Integrated Volume Compression and Visualization

Tzi-cker Chiueh, Chuan-kai Yang
Experimental Computer System laboratory
Taosong He
Bell Laboratories of Lucent Technologies
Hanspeter Pfister
Mitsubishi Electric Research Laboratory
Arie Kaufman
Center for Visual Computing

Deparment of Computer Science
State University of New York at Stony Brook
Stony Brook, NY 11794-4400

Abstract

Volumetric data sets require enormous storage capacity even at moderate resolution levels. The excessive storage demands not only stress the capacity of the underlying storage and communications systems, but also seriously limit the speed of volume rendering due to data movement and manipulation. A novel volumetric data visualization scheme is proposed and implemented in this work that renders 2D images directly from compressed 3D data sets. The novelty of this algorithm is that rendering is performed on the compressed representation of the volumetric data *without pre-decompression*. As a result, the overheads associated with both data movement and rendering processing are significantly reduced. The proposed algorithm generalizes previously proposed whole-volume frequency-domain rendering schemes by first dividing the 3D data set into subcubes, transforming each subcube to a frequency-domain representation, and applying the Fourier Projection Theorem to produce the projected 2D images according to given viewing angles. Compared to the whole-volume approach, the subcube-based scheme not only achieves higher compression efficiency by exploiting local coherency, but also improves the quality of resultant rendering images because it approximates the occlusion effect on a subcube by subcube basis.

CR Categories and Subject Descriptors: I.3.8 [Computer Graphics]: Applications; I.4.5 [Image Processing]: Reconstruction.
Additional Keywords: Volume Compression, Fourier Projection Theorem, Discrete Hartley Transform, Image Compositing.

1 INTRODUCTION

With the advent of 3D medical imaging devices such as Computer Tomography (CT), Nuclear Magnetic Resonance (NMR), and 3D ultrasound systems, more and more medical diagnostic data are now represented in a volumetric format. Meanwhile, scientists and researchers develop a large variety of computational models to study and understand various physical phenomenon. Results from these numerical simulations typically reflect certain aspects of the underlying physical world and therefore inherently exhibit a 3D structure. An important characteristic of volumetric data sets is their very large storage requirements. As an example, a 1024x1024x1024 volume with each voxel represented by 24 bits will require 3 Gbytes of storage space. Excessive storage demands exact the underlying I/O and communications subsystems, as well as lengthen the end-to-end rendering delay. Compression is one possible solution toward this problem.

Although faster CPUs help mitigate the overhead associated with compression/decompression, the compressed data set, when explicitly decompressed, still incurs significant data movement overhead on the memory bus (not I/O bus), which is proportional to the size of the uncompressed data set. Because the performance of modern RISC processors is critically dependent upon the reduction of main memory traffic, it is essential to maintain the data in the compressed form as long as possible. This paper proposes an algorithm that performs volume rendering directly on compressed data sets, thus avoiding the decompression step at run time and its associated performance overhead.

This algorithm is a generalization of whole-volume frequency-domain 3D rendering algorithm independently developed by Dunne, Napel, and Rutt [DNR90], and Malzbender and Kitson [MK93, Mal93], both of which in turn are based on the *Fourier Projection Theorem*. We take a cube-based approach by first subdividing the data volume into subcubes, then applying a Fourier-like transform to each of these subcubes, and finally quantizing the resulting coefficients according to the dynamic range of the coefficient values in the subcubes. To implement spatial-domain ray casting [Sab88], we apply the *Fourier Projection Theorem* on the quantized frequency-domain representation of each subcube, and perform spatial-domain compositing by treating each subcube as an indivisible macro-voxel with its aggregate opacity and intensity values.

None of the previous works [NH92, NH93, YL95, FY94] in this area allows direct rendering from compressed data without decompression. Compared to these efforts, the proposed integrated volume compression and visualization algorithm exhibits the following two advantages:

High compression efficiency: Transform coding compacts the energy of the spatial-domain signal to few transform-domain coefficients. Consequently, higher compression efficiency is expected compared to other compression approaches. In addition, the proposed subcube-based approach can exploit the coherency in local regions more effectively than the whole-volume approach [DNR90] [Mal93] and therefore attain better compression ratio.

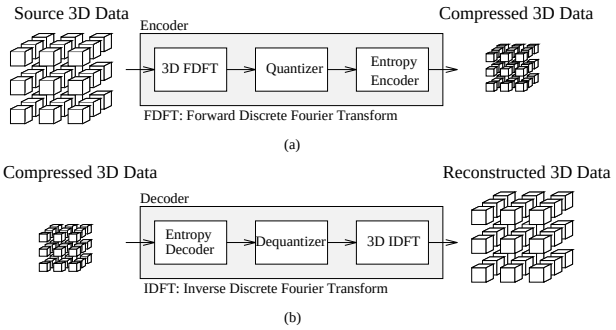

Figure 1: *The data flows of the proposed volume (a) compression and (b) decompression algorithms.*

Better rendering quality: The image quality of whole-volume frequency-domain volume rendering is usually poor because it simulates the interaction between light and voxels by a simple line integral along the view direction across the volume. The result is essentially a X-ray-like image without any occlusion effects. Totsuka and Levoy [TL93] proposed a linear approximation to the exponential attenuation [Sab88] and an alternative shading model to fit the computation within the frequency-domain rendering framework. Although their images show improved visual depth cues, the effect of the linear attenuation model and the lack of occlusion is still noticeable. We address these problems by first applying the Fourier Projection Theorem on a subcube-by-subcube basis to derive the aggregate intensity and opacity values for each subcube; then we use spatial compositing (e.g., [Lev88, Wes90]) to combine these intensity and opacity values according to an exponential attenuation model [Sab88]. The resulting images thus show improved occlusion and attenuation effects.

