FAST DATA-PARALLEL RENDERING OF DIGITAL VOLUME IMAGES

By

Song ZOU

A Thesis

submitted in partial fulfillment of the requirements

for the Degree of Master of Philosophy

Division of Information Engineering

The Chinese University of Hong Kong

June 1995

香港中文大學
大學
圖書館系統
2 7 JUL 1998
UNIVERSITY
LIBRARY SYSTEM
THE CHINESE UNIVERSITY OF HONG KONG

Acknowledgement

I would like to express my gratitude to my supervisor, Dr. W. Y. Ng, for his always being supportive and his guidance and help throughout the two years.

I would also like to thank the members of Visual Information Lab, H.Z. Li, W. Jiang, J. Yao, W. K. Cheung, Maxi Hui and Edward Yau, for their helpful discussion, advice and support.

Finally, this thesis is dedicated to my parents and Amanda Kuet.

Abstract

In the fast growing field of visualization in scientific and medical data, volume rendering has emerged as a useful technique for inspecting volumetric objects. With appropriate assignment of color, degree of translucence and brightness to voxels, the rendered scene may be seen "through" and its structure revealed. With different perspectives, the spatial relationship within the volume may be examined. However, since the best parameters for rendering cannot be determined as a priori, it is desirable to adjust them interactively.

Interactive volume rendering is highly computation intensive since operations are often repeated on large volume of data. SIMD parallel machines, where one instruction is carried simultaneously on multiple processors, suit for the data parallel algorithms in volume rendering.

This thesis discusses data parallel approaches to volume rendering. Colors and opacities are assigned to each voxel. Then, they are resampled and composited according to different perspectives. Techniques in different phases of rendering such as data parallel resampling, ray composition as well as the implementation issues on the parallel platform are given particular attention.

In data parallel resampling, we present a new matching algorithm which is minimum in mismatch distance and flexible for different resampling schemes.

In parallel implementation, we propose a binary swap algorithm to reorder the data in the SIMD processor array with low communication overhead. The Monte Carlo ray composition is a fast algorithm for efficient ray composition.

Contents

Chapter 1

Introduction

The technology advances in computed tomography (CT) and magnetic resonance imaging (MRI) , confocal scanning microscopy, etc. as well as large scale computer simulation complex systems such as fluid flow etc. have made 3D digital image become popular. The analysis of such data set is by means of visualization. Volume rendering, as an emerging field in the area of *Visualization in scientific computing* (ViSC) provides the ideal tool of such analysis.

By means of visualization, meaningful information from complex volume data is presented by interactive graphics and imaging. Two kinds of techniques are often used here, namely the graphic method and the volumetric method. In a graphic approach, geometric surface primitives, such as polygons, curves are fitted to the volume data with a binary classification to extract such primitives. Therefore, the error due to the inaccuracy in classification may lead to visual artifacts in the generated image. These artifacts may manifest themselves as spurious surfaces (false positives) or erroneous holes (false negatives)[5]. Volume Rendering, the volumetric technique, starts directly from the voxels and avoids any binary classification. It is supposed to be free of visual artifacts due to the error of extraction and possible to interpret complex volume structure with no obvious graphic primitives. In addition, the volumetric method may interpret volume data as transparent material with interior structures being seen through the translucent surface.

When volume data are rendered from 3D to the 2D screen, it is inevitable that the information shown is significantly reduced. However, interactive speed to different perspectives and opacities may compensate the loss by providing time as another dimension. To be interactive, the rendering speed should be at least two or three frames per second, for which highly intensive computation is required considering the size of volume data. For example, a volume of 256^3 will need a 500 Mflops computation power to afford 5 operations per voxel, 2 times per second. The development of parallel machine, especially the SIMD (Single Instruction Multiple Data) parallel architecture provides a possible solution to the requirement of interactivity. An SIMD machine, where an instruction is executed simultaneously on multiple data, may reduce the processing time significantly by efficiently exploiting the parallelism in the rendering process.

Recent years, there have been many researches on volumetric rendering as well as their implementations on parallel platforms. There will be a survey on the related researches in chapter 2.

