Lossless Compression of Volume Data

James E. Fowler† Roni Yagel‡
†Department of Electrical Engineering
‡Department of Computer and Information Science
The Ohio State University
Columbus, Ohio 43210

Abstract

Data in volume form consumes an extraordinary amount of storage space. For efficient storage and transmission of such data, compression algorithms are imperative. However, most volumetric datasets are used in biomedicine and other scientific applications where lossy compression is unacceptable. We present a lossless data-compression algorithm which, being oriented specifically for volume data, achieves greater compression performance than generic compression algorithms that are typically available on modern computer systems. Our algorithm is a combination of differential pulse-code modulation (DPCM) and Huffman coding and results in compression of around 50% for a set of volume data files.

I. Introduction

Compression for efficient storage and transmission of digital data has become routine as the application of such data has grown. Several common data-compression programs are readily available on many computers to fight the burgeoning demand for storage space. These programs are typically generic; that is, they can compress data regardless of its nature. It is common that a single program is used to compress not only English text, but also source code, executable binaries, and raw data. Although generic programs have some advantages, such as greater portability, it may be worthwhile, in cases in which the demand for storage space is particularly high, to consider compression algorithms that have been tailored specifically for a single application.

One such application is volume graphics, the extension of computer graphics that addresses rendering directly from volume data. Since volume size grows as the cube of the resolution, even a moderate-resolution volume requires a great amount of storage space. For example, consider a computer-tomography (CT) volume of $256 \times 256 \times 256$ voxels. If each voxel has 16 bits of precision, a total of 32 Mbytes is needed to store this volume. It is easy to see that a collection of several such volume images is unwieldy to maintain with today's technology.

In this paper, we present a data-compression algorithm which, being oriented specifically for volume data, will alleviate some of the storage-space problems associated with volume graphics. First, we present a very brief overview of general data-compression techniques. We then investigate previous attempts at volume compression. We follow this discussion with the details of our volume-compression algorithm. We conclude by comparing the performance of our algorithm on real volume data with that of several popular generic compression programs.

II. Data Compression Techniques

Data compression is the encoding of a body of data, D, as a smaller body of data, $\hat{D}$ [1]. There are two general types of data compression: *lossy* and *lossless*. In lossless data compression, it is possible to reconstruct exactly the original data D given $\hat{D}$. Such reconstruction is called *decompression*. Lossless compression is commonly used in applications, such as text compression, where the loss of even a single bit is unacceptable. On the other hand, in lossy compression, decompression produces only an approximation, $\tilde{D}$, to the original data D. Lossy compression is often used for applications such as image compression. Oftentimes with image compression, it is important that the decompressed image only *looks* like the original image; usually, it is not necessary to reproduce the image exactly, particularly if the image is being used for entertainment purposes. However, for applications such as medical imaging, lossy compression is unacceptable. Medical images are often used to identify tumors and other anomalies; distortion of the image due to compression may cause a false identification of a nonexistent tumor or the overlooking of a real one.

In this paper, we will limit ourselves to the consideration of only lossless compression, as lossy methods are generally unfit for data archiving. Moreover, volumes are used mainly in scientific applications where

Table 1: Voxel Entropies (in bits/voxel) of the Volume Dataset

Volume	entropy
3dknee.vox	9.58
CThead.vox	7.91
MRbrain.vox	7.93
3dhead.vox	8.65
sod.30.vox	3.28
hipiph.vox	10.29
rna.vox	7.97

great effort is invested in reconstructing high-quality volumes. These are used for diagnosis, treatment planning, and scientific observation for which lossy compression is unacceptable. Below, we discuss some of the more common lossless techniques. Storer provides a detailed discussion of these and other lossless-compression algorithms in [1].

III. Lossless Compression

One of the most common lossless compression techniques is entropy coding, which capitalizes on the basic tenant of information theory, entropy. Given a data source that outputs symbols from an alphabet, Σ, the entropy of the source is defined as

$$H = -\sum_{i=1}^{k} p_i \log_2 p_i \qquad (1)$$

where $k = |\Sigma|$ and p_i is the probability of occurence of the i^{th} symbol of Σ. It is assumed that the source outputs the current symbol independently of the past outputs. Shannon's fundamental source-coding theorem states that it is possible to code this source, without distortion, with an average of $H + \epsilon$ bits/symbol, where $\epsilon > 0$ is an arbitrarily small quantity [2]. The most common form of entropy encoding is Huffman encoding [3], which attempts to match the source entropy by assigning shorter codes to source symbols that occur frequently and longer codes to those symbols with small probability of occurence. Table 1 shows the entropy of the test volumes used later to generate results. These volumes are from the Chapel Hill volume-rendering test dataset and are taken from actual MRI, CT, and electron-density-map data.