The rest of this paper is organized as follows. In Section 2, the basic algorithm for volume data compression is presented. In Section 3, the concept of Fourier volume rendering is introduced and the proposed compression-domain volume rendering algorithm is described in detail. That section also presents different possibilities of generating occlusion effects and non-linear attenuation in subcube-based frequency domain rendering. The results are presented in Section 4. Section 5 concludes this paper by summarizing the main results and outlining plans for future work.

2 VOLUME DATA COMPRESSION

The proposed volume data compression algorithm is based on transform coding and is actually a generalization of the JPEG still image compression algorithm [Wal91] to the 3D case, with one important exception: the transform is a discrete Fourier transform rather than a discrete cosine transform (DCT).

Figure 1 shows the data flow of the proposed compression/decompression algorithm. An $N \times N \times N$ volume is decomposed into $M \times M \times M$ subcubes, each of which first goes through a 3D discrete Fourier transform as follows:

$$F(u,v,w) = \frac{1}{M^3} \sum_{x=0}^{M-1} \sum_{y=0}^{M-1} \sum_{z=0}^{M-1} f(x,y,z) e^{-2\pi i \frac{xu+yv+zw}{M}} \quad (1)$$

For simplicity and without loss of generality we assume that N is an integral multiple of M. Each of the 3D Fourier coefficients in each subcube is then quantized using the equation

$$F^Q(u,v,w) = Round\left(\frac{F(u,v,w)}{Q(u,v,w)}\right) \quad (2)$$

where $Q(u,v,w)$ is the quantization table entry corresponding to $F(u,v,w)$ and represents the quantization step. The resulting 3D quantized frequency coefficients within a cube are organized as a linear sequence through a 3D zig-zag order. A 3D zig-zag order through a cube is the traversal order in which (u_1, v_1, w_1) precedes (u_2, v_2, w_2) if $u_1+v_1+w_1 < u_2+v_2+w_2$, and the (u,v,w) tuples on the plane $u+v+w = K$ follow a 2D zig-zag order. The resulting linear sequence of Fourier transform coefficients is fed into an entropy encoder that in turn consists of run-length coding and Huffman coding. Decompression is done by reversing the above process.

Because the above compression algorithm is based on transform coding, it is inherently lossy. Most of the compression gain is realized by the quantization step, with subsequent steps corresponding to a lossless compression scheme. In fact, manipulating the contents of the quantization table is the most effective way of making the tradeoff between compression ratios and reconstruction errors. As for the compression/decompression overhead, the entropy encoding step involves only table look-up and bit compaction/expansion and therefore accounts for a small portion of the overall delay. Because the entropy encoding step plays a rather minor role in the space and time performance of the proposed volume compression scheme, we will use the term *compressed volume data* to refer to the sets of quantized 3D Fourier coefficients from the subcubes of a volume data set.

In the case of JPEG, a recommended set of quantization tables have been developed through extensive testing and measurements of a large number of images. Because there isn't much previous research on volume data compression, a theoretical understanding of a good choice of $Q(u,v,w)$'s in Equation 2, i.e., the entries in the quantization table, is still lacking. Ideally it is preferable to take advantage of the varying perceptive significance of different frequency coefficients by allocating to each of them a different number of bits. For example, Table 1 shows the variance distribution of the 3D Fourier coefficients using 4x4x4 cubes as basic units. The fact that few coefficients have a much larger magnitude than the others demonstrate the energy concentration capability of transform coding. Presumably one could base each coefficient's number of representation bits on the size of its corresponding variance. However, the fact that our experience in the interaction between volume rendering and volume data compression is rather limited forces us to adopt a simpler approach.

In our implementation we use the same quantization step $Q(u,v,w)$ for all coefficients in a subcube. We calculate the dynamic range R_i of coefficients in each subcube i as

$$R_i = |max\_value_i - min\_value_i| \quad (3)$$

where max_value_i and min_value_i are the maximum and minimum AC coefficient values of the i-th subcube, respectively. The quantization step for the i-th subcube is then chosen to be

$$Q(u,v,w) = \frac{R_i}{2^c}, \qquad \text{u,v,w = 0, 1,..., M - 1} \quad (4)$$

with c being a fixed constant. We found that the total Mean Square Error, defined as

$$MSQE = E[F(u,v,w) - F^Q(u,v,w)]^2 \quad (5)$$

	z = 0 x →					z = 1 x →			
y ↓	119.13	0.99	0.39	0.45	y ↓	19.21	0.56	0.40	0.83
	11.58	1.30	1.01	0.06		9.00	0.49	0.25	0.48
	8.89	0.46	0.24	0.47		8.97	0.07	0.25	2.62
	18.94	0.24	0.00	2.06		14.70	0.82	0.22	1.06
	z = 2 x →					z = 3 x →			
y ↓	11.58	0.56	0.17	0.83	y ↓	11.58	0.84	0.17	0.42
	18.76	2.16	0.00	0.22		23.44	3.56	0.28	0.00
	14.77	0.00	0.21	3.59		14.77	0.94	0.21	0.93
	6.26	1.25	0.00	0.07		9.07	0.48	0.24	0.48

Table 1: *Typical variance distribution of the 3D Fourier coefficients.*

Subdivision	Original Size	Compressed Size	Compression Ratio
High-potential iron protein molecule			
$2 \times 2 \times 2$	287,496	44,967	6 : 1
$4 \times 4 \times 4$	287,496	9,255	31 : 1
$8 \times 8 \times 8$	287,496	3,328	86 : 1
Lobster (CT dataset)			
$4 \times 4 \times 4$	3,481,600	108,045	32 : 1
$8 \times 8 \times 8$	3,481,600	36,047	96 : 1

Table 2: *Achievable compression efficiency for different datasets using different subcube size.*

was acceptable for $c \geq 10$, where $E[]$ is the expectation operator. Table 2 shows the compression ratios for different subcube sizes using the above quantization strategy.