We based our research on Levoy[17]'s volume rendering model shown in figure 1.1. The voxel value is used as the input of two independent processes, namely the shading process and the classification process. The shading process maps the input value to color and yields the effect of illusion of smooth surfaces. The classification process maps the input value to opacity, which is the attenuation

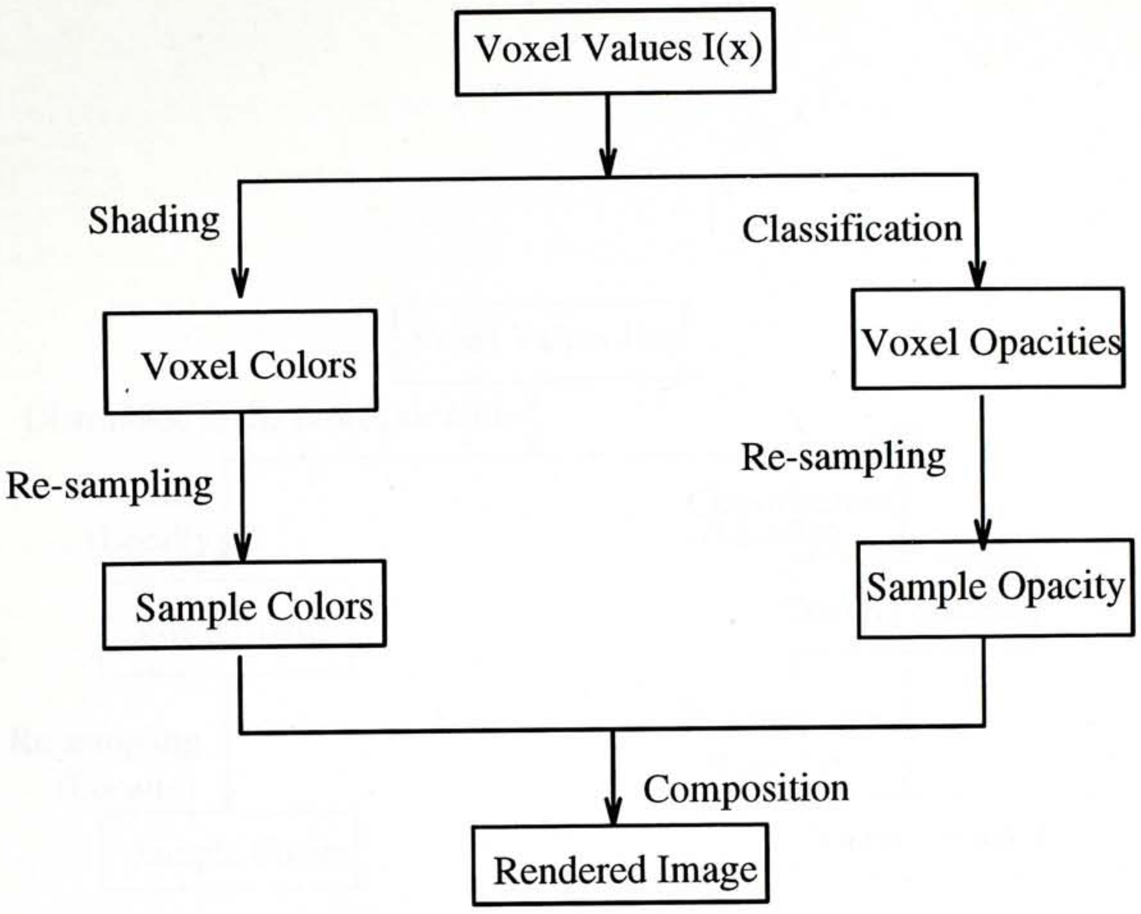

Figure 1.1: The rendering pipeline

property of each voxel. The colors and opacities are then resampled according to a given perspective. The final step, composition, composites the voxels in the order of back to front or front to back of the viewing perspective.

This thesis discusses the parallel implementation of the volumetric rendering technique. In parallel algorithms, the computation is carried simultaneously by many processors. Based on the rendering pipeline described above, the volume data are distributed into the parallel processors. While the processes of shading, classification and resampling are possible to be carried independently on each processor using only local information, the composition must combine the data on the entire ray which are often in different processors. Moreover, in order to display the rendered image, it is necessary to have it laid in the processor

