PROGRESSIVE TRANSMISSION AND RENDERING OF FOVEATED VOLUME DATA

CHEN CHEN

NATIONAL UNIVERSITY OF SINGAPORE

2005

Name: Chen Chen
Degree: Master of Computer Science
Dept: School Of Computing
Thesis Title: Progressive Transmission and Rendering of Foveated Volume Data

Abstract

With development of biotechnology and medical techniques, large scale volume data sets that cannot be achieved in local hard disk are produced for research purpose. In this project, I surveyed on various volume rendering schemes including two steps, data preprocessing and fast volume rendering algorithm, which compress and extract required data for rendering. Two progressive transmission/rendering schemes using wavelet foveation are implemented and compared in this project, Region-Based and Coarse-To-Finer. Region-Based scheme transmits and renders the fovea requested by the user on the client site at highest resolution first, and iteratively expands the fovea layer by layer towards the peripheral. While Coarse-To-Finer gives a rough preview image at low resolution in a short time at the client site and progressively refines the fovea from peripheral to the fovea center.

Keywords:

Volume Rendering, Progressive rendering, Wavelet Foveation

PROGRESSIVE TRANSMISSION AND RENDERING OF FOVEATED VOLUME DATA

CHEN CHEN
(B.Comp(Hons.), NUS)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF SCIENCE

SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF SINGAPORE

2005

Acknowledgement

I would like to thank my project supervisor, Dr. Huang zhiyong. He guides me through the whole project.

I also would like to thank Dr. Chang Ee-Chien, and Yu Hang(Ph.D. candidate). They gave me some good suggestions.

Contents

Summary

With development of biotechnology and medical techniques, large scale volume data sets that cannot be achieved in local hard disk are produced for research purpose. In this project, I surveyed on various volume rendering schemes including two steps, data preprocessing and fast volume rendering algorithm, which compress and extract required data for rendering. Two progressive transmission/rendering schemes using wavelet foveation are implemented and compared in this project, Region-Based and Coarse-To-Fine. Region-Based scheme transmits and renders the fovea requested by the user on the client site at highest resolution first, and iteratively expands the fovea layer by layer towards the peripheral. While Coarse-To-Fine gives a rough preview image at low resolution in a short time at the client site and progressively refines the fovea from peripheral to the fovea center.

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Modern scanners such as CT and MRI provide detail cross-sections of objects at a very high resolution. These data sets occupy huge amount of memory space ranges from several hundreds of megabytes to about one hundred gigabytes. Some super computer centers maintain large data repositories whose data sets will be accessed through networks and rendered in real time for researchers around the world to collaborate in their research.

The network speed is usually too slow compared with the large size of the scientific data sets. Lossy compression schemes are proposed by researchers to reduce the size of volume data. Another design issue is priority selection. Data will be reordered with priority. The importance of each voxel can be decided either by the client's current view direction or manually specified a region of interest(ROI) by the viewer.

1.2 Wavelet Foveated Volume

Wavelet transform is reported to be the best compression scheme among all, which makes use of the coherence in the data set. A large portion(up to 90%) of the coefficients will be zero after wavelet transform. However, the data size is still too large after wavelet compression if the user wants to access the data interactively. A foveated volume is a non-uniform sampled volume

whose resolution is highest at the fovea(a user specified ROI), but falls off as the distance with the fovea increases. The space variant of our visual system suggests that if one's gaze point is fixed on the fovea, he cannot distinguish the foveated image from the original one. Transform the original volume set into a foveated volume will tremendously reduce the size of the data set. It can also be regarded as a priority selection scheme.

1.3 Remote Progressive Transmission and Rendering

Coefficients in wavelet foveated volume are transmitted and rendered progressively at client site according to their priority. There are two ways of progressive transmission according to the way they assign the priority. One is Region-Based, that is, the priority of an coefficient will be inversely proportional to its distance to the center fovea. The center fovea will have the highest priority thus will be transmitted and rendered first at client site with highest resolution. The user at the client site will see a small portion of the volume which he/she is most interested at, it will then be expanded iteratively until the whole foveated volume is received. The other is Coarse-To-Fine algorithm. Higher priority will be assigned to lower level of details. The average coefficients will have the highest priority. According to this policy, a rough average volume will be transmitted and rendered first. More detail coefficients will arrive successively to refine the fovea from peripheral to center. The center fovea will be received and refined last. Each of these two methods have its good and bad points. In our successive sections, we

will describe it in more details.

1.4 Contribution

In this project, we implemented a server-client volume rendering system. 3D data sets are stored in server site. Suppose the client wants to view a volume data V, he sends a request of region-of-interest(ROI), which is represented as $ROI = \{(x, y, z), d\}$. (x, y, z) is the coordinate of the center of the fovea in spatial domain, and d represents the size of the fovea to be $d * d * d$. The input of the server part will be C_W, which is the blockwisely compressed wavelet form of volume data V, and $ROI = \{(x, y, z), d\}$, which is received from the client site. The output of the server site will be a wavelet foveated volume extracted from C_W named F_W. Data blocks of F_W will be progressively transmitted and rendered at the client site, so that the client can see an intermediate rendering result without waiting for the whole volume to arrive. More detail coefficients will arrive in success to refine the rendering result.

The rendering time of the rendering algorithm required depends only on the size of selected data. Progressive rendering algorithm will have the same final rendering result with normal rendering. We have implemented two transmission/rendering schemes. The results are compared and reported. It shows that the proposed framework can achieve the goal of fast rendering on a client-server system.

1.5 Outline

In Section 2, previous work in two main related research fields are surveyed, compression scheme and volume rendering algorithm. We discuss what are the good and bad points of these various methods. In Section 3, we proposed a server-client progressive rendering algorithm. In Section 4, performance statistics and image quality comparisons of two schemes are reported. Finally, we talk about future work and conclude in section 5 and 6.

2 Literature Survey

The huge amount of data is the most significant problem both in data transmission and volume rendering. To achieve fast volume rendering in a server-client approach, two steps are included: data preprocessing, which includes data compression and data extraction, and fast rendering.

2.1 Data Preprocessing

In order to transmit data efficiently through a low/medium network and load the huge data into the core memory of system, it is compulsory to preprocess the huge volume data into smaller size. The intuitive method is data compression.

A good compression scheme can be judged from many factors: encoding/decoding speed, random access, compression rate and visual quality. The well known vector quantization is time consuming to produce an optimal codebook and generates low quality image with high compression rate. Huffman encoding is a prefix-free variable-length code, but produces low compression ratio and low decoding speed. Comparing with these traditional compression schemes, wavelet-based compression provides good visual quality with high compression rate. Many researchers employed wavelet transform in their compression scheme in volume rendering algorithm.

Wavelet-based compression schemes vary from each other, but they usually contain four steps: wavelet transformation, normalization, thresholding,

and encoding. First, transform the data into wavelet domain. Normalization is to normalize all coefficients to the interval [0,1]. Thresholding step discards those coefficients that are smaller than a certain threshold value. Encoding is to encode the data using some coding schemes like run-length or Huffman.

After data compression, the data is small enough to be loaded into the core memory of the current system. However, in a server-client volume rendering system, it is still too large for transmission through the network, whose speed is usually tens of KB per second. Long time will be taken before all data reach the client site. Data extraction is employed further to extract the most important data and transfer with priority for fast rendering.

2.1.1 Various Methods

P. Lacroute et al. [16] is a scheme performed in spatial domain. They encode the data set in the scanline order from front to back using run-length encoding. The encoded data consists of two types of runs, transparent and non-transparent, defined by a user-specified threshold.

R. Grosso et al. [21] shows that obtaining the value of a wavelet coefficient from the compressed data is one of the most time consuming steps during the whole rendering process. Thus, they develop a data structure which is faster than the traditional sequential encoding of the coefficients, which is a sparse 3D array. The seven wavelet components corresponds to a level are independently saved and compressed in a sparse 3D array. After the normal process: normalization, thresholding and quantization, all coeffi-

cients corresponding to the z direction, i.e. for an index triple (i, j, k), those coefficients corresponding to the k index, are run-length encoded, eliminating all zero entries. A table lookup, which corresponds to i, j position of a 3D array, is used to store the address of each run-length encoded array. In addition, caching of the last k index is also used to accelerate the access speed.

I. Ihm et al. [6] proposes a scheme named zerobit, which achieves both fast random access and good compression ratio to the wavelet-compressed data. Because of the usual spatial coherence in the volume data, more than 90% of the coefficients are zero after performing Haar wavelet transformation and thresholding. This implies that zero coefficients will appear in thick chunks. Making use of this property, I. Ihm et al. divides the volume data into unit blocks of size 16X16X16 and then into cells of size 4X4X4. Three data structures are used here, cell tag table, significant map and byte stream. Cell tag table will tag with zero for those cells with all zero coefficients, and with positive index which point to the significant map for those cells that contains at least one non-zero coefficient. Significant map contains a bit map list which 0 and 1 corresponds to zero and non-zero coefficients respectively and an offset value in the byte stream for a cell.

Using the cell tag table and significant map, zero coefficients, which occupy more than 90% of the volume data, are compressed and can be returned in one or two memory access. Byte stream only stores those non-zero coefficients and will only be accessed when a non-zero coefficient is requested.

Kim, T. et al.[9] divides the volume into 8X8X8 block and perform generic

wavelet compression involves normalization, thresholding and run-length encoding. The different is, they assign higher error threshold for higher frequency levels and the coefficients are run-length encoded according to the reconstruction sequence from coarser to finer level. The entire encoding data contains a block index table and a run-length table. Block index table contains the average value of a block and a pointer to the index of its detail coefficients of the block in the run-length table.

In contrast with the above schemes, the later ones are view-dependent schemes which extract the most important data from the whole data set depending on the view direction, distance or user interest.

S. Guthe et al. [22] uses a blockwise hierarchical compression scheme. As the above schemes do, divide the volumes into cubic blocks of (2k)3, then apply wavelet transform to obtain 8 blocks of k3 voxels. Group 8 adjacent low pass filtered blocks to again get a block of (2k)3 voxels. Repeat this procedure until a single block is left. The resulting data structure is an octree. Each node, except the leaf nodes has 8 child nodes whose resolution is twice as high as that of a parent node. Only the root node has the low-pass filtered block. Other nodes only contain 7 detail coefficient blocks. For reconstruction of a child node, one must reconstruct its parent node first. An importance value is assigned to each node during the compression, which equals the L2 error of its subtree compared with the original data divides by the depth of the nearest voxel in the node. Those nodes with larger L2 error and smaller distance will be rendered first. The data in each node is encoded using run-length encoding combined with a fixed Huffman encoder.

The compression rate for run-length Huffman coding at a lossless setting is 10-15%.