3 RENDERING FROM COMPRESSED VOLUME

Given the compressed volume data, i.e., the Fourier coefficients of the volume's subcubes, one can apply the Fourier Projection Theorem to compute the projection image of each subcube, and performs a spatial-domain compositing by treating each subcube as a macro-voxel with its own color and opacity values derived from the projection image. Figure 2 shows the basic rendering process.

3.1 Generation of Subcube Projection Image

To derive a subcube's projection image from its compressed representation, we use a new class of volume rendering algorithms [DNR90, MK93, Mal93, Lev92, TL93] that are based on the Fourier Projection Theorem, which was originally developed for Computer Tomography (CT) 3D reconstruction.

To facilitate the exposition of the theorem [MO74], we start with the 2D case and then generalize the theorem to the 3D case. Given a 2D spatial distribution $f(x,y)$, the 1D projection of $f(x,y)$ along a line specified as $x\cos\theta + y\sin\theta = R$ is a line integral and is given by

$$\begin{aligned} g(\theta,R) &= \int\int f(x,y)\delta(x\cos\theta + y\sin\theta - R)\,dxdy \\ &= \int_0^{2\pi}\int_0^{\infty} f(r,\phi)\delta[r\cos(\theta-\phi) - R]\,r\,d\phi dr \end{aligned} \quad (6)$$

The term $g(\theta,R)$ with a fixed θ stands for the projection of $f(x,y)$ along the direction that is at an angle $\theta - \frac{\pi}{2}$ with respect to the X axis. To see how this is related to the 2D Fourier transform $F(u,v)$ of $f(x,y)$, we start with the definition

$$F(u,v) = \int\int f(x,y)e^{-i2\pi ux}e^{-2\pi vy}dxdy \quad (7)$$

Expressing $F(u,v)$ in polar coordinates with $u = \rho\cos\beta, v = \rho\sin\beta$ gives

$$F(\rho,\beta) = \int\int f(x,y)e^{-i2\pi\rho(x\cos\beta + y\sin\beta)}dxdy \quad (8)$$

By exploiting the property of a $\delta(.)$ function, this can be transformed to

$$\begin{aligned} & F(\rho,\beta) \\ &= \int\int\int f(x,y)\times \\ &\quad \delta(x\cos\beta + y\sin\beta - R)e^{-i2\pi\rho R}dxdydR \\ &= \int g(\beta,R)e^{-i2\pi\rho R}dR \end{aligned} \quad (9)$$

Therefore,

$$F(\rho,\beta) = \mathcal{F}_{1D}[g(\beta,R)] \quad (10)$$

where $\mathcal{F}_{1D}$ represents a 1D Fourier transform operator. In other words, the 1D Fourier transform of a 1D projection of $f(x,y)$ along the direction specified by the angle β, is a 1D line across the center of the 2D Fourier transform of $f(x,y)$ whose normal vector is along the direction specified by the angle β. The generalization of this theorem to the 3D case is straightforward: The 2D Fourier transform of a 2D projection of a 3D distribution $f(x,y,z)$ along the direction specified by the angle β, is a 2D plane across the center of the 3D Fourier transform of $f(x,y,z)$ whose normal vector is along the direction specified by the angle β.

Discrete cosine transform (DCT) exhibits better energy compaction properties and is used in the JPEG still picture compression standard [RY90, Wal91]. However, the fact that the DCT does not possess the separability property, i.e.,

$$\cos(x)\times\cos(y)\times\cos(z) \neq \cos(x+y+z) \quad (11)$$

prevents a DCT-based projection theorem. In the above derivation we make use of this property to go from Equation 7 to Equation 8, but it no longer holds if the transform used is a DCT.

The Fourier Projection Theorem also applies to discrete Hartley transform (DHT). The 3D DHT is defined as

$$\begin{aligned} & H(u,v,w) = \\ & \quad S\times\textstyle\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}\sum_{z=0}^{N-1} f(x,y,z)\times \\ & \quad [\cos(2\pi\tfrac{ux+vy+wz}{N}) + \sin(2\pi\tfrac{ux+vy+wz}{N})] \end{aligned} \quad (12)$$

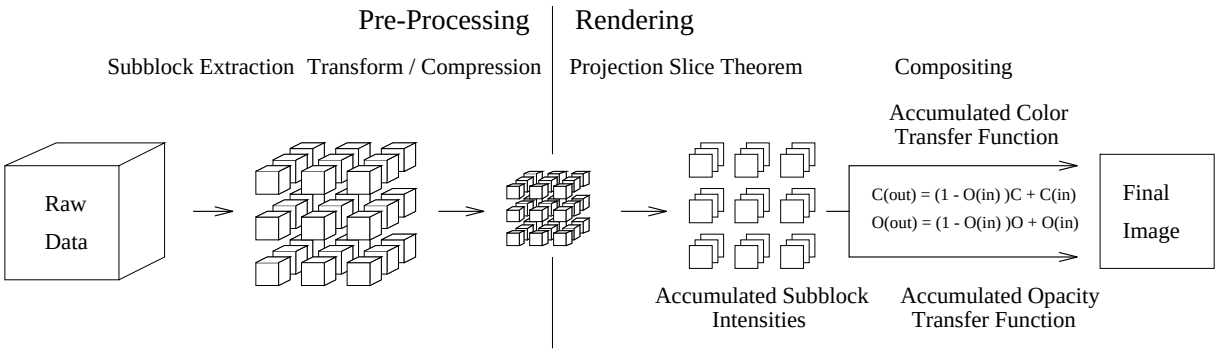

Figure 2: *Integrated volume data compression and rendering. Each subcube is first independently Fourier-transformed. Then the projection image of each subcube is computed through the Fourier Projection Theorem. The final rendered image results from compositing the subcubes' projection images according to the view angle and opacity/color transfer functions.*