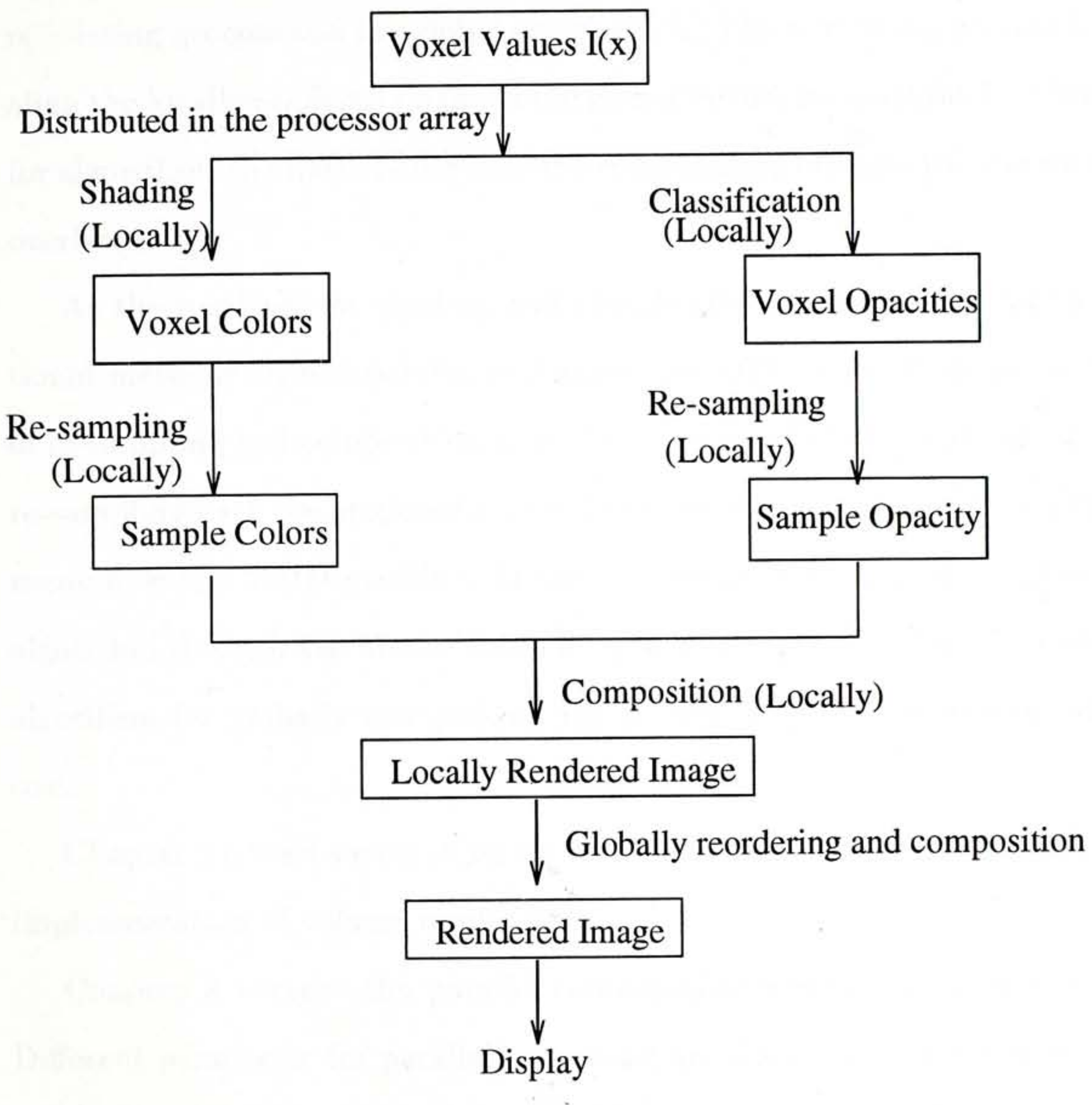

Figure 1.2: The rendering pipeline

array according to a certain virtualization scheme. Therefore, communication between the processors is inevitable.

The parallel rendering pipeline designed in this thesis is shown in figure 1.2. The composition is pipelined into three parts: the local composition, the reordering process and the global composition. The reordering process is used to align the locally rendered image to the global virtualization model. It is essential for algorithms to minimize not only the computation but also the communication overhead.

As the methods for shading and classification may be identical to conventional methods on non-parallel platforms, special focuses are given on the part of re-sampling and composition as well as their parallel implementations. In the re-sampling part, we proposed a new algorithm which could be efficiently implemented on the SIMD machine. In the ray composition part, we proposed a fast algorithm through the Monte Carlo integration. Moreover, there is a reordering algorithm for globally compositing the volume at a minimum communication cost.

Chapter 2 gives a survey of recent researches on volume rendering and parallel implementation of volume rendering.

Chapter 3 surveys the parallel computation models for volume rendering. Different paradigms for parallel processing are discussed and compared. Detail descriptions are given to the SIMD architecture: the model, the communication mechanism and the display mechanism.