Another common technique of lossless compression is the family of textural-substitution algorithms, of which Lempel-Ziv [4, 5, 6] is the most popular. The Lempel-Ziv (LZ) algorithm codes repeated variable-length substrings as fixed-length codes. LZ coding does not rely on prior knowledge of the source statistics but rather determines a "dictionary" of substrings from the source as coding progresses. For a well-behaved source, the dictionary will contain strings which have an almost equal probability of occurence. Thus, dictionary strings containing frequently used symbols are longer than those strings containing infrequent symbols [6]. For a sufficiently well-behaved source, LZ coding achieves the lower bit-rate bound given by the source entropy of (1).

These lossless compression techniques have been implemented in programs found on many UNIX systems. `gzip` and `zip` (and `PKZIP` on MSDOS systems) implement LZ compression as described above. `compress` implements LZW compression [6], a variant of the basic LZ algorithm. `pack` implements Huffman coding. Each of these programs is designed to compress computer files regardless of the type of data they contain. Below, we compare the performance of each of these programs with our volume-compression algorithm on several volume data files.

IV. Volume Compression

Despite its extraordinary size, much of the information in a volume array is redundant. Although all data-compression techniques exploit data redundancy to achieve compression, different techniques do so differently. Lossless compression techniques such as Huffman coding and LZ coding as discussed above can be thought of as removing "statistical redundancy" from the data. However, since we know that our data is in the form of a volume, and that volumes possess some local spatial coherence, we can remove this "spatial redundancy" as well to achieve greater compression.

There has been relatively little research reported that deals directly with the issue of lossless volumetric compression. One early scheme was developed for the archiving of CT-image volumes and used run-length encoding for very simple, yet very fast, lossless compression [7]. Compression of 40% was typical for this scheme; however, a significant amount of this compression was due to the fact that 12-bit voxel integers were stored with 16 bits each in the original volume.

There has been more interest in lossy volume compression. Much of the early work in this area stemmed from attempts to compress television images by considering the sequence of frames as a volume. The use of 3D transforms [8] is one approach that is equally applicable to general volumes as it is to television data. An example of this approach is [9], in which a Hadamard transform is used for real-time television compression.

More recently, there has been some interest in lossy compression coupled with volume rendering [10, 11, 12]. One proposal [10, 11] uses vector quantization to compress the volume, resulting in greater speed of the rendering process. Since vector quantization is lossy, this algorithm is not suitable for archival purposes. In [12], a discrete Hartley transform is used for compression. Again, this approach is lossy and is intended to expedite the rendering process. Below, we present an algorithm designed specifically for lossless compression of volume data that yields better compression than [7] and can be used for archiving. Additionally, it achieves greater compression than the generic algorithms discussed in the previous section.