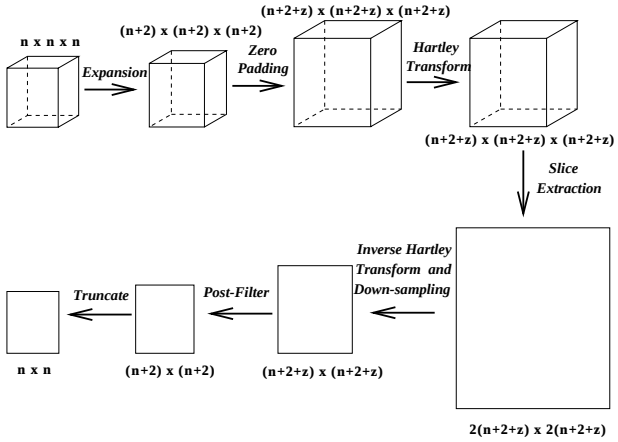

Figure 3: *The evolution of a subcube as its projection image is generated through the Fourier Projection Theorem.*

where S is $\frac{1}{N^3}$ for the forward and 1 for the inverse transform. Because the DHT requires only real and no imaginary number manipulation and assumes the same format for forward and reverse transformations, it is used in our algorithm implementation.

To apply the Fourier Projection Theorem in the context of the proposed algorithm, we need to address the following two issues. First, the original formulation is in the continuous domain whereas our algorithm is supposed to work in the discrete domain. Second, because our algorithm applies the Fourier Projection Theorem on a subcube by subcube basis, it is critical to eliminate the aliasing effect around the boundary region of the subcube's projection image.

Figure 3 shows how a subcube in the original data volume evolves before its projection image is generated. Before being transformed into the frequency-domain representation, an $n \times n \times n$ subcube is first expanded into $(n+2) \times (n+2) \times (n+2)$ subcube by including voxels from its neighboring subcubes. In the case of boundary subcubes, zero padding is assumed. This expansion provides overlap between the projection images from neighboring subcubes so that during spatial-domain compositing those rays that traverse through the boundaries of subcubes always have the same number of projection values for interpolation as the other rays. Then each subcube is zero-padded and grows to $(n+2+Z) \times (n+2+Z) \times (n+2+Z)$. This zero padding provides a protection zone for the original data against aliasing, which tends to degrade the boundary region. In the course of applying the Fourier Projection Theorem, a slice of the dimension $2(n+2+Z) \times 2(n+2+Z)$ is extracted from the $(n+2+Z) \times (n+2+Z) \times (n+2+Z)$ frequency subcube, with zero filled in those pixels that are not defined in the subcube. The reason that the extracted slice is twice the size along each dimension is to accommodate the fact that the projection image along a non-orthonormal direction is larger than those along the orthonormal directions, which are of the same dimension as the subcube. For example, the support of a 1D projection across a $X \times X$ 2D block is X if the projection angle is orthonormal, i.e., multiples of $\frac{1}{2}\pi$. However, if the projection angle is non-orthonormal, say $\frac{1}{4}\pi$, then the support of the 1D projection is $\sqrt{2}X$. It can be shown that doubling the dimension of the extracted slice is sufficient to accommodate all possible projection angles.

Because zero padding in the frequency domain corresponds to super-sampling in the spatial domain, the subcube's projection image is computed by taking a 2D inverse transform of the extracted slide every other sampling point along each dimension, and the resultant dimension is $(n+2+Z) \times (n+2+Z)$. Next a post-filtering step is applied to this projection image to eliminate the aliasing effect and arrive at a $(n+2) \times (n+2)$ image, of which only the $n \times n$ portion is contributed by the subcube itself.

3.2 Summation of Subimages

The proposed compression scheme applies the Fourier Projection Theorem not on the whole volume but on individual subcubes. In this section we show that the summation of the resulting subimages is equivalent to whole-volume frequency domain rendering, and the next section describes how spatial compositing between subimages can include the occlusion effect through opacity transfer functions.

To simplify the explanation, we again describe the algorithm in the context of 2D images. From 2D to 3D, it is just a straightforward generalization. In Figure 4 (a), a rectangular 2D image $\mathbf{I}$ is projected along the direction represented by the arrows. According to the Fourier Projection Theorem, the projection of $\mathbf{I}$ along the projection angle θ is a 1D signal, which is the inverse Fourier transform of the line that crosses the center of the 2D Fourier transform of $\mathbf{I}$ also at the angle θ. Mathematically,

$$P_\theta(\mathbf{I}) = \mathcal{F}^{-1}(X_\theta(\mathcal{F}(\mathbf{I}))) \tag{13}$$

where the X_θ operator means extracting the line that crosses the

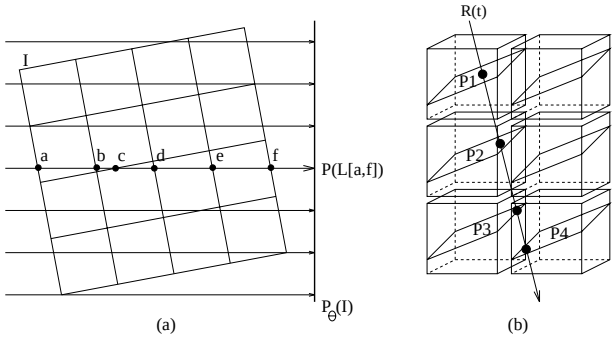

Figure 4: *Summation of projection sums from (a) each block in the 2D case and (b) from each subcure in the 3D case along the viewing ray.*