Chapter 4 presents the preprocessing part. The voxel values are loaded into the processor array in a block by block manner; the color of each voxel is obtained by a shading process; the opacity of each voxel is obtained by a classification process.

Chapter 5 focuses on data parallel rotation and resampling algorithms. By leaving the data in place, the coordinates in the object space will be mapped to the viewing space for consequent resampling. A new fast data parallel rotation and resampling scheme on the SIMD machine is given.

Chapter 6 gives a data reordering algorithm to align the resampling and local composition results to the viewing space layout in the processor. Only regular communication patterns are used.

Chapter 7 surveys the previous techniques for ray composition, presents a new ray composition method by Monte Carlo integration, demonstrates its use in parallel platforms. Although there are rooms for improvement, it does have some advantages over the conventional methods.

Chapter 8 gives the conclusion and possible future works.

Chapter 2

Related works

The basic idea of volume rendering is to simulate light transmission through the volume. In 1984, Kajiya[14] modeled the problem as a radiation transport problem with a complex differential equation. Since then, other simplified methods for fast implementation are also presented. Generally, three kinds of effects of the materials in the volume are shown[4]: (a) they are luminous and emit light; (b) they act as translucent filters absorbing the incoming light and (c) they contain surfaces or particle scatterers which attenuate as well as reflect light. Separating the processing of shading and classification, the Levoy's model described in chapter 1 provides a simple yet efficient way of volume rendering. Many variants of it have also been proposed; most of them give improvement in one or two components of the rendering pipeline(figure 1.1). Since shading and classification are able to be pre-processed, resampling and composition are the critical parts when interactivity is concerned. As the focus of this thesis is fast algorithms of volume rendering, we will give a brief survey of the researches on resampling and composition methods.

One way to accelerate resampling and composition is by exploiting the spatial coherence: Data are organized as octree where those with same properties are grouped as one cell. The cells are processed first to provide a perspective independent footprint. As the result, the number of voxels to be processed for resampling and composition is reduced. Progressiveness is achieved by refining the hierarchical data structure.

To evaluate the footprint of each cell, Westover[26] presents an algorithm that allows the renderer to use a pre-computed footprint function table to build the view-transformed footprint of a particular view.

Another way is to take advantage of the properties of certain transforms to reverse the order of composition and resampling. In this way, resampling is only carried on the composited image which is two dimensional only. Fourier transforms[25], wavelet transforms[20] and shear-warp transforms[15] are used for this purpose.

However, not all methods mentioned aboved are suitable for parallel implementation, which requires operations use only local information and the workloads on each processor are about the same to fully utilize the SIMD machine. Hsu[13] presents an implementation on the DEC-mpp SIMD machine. The data are allocated in the processors in a cell by cell manner; the composition is carried in the order of shooting light.

2.1 Spatial domain methods

Levoy[18] improved his model by hierarchical enumeration and adaptive termination for fast processing. Hierarchical enumeration is by grouping the voxels with opacities zero together and processing only once when compositing. Adaptive termination is to terminate the ray tracing when the rest of the ray has less than the threshold significance to the final composited image. These methods reduce the amount of computation. However, they are not suitable for parallel implementation because global information of the volume is needed.

Westover[26]'s footprint evaluation provided an approach suitable for parallel implementation. The view-transformed footprint of every sub-cell of the volume is built by the pre-computed footprint table independently.

Based on Westover's method, the hierarchical splatting by Laur and Hanrahan[16] uses a pyramidal volume representation. An octree is used to fit the pyramid given the user-supplied precision. This octree is then drawn using a set of footprints, each scales to match the size of the projection of a cell. This method significantly reduces the number of cells. Moreover, it is a progressive method which is very useful for interactive applications. It is ideal for volumes with simple structures. For medical data, there may be not much voxels with the same properties that can be grouped together in the octree, therefore the number of cells is not reduced that much to justify the cost of providing such a hierarchical data structure.

2.2 Transformation based methods

Totsuka and Levoy[25] proposed the frequency domain rendering method due to the Fourier Projection Slice Theorem: once volume data are Fourier transformed, an (orthographic) image for any viewing direction can be obtained by inversely transforming a 2D slice extracted from the 3D spectrum at the given orientation.

Linear depth cueing and directional diffuse reflection are used. Depth cueing is obtained by weighting voxels according to their distance from the observer. The resampling process is only used for extracting the 2D slice and the cost of implementation is dominated by the 2D inverse Fourier Transform. However, since the composition is not a linear process to the voxel value, it cannot be replaced by a linear transformation. Only the linear process such as depth cueing may be simulated.