A. Norton et al. [1] is also a blockwise compression scheme and the importance of a subcube depends on its visibility, that is, the view point and the data's opacity. Those subcubes with importance value larger than a threshold T will be regarded as frontal cubes and will be extracted. The server will automatically decide the coefficient threshold C such that the client can download the set of selected coefficients in less than one second. This threshold value is stored in the server repository to refine the client repository progressively according to the threshold value. The importance threshold T will be decreased by a factor of 0.7 when the frontal subcubes' compression rate reaches a target ratio, 0.1 for Haar wavelet. The image resulting from the compressed data will converge to the image that results from uncompressed data when importance threshold reaches 0. This algorithm is responsive since when the view point changes, there will be a few seconds' delay until the requested data available.

2.1.2 Comparison

Accessing the compressed data is the most time-consuming procedure. R. Grosso et al. [21] try to make some improvement in random access, so they indexed the i, j coordinate of the voxel. However, this scheme only works when the major axis is z, and gives low access and reconstruction rate when the major axis is x or y. Moreover, they ignore the coherence of data in 3D

and cannot achieve best compression rate. I. Ihm et al. [6] used a zerobit scheme to conquer this problem. Their scheme achieves random access while sacrificing some of the compression rate, since two successive coefficients with same value are redundantly stored. Kim, T. et al.[9] achieve random accessibility and high compression rate, but each cube are run-length coded, which gives lower reconstruction speed comparing with I. Ihm et al. [6].

Different with the above schemes, some schemes extracts important parts of the data set for future rendering to save transmission and memory cost. S. Guthe et al. [22] use an octree structure to store data in multi-resolution. In order to save the reconstruction time, recent used blocks will be stored in cache. A. Norton et al. [1] use almost the same structure as S. Guthe et al.[22] do. The different is that they transmit the subcubes progressively and define importance value of each voxel differently. A threshold value is associated with each subcube to facilitate progressive refinement. The server will decide the threshold so that the amount of data can be transmitted through the low/average bandwidth network in less than one second.

2.2 Fast Rendering Algorithms

There are four techniques that are particular popular in volume rendering: raycasting, splatting, shear-warp and texture-mapping hardware-based approaches [15]. Raycasting and splatting gives better image quality while has lower rendering speed as contrast to shear-warp and texture-mapping. Raycasting is to cast a ray for each image pixel into the volume and accumulate

Methods	Wavelet-Based	Output	Data Access Mode	Compression Rate	Reconstruction of Single Voxel	Progressive Refinement
Lacroute et al. [16]	NO	a list of voxel scanline	sequential	-	-	NO
R.Grosso et al. [21]	YES	whole data set	random access on i,j index	✓✓	O(n)	NO
Ihm et al.[6]	YES	whole data set	random access	✓✓	O(1)	NO
Kim et al.[9]	YES	whole data set	random access	✓✓✓	O(1)	NO
Guthe et al. [22]	YES	significant and near cubes	hierarchical	✓✓✓	O(logn)	NO
Norton et al.[1]	YES	visible coefficients of frontal cubes	hierarchical	✓✓	O(logn)	YES

Table 1: Comparison of various data proprocessing methods (more ' ✓' means higher compression rate and '-' means not applicable)

the sample value along the ray by resampling. In contrast with ray casting, a feed backward method, splatting, which is first proposed by Westover [11], is a feed forward algorithm calculating the footprint which is the weight of the voxel contributing to its neighboring pixel. Both methods have been combined with wavelet transform in order to reduce the size of volume data to fit into core memory [12] [13].

Texture-mapping is a hardware based method which is very fast but gives lower image quality. When the data set is too large that the texture is hard to be loaded totally into texture memory, texture swap will occur and rendering speed will reduce dramatically. Shear-warp [16] is the fastest software-based algorithm and gives similar image quality with raycasting and splatting. However, when the magnification is high, significant aliasing is present. For fast volume rendering, we focus on the later two algorithms: texture-

mapping and shear-warp below.

2.2.1 Various Methods

Texture-map has two approaches, 2D texture-mapping and 3D texture-mapping. P. Pinnamaneni et al. [17][18] uses 2D texture mapping in their algorithm. They use three sets of perpendicular 2-D cross-sections, which are mapped onto polygonal 2-D planes. Each pixel in a cross-section is assigned a transparency α value. The transparency transfer function will determines the appearance of the 3-D reconstruction. It can be a simulated X-ray view or normal images. The rendered image will first be displayed in a coarser resolution and being refined later when more data are received. The accuracy of the image depends on the number of slices. Simple duplication or interpolation will be used when rendering lower resolution cross-sections. This algorithm is suitable for preview images and web-based rendering.

The algorithm is later extended into 3D texture-mapping in [19]. The voxel array is interpreted as a 3D texture defined in 3D space and the three-dimensional data is loaded as texture block into the texture buffer. A parallel stack of polygon planes that are orthogonal to the screen are used for texture mapping. Each polygon plane vertex is associated with texture value by trilinearly interpolation.

Guthe et al. [22] uses 3D texture-mapping rendering algorithm in his scheme. 3D texture is created and loaded onto the graphics hardware. Then viewplane aligned slices will be tiled in a back to front order. Texture-

mapping always has the problem when the data set is too large and cannot be loaded into the texture memory. Guthe solves this problem by using multi-resolution rendering. The resolution of a cube is proportional to its importance value. Since an octree wavelet compression scheme is used, Guthe adds a cache to store decompressed higher level nodes which will be used frequently, so that the renderer doesn't need to decompress the node from the tree's root node everytime.

P. Lacroute et al.[16] proposed a software-based fast rendering algorithm shear-warp factorization. Shear-warp factorization has the property that rows of voxels in the volume are aligned with rows of pixels in the intermediate image. This property has some good points: 1. only simple translation and re-sampling is needed. 2. Every voxel in the same slice has the same re-sampling weight.

Philippe Lacroute and Marc Levoy develop volume rendering algorithm based on shear-warp factorization in the space domain which take advantage of this property. They use run-length encoding of the voxel scanline which contains two types of runs, transparent and non-transparent. The intermediate image also uses run-length encoding which contains opaque and non-opaque pixels. All these transparent, non-transparent, opaque and non-opaque are decided using user-specified thresholds. The intermediate image is computed on the fly during rendering. Those transparent voxels and voxels that are occluded by the opaque intermediate pixels will be skipped when computing intermediate image. The run-length encoding of voxels saves much space since 70-95% of the voxels are transparent. The skip of those

transparent voxel and opaque pixels will accelerate the rendering process. The algorithm provides quite good compression rate, decoding, encoding speed and visual quality. However, it doesn't allow quick random access, which might be a weak point.

Using this rendering technique, the resulting image will be more like we see through our eye rather than a x-ray image.

J. P. Schulze et al. [7] gives a formal proof for the correctness of shear-warp algorithm and extends the algorithm to perspective view. H.Yu [4] proposed a rotation algorithm approximated by shear-warp factorization.

Kim et al.[9] proposed a scheme combined wavelet transformation with shear-warp factorization. The voxel slices' density value from the wavelet coefficients are reconstructed on the fly and transparent voxels lines are skipped during rendering. The renderer will start from the first non-transparent voxel line in each slice. Because wavelet transformation is employed, the space required is smaller than P. Lacroute's scheme.

2.2.2 Comparison

From Table 2, we can see, 2D texture-mapping is slower than 3D texture-mapping algorithms by a factor of 3. However, it requires less memory space since the texture slices don't need to be loaded into the texture memory in one time. Texture-map algorithms are generally faster than shear-warp algorithms but gives lower image quality. It can be improved by using more

Method	Output Resolution	Output Quality	Rendering Algorithm	Rendering Speed	Memory Cost
Pinnamaneni et al. [17]	Uniform	★	2D texture-map	$3 * n(slice)$ $*n(voxel\_in\_slice)$	◇◇
Pinnamaneni et al. [19]	Uniform	★★	3D texture-map	$n(slice)$ $*n(voxel_i n_s lice)$	◇◇◇
Guthe et al. [22]	Multi-resolution	★★	3D texture-map	$n(slice)$ $*n(voxel_i n_s lice)$	◇
Lacroute et al. [16]	Uniform	★★★	Shear-warp	$n(voxel)-$ $n(transparent)$	◇◇◇
Kim et al.[9]	Uniform	★★★	Shear-warp	$n(voxel)$ $-n(transparent)$	◇◇

Table 2: Comparison of various rendering schemes (more ★ means more accurate rendering, and more ◇ means the memory cost is more expensive)

texture slices, but rendering speed is inverse proportional with the number of texture slices used. 3D Texture-mapping is fast but when dataset is very large and cannot be loaded into the texture memory, swaps will occur and rendering speed will decrease. Guthe et al. [22] solve this problem by using multi-resolution display. Important voxels are displayed in higher resolution while less significant and farer voxels are displayed in lower resolution.

2.3 parallel-programming

Another approach to accelerate rendering process is parallel programming. The basic idea is to divide the process into separate jobs with equal complexity, distribute to processors respectively and finally combine the results of each processor.

P. Schroder et al. [20] finds that by setting the ray distance along x-axis

and y-axis to be 1, and tiling the volume accordingly, all rays will enter voxels at the same voxel local coordinates. The volume can be divided into sets of voxels with equal size and each set corresponds to one ray. Voxels in each set tile perfectly without holes or overlaps. This reveals the possibility for parallel programming. We can assume there are as many processors as there are rays entering at the front most face and each processor holds a accumulated integral and a volume data set.

K. Sano et al. [10] proposed an parallel algorithm for shear-warp factorization. The basic idea is to divide the volume into subvolumes with equal number of opaque voxels along the major axis. Each processor will be in charge of one subvolume and finally composite the subvolume image to obtain the final result.

3 Proposed Algorithm

3.1 Scheme overview

This scheme aimed at web-based fast rendering of foveated volume which contains usual three steps: data preprocessing, progressive transmission and progressive volume rendering.

Volume data is first transformed to wavelet form and divided into 8X8 blocks, each block is encoded using RLE(run-length encoding)in server site. Experiments shows that volume data will be compressed up to 1/10 of the original size after this operation. Blockwise structure is employed in order to enable efficient data compression as well as convenient communication between server and client. When a client request is received, a compressed foveated volume is extracted. A foveated volume has highest resolution at the fovea while the resolution falls off when the distance with the fovea increases. We use wavelet foveation to approximate foveation operator. For an original data set with size 1024^3 and fovea size 48^3, the wavelet foveated volume will have approximately $24^3 * 36 = 497664$ coefficients, which is reduced by a factor of 2,000 comparing with the original size. Details of wavelet transform and extraction of foveated volume will be described in subsection 3.2.1 and 3.2.2.