center at angle θ. Now let us decompose the image into 16 subimages, $\mathbf{SI}_{ij}$, where $i, j = 0, 1, 2, 3$, and apply the Fourier Projection Theorem to each of them. Consider a particular projection ray, say L in the figure, and let us denote the line integral of $\mathbf{I}$ between a segment of L as $P(L[m,n])$, where m and n represent points on L. By definition,

$$\begin{aligned} P(L[a,f]) &= P(L[a,b]) + P(L[b,c]) + P(L[c,d]) + \\ &\quad P(L[d,e]) + P(L[e,f]) \end{aligned} \tag{14}$$

where $P(L[a,f])$ represents a point in the 1D signal $P_\theta(\mathbf{I})$. Similarly, one can view $P(L[a,b])$ as a point in the projection of the subimage $\mathbf{SI}_{01}$, i.e., in $P_\theta(\mathbf{SI}_{01})$, and the same interpretation applies to the other segments of L. Because both Fourier transform and the projection operation are linear operators, $P_\theta(\mathbf{I})$ can thus be rewritten as

$$\begin{aligned} P_\theta(\mathbf{I}) &= \sum_{i=0}^{N}\sum_{j=0}^{M} P_\theta(\mathbf{SI}_{ij}) \\ &= \sum_{i=0}^{N}\sum_{j=0}^{M} \mathcal{F}^{-1}(X_\theta(\mathcal{F}(\mathbf{SI}_{ij}))) \end{aligned} \tag{15}$$

where N and M equal three in our example, but in general they depend on how the image is decomposed into blocks. What Equation 15 says is that one can apply the Fourier Projection Theorem to each of the subimages and sum the projections up, to compute the projection of a 2D image.

In the 3D case we cast a 2D array of rays into the data volume along the view angle to get the projected image. Figure 4 (b) shows a viewing ray $R(t)$ perpendicular to the view plane intersecting the image planes of the subcubes at points P1 through P4. Each image plane is generated using the Fourier Projection Theorem on the subcube, and the sum of all projected intensities $\sum_{i=1}^{4} P_i$ is equivalent to the projection of the whole data volume along the ray $R(t)$.

3.3 Compositing of Subimages

Most contemporary volume rendering algorithms use more sophisticated modulation schemes that take into account the physical affects of light absorption, scattering, and shading [Sab88]. Let us assume that each voxel of the data volume is characterized by a density $\nu(x,y,z)$ and a color value $\gamma(x,y,z)$ such that the voxel can emit light with an energy of $\nu\gamma$ per unit length and can absorb light with an opacity of $\nu\tau$ per unit length, where τ is the attenuation constant. Consider a viewing ray $R(t)$ that shoots through the data volume and is parameterized by t with $R(a)$ and $R(b)$ as the start and end points. If one ignores the scattering effect, the total energy arriving at $R(b)$ due to emission and absorption along the ray from $R(a)$ to $R(b)$ is given by

$$I_{R(b)} = \int_a^b \nu(R(t))\gamma(R(t)) e^{[-\int_a^t \tau\nu(R(u))\,du]}\,dt \tag{16}$$

If one simplifies the illumination model further by ignoring the absorption effect, then the equation is reduced to

$$I_{R(b)} = \int_a^b \nu(R(t))\gamma(R(t))\,dt \tag{17}$$

This can be computed by directly applying the Fourier Projection Theorem because each projection image pixel is the result of a line integration along the viewing ray. However, the Fourier Projection Theorem cannot be applied to Equation 16 because of the exponential attenuation term in the integral. Levoy [Lev92] proposed to approximate the exponential attenuation mechanism with the first-order term of its Taylor expansion by changing Equation 16 to

$$I_{R(b)} = \int_a^b \nu(R(t))\gamma(R(t))[1 - \tau\nu(R(a))(t-a)]dt \tag{18}$$

Based on Equation 18, the absorption effect is simulated using three additional volumes each voxel of which is pre-multiplied with its X, Y, and Z coordinates respectively before applying the 3D forward Fourier transform [Lev92]. Unfortunately this approach requires four times as much storage overhead compared to spatial-domain rendering schemes, aggravating the already serious storage problem of volume datasets.

A subsequent paper by Totsuka and Levoy [TL93] described frequency-domain rendering methods with depth cueing and directional shading using a linear approximation to the Lambertian reflection model that require only one copy of the dataset. The quality of the resultant images is not particularly good compared to volume rendering of the spatial domain representation. The reason is that the first-order approximation to the exponential absorption function, although greatly simplifying computation, significantly distorts the attenuation effects of the original illumination model. In addition, this technique cannot be applied to arbitrary opacity transfer functions.

The proposed approach applies block-by-block compositing based on the projection subimages that are derived through the Fourier Projection Theorem from each subcube. That is, given the line integrals along the viewing rays for each subcube, how should the aggregate color and opacity values contributed by each subcube be calculated so that each subcube can be treated as a macro-voxel for spatial-domain compositing? The attenuation among consecutive voxels along a ray is modeled using a spatial compositing approximation to Equation 16 [PD84]. The discrete front-to-back compositing formula is defined as:

$$C_{acc,out} = (1 - O_{acc,in})C_v O_v + C_{acc,in} \tag{19}$$

$$O_{acc,out} = (1 - O_{acc,in})O_v + O_{acc,in} \tag{20}$$

where O_v and C_v represent the absorption and emission parameters for the voxel v, and $C_{acc,in}$ ($O_{acc,in}$) and $C_{acc,out}$ ($O_{acc,out}$) represent the accumulative color (opacity) values into and out of the

voxel, respectively. O_v and C_v are usually derived from the voxel's raw data value through a *color* and a *opacity transfer function*.