Muraki[20] proposed a hierarchical method based on wavelet transform. By using a smoothly decreasing 3D orthonormal wavelet, a hierarchical structure of local primitives at different scales is obtained. However, again the nonlinearity makes the process of composition difficult to replicate.

Lacroute and Levoy[15]'s shear-warp algorithm decomposes the transformation into a two-pass process. The composition is after the first and the resampling after the second. The advantage of this method is that the resampling process is in 2D only; and the disadvantage is resampling quality may be lowered.

2.3 Parallel Implementation

Hsu[13] implemented his parallel volume rendering algorithms on the Dec-mpp SIMD machine: Volume data are divided into many identical cubes and distributed in the parallel processors; the composition is by shooting the ray into the voxels and the samples are computed as the ray being traced. However, the processors are not fully utilized since some processors have to wait for the ray to arrive.

Schröder[23]'s implementation avoids this problem by transforming the coordinate only and leaving the data in their place. A five-pass decomposition of rotation transform is used. However, overhead is introduced to keep track of the relative movement of neighbors and the quality of resampling is lowered due to sequential lossy filtering. Finally, the communication overhead occurs when the data are aligned to the viewing space for composition.

Ma et al.[19] proposed a binary swap algorithm during the time the communication scheme in this thesis was derived. His implementation is on a group of workstation. Similar to the methods used in this thesis, the binary swap algorithm uses only local communication to align the volume in the processors to the viewing space.

Chapter 3

Parallel computation model

3.1 Introduction

Before we explore the problem, we explore the tools: the platform able to render the volume data fast and efficiently. Such platform shall have sufficient computation power as well as the mechanism to display the result of interactive volume rendering.

The huge demand for computation capacity recent years has led to the widespread availability of parallel processing facilities. By combining the resources of certain number of processors, the theoretical system performance may be increased to match the processing demand of interactive volume rendering. For example, DEC-mpp 12000 Massively Parallel Computer, with 8192 processor elements, has a peak performance of 665 MFLOPS and 13000 MIPS, which is able for more than 5 operations per voxel and 2 times per second for a 256^3 volume.

In this chapter, the general concepts of parallel processing are reviewed.

Then within the context of volume rendering, systems with different paradigms are described and compared. Focus is on the SIMD paradigm: its architecture, communication mechanism and display mechanism.

3.2 Classifications of Parallel Computers

A number of schemes have been designed for classifying the types of parallel computation model[9]. The most widely quoted is the *Flynn's Taxonomy*[8], where different types of parallel computers are distinguished by the relationship between the instructions executed by a machine, and the data upon which these instructions operate. Whether the instruction stream and the data stream are multiple or single results in the following four categories:

Single Instruction stream, Single Data stream (SISD), which is the conventional sequential computer. A single instruction processing unit executes sequentially the operations in the instruction stream on one data stream, although the instructions may be pipelined.

Single Instruction stream, Multiple Data stream (SIMD), which usually consists of a grid of Parallel Element(PE)s and a central processor. The central processor broadcasts an instruction to the PEs , which execute the same instruction on separate data.

Multiple Instruction stream, Single Data Stream (MISD). In this class, processors execute distinct instruction streams on the same stream of data. No practical realization of this category has been forthcoming.

Multiple Instruction stream, Multiple Data Stream (MIMD). In contrast to SIMD systems, the MIMD systems consist of distinct processors which operate asynchronously, each on a distinct set of data, without central processor. Also, there are interaction mechanisms between the processors for possibly sharing the resources.

The most powerful paradigms of parallel processing which have so far seen extensive use are the MIMD and SIMD models[21].

The MIMD model suits for control-parallel algorithms where distinct tasks are distributed between the processors for concurrent processing. Mechanisms to support asynchronous communication are supported in addition to a programming environment which provides task scheduling, allocation, synchronization and coordination.

The SIMD model, on the other hand, suits for data-parallel algorithms. Not only a single instruction is carried simultaneously on all the PEs, but also the communication within the processors is synchronous. This simplicity make the number of PEs supported by the SIMD model much more than the number by the MIMD model. In addition, design as well as implementation of the algorithms is much easier.

Volume rendering algorithms, where most of the operations should be repeated on the vast amount of volume data, are data parallel in general. Moreover, the communication incurred in parallel volume rendering is synchronous in general. These make the SIMD model well suited for volume rendering. In the following sections, we will discuss the SIMD model: its architecture, its communication mechanism and its display mechanism.