Data is then progressively transmitted in sequence. Data blocks of the wavelet foveated volume are reordered according to its priority. Two priority assignment schemes will associate with two transmission and rendering

schemes. One is Region-Based and the other is Coarse-To-Fine. Two transmission scheme and their reordering of the data blocks will be discussed in detail in section 3.3.

To be consistent with progressive data transmission, progressive rendering algorithm is employed. In Region-Based scheme, the volume are displayed on client site layer by layer from fovea toward the peripheral. The fovea area will be rendered in high resolution using data received earlier and in each successive step, one immediate outer layer is displayed in half resolution of its inner layer. In Coarse-To-Fine scheme, the client will first show a rough average image of the volume and iteratively refines the fovea area. Two progressive rendering algorithms and an efficient rendering equation are described in section 3.4.1 and 3.4.2 respectively. A fast rendering algorithm, wavelet shear-warp factorization, discussed in section 3.4.3, is used in our scheme to accelerate rendering of non-orthogonal viewing direction.

3.2 Data Preprocessing

3.2.1 Data Compression

Volume data sets usually range from several hundreds of megabytes to about one hundred gigabytes which is hard to be load into main memory. Furthermore, in a server-client paradigm, the network speed is usually quite low, about tens of KB per second. For a data set with size 1024^3 and fovea size 48^3 in spatial domain, the required data is larger than 1M whose waiting time is too long in a low/medium speed network. Data compression is an

intuitive method to solve this "large data size" problem. Haar wavelet is computational simple and has a good correspondence with octree that enables easy data searching and reconstruction. After Haar wavelet transform, we apply RLE(run-length encoding) on the wavelet volume block by block, which shows a good compression result.

Haar Wavelet Transform We will illustrate it in 2D case first. Let I be an image of size $2^N \times 2^N$. Four images can be generated from I after one pass of Haar wavelet transform, where

$$\begin{aligned} I_{00}(i,j) &= \frac{I(2i,2j) + I(2i+1,2j) + I(2i,2j+1) + I(2i+1,2j+1)}{4} \\ I_{10}(i,j) &= I(2i+1,2j) \\ I_{01}(i,j) &= I(2i,2j+1) \\ I_{11}(i,j) &= I(2i+1,2j+1) \\ for i,j &= 0 \ldots 2^{N-1} - 1 \end{aligned}$$

As illustrated in figure 1, Image I_{00} is an average image of image I with half resolution. The rest three images are called detail images. We recursively perform this process on the average image of each iteration until the average image has only 1 pixel or reaches some pre-defined resolution. The result will be the wavelet representation of the original image I.

This procedure is invertible. Using the following equation, we can get

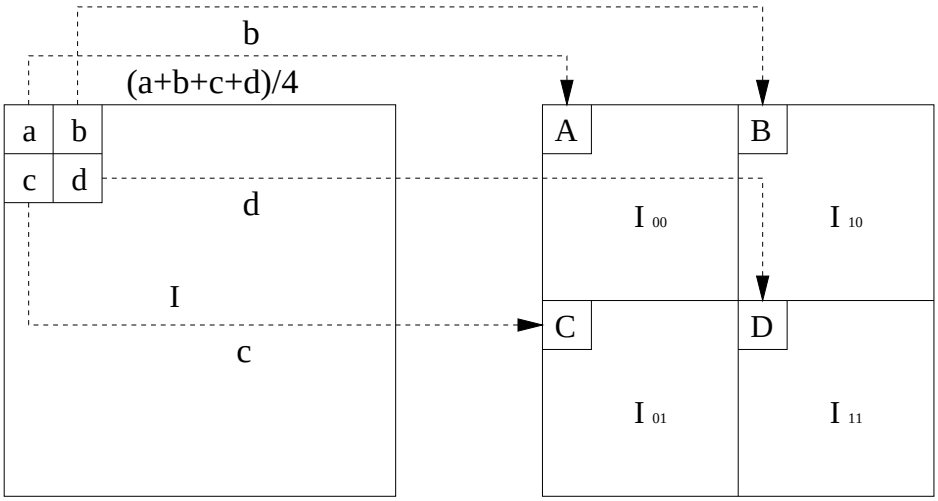


Figure 1: Illustration of wavelet transformation

back the original values of image I.

$$
\begin{aligned}
a &= 4A - B - C - D \\
b &= B \\
c &= C \\
d &= D
\end{aligned}
$$

This is the basic idea of wavelet transform. In this project, we apply similar wavelet transform to 3D volume data. The transfer operation we used can be described using the following equations:

$$
\begin{aligned}
A &= a + b + c + d + e + f + g + h \\
B &= a - b \\
C &= a - c \\
D &= a - d
\end{aligned}
$$

...

This operation is also invertible by applying the following equations:

$$\begin{aligned} a &= \frac{A+B+C+D+E+F+G+H}{8} \\ b &= a - B \\ c &= a - C \\ d &= a - D \\ &\dots \end{aligned}$$

a,b,c,d,e,f,g,h are the eight adjacent voxels in the 3D volume data. In our operator, we are taking the sum instead of average is to avoid arithmatic division, so that the transformation can be performed more efficiently. A volume data will normally have large smooth area. Intensity of adjacent voxels are highly correlated. Taking difference between adjacent voxels results in small values including many successive zeros in detailed images, which can be compressed greatly.

Blockwise hierarchical compression scheme In the server site, we use a blockwise hierarchical compression scheme similar with S. Guthe et al [22]. Divide the volumes into blocks of $(2k)^3$, then apply generic Haar wavelet transform on it to get 8 blocks of size k^3. One of them will be average block, the other 7 are detail blocks. Group 8 adjacent average blocks again to obtain a block of size $(2k)^3$. Repeat this procedure until the size of the average image reaches m. These m blocks are the low pass filtered volume of the whole

volume data. The value of k has a tradeoff between compression rate and date redundancy during transmission. Since the block will be regarded as a unit during data transmission. Large k value will cause data redundancy since we will transmit more data than we actually needed. Small k value usually gives low compression rate. In our case, k will be 8 and m will be 16.

Each block in the average image corresponds to an octree. Each node in the octree, except the leaf nodes and root node, consists of 7 detail blocks of size k^3, totally $7 * k^3$ coefficients and has 8 child nodes. The leaf nodes only have 7 detail blocks without child nodes. The root node has an additional average block, besides the 7 detail blocks and 8 child nodes. The root node gives a very rough approximation of one portion of the data set and the resolution can be increased by a factor of 2 by going downwards to the a child node. We can illustrate it using a 2D example. However, 3D case is very similar with 2D case.

Give a 2D data set of size $(8k)^2 * blk\_sz$, repeat the procedure described previously 2 times and obtain 4 average blocks marked as A_{00},A_{01},A_{10} and A_{11}(figure 2). Each of them associates with a quad-tree (figure 3). Take A_{00} as example, the coefficients marked in grey color are all the component nodes in quad-tree of A_{00}. Local coordinate in a level v corresponds with 3 blocks: HL, LH and HH, which are the 3 components of node(v,i,j) in the quad-tree. Four child nodes of this node (v,i,j) in level v+1 can be simply identified by (2i,2j), (2i,2j+1), (2i+1,2j) and (2i+1,2j+1), which forms a pointless quad-tree.

L0		**L1**		**L2**			
A_{00}	A_{01}	HL_{00}	HL_{01}	HL_{00}	HL_{01}	HL_{02}	HL_{03}
A_{10}	A_{11}	HL_{10}	HL_{11}	HL_{10}	HL_{11}	HL_{12}	HL_{13}
LH_{00}	LH_{01}	HH_{00}	HH_{01}	HL_{20}	HL_{21}	HL_{22}	HL_{23}
LH_{10}	LH_{11}	HH_{10}	HH_{11}	HL_{30}	HL_{31}	HL_{32}	HL_{33}
LH_{00}	LH_{01}	LH_{02}	LH_{03}	HH_{00}	HH_{01}	HH_{02}	HH_{03}
LH_{10}	LH_{11}	LH_{12}	LH_{13}	HH_{10}	HH_{11}	HH_{12}	HH_{13}
LH_{20}	LH_{21}	LH_{22}	LH_{23}	HH_{20}	HH_{21}	HH_{22}	HH_{23}
LH_{30}	LH_{31}	LH_{32}	LH_{33}	HH_{30}	HH_{31}	HH_{32}	HH_{33}

Figure 2: Wavelet Transform of a $(8k)^2 * blk\_sz$ data set

3D case is almost the same with 2D, the only difference is each node will have 7 component nodes: LLH, LHL, LHH, HLL, HLH, HHL and HHH, and each node has 8 child nodes.

After wavelet volume are divided into data blocks. RLE(run-length encoding) is performed on it block by block in order of front to back, top to bottom and left to right. The compressed block is a list of value(v) and length(l) pairs. Value(v) is the integer value of a coefficients, and length(l) is the number of successive integers with this same value. During decoding, we simply get the (v,l) pair one by one in order, and fill l v's into the data sequence each time.

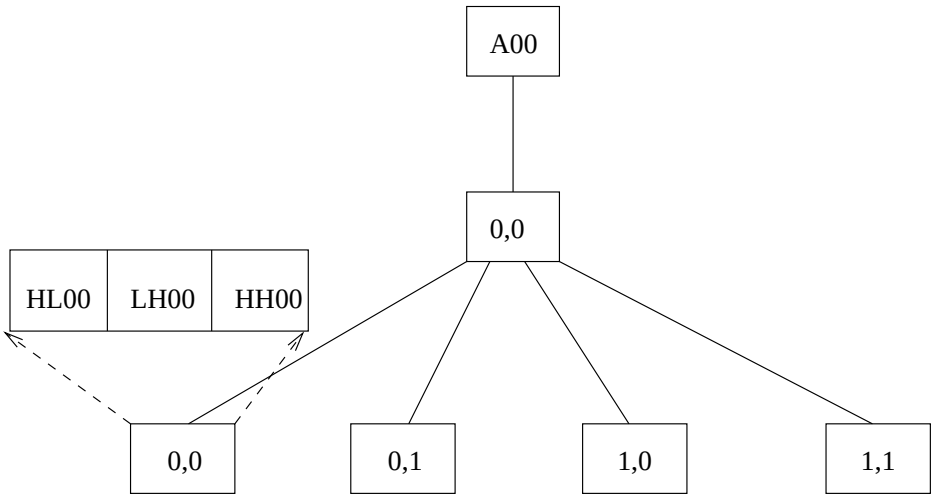


Figure 3: Quad-tree of data set in Figure 2

Take an example, given a data list L=1,1,1,1,0,0,5,0,0,0,0,0,0,0,0. The run-length encoded data R will be (1,4),(0,2),(5,1),(0,8).