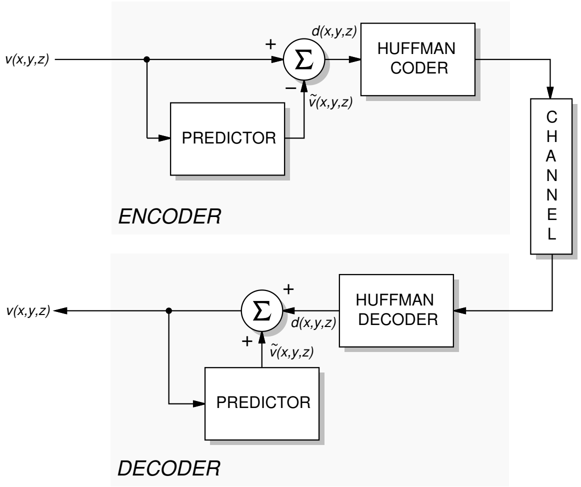

Figure 1: Block diagram of the volumetric compression algorithm

V. Algorithm Details

The algorithm we propose for volume compression is a combination of differential pulse-code modulation (DPCM) and Huffman coding. Both techniques have been used extensively in compression of 2D images; for an overview, see [8]. Theoretical performance of entropy encoding to improve DPCM is discussed in [13]. DPCM belongs to a family of compression algorithms known as predictive techniques. In a volume, adjacent voxels usually possess a high degree of spatial coherence; that is, adjacent voxel values are highly correlated. Predictive techniques exploit this spatial coherence and encode only the new information between successive voxels. Predictive techniques feature a *predictor* that calculates a value from previously encoded voxels. This *predicted value* is subtracted from the current voxel. Thus, only the new information provided by the current voxel is encoded.

Fig. 1 shows the block diagram of our volume-compression algorithm. The volume, $v(x,y,z)$, is read in raster-scan order. A predicted value, $\tilde{v}(x,y,z)$, is subtracted from each voxel value, creating a stream of difference values, $d(x,y,z)$, which are then Huffman coded. The *channel* shown in Fig. 1 is the archive system which is, in most cases, a disk drive.

A. The Predictor

The predictor is of the following form:

$$\begin{aligned}\tilde{v}(x,y,z) = \\ & a_1 v(x-1,y,z) + a_2 v(x,y-1,z) \\ & + a_3 v(x,y,z-1) \qquad (2)\end{aligned}$$

where $v(x,y,z)$ is the voxel array. Note that the predictor is *causal*; that is, the prediction is formed from only those values which have already been processed. Causality is required so that the decoder can track the operation of the encoder and generate the same predicted values as the encoder.

The predictor coefficients, a_i, are determined so that the predictor is optimal in the mean-square-error sense. We must calculate new preditor coefficients for each volume which we encode. The basic derivation of this calculation follows; more detail on optimal linear predictors can be found in [8].

To be optimal, the predictor must minimize the variance of the error sequence. The error sequence is

$$\epsilon(x,y,z) = v(x,y,z) - \tilde{v}(x,y,z) \qquad (3)$$

and its variance is

$$\rho^2 = E[\epsilon^2(x,y,z)] \qquad (4)$$

where $E[\cdot]$ is the probabilistic expectation operator. This minimization occurs when the error is orthogonal to the voxels upon which we are basing the prediction:

$$E[\epsilon(x,y,z)v(x-1,y,z)] = 0 \qquad (5)$$

$$E[\epsilon(x,y,z)v(x,y-1,z)] = 0 \qquad (6)$$

$$E[\epsilon(x,y,z)v(x,y,z-1)] = 0 \qquad (7)$$

After some algebra, we arrive at:

$$\begin{bmatrix} r(0,0,0) & r(1,-1,0) & r(1,0,-1) \\ r(-1,1,0) & r(0,0,0) & r(0,1,-1) \\ r(-1,0,1) & r(0,-1,1) & r(0,0,0) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} r(1,0,0) \\ r(0,1,0) \\ r(0,0,1) \end{bmatrix} \quad (8)$$

where $r(x,y,z)$ is the covariance of $v(x,y,z)$,

$$r(x,y,z) = E[v(x',y',z')v(x'-x,y'-y,z'-z)] \quad (9)$$

and x', y', and z' are dummy variables over which the expectation is performed. Once we have determined the covariance, the predictor coefficients, a_i, are determined by solving the linear system of (8).

In presenting this derivation, we make several assumptions. First, we assume that $v(x,y,z)$ has zero mean. In general, real volume data will have a non-zero mean which we will have to subtract from $v(x,y,z)$ before we use (2). Secondly, we assume that the volume is *wide-sense stationary*; that is, the statistics (mean and covariance) do not vary throughout the volume. For real volume data, this stationarity does not hold – we will return to this point later.

The predictor presented here is optimal in the mean-square-error sense. In many cases, this optimality may be overkill; good results may oftentimes be obtained from a simple predictor whose coefficients are fixed in advance. One such simple predictor is obtained by setting $a_1 = a_2 = a_3 = \frac{1}{3}$, which works reasonably well in practice if the voxel-sampling lattice is isotropic; i.e, the voxels are cubic. The advantage of the optimal predictor is that it will automatically compensate for non-isotropic sampling. In addition, calculation of the optimal predictor is not very costly, as it takes only about 6% of the total compression time.