3.3 The SIMD machine architectures

A massively data parallel system is a system with more than one thousand processors. The system used in this thesis is a massively data parallel system: the DEC-mpp 12000 which is identical to the Maspar System[3]. It consists of a front end and a Data Parallel Unit (DPU). The front end includes a workstation that runs an implementation of a UNIX operating system and standard I/O. The DPU consists of an Array Control Unit(ACU), an array of 8,192 Processor Element(PE)s (extentable to 16,384PEs), and communication mechanisms. The DPU has its own I/O and display functions.

The volume data are first read through the I/O functions built in the DPU for consequent operations in the PEs. Finally the rendered image stored in the PEs is displayed through the display function built in the DPU. Two kinds of overheads are used to evaluate the algorithms, namely, the computation overhead and the communication overhead. The computation overhead is measured by the number of additions, multiplications and comparisons. For the example of the DEC-mpp system 1200, the timing for addition and multiplication in PEs of character operands is 6 and 41 timing clocks respectively, which means that multiplication is much more expensive than addition. In our algorithms, additions are preferred.

The communication overhead is the time for the processors in the DPU to exchange their data. In the next section, we will discuss the communication mechanism in the SIMD architecture and measure the communication overhead.

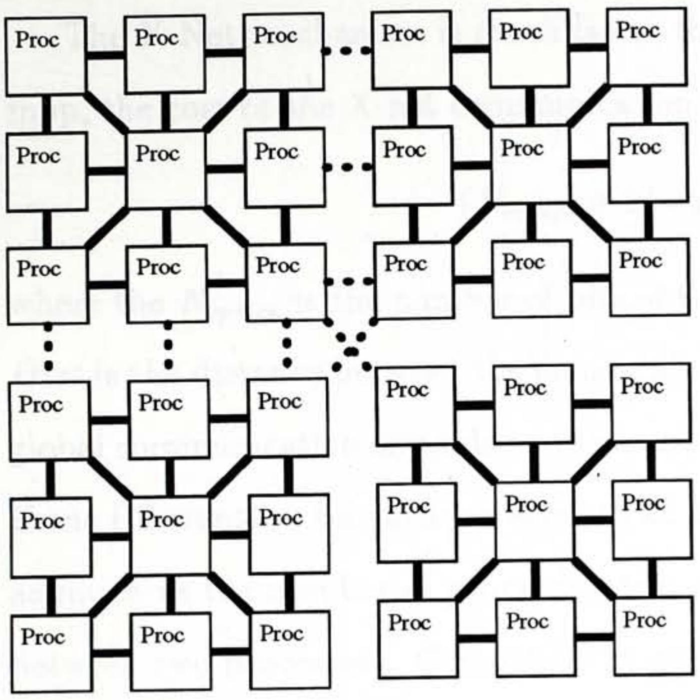

Figure 3.1: The processors elements arrays in the SIMD structure

3.4 The communication within the parallel processors

The communication of the massive parallel machine consists of the communication between the PEs within the DPU as well as the communication between the DPU and the front end.

We will focus on the communication within the parallel processors first. Figure 3.1 shows an example of the PE grids inter-connected in eight directions: north, northeast, east, southeast, south, southwest and northwest. There are two ways of communication within the parallel processors, namely the X-Net Communication and the Global Router Communication. The X-Net mechanism allows simultaneous data transfer in a single direction for a fixed distance; the Global Router enables an arbitrary subset of processors to communicate with any other subset of processors.

The X-Net mechanism is much faster than the Global Router. In the DEC-mpp, the cost of the X-net communication is:

$$(N_{opsize} + 2) * Dist + 6 \tag{3.1}$$

where the N_{opsize} is the number of bits of the operand to be transferred and the $Dist$ is the distance between the communicating parallel processors. The cost of global communication depends on the number of collisions in the communication. If one PE wants to broadcast a value to all other PEs, it may take approximately as much as the number of processors times the time a random communication between two processors. Generally, the global communication costs much more than the X-net communication. For example, a random communication pattern with all PEs participating take an average of 5000 timing clocks for a 32 bit operand while the X-Net communication takes much less according to equation 3.1.

Since the success of our algorithms heavily depends on a low communication overhead, the X-Net communication is used whenever possible.

3.5 The parallel display mechanisms

The parallel display mechanism is used in the final step to display the rendered results laid in the PEs. Sending the data from PEs to the front end for display, it is equivalent to the communication between the front end and the DPU.