To treat each subcube as a macro-voxel, one needs to compute the aggregate color and opacity values contributed by each subcube. From Equations 19 and 20, the aggregate color contributed by a subcube that intersects with a projection ray is

$$C_{subcube} = \sum_{i=1}^{n} [C_i * O_i * \prod_{j=1}^{i-1} (1 - O_j)] \quad (21)$$

where we assume there are n voxels on the projection ray that fall within the given subcube. Similarly the aggregate opacity contributed by such a subcube is

$$O_{subcube} = 1 - \prod_{i=1}^{n} (1 - O_i) \quad (22)$$

However, the only information about each subcube after the application of the Fourier Projection Theorem is the line integrals through the subcube along the projection angle, that is, $\sum_{i=1}^{n} D_i$, where D_i is the raw data value associated with the i-th voxel. When the opacity transfer function assumes a negative exponential form:

$$O_i = 1 - e^{-K * D_i} \quad (23)$$

then it can be shown that $O_{subcube}$ can be computed exactly from $\sum_{i=1}^{n} D_i$ as follows:

$$O_{subcube} = 1 - e^{-K * \sum_{i=1}^{n} D_i} \quad (24)$$

For arbitrary opacity transfer functions, the only way to reconstruct $O_{subcube}$ exactly is to prepare a second volume from the original data volume by applying the following transformation to each voxel:

$$D_{new,\,i} = \log(1 - O(D_i)) \quad (25)$$

where $O()$ is any chosen opacity transfer function. Unfortunately, this approach is only applicable if the opacity transfer function is fixed, i.e., no interactive classification.

To allow the flexibility of changing the opacity transfer function interactively at run time, we choose the following method to approximate $O_{subcube}$ given $\sum_{i=1}^{n} D_i$. The basic idea is to assume that the voxel values within a subcube are reasonably close to each other so that one can use the average values for each of the voxels along the ray. Let $D_{avg} = \frac{\sum_{i=1}^{n} D_i}{n}$. By substituting D_{avg} for every voxel in Equation 22, we get

$$O_{subcube} = 1 - [1 - O(D_{avg})]^n \quad (26)$$

where $O()$ is the chosen opacity transfer function The smaller the subcube is, the more accurate this approximation of $O_{subcube}$ is.

The aggregate color contributed by a subcube, $C_{subcube}$, can be derived through two possible approaches. If the color transfer function is linear with respect to the raw voxel value, i.e., $C_i = K * D_i$, which is the case most of the time, the aggregate color can be approximated by ignoring the attenuation due to voxels within the subcube:

$$C_{subcube} = K * \sum_{i=1}^{n} D_i \quad (27)$$

When opacity is discounted, e.g., X-ray-like projections, this approximation method is preferable. Alternatively, one can apply the same idea of using the average value for every voxel on the ray and plugging it into Equation 21:

$$C_{subcube} = C(D_{avg}) * O(D_{avg}) * \frac{1 - [1 - O(D_{avg})]^n}{O(D_{avg})} \quad (28)$$

In this case, the color transfer function $C()$ doesn't have to be linear with respect to the raw data. Again the smaller the subcube is, the more accurate this approximation of $C_{subcube}$ is.

Given the aggregate color and opacity approximations for each subcube from its projection sums, we then apply subcube-by-subcube spatial domain compositing using the following equations:

$$C_{acc,out} = (1 - O_{acc,in}) C_{subcube} + C_{acc,in} \quad (29)$$

$$O_{acc,out} = (1 - O_{acc,in}) O_{subcube} + O_{acc,in} \quad (30)$$

where $C_{acc,in}$, $C_{acc,out}$, $O_{acc,in}$ and $O_{acc,out}$ are with respect to a subcube, not a voxel. Note that Equation 29 is different from Equation 19 because the occlusion effect due to each voxel has been captured in the aggregate color calculation, and should not be repeated in subcube-level compositing.

4 RESULT

Compared to conventional voxel-by-voxel ray casting algorithms, the proposed compression domain rendering algorithm may introduce two sources of errors. First, the projection sums computed from the Fourier Projection Theorem may not be exact due to aliasing. This error may be significant since the aliasing effect tends to be more serious at the subcube boundaries, which in turn could be in the middle of the data volume. Secondly, the calculations of the subcube's aggregate color and opacity values from the projection sums are just approximations, which may lead to further deviation from the result of the ray casting algorithm.

All the reported measurements below are 2D mean square errors (MSE) from the *head* data set. Table 3 shows the average mean square errors between the projection sums derived from the Fourier Projection Theorem and those from spatial domain summing, for subcubes of different sizes from different viewing angles. The projection angles are specified in the second row in terms of multiples of π. Mean square errors for orthonormal viewing angles are usually smaller than those for non-orthonormal ones, because non-orthonormal projections require interpolation for sample values on the rays, and thus tend to suffer more from aliasing. For a given non-orthonormal viewing angle, the mean square error increases as the subcube size decreases. This is because the MSE calculation tends to amplify the aliasing error, which in itself is independent of the subcube size, when the subcubes are smaller. For orthonormal viewing angles, such trends are less obvious, because most of the MSE in this case is mainly due to floating-point rounding.

