\bar{\gamma} = 0.07$



(e) $\bar{\gamma} = 0.05$ (f) $\bar{\gamma} = 0.03$

Figure 7: A Reconstructed Slice and Ray-cast Images