For simplicity, we have designed our predictor using causal voxels from only the 6-neighborhood; i.e. those voxels that share a face with the current voxel. Alternatively, one could include voxels from the 18- or 26-neighborhoods (respectively, those voxels sharing an edge or a vertex with the current voxel). As these voxels are more distant from the current voxel, they will be less effective in prediction; however, further study is warranted to determine whether the improvement they can offer to the prediction is worth the additional complexity of calculating more predictor coefficients.

B. The Huffman Coder

An algorithm for the construction of a codebook for Huffman coding is presented in [8]. In our volume-compression algorithm, we use a variant of Huffman coding called *truncated Huffman coding* [8]. If the number of possible source symbols is L, the longest possible Huffman code can have as many as L bits. Thus, in practical situations that require a large L, a truncated Huffman code is used. In truncated Huffman coding, the first L_1 symbols ($L_1 < L$) are Huffman coded and the remaining $L - L_1$ symbols are assigned a fixed-length code. Thus, the maximum code length is restricted to a manageable length. For a volume with 16 bits per voxel, $L = 131,071$ (16 bits/voxel implies difference values in the range -66535 to 65536). For the lack of a suitable heuristic for choosing L_1, we arbitrarily set $L_1 = 256$.

C. The Compression Algorithm

Our volume compression algorithm is given below. To compress a given volume of size XYZ voxels, the following steps are performed:

1. Scan the volume to estimate the mean:

$$\overline{v} = \frac{1}{XYZ} \sum_{x=0}^{X-1} \sum_{y=0}^{Y-1} \sum_{z=0}^{Z-1} v(x,y,z) \quad (10)$$

2. Scan the volume to estimate the covariance:

$$\begin{aligned} r(x,y,z) = {} & \\ & \frac{1}{(X-2)(Y-2)(Z-2)} \times \\ & \sum_{x'=1}^{X-2} \sum_{y'=1}^{Y-2} \sum_{z'=1}^{Z-2} \left[\left(v(x',y',z') - \overline{v}\right) \times \right. \\ & \left. \left(v(x'+x,y'+y,z'+z) - \overline{v}\right) \right] \quad (11) \end{aligned}$$

for $x, y, z \in \{-1, 0, 1\}$.

3. Calculate the predictor coefficients, a_1, a_2, and a_3, from (8) using the estimated covariance values.

4. Scan the volume using the predictor to calculate the difference values, $d(x,y,z)$:

$$\begin{aligned} d(x,y,z) &= v(x,y,z) - \tilde{v}(x,y,z) && (12) \\ \tilde{v}(x,y,z) &= a_1 v_1(x,y,z) + a_2 v_2(x,y,z) + \\ & \quad a_3 v_3(x,y,z) && (13) \end{aligned}$$

$$v_1(x,y,z) = \begin{cases} 0 & \text{if } x = 0 \\ v(x-1,y,z) - \overline{v} & \text{else} \end{cases} \quad (14)$$

$$v_2(x,y,z) = \begin{cases} 0 & \text{if } y = 0 \\ v(x,y-1,z) - \overline{v} & \text{else} \end{cases}$$

$$(15)$$

$$v_3(x,y,z) = \begin{cases} 0 & \text{if } z = 0 \\ v(x,y,z-1) - \overline{v} & \text{else} \end{cases} \quad (16)$$

5. Count the number of times $d(x,y,z) = s_i$ for each possible s_i. These counts are the estimates of the symbol probabilities p_i.
6. Determine a truncated Huffman code table using the p_i probabilities.
7. Scan the volume once again, using the predictor to calculate the difference values, and the Huffman code table to encode the difference values, producing the compressed volume.

In the compressed volume, it is necessary to store the following, in addition to the coded difference values:

- The estimated mean, $\overline{v}$.
- The three predictor coefficients, a_1, a_2, and a_3.
- The Huffman code table, which consists of the source symbols (difference values), s_i, and the assigned Huffman codes.