Table 4 shows the mean square errors between the rendered images computed from voxel-by-voxel ray casting and those from subcube-by-subcube compositing using the aggregate color and opacity values approximated from spatial-domain projection sums, for different opacity transfer functions and different viewing angles. The reason we choose spatial-domain rather than the projection sums calculated from the Fourier Projection Theorem is to isolate the errors due only to the aggregate color and opacity approximation scheme. The linear opacity transfer function, $O_i = K * D_i$, chooses K to be the inverse of the difference between the largest and smallest density values in the volume, whereas the exponential opacity transfer function, as defined in Equation 23, uses the same

Subcube Size	Orthonormal		Non-orthonormal	
	$< 0\ 0\ 0 >$	$< 0.5\ 0\ 1 >$	$< 0.25\ 0.25\ 0.25 >$	$< 0.48\ 0.1\ 0.95 >$
4 x 4 x 4	0	0.153	25.543	35.980
8 x 8 x 8	0.529	1.536	15.990	23.536
16 x 16 x 16	0.433	1.026	9.980	18.336
32 x 32 x 32	0.301	1.006	4.670	10.312
64 x 64 x 64	0.149	0.217	3.252	3.513

Table 3: *The average mean square errors between the projections derived from the Fourier Projection Theorem and those from spatial domain summing, for various viewing angles. The color values are normalized to the range between 0 and 255.*

Subcube Size	Exponential		Linear	
	$< 0.5\ 0\ 1 >$	$< 0.48\ 0.1\ 0.95 >$	$< 0.5\ 0\ 1 >$	$< 0.48\ 0.1\ 0.95 >$
2 x 2 x 2	3.115	5.339	4.935	8.377
4 x 4 x 4	7.0699	8.001	9.129	10.560
8 x 8 x 8	10.870	12.698	12.699	14.958
16 x 16 x 16	16.775	17.314	18.397	19.402
32 x 32 x 32	25.781	26.089	27.778	28.321
64 x 64 x 64	32.750	37.336	34.579	39.601

Table 4: *The mean square errors between the rendered images using voxel-by-voxel ray casting and those from subcube-by-subcube compositing based on aggregate color and opacity approximation values, for different opacity transfer functions and viewing angles. The color values are normalized to the range between 0 and 255.*

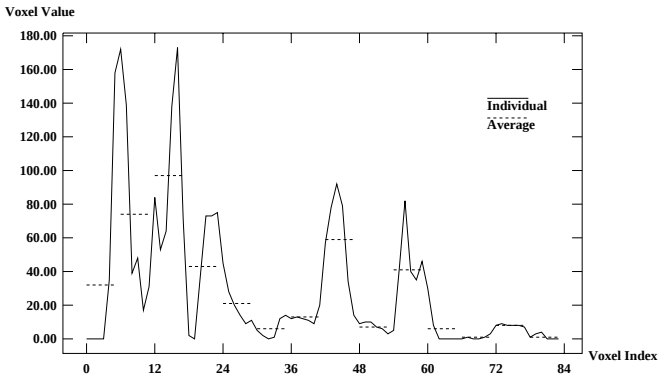

Figure 5: *The sequence of voxel values encountered by a sample projection ray, as shown as the solid line, and the average approximation voxel value used for each subcube, as shown as the dashed line.*

value for K. As expected, the smaller the subcube, the smaller the MSE. MSEs for orthonormal projections are smaller than those for non-orthonormal, because the aliasing effect is less serious. Because the aggregate opacity approximation is exact when the opacity transfer function is exponential, the MSEs for the exponential opacity case are smaller than the MSEs for the linear opacity case.

To compare the rendered images from voxel-by-voxel ray casting and from the compression domain rendering algorithm, we render the *head* data set using the two algorithms under different opacity transfer functions. Figure 6 and 9 show the rendered images using ray casting and compression domain rendering, assuming that there is no opacity, i.e., only X-ray effect. In this case, both the color and opacity approximation equations are exact, and therefore they look almost identical. Figure 7 and 10 show the rendered images using ray casting and compression domain rendering, assuming an exponential opacity transfer function. In this case, the aggregate opacity approximation equation is exact, but the color opacity approximation equation is not. At the subcube size of 4^3, the image from the proposed method is quite comparable to that from ray casting. Figure 8 and 11 show the rendered images using ray casting and compression domain rendering, assuming a linear opacity transfer function, i.e., the opacity is linearly proportional to the data density. In this case, neither the color nor the opacity approximation equation is exact. At the subcube size of 4^3, the two rendered images still look comparable, although the difference is more significant compared to the exponential opacity case.

To understand how the difference arises between the rendered images from the proposed scheme and ray casting, we trace the set of voxels encountered by the rays that are responsible for the portion of the image that shows the most serious discrepancy. Figure 5 shows a partial sequence of voxel values encountered by such a ray, and the average approximation for each subcube used by our approach. Because the voxel value sequence fluctuates noticeabaly within each subcube, the average approximations fail to capture the true ray casting computation. As a result, there is a significant difference in the final results produced by two methods for these rays.

5 CONCLUSION

By applying the Fourier Projection Theorem on a subcube by subcube basis, and performing spatial-domain compositing by treating each subcube as an indivisible macro voxel, we have integrated volume data compression and volume rendering in a unified framework. The preliminary experiment results show both high compres-

sion ratios and improved image quality over previous frequency domain rendering approaches.

There are several directions from this research that we are currently exploring. First, we are developing a strategy to choose the quantization table for a given data volume that produces the best compression ratio while maintaining the same reconstructed data quality. Secondly, we are examining the interaction between volume rendering and compression. In particular, we are interested in identifying the compression algorithm parameters to which the rendering algorithm is most sensitive. Thirdly, we are investigating the feasibility of using variable subcube size in the compression domain rendering algorithm, so as to exploit data-dependent optimization and achieve higher compression efficiency and better rendering quality.

Acknowledgement

This research is supported by an NSF Career Award MIP9502067 and a contract 95F138600000 from Community Management Staff's Massive Digital Data System Program.