It is very important to understand the data format used for display, since the identical data set may result in different images if different data formats are used. Suppose the size of the image is equal to the size of the processor array, the format of display becomes trivial: the coordinates of each processor

(0,0) (1,0)	(2,0) (3,0)	(4,0) (5,0)	(6,0) (7,0)
(0,1) (1,1)	(2,1) (3,1)	(4,1) (5,1)	(6,1) (7,1)
(0,2) (1,2)	(2,2) (3,2)	(4,2) (5,2)	(6,2) (7,2)
(0,3) (1,3)	(2,3) (3,3)	(4,3) (5,3)	(6,3) (7,3)
(0,4) (1,4)	(2,4) (3,4)	(4,4) (5,4)	(6,4) (7,4)
(0,5) (1,5)	(2,5) (3,5)	(4,5) (5,5)	(6,5) (7,5)
(0,6) (1,6)	(2,6) (3,6)	(4,6) (5,6)	(6,6) (7,6)
(0,7) (1,7)	(2,7) (3,7)	(4,7) (5,7)	(6,7) (7,7)

The hierarchical model

(0,0) (4,0)	(1,0) (5,0)	(2,0) (6,0)	(3,0) (7,0)
(0,4) (4,4)	(1,4) (5,4)	(2,4) (6,4)	(3,4) (7,4)
(0,1) (4,1)	(1,1) (5,1)	(2,1) (6,1)	(3,1) (7,1)
(0,5) (5,5)	(1,5) (5,5)	(2,5) (6,5)	(3,5) (7,5)
(0,2) (4,2)	(1,2) (5,2)	(2,2) (6,2)	(3,2) (7,2)
(0,6) (4,6)	(1,6) (5,6)	(2,6) (6,6)	(3,6) (7,6)
(0,3) (4,3)	(1,3) (5,3)	(2,3) (6,3)	(3,3) (7,3)
(0,7) (4,7)	(1,7) (5,7)	(2,7) (6,7)	(3,7) (7,7)

The cut and stack model

Figure 3.2: The hierarchical model and the cut and stack model

become the coordinates of the displayed image. However, in most of the cases, the image size is greater than the size of the PE array. Therefore, some kinds of virtualization techniques are used. In the DEC-mpp, two kinds of virtualizations are supported, namely Hierarchical model and Cut-and-stack model.

Suppose there are 2^n pixels in the image and 2^N $(N \leq n)$ processors, in either X or Y direction, where the data coordinate is $x(y)$ and the processor coordinate is $X(Y)$.

In Hierarchical virtualization, virtual neighbors (neighboring pixels in the image) are stored on the same PE whenever possible. The relationship between the processor coordinate and the data coordinate is :

$$x \ DIV \ 2^N = X \tag{3.2}$$

If the coordinates are represented in binary, the N most significant bits of the data coordinate are identical to the processor coordinate.

In the Cut-and-stack virtualization, virtual neighbors are stored on the neighboring processor when ever possible. The relationship between the processor coordinate and the data coordinate is :

$$x \ MOD \ 2^N = X \tag{3.3}$$

If the coordinates are represented in binary, the N least significant bits of the data coordinate are identical to the processor coordinate.

Figure 3.2 assumes an 8×8 image and a processor mesh of 4×4. The layout on the left is according to the Hierarchical model and the one on the right is according to the Cut and stack model.

It is essential that the final result of the rendered image is laid in the processor array according to either one of the virtualization models to be displayed.

Chapter 4

Data preparation

4.1 Introduction

This chapter includes three parts: the original data layout in the processor array, the shading process and the classification process. These topics are not the main points of this thesis and are presented mainly for completeness.

The topics in this chapter are regarded as the preprocessing steps. The original data should firstly be distributed in the parallel processor array. According to the rendering pipeline described in chapter 1, the only information provided is the voxel value; the materials which composite each voxel are often not provided. Opacity and color of the voxel, which are decided by which material it is composed of, should be determined by the shading and classification processes respectively.