One will notice that, in the course of compressing a volume, the compression algorithm scans the volume four times. The first two scans of the volume are used to estimate the mean and variance, respectively, of the voxels. Theoretically these two estimations could be done simultaneously in one scan; however, greater mathematical precision is obtained in the estimate of the variance when the mean is already available. Since the mean and variance statistics are needed only for the optimal predictor, if one chooses to use the simple predictor as discussed previously, these two scans for statistics calculation are unnecessary. The last two scans of the volume are much more computationally intensive than the first two. The last scan calculates the same difference values that were calculated during the third scan of the volume; however, to prevent excessive memory costs, these differences values are not saved, making a fourth scan of the volume necessary.

D. The Decompression Algorithm

The decompression algorithm is as follows:

1. Read the estimated mean, $\overline{v}$, and the predictor coefficients, a_1, a_2, and a_3, from the compressed file.
2. Read the Huffman table from the compressed file.
3. For each voxel in the reconstructed volume $v(x,y,z)$, calculate the predicted value $\tilde{v}(x,y,z)$ using (13). Read the next Huffman code from the compressed file and look up the difference value, $d(x,y,z)$, in the Huffman table for that code. Reconstruct the current voxel:

$$v(x,y,z) = \tilde{v}(x,y,z) + d(x,y,z) \quad (17)$$

Table 2: Exectution Times for the Compression Programs on `3dhead.vox`. These times are for a HP 9000 Series 735.

Method	Time (min) Compress	Decompress
`compvox`	4.7	1.4
`compress`	0.4	0.2
`zip`	0.8	0.5
`gzip`	1.4	0.2
`pack`	0.2	0.3

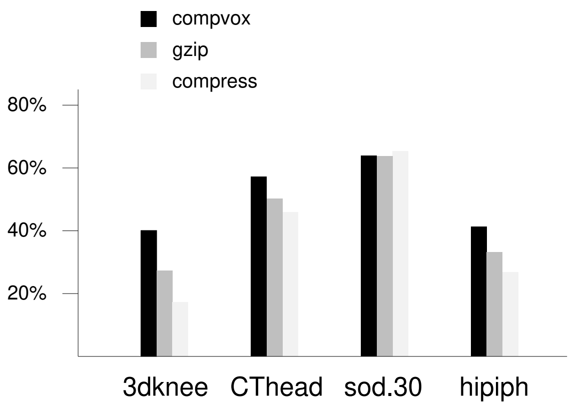

Figure 2: Percentage compression of several data-compression programs on volume data

VI. Results

Table 2 compares the execution times for the different compression programs on one volume. Times for both compression and decompression are given.

Table 3 compares the compression performance of our volume compression algorithm, `compvox`, with the `compress`, `zip`, `gzip`, and `pack` programs. Our algorithm achieves the greatest compression for all the volumes except one. The volumes used here are from the Chapel Hill volume-rendering test dataset and are taken from actual MRI, CT, and electron-

Table 3: Percentage Compression of Several Data-Compression Programs on Volume Data

	Original Data		`compress`	`zip`	`gzip`	`pack`	`compvox`
Volume	Resolution	Size (Mb)	% Comp.	% Comp.	% Comp.	% Comp.	% Comp.
3dknee.vox	$256 \times 256 \times 127$	15.9	17.2%	24.9%	27.3%	27.3%	40.1%
CThead.vox	$256 \times 256 \times 113$	14.1	46.0%	48.9%	50.3%	35.9%	53.3%
MRbrain.vox	$256 \times 256 \times 109$	13.6	40.2%	43.2%	45.0%	37.2%	52.7%
3dhead.vox	$256 \times 256 \times 109$	13.6	29.9%	33.8%	36.1%	32.0%	48.5%
sod.30.vox	$97 \times 97 \times 116$	1.0	65.3%	62.3%	63.8%	58.4%	63.7%
hipiph.vox	$64 \times 64 \times 64$	0.5	26.8%	28.5%	33.2%	20.7%	41.9%
rna.vox	$100 \times 120 \times 16$	0.2	33.1%	35.8%	37.5%	35.3%	55.4%
Average			36.9%	39.6%	41.9%	35.3%	50.8%

density-map data. Using `compvox` on the entire collection of these volumes saves 9 megabytes over `compress`. Fig. 2 is a bar graph of some of the data of Table 3 comparing the performance of `compvox`, `gzip`, and `compress` for all the volumes.