References

[DNR90] S. Dunne, S. Napel, and B. Rutt. Fast reprojection of volume data. In *Proceedings of the 1st Conference on Visualization in Biomedical Computing*, pages 11–18, 1990.

[FY94] J. Fowler and R. Yagel. Lossless compression of volume data. In *Proceedings of Visualization '94*, pages 43–50, October 1994.

[Lev88] M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics & Applications*, 8(5):29–37, May 1988.

[Lev92] M. Levoy. Volume rendering using the fourier projection-slice theorem. In *Proceedings of Graphics Interface '92*, pages 61–69. Canadian Information Processing Society, 1992.

[Mal93] T. Malzbender. Fourier volume rendering. *ACM Transactions on Graphics*, 12(3):233–250, July 1993.

[MK93] T. Malzbender and F. Kitson. A fourier technique for volume rendering. In H. Hagen, H. Müller, and G. M. Nielson, editors, *Focus on Scientific Visualization*, pages 305–316. Springer Verlag, 1993.

[MO74] R. Mersereau and A. Oppenheim. Digital reconstruction of multidimensional signals from their projections. *Proceedings of the IEEE*, 62(10):1319–1338, October 1974.

[NH92] P. Ning and L. Hesselink. Vector quantization for volume rendering. In *Proceedings of the 1992 Workshop on Volume Visualization*, pages 69–74, October 1992.

[NH93] P. Ning and L. Hesselink. Fast volume renderign of compressed data. In *Proceedings of Visualization '93*, pages 11–18, October 1993.

[PD84] T. Porter and T. Duff. Compositing digital images. *Computer Graphics*, 18(3), July 1984.

[RY90] K. R. Rao and P. Yip. *Discrete Cosine Transform, Algorithms, Advantages, Applications*. Academic Press Inc., 1990.

[Sab88] P. Sabella. A rendering algorithm for visualizing 3D scalar data. *Computer Graphics*, 22(4), August 1988.

[TL93] T. Totsuka and M. Levoy. Frequency domain volume rendering. In *Computer Graphics Proceedings, Annual Conference Series*, pages 271–278. Proceedings of SIGGRAPH 93, 1993.

[Wal91] G. K. Wallace. The JPEG still picture compression standard. *Communications of the ACM*, 34(4):30–44, April 1991.

[Wes90] L. Westover. Footprint evaluation for volume rendering. *Computer Graphics, Proc. SIGGRAPH '90*, 24(4), August 1990.

[YL95] B.-L. Yeo and B. Liu. Volume rendering of dct-based compressed 3d scalar data. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):29–43, March 1995.

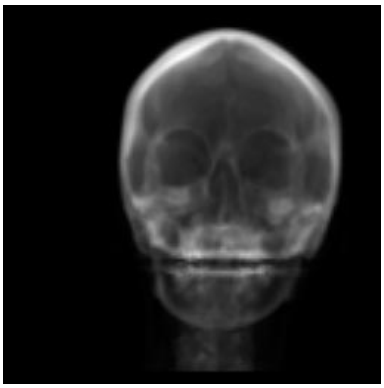

Figure 6: *A ray-cast image of the head data set, using the X-ray-like opacity model.*

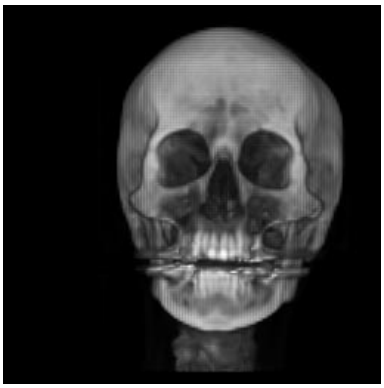

Figure 7: *A ray-cast image of the head data set, using an exponential opacity transfer function.*

Figure 8: *A ray-cast image of the head data set, using a linear opacity transfer function.*

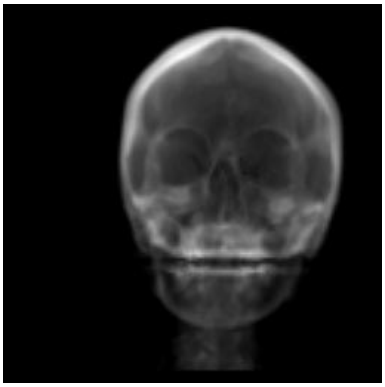

Figure 9: *A compression-domain rendered image of the head data set, using the X-ray-like opacity model. The subcube size is 4x4x4.*

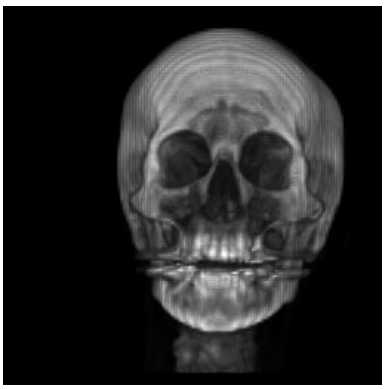

Figure 10: *A compression-domain rendered image of the head data set, using an exponential opacity transfer function. The subcube size is 4x4x4.*

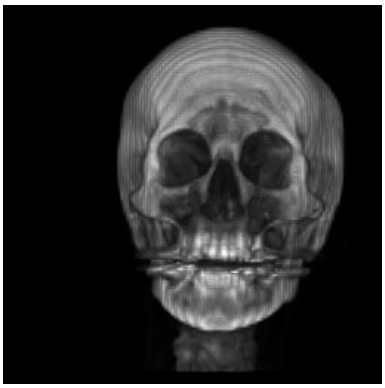

Figure 11: *A compression-domain rendered image of the head data set, using a linear opacity transfer function. The subcube size is 4x4x4.*