3.2.2 Derive foveated volume in wavelet domain

We have a wavelet volume W after wavelet transform. Suppose the client wants to view a volume data V, he gives a request of ROI which is represented as the center of the fovea, that is, ROI(x,y,z) and the size of the ROI r^3. The inputs of the server part are W, which is the wavelet representation of V, and $ROI\{(x, y, z), r^3\}$, our task is to compute W_f from W, when a client request is received.

We call the lowest resolution image in the wavelet representation W as layer 0 image, and the second lowest resolution image as layer 1 image and so on. layer 0 is actually the average image of W, and layer i image can be ob-

tained by performing inverse wavelet transformation to layer i-1 with the corresponding detailed coefficients(corresponding detailed coefficients are layer i details). The highest resolution image in W is denoted as layer max_layer. In general, the volume data is of size W*H*D, and the number of layers is $log_2\frac{W}{m}$, $log_2\frac{H}{m}$ and $log_2\frac{D}{m}$ respectively (assuming W, H and D are multiples of m and m is power of 2). In case the volume data V is cubic with size of N^3(N is power of 2), and the wavelet transform is processed until the average image has size m^3, $max\_layer = log_2\frac{N}{m}$.

W_f is extracted from W. It consists of all voxels that are needed for reconstructing ROI_i, where ROI_i is the region of interest in the ith layer of W. The center of ROI_i is $(\lfloor\frac{x}{2^{max\_layer-i}}\rfloor), \lfloor\frac{y}{2^{max\_layer-i}}\rfloor, \lfloor\frac{z}{2^{max\_layer-i}}\rfloor)$ and the size of ROI_i is r^3.

For convenience of calculation, the boundary of ROI_i can be defined as

$$begin_i.x = (x/2^{(max\_layer-i)} - s/2) \bigoplus (0XFFFE) \quad (1)$$

$$begin_i.y = (y/2^{(max\_layer-i)} - s/2) \bigoplus (0XFFFE) \quad (2)$$

$$begin_i.z = (z/2^{(max\_layer-i)} - s/2) \bigoplus (0XFFFE) \quad (3)$$

$$end_i.x = (x/2^{(max\_layer-i)} + s/2) \bigoplus (0XFFFE) - 1 \quad (4)$$

$$end_i.y = (y/2^{(max\_layer-i)} + s/2) \bigoplus (0XFFFE) - 1 \quad (5)$$

$$end_i.z = (z/2^{(max\_layer-i)} + s/2) \bigoplus (0XFFFE) - 1 \quad (6)$$

$$begin_i.x, begin_i.y, begin_i.z, end_i.x, end_i.y, end_i.z \in [0, minlen * 2^i - 1]$$

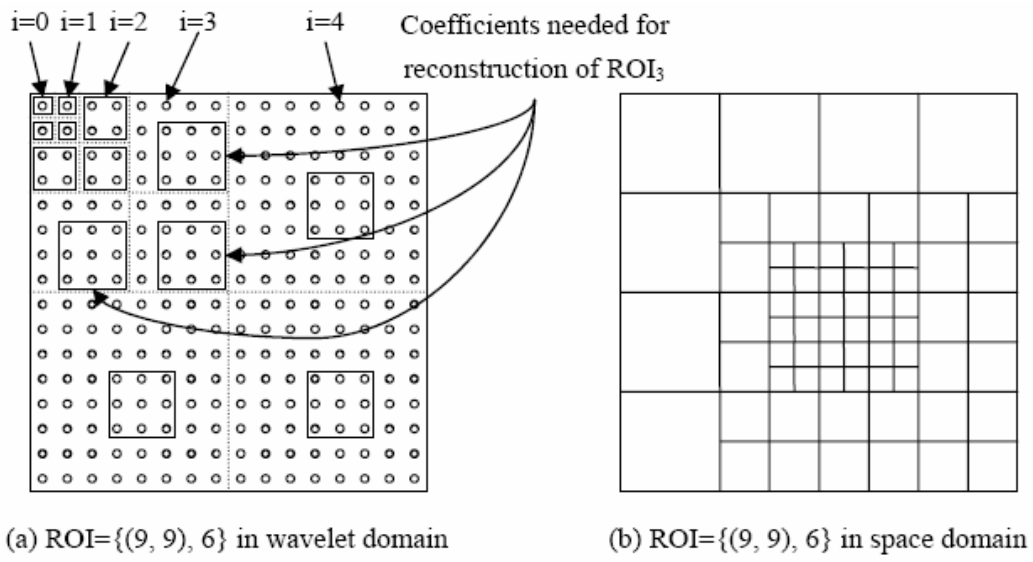


Figure 4: Extraction of the foveated data in wavelet domain

Figure 4 gives an 2D example of ROI=$\{(9,9),6\}$ for a 16*16 image, where m=1, that is, the average image size is 1^2. In this example, $max\_layer = log_2 16 = 4$. Take layer 3 as an example, $ROI_3 = \{(4,4),6\}$, pixels in layer 3 image with coordinate (i,j), $i, j \in [2,7]$ are in ROI_3. In order to reconstruct ROI_3, layer 3 details with coordinate (i,j), $i, j \in [1,3]$ are needed.

Figure 4(b) represents the reconstructed foveated image.

3.3 Progressive Transmission

Given a data set with size 1024^3, fovea size 48^3 and let the size of average image to be 16^3. The wavelet form of this data set will have $log_2(1024 \setminus 16) = 6$ layers. Fovea size in wavelet domain will be 24^3 and each layer has 7 passes. The wavelet foveated volume will have approximately $24^3 * (5*7+1) = 497664$

coefficients, the inner most layer has only 1 average image instead of 7 detailed images. Even though blocks are run-length encoded and compressed, the size of data to be sent is still quite large. In a $low \setminus medium$ speed network, the user needs to wait for a few seconds before all of the data needed reaches the client site. Progressive transmission is to re-order the coefficients such that more important coefficients will be regarded as high priority and being sent first. These coefficients can be rendered in the client site separately with other data. Less important coefficients will be sent progressively, step by step, to refine the result image.

There are two ways of progressive transmission in wavelet volume rendering scheme. One is to transmit a rough average image first and refine the fovea from outer ROI_{i+1} to inner ROI_i using detail coefficients. The other is to transmit a full resolution ROI_{max_layer} first, and expand the fovea area from $ROI_{max\_layer-1}$ to ROI_0 with decreasing resolution until the full foveated volume is transmitted. We will explain the two methods in detail and illustrate using examples.

The first method is intuitive to implement. We just send the wavelet foveated volume data from inner layer to outer layer progressively. Coefficients in each layer should be wrapped as an atomic package.

The basic idea of the second method is to send all data blocks for constructing ROI_i in each iteration. At the first iteration, boundary of ROI_{max_layer} will be computed and data blocks in layer $max\_layer - 1$ that contain coeffi-

cients for ROI_{max_layer} will be added into the data queue. All parent blocks of these blocks will also be added into the data queue. A block is regarded as a parent block of another block if it's a parent node of that block in the oc-tree they belong to. The index of the data block will be added into the data list when it is added into the data queue. In iteration i, boundary of $ROI_{max\_layer-i+1}$ will be computed. Data blocks in layer $max\_layer - i$ that contain coefficients for $ROI_{max\_layer-i+1}$ and all parent blocks of these blocks will be added into the data queue if that block does not exist in the data list. Repeat this procedure until coefficients of ROI_0 are added into the data queue. If the data list doesn't contain all data blocks in average image after above procedure, append the remaining average blocks into the data queue.

Let's illustrate the procedure using a 2D example. Given a 2D image with size $128 * 128$, where m=16, k=8 and ROI=(86,86),45. The average image has size $16 * 16$, which contains 4 data blocks, the fovea centered at coordinate (86,86) with size (45,45). In this example, $max\_layer = log_2(128/16) = log_2 8 = 3$.

Take ROI_2 as an example, $ROI_2 = \{(\lfloor \frac{x}{2^{max\_layer-i}} \rfloor), \lfloor \frac{y}{2^{max\_layer-i}} \rfloor), d\} = \{(\lfloor \frac{86}{2^2} \rfloor), \lfloor \frac{86}{2^2} \rfloor), 45\} = \{(43, 43), 45\}$.

$$
\begin{aligned}
begin_i.x &= (43 - 45/2) \bigoplus (0XFFFE) = 20 \\
begin_i.y &= (43 - 45/2) \bigoplus (0XFFFE) = 20 \\
end_i.x &= (43 + 45/2) \bigoplus (0XFFFE) - 1 = 63
\end{aligned}
$$

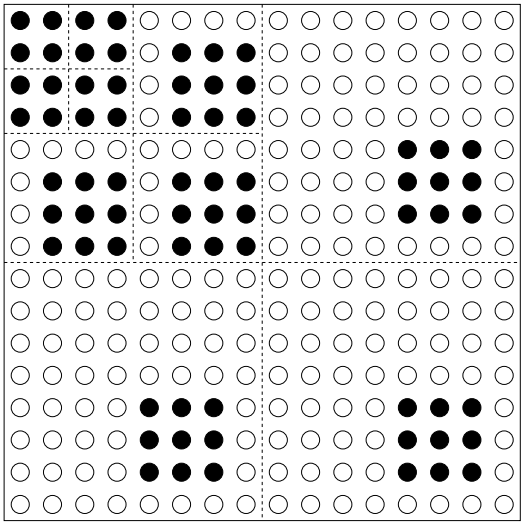


Figure 5: Wavelet Foveated Image

$$end_i.y \quad = (43 + 45/2) \bigoplus (0XFFFE) - 1 = 63$$

$$begin_i.x, begin_i.y, end_i.x, end_i.y \in [0, 63]$$

Pixels in layer 2 image with coordinate (i,j), $i, j \in [20, 63]$ are in ROI_2. In order to reconstruct ROI_2, layer 2 details with local coordinates (i,j), $i, j \in [10, 31]$ are needed, represented as block index (blk_i, blk_j), $blk_i, blk_j \in [1, 3]$. Similarly,

$ROI_3 = \{(86, 86), 45\} = [64, 107]$, level 3 details: (blk_i, blk_j), $blk_i, blk_j \in [4, 6]$

$ROI_2 = \{(43, 43), 45\} = [20, 63]$, level 2 details: (blk_i, blk_j), $blk_i, blk_j \in [1, 3]$

$ROI_1 = \{(21, 21), 45\} = [0, 31]$, level 1 details: (blk_i, blk_j), $blk_i, blk_j \in [0, 1]$

$ROI_0 = \{(10, 10), 45\} = [0, 15]$, average blocks: (blk_i, blk_j), $blk_i, blk_j \in [0, 1]$

Figure 5 draws out this wavelet foveated image, each circle represents one data block with size $8 * 8$. Circles in black are data blocks that belongs to

$ROI_i, i \in [0, 3]$.