VII. Conclusions

In this paper, we have presented an algorithm that is designed specifically for the compression of volume data. From the results presented in Tables 2 and 3, one could conclude that, although our volume-compression algorithm yields compression superior to generic compression routines such as `compress`, `zip`, `gzip`, and `pack`, it requires significantly more processing time. However, the current implementation of `compvox` has not been optimized for execution speed; rather, it is prototype code that has been made flexible to facilitate modifications during the development of the algorithm. Greater attention to speed issues would make our code more competitive with the generic routines.

As discussed previously, the optimal predictor that we design for each volume assumes that the statistics (mean and covariance) are wide-sense stationary throughout the volume. In any real volume, the best we can obtain is only some local stationarity; that is, the statistics are invariant only over small portions of the volume. One could obtain better predictions (and, thus, better compression) by designing a number of different predictors, each optimized to a particular section of the volume, rather than choosing one predictor for the entire volume. We propose dividing the volume into an octtree, as is done in [14]. A separate predictor would be designed for each leaf node. A suitable criterion would determine the sizes of the leaf nodes based on the following tradeoff. With small leaf nodes, the predictors will be better able to exploit local stationarity. However, if the leaves are too small, not enough data would be available to precisely estimate the local voxel statistics. Somewhat similar approaches, wherein the predictor is changed as local statistics change, have been developed for 2D image compression; these approaches fall under the general heading of adaptive DPCM and are discussed in [8].

Localization of the predictor is one avenue we are currently pursuing to improve the compression performance of our algorithm. Additionally, we are examining whether increasing the neighborhood of the predictor from the 6-neighborhood to the 18- or 26-neighborhood will appreciably improve performance, and a heuristic for determining the L_1 value for the truncated Huffman code is under exploration.

Acknowledgemets

James E. Fowler is supported by a PhD Scholarship from AT&T. This research was supported by the National Science Foundation under grant CCR-9211288. We thank the University of North Carolina at Chapel Hill for the use of the volumetric datasets.

References

[1] J. A. Storer, *Data Compression: Methods and Theory.* Principles of Computer Science Series, Rockville, Maryland: Computer Science Press, 1988.

[2] C. E. Shannon, "A mathematical theory of communication," in *Key Papers in The Development of Information Theory* (D. Slepian, ed.), pp. 5–18, New York: IEEE Press, 1948.

[3] D. A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes," *Proceedings of the IRE*, vol. 40, pp. 1082–1101, 1952.

[4] J. Ziv and A. Lempel, "Compression of Individual Sequences Via Variable-Rate Coding," *IEEE Transactions on Information Theory*, vol. 24, no. 5, pp. 530–536, 1978.

[5] J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression," *IEEE Transactions on Information Theory*, vol. 23, no. 3, pp. 337–343, 1977.

[6] T. A. Welch, "A Technique for High-Performance Data Compression," *Computer*, vol. 17, pp. 8–19, June 1984.

[7] M. L. Rhodes, J. F. Quinn, and B. Rosner, "Data Compression Techniques for CT Image Archiving," *Journal of Computer Assisted Tomography*, vol. 7, pp. 1124–1126, December 1983.

[8] A. K. Jain, *Fundamentals of Digital Image Processing.* Englewood Cliffs, NJ: Prentice Hall, 1989.

[9] S. C. Knauer, "Real-Time Video Compression Algorithm for Hadamard Transform Processing," *IEEE Transactions on Electromagnetic Compatibility*, vol. EMC-18, pp. 28–36, February 1976.

[10] P. Ning and L. Hesselink, "Fast Volume Rendering of Compressed Data," in *Proceedings of IEEE Visualization*, (San Jose), pp. 11–18, October 1993.

[11] P. Ning and L. Hesselink, "Vector Quantization for Volume Rendering," in *Proceedings of the Boston Workshop on Volume Visualization*, pp. 69–74, ACM Press, October 1992.

[12] C. Tzi-Cker, H. Taosong, A. Kaufman, and H. Pfister, "Compression Domain Volume Rendering," 1994. Technical Report 94.01.04, SUNY at Stony Brook.

[13] J. B. O'Neal, "Differential Pulse-Code Modulation with Entropy Coding," *IEEE Transactions on Information Theory*, vol. IT-22, pp. 169–174, March 1976.

[14] D. Laur and P. Hanrahan, "Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering," *Computer Graphics*, vol. 25, pp. 285–288, July 1991.