Figure 6 and 7 shows the procedure of the first and second transmission methods, separately. Black circles represents data blocks being sent in each step. In method 2, there are only 2 steps rather than 4. That is because we use a blockwise structure, each block is regarded as an atomic unit. During each iteration, we may send more data than there is actually needed to reconstruct ROI_i. Thus, detail coordinates needed in step 3 and 4 have been sent in previous steps as redundant data.

3.4 Rendering Algorithm

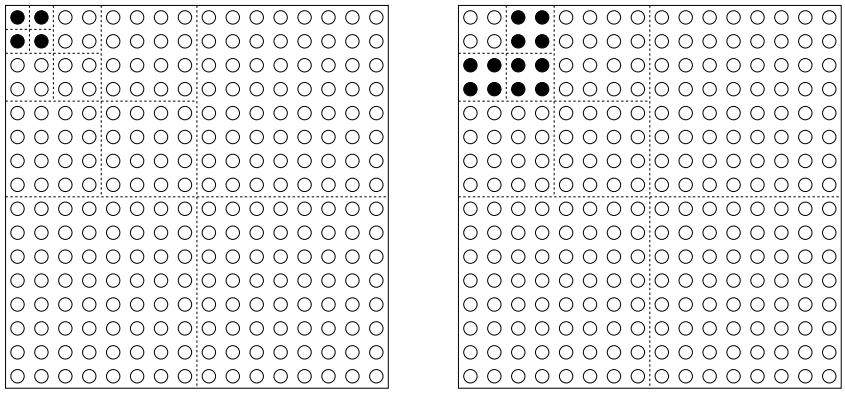
3.4.1 Rendering Equation

In context of direct volume rendering, each element of the volume data is called a voxel and is assigned a density value. Light shoots into the volume, during traverse through the volume, the light is emitted and absorbed by the voxels. The finial intensity result reaches the viewer will be defined as

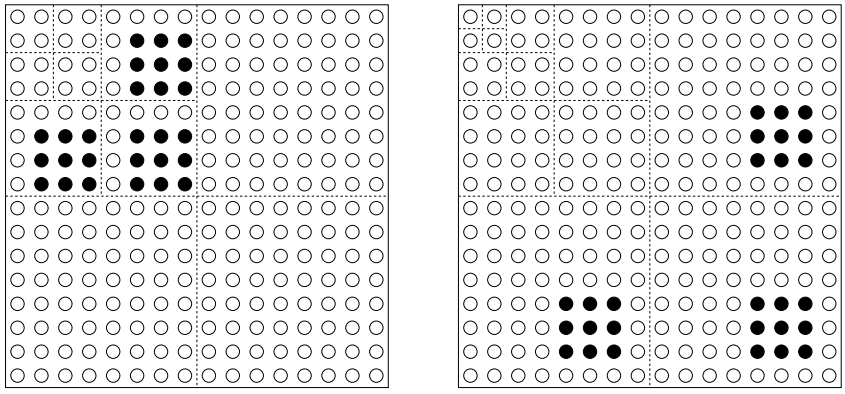
$$I(t1, t2) = \int_{t1}^{t2} q(t) e^{-\int_{t1}^{t} \alpha(s) ds} dt$$

where t1 and t2 are the starting and ending point of the viewing ray. q(t) and $\alpha(s)$ are intensity and opacity function respectively.

For convenience computation, this continuous rendering function can be reduced to discrete form with the assumption that for a certain segment i,

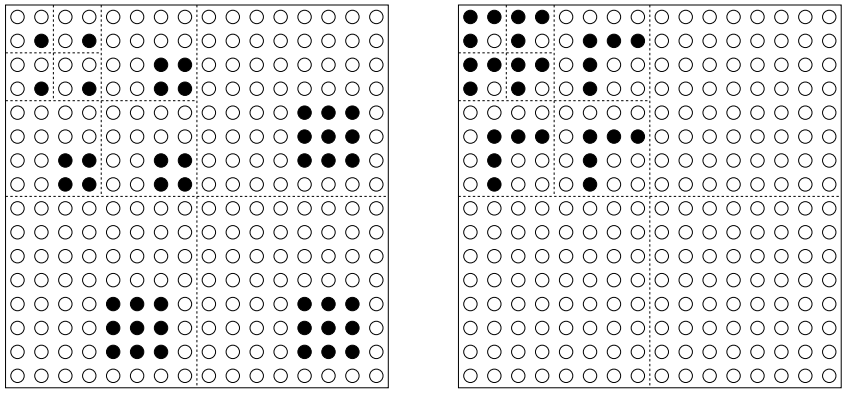


Step 1 Step 2



Step 3 Step 4

Figure 6: 4 iterations of progressive transmission method 1.



Step 1 Step 2

Figure 7: 2 iterations of progressive transmission method 2.

the intensity function and opacity function remains constant, q_i and α_i. We assume that each voxel represents a segment. The outer integral of the above formula can be replaced by a Eulerian sum over accumulated opacity.

$$I = \sum_{k=0}^{n} q_k \alpha_k \prod_{i=0}^{k-1} (1 - \alpha_i)$$

In our work, the foveated volume has various resolution instead of uniform resolution. Thus, we need to render subvolumes that contain more than one segment(voxel) whose voxels share the same intensity value q and opacity value α. An optimized formula can be used to save computational time when rendering such subvolumes.

$$I_{subvolume} = \sum_{i=1}^{n} q\alpha \prod_{j=1}^{i-1} (1 - \alpha) = \sum_{i=1}^{n} q\alpha (1 - \alpha)^{i-1}$$

n is the number of voxels of the subvolume that share the same intensity and opacity value.

For two subvolumes S_a and S_b with size n_a and n_b. Voxels in S_a and S_b have intensity and opacity value q_a and α_b, respectively. S_a is in front of S_b, that is, S_a is "over" S_b. The intensity of S_a,S_b and S_{final} can be defined as:

$$\begin{aligned} I_{S_a} &= \sum_{i=1}^{n_a} q_a \alpha_a (1 - \alpha_a)^{i-1} \\ I_{S_b} &= \sum_{i=1}^{n_b} q_b \alpha_b (1 - \alpha_b)^{i-1} \\ I_{final} &= I_{S_a} + (1 - \alpha_a)^{n_a} * I_{S_b} \end{aligned}$$

3.4.2 Progressive Rendering with orthogonal viewing direction

An intuitive way to get a rendering image is to wait for all coefficients arriving at the client site, reconstruct V_f and implement a direct rendering on it. However, this method is time and resource consuming. Sometimes, it's even impossible due to the limitation of client resource. Our goal is to avoid reconstruction of V_f and renders the volume data that are progressively transmitted to the client site iteration by iteration. During each iteration, we compute and store some of the rendering attributes for each ray as a one-value data item to avoid multiple process.

To be consistent with the two progressive transmission methods, there are also two ways of rendering. One is from pheripheral to center, that is from coarse to fine resolution. The other is a region-based method which processed in a reverse order.

Rendering Algorithm 1: Region-Based

- *Partial Volume reconstruction*

  In server-client rendering system, data are transmitted to the client site layer by layer. Thus, we need to reconstruct a partial volume and give the user an intermediate rendering result when some part of the data are received. We will describe given a boundary of the volume data in spatial domain, how to pick wavelet coordinates from a whole wavelet volume and reconstruct the volume in general.

Given a starting and ending coordinate $(begin_x, begin_y, begin_z)$ and (end_x, end_y, end_z) of a volume and a desired resolution scale 2^k(as a power of 2). We need to determine its wavelet coefficients in each layer and reconstruct the original volume data at a desired resolution.

Define average image and inner most layer to be layer 0 and outer most layer, that is, the desired resolution layer, to be layer l.

Wavelet coefficients at layer i is from $(begin_x/2^{(l-i)}, begin_y/2^{(l-i)}, begin_y/2^{(l-i)})$ to $(end_x/2^{(l-i)}, end_y/2^{(l-i)}, end_z/2^{(l-i)})$. Reconstruction starts from the average image and the inner most detail coefficients until desired resolution is reached.

- *Progressive Rendering*

  Given a mask with Mask_size=(s,s,s), Mask_center=(x,y,z), num_layer=L and min_length=minlen.

  The boundary of each ROI_i are consistent with the definition in transmission part. In first iteration, the center fovea is reconstructed and rendered using partial volume reconstruction, which is described in previous section. In a later iteration i, 6 sub-volumes: top, bottom, left, right, front and back are separately reconstructed at a resolution scale 2^{i-1}, rendered using standard rendering algorithm and combine the rendered result to previous rendering image.

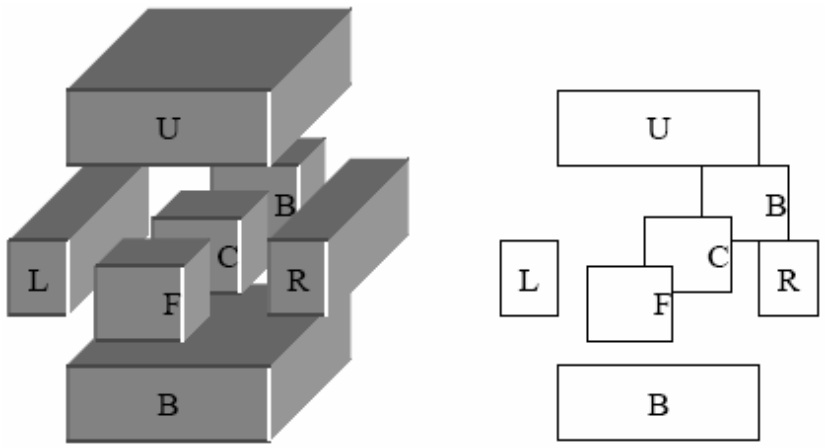


Figure 8: Image Composition

- *Image composition*

  Top, bottom, left and right rendering images obtained in each iteration are simply added to the previous rendering image at their right position, while front and back image needs image composition. For each rendering result, the system obtains both density value and alpha value of each pixel. An *over* operator is performed on front, inner and back image, that is, $(front)$ *over* $(inner)$ *over* $(back)$.

  over operator can be performed pixel by pixel. Given image pixel $A(d_a, \alpha_a)$ and $B(d_b, \alpha_b)$. Resulting pixel is $C(d_c, \alpha_c)$.

$$C = (A)over(B)$$

$$d_c = \alpha_a d_a + (1 - \alpha_a) * d_b$$

Rendering Algorithm 2: Coarse to Fine

Definition of ROI_0 to ROI_{max_layer} are exactly as described in algorithm 1. We start the rendering procedure from ROI_0, which is the average block to the inner most layer ROI_{max_layer}. Each layer, exclude the inner most layer ROI_{max_layer}, can be divided into seven parts, denoted as, *Top, Bottom, Left, Right, Front, Back* and *Center*. The *Center* part is actually the average information for the inner layer and will be kept for the next iteration.

Take layer i ROI_i as an example, which is constructed by the *Center* part of ROI_{i-1} and level i detail coefficents. It is divided into seven parts, and rendered using a slightly modified rendering equation, which is described in the previous section. In ROI_i, the number of voxels each voxel represent is $2^{max\_layer-i}$, so the rendering equation for each voxel in ROI_i should be $I = \sum_{i=1}^{2^{max\_layer-i}} q\alpha(1-\alpha)^{i-1}$.

Seven parts will be rendered seperately and integrated. The center part C_i and the rendering result of *Front* and *Back*, RF_i and RB_i part will be taken down for next iteration. C_i is the average pass for iteration $i+1$, and the rendering result of iteration $i+1$, R_{i+1} will be combined with RF_i and RB_i using the following equation to refine the rendering result of iteration i.

$$R = (RF_i + (1 - alphaRF_i) * R_{i+1}) + (1 - alphaRFi * alphaR_{i+1}) * RB_i$$

Repeat this procedure until the inner most layer is reached.

3.4.3 Rendering Foveated Volume using shear-warp factorization with non-orthogonal viewing direction

When viewing from a non-orthogonal direction, the volume transformation can be simplified by transform each slice to an intermediate coordinate system, sheared object space, for which there is a very simple mapping from the object coordinate system and allows efficient projection. Then warp the intermediate image to the image plane to produce the final result.

Given a viewing direction θ, adjacent voxel slice distance b and adjacent pixel distance d, the rotation can be approximated by shifting each slice to the right by a distance of $l = b * \tan\theta$. To ensure the pixel distance in the resulting image is still d after shearing, the ray distance in the sheared object coordinate should be $m = d/\cos\theta$ (Figure 9). After shearing, re-sampling by interpolation should be performed to get the voxel density value along the viewing ray.

In our scheme, we use wavelet shear-warp factorization. The difference with the above shearp-warp algorithm is we perform it in the wavelet domain instead of spatial domain. Factorization of the low sub-band LLL_i coefficients in fovea area in each level will approximate that of the whole volume. As mentioned above, coefficients in fovea area of all levels is approximately 2000 times smaller than the original volume data size, thus wavelet shear-warp avoids tedious shearing of the whole data set and tremendously reduces the computational complexity. After shearing of wavelet coefficients,

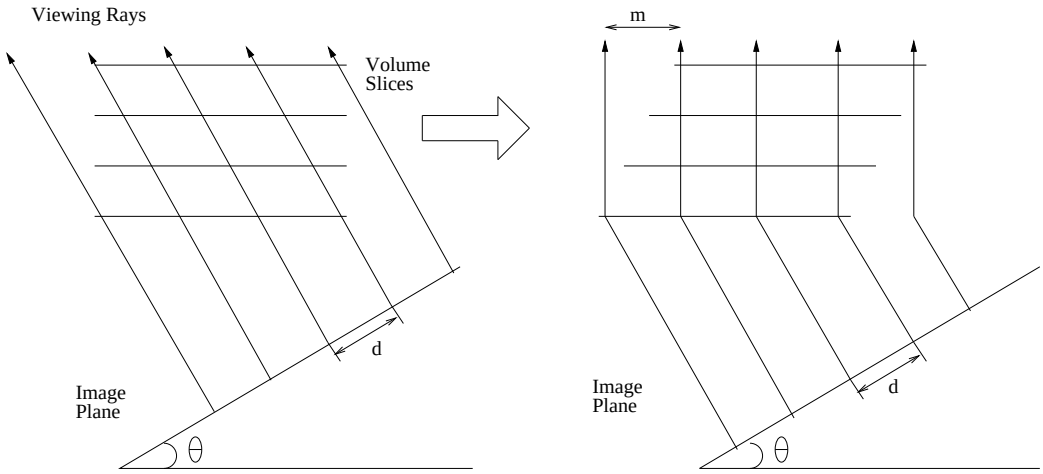


Figure 9: Volume being transformed to sheared object space

they will be input to the algorithm mentioned in orthogonal rendering and produce an intermediate image by normal projection. Warp function will then be executed on the intermediate image to produce the final image.

4 Comparison and Experiment Result of Two Schemes

In our experiment, we use cross-section of human head as our example. The size of the volume data is 256^3, size of the average image is 16^3, thus $max\_layer = log_2(256/16) = 4$. Wavelet volume W is divided into blocks of size 8^3. Given $mask\_center = \{100, 105, 75\}$ and $mask\_size = \{50, 50, 50\}$, we will compare in each iteration, the result image, data amount being transmitted and time of two schemes. DC_j and BI_j represents the index range of detail coefficients and data blocks of ROI_i at layer j of wavelet volume W.

ROI_4: $DC_{3x} = [38, 63]$,$DC_{3y} = [40, 65]$,$DC_{3z} = [24, 49]$

$BI_{3x} = [4, 7]$,$BI_{3y} = [5, 8]$,$BI_{3z} = [3, 6]$

ROI_3: $DC_{2x} = [12, 35]$,$DC_{2y} = [14, 37]$,$DC_{2z} = [6, 29]$

$BI_{2x} = [1, 4]$,$BI_{2y} = [1, 4]$,$BI_{2z} = [0, 3]$

ROI_2: $DC_{1x} = [0, 23]$,$DC_{1y} = [0, 23]$, $DC_{1z} = [0, 19]$

$BI_{1x} = [0, 2]$,$BI_{1y} = [0, 2]$,$BI_{1z} = [0, 2]$

ROI_1: $DC_{0x} = [0, 15]$, $DC_{0y} = [0, 15]$, $DC_{0z} = [0, 15]$

$BI_{0x} = [0, 1]$,$BI_{0y} = [0, 1]$,$BI_{0z} = [0, 1]$

Figure 10 gives a full rendering result for this volume data. Four snapshots in Figure 11 shows the four iterations for scheme 1: Region-Based and Figure 12 shows the four iterations for scheme 2 Coarse-To-Fine. The experiment shows that, in the first Region-Based algorithm (Figure 11), the client can see his most interested region-of-interest(ROI)specified by himself at the

Experiment Result				
		Data Amount	Image	Transmit/Render Time
Iteration I	Scheme 1	618 BLKs	Figure11(a)	0.381/0.0472 sec
	Scheme 2	64 BLKs	Figure12(a)	0.1468/0.047 sec
Iteration II	Scheme 1	436 BLKs	Figure11(b)	0.125/0.0154 sec
	Scheme 2	189 BLKs	Figure12(b)	0.1154/0.0222 sec
Iteration III	Scheme 1	95 BLKs	Figure11(c)	0.0938/0.0156
	Scheme 2	448 BLKs	Figure12(c)	0.1404/0.0156 sec
Iteration IV	Scheme 1	0 BLKs	Figure11(d)	0.0/0.0 sec
	Scheme 2	448 BLKs	Figure12(d)	0.1092/0.0158 sec
Summary	Scheme 1	1149 BLKs	–	0.5998/0.0782 sec
	Scheme 2	1149 BLKs	–	0.5118/0.1006 sec

Table 3: Experiment Results of two progressive rendering scheme for human head

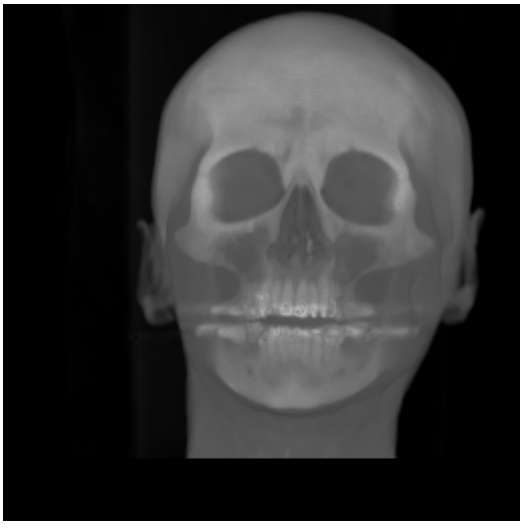
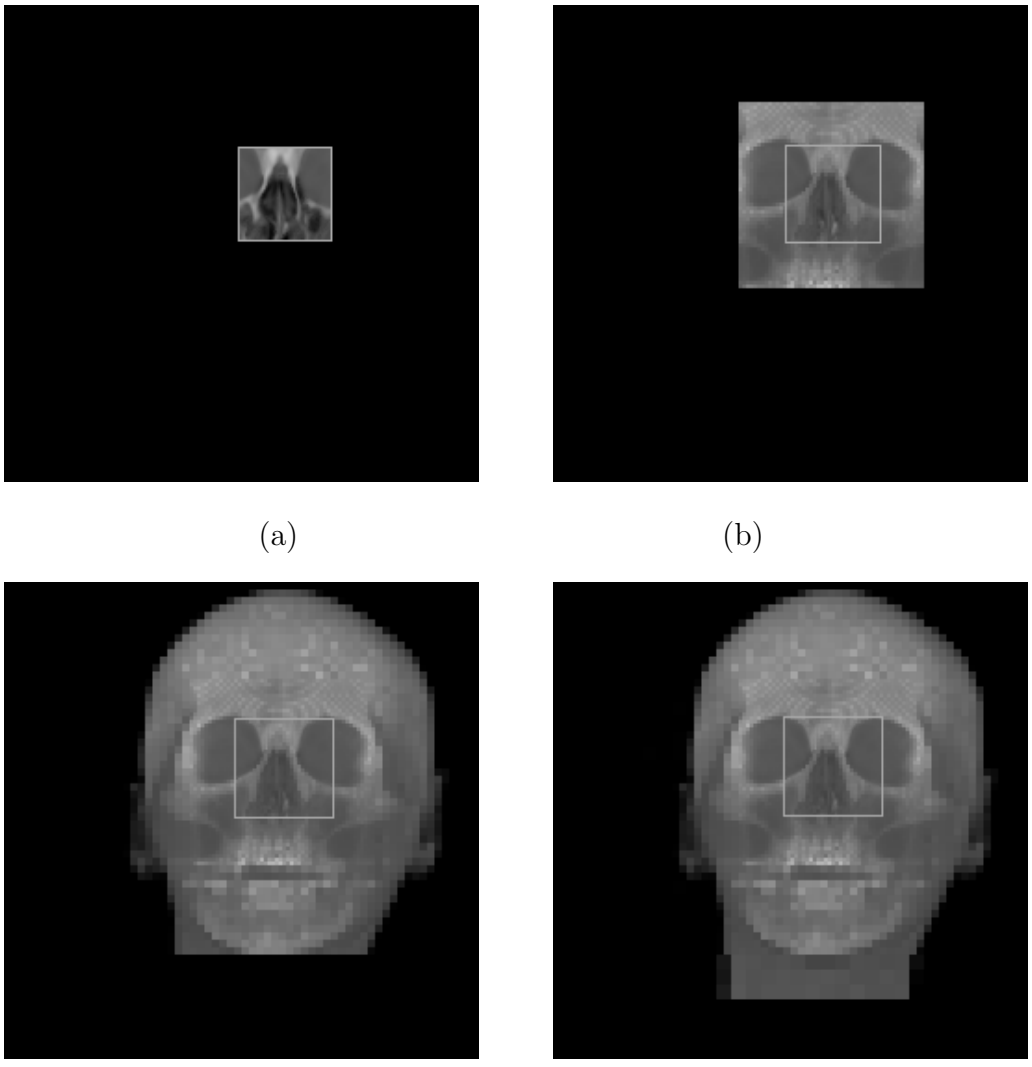


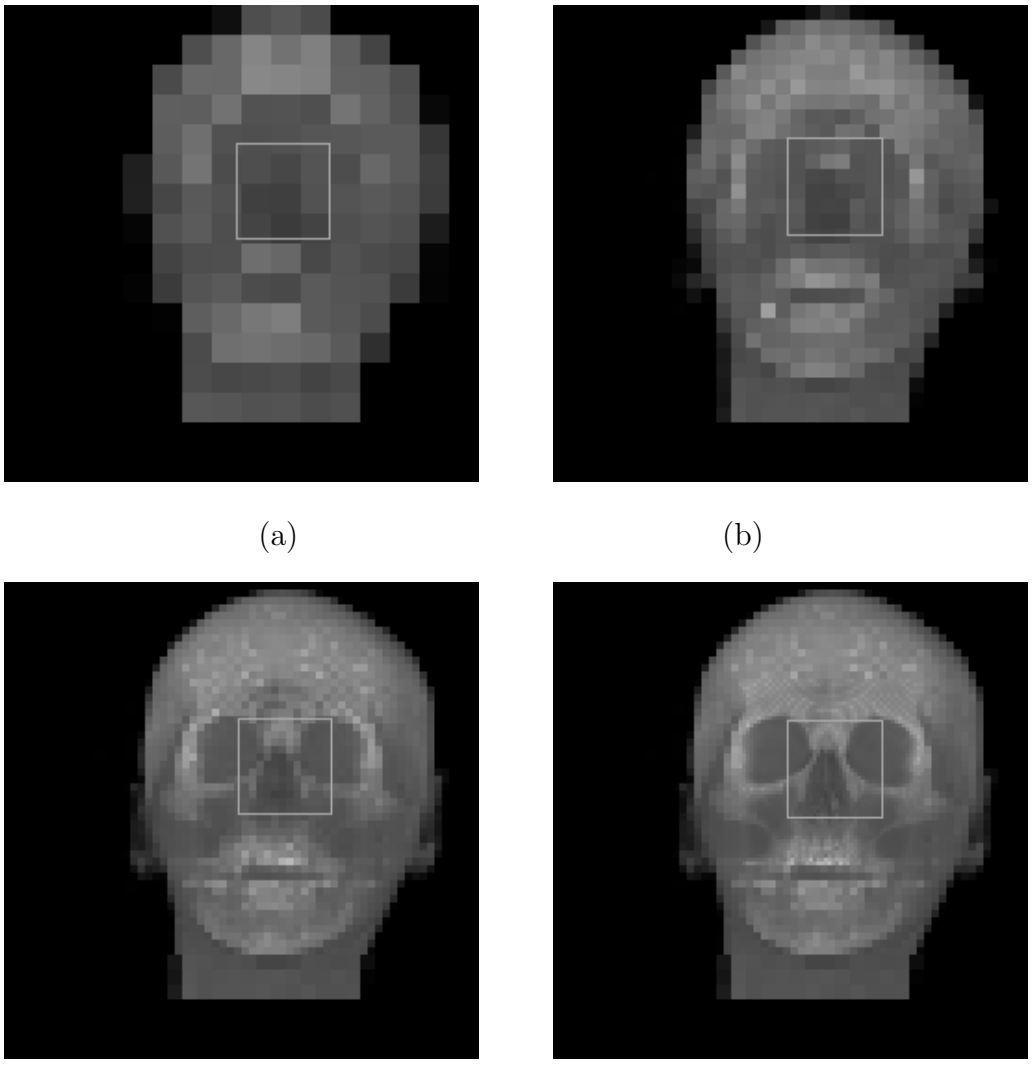
Figure 10: Rendering Result of Direct Rendering



(a) (b)

(c) (d)

Figure 11: 4 Iterations of Progressive Rendering Scheme 1: Region-Based.Fovea at (100,105,75)



(a) (b)

(c) (d)

Figure 12: 4 Iterations of Progressive Rendering Scheme 2: Coarse-To-Fine. Fovea at (100,105,75)

first iteration. The client will have to wait for more than two times of the time (0.5118 second comparing to 0.381 second) for the inner most ROI to arrive in second algorithm: Coarse-To-Fine, which is not desirable. Furthermore, region-based rendering algorithm can give a clearer rendering result of ROI. In Coarse-to-Fine algorithm(Figure 12), we can see that low resolution details arrive first and high resolution inner layer arrives later. Thus its clarity will be interfered by low resolution coefficients. The user cannot have a clear view of his most interested ROI even when all detail coefficients have been received. However, the client waits less time for the first rendering result to be displayed at the client site in Coarse-To-Fine algorithm, 0.1468 second comparing to 0.381 second.

Another example we take here is cross-section of human dummy with size 256^3. Size of the average image is 16^3, $max\_layer = 4$. Given $mask\_center = 80, 150, 100$ and $mask\_size = 50, 50, 50$.

ROI_4: $DC_{3x} = [28, 51]$,$DC_{3y} = [62, 85]$,$DC_{3z} = [38, 61]$

$BI_{3x} = [3, 6]$,$BI_{3y} = [7, 10]$,$BI_{3z} = [4, 7]$

ROI_3: $DC_{2x} = [8, 31]$,$DC_{2y} = [24, 47]$,$DC_{2z} = [22, 35]$

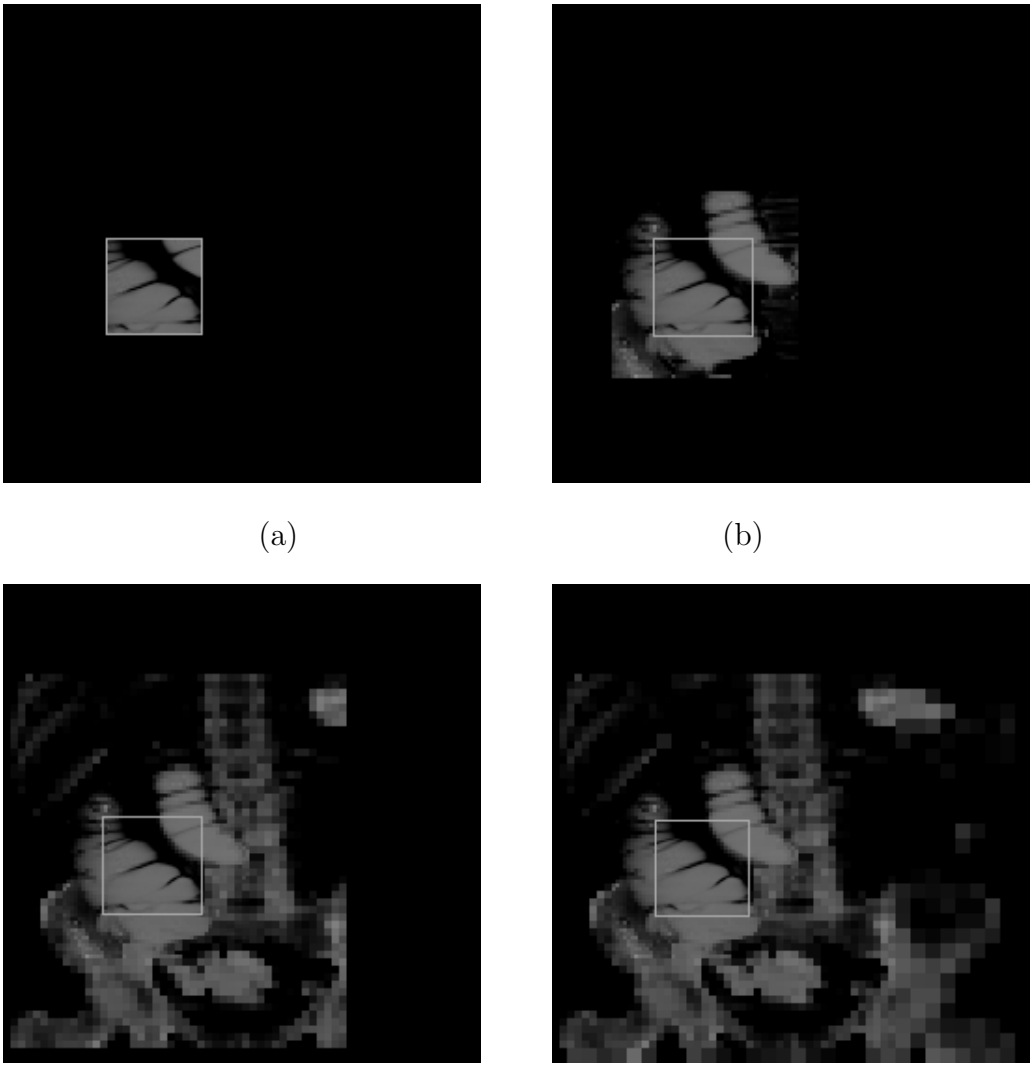
$BI_{2x} = [1, 3]$,$BI_{2y} = [3, 5]$,$BI_{2z} = [2, 4]$

ROI_2: $DC_{1x} = [0, 21]$,$DC_{1y} = [6, 29]$, $DC_{1z} = [0, 23]$

$BI_{1x} = [0, 2]$,$BI_{1y} = [0, 3]$,$BI_{1z} = [0, 2]$

ROI_1: $DC_{0x} = [0, 15]$, $DC_{0y} = [0, 15]$, $DC_{0z} = [0, 15]$

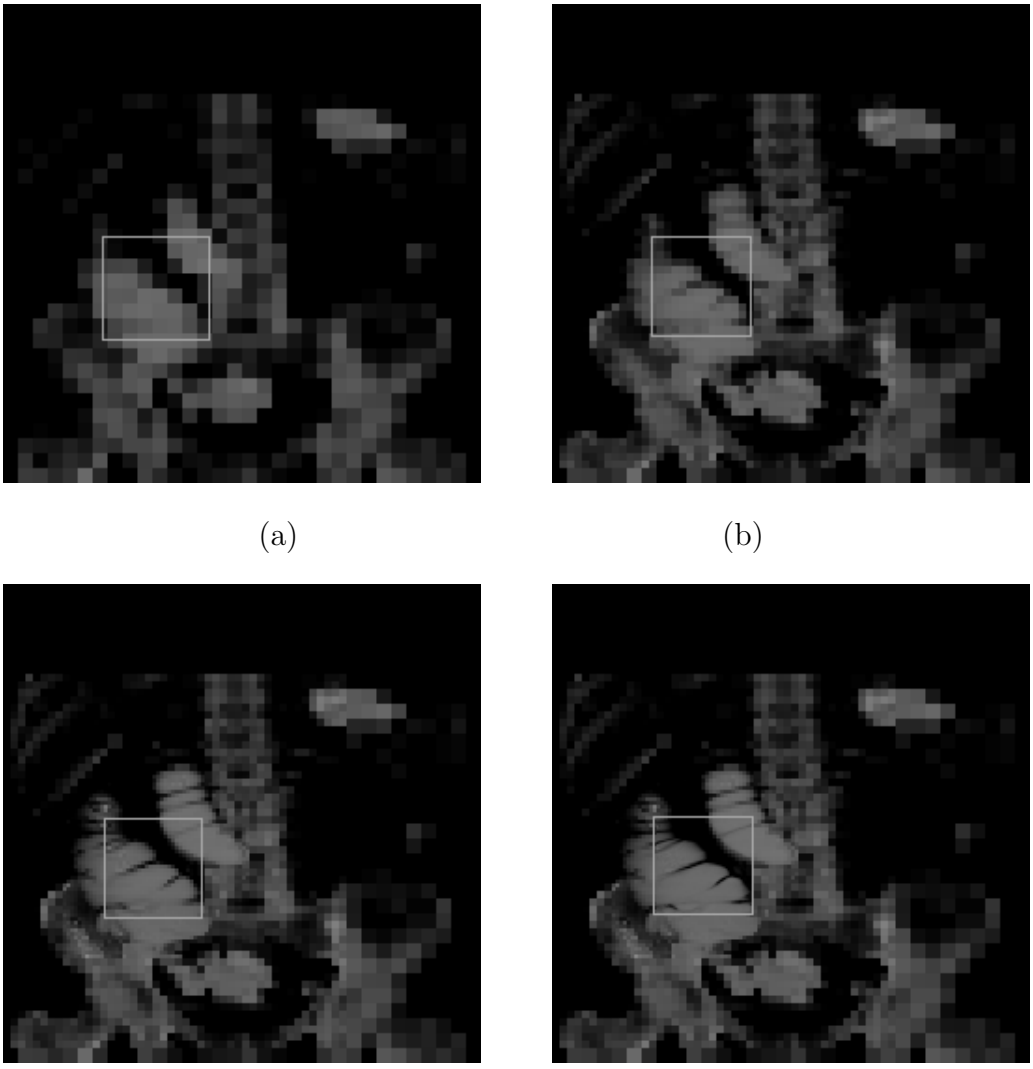
$BI_{0x} = [0, 1]$,$BI_{0y} = [0, 1]$,$BI_{0z} = [0, 1]$



(a) (b)

(c) (d)

Figure 13: 4 Iterations of Progressive Rendering Scheme 1: Region-Based.(Fovea at 80.150,100)



(a) (b)

(c) (d)

Figure 14: 4 Iterations of Progressive Rendering Scheme 2: Coarse-To-Fine. Fovea at(80.150,100)

Experiment Result				
		Data Amount	Image	Transmit/Render Time
Iteration I	Scheme 1	618 BLKs	Figure13(a)	0.372/0.0441 sec
	Scheme 2	64 BLKs	Figure14(a)	0.1470/0.045 sec
Iteration II	Scheme 1	107 BLKs	Figure13(b)	0.052/0.0113 sec
	Scheme 2	252 BLKs	Figure14(b)	0.2041/0.0126 sec
Iteration III	Scheme 1	228 BLKs	Figure13(c)	0.1242/0.0158
	Scheme 2	189 BLKs	Figure14(c)	0.1012/0.0125 sec
Iteration IV	Scheme 1	0 BLKs	Figure13(d)	0.0/0.0 sec
	Scheme 2	448 BLKs	Figure14(d)	0.0861/0.0230 sec
Summary	Scheme 1	953 BLKs	–	0.5482/0.0712 sec
	Scheme 2	953 BLKs	–	0.5384/0.0931 sec

Table 4: Experiment Results of two progressive rendering scheme for human dummy

5 Conclusion

The wavelet-based rendering scheme proposed in this paper aimed at web-based accessing to large scale volume data stored on server repository and multi-resolution progressive rendering on client site.

There are two progressive modes, Region-Based and Coarse-To-Fine. In the first mode, the user on the client site may see the fovea area first and the image will expand iteratively towards the peripheral. In the second mode, the user can immediately sees a preview image before all detail coefficients are received and pixels in the image will be progressively refined depend on the distance to the user specified fovea. Two progressive transmission and rendering scheme are defined for these two modes. Coarse-To-Fine is commonly used in progressive transmission. We proposed Region-Based scheme, which will transmit and render the most interest part of the volume data to

the user at the client site. Both two schemes are compared by both result image and their transmission/process time. Region-Based algorithm has a better view of the fovea at the first sight, while Coarse-To-Fine has shorter waiting time to view the first preview image.

Run-length Encode is used for wavelet compression in both scheme. BLOCK is a convenient unit for both simple communication between server and client and efficient compression of the data.

Wavelet shear-warp factorization is used to accelerate rendering of a non-orthogonal viewing direction which avoids shearing of the whole data set and shears only the fovea instead.

6 Future Work

In our project, when dealing with some large data sets, tremendously large memory are required, since we import the whole data set into the memory at one time when doing wavelet transformation and data extraction. Solution for this problem is to cut the volume data sets that exceeding a certain size into several smaller data sets and build a location map to indicate the actual position of each small data segments. Because of the locality of wavelet transformation, doing wavelet transform for each data segment separately is possible. When a client request is received, the system can refer to the the exact part of the data according to the location map and the coordinate given by client instead of loading the whole data set.

On the client site, higher level-of-detail coefficients are highly reusable, even fovea stated by the user can have significant overlap between each other. Thus, those frequently used BLOCKS can be cached to avoid redundant data transmission in successive client requests.

References

[1] A. Norton and A. Rockwood, Enabling View-dependent Progressive Volume Visualization on the Grid. IEEE Computer Graphics and Applications, 2003

[2] E.C. Chang, S.Mallat, and C. Yap. Wavelet foveation. Journal of Applied and Computational Harmonic Analysis, pages 312-336. 2000

[3] G.R. Thoma, L.R. Long, Compressing and transmitting visible human images. Journal Multimedia, IEEE , Volume: 4 , Issue: 2 , Pages:36 - 45, April-June 1997

[4] H.Yu, V.T. Nguyen and E.C. Chang Rotation Of Foveated Image In the Wavelet Domain

[5] H. Yu and E.C. Chang, Fast Rendering of Rotated Volume Data Using Wavelet Foveation and Shear-warp Factorization.

[6] I. Ihm and S. Park. Wavelet-based 3D compression scheme for very large volume data. Computer Graphics Forum, pages 3-5, 1999

[7] J. P. Schulze, R. Niemeier and U. Lang. The Perspective Shear-Warp Algorithm in A Virtual Environment. Proceedings of the conference on Visualization '01, San Diego, California pages 207-214, 2001

[8] J. Schermann, J. L. Barron, and I. Gargantini. 3D foveated visualization on the web. Proc. Of SPIE on Internet Imaging: Electronic Imaging, pages 349-360, 2001

[9] Kim, T., Shin, Y. An Efficient Wavelet-Based Compression Method for Volume Rendering. Proceedings. 7th Pacific Conference: Computer Graphics and Applications. Pages 147 - 156, 5-7 Oct. 1999

[10] K. Sano, H. Kitajima, H. Kobayashi and T. Nakamura, Parallel processing of the shear-warp factorization with the binary-swap method on a distributed-memory multiprocessor system. Proceedings of the IEEE symposium on Parallel rendering, Phoenix, Arizona, United States, Pages: 87 - ff. 1997

[11] L. Westover. Footprint Evaluation for Volume Rendering. Proceedings of the 17th annual conference on Computer graphics and interactive techniques. Pages 367-376. 1990

[12] M.A. Westenberg and Jos B.T.M. Roerdink, X-Ray Volume Rendering by Hierarchical Wavelet Splatting. International Conference on Pattern Recognition (ICPR'00)-Volume 3, Barcelona, Spain, September 03 - 08, 2000

[13] M. Gross, L. Lippert, R. Dittrich, and S. H?ring. Two methods for wavelet-based volume rendering. Technical Report 247, ETH-Zrich, 1996

[14] M. Levoy and R. Whitaker. Gaze-directed volume rendering. In Proceedings of the 1990 symposium on Interactive 3D graphics, pages 217-223. ACM Press, 1990.

[15] M. Meissner, J. Huang, D. Bartz, K. Mueller, and R. Crawfis A practical Evaluation of Popular Volume Rendering Algorithms. Proceedings of the 2000 IEEE symposium on Volume visualization. Pages 81-90 2000

[16] P. Lacroute and M. Levoy, Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation. Proceeding. SIGGRAPH '94, Orlando, Florida, pages 451-458. July, 1994

[17] P. Pinnamaneni, S. Saladi, J. Meyer, 3D Haar Wavelet Transformation and Texture-Based 3D Reconstruction of Biomedical Data Sets. Visualization, Imaging and Image Processing (VIIP 2001), The International Association of Science and Technology for Development (IASTED), Marbella, Spain, ACTA Press, pages 389 - 394, Sept. 3 - 5, 2001

[18] P. Pinnamaneni, S. Saladi, J. Meyer, Remote transformation and local 3D reconstruction and visualization of biomedical data sets in Java3D. Journal Visualization and Data Analysis, Proc. SPIE Vol. 4665, p. 44-54, 2002

[19] P. Pinnamaneni, Wavelet-Based Volume Rendering. Ph.D thesis 2003

[20] P. Schroder and G. Stoll, Data Parallel Volume Rendring as Line Drawing. Proceedings of the 1992 workshop on Volume visualization. Boston, Massachusetts, United States Pages 25-32. 1992

[21] R. Grosso and T. Ertl and J. Aschoff, Efficient Data Structures for Volume Rendering of Wavelet-Compressed Data. The 4th International Conference in Central Europe on Computer Graphics and Visualization'96 (WSCG96) February 12.- 16., 1996

[22] S. Guthe, M. Wand, J.s Gonser and W. Straber, Interactive Rendering of Large Volume Data Sets. Proceeding IEEE Visualization 2002, IEEE CS Press, pages 53-60. 2002