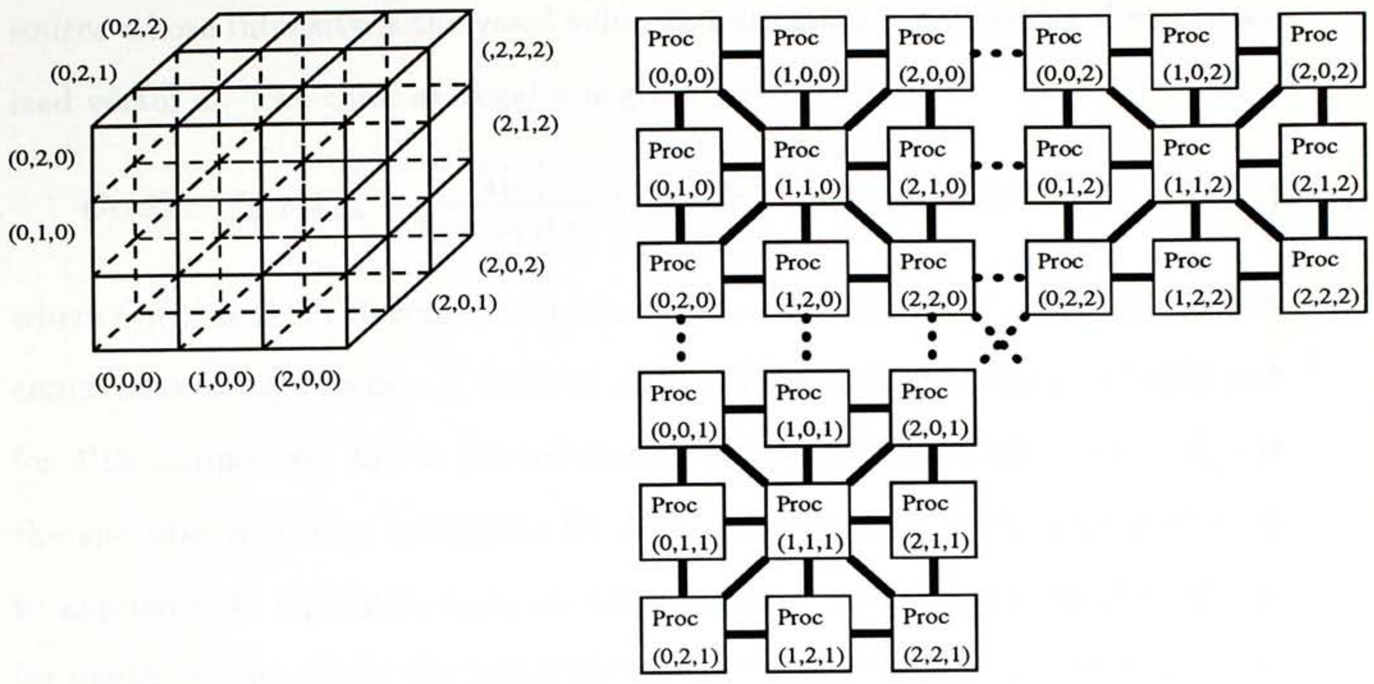

Figure 4.1: The Data layout in the processors

4.2 Original data layout in the processor array

The original data are three dimensional digital data. To be held in a two dimensional processor array, they are organized in each processor in a block by block manner. That means every processor holds a cell of the whole image. (See figure 4.1)

4.3 Shading

The shading process maps the original voxel value to color and provides a satisfactory illusion of smooth surfaces at a reasonable cost. The shading model chosen was developed by Phong[2]. Every voxel is regarded as a parallel light source whose intensity is the voxel value and direction is defined by the normalized vector L. The color at voxel x is given by:

$$C_\lambda(x) = f_\lambda(x)k_{a,\lambda} + \frac{V_\lambda(x)}{k_1 + k_2 d(x)}[k_{d,\lambda}(N(x) \cdot L) + k_{a,\lambda}(N(x) \cdot H)^n] \qquad (4.1)$$

where $C_\lambda(x)$ is the λ'th component of color at voxel location x[1]; $f_\lambda(x)$ is the λ'th component of light source at location x; $k_{a,\lambda}$ is the ambient reflection coefficient for λ'th component; $k_{d,\lambda}$ is the reflection coefficient for λ'th component; $k_{s,\lambda}$ is the specular reflection coefficient for λ'th component; n is the exponent used to approximate highlight; k_1,k_2 are the constants used in linear approximation for depth-cueing, $d(x)$is the perpendicular distance from picture plane to voxel location x;$N(x)$ is the surface normal at voxel location x; H is the normalized vector in the direction of maximum highlight.

Since a parallel light is used, L is a constant. Furthermore

$$H = \frac{V + L}{|V + L|} \qquad (4.2)$$

where V is the normalized vector in direction of observer. Since the projection is orthographic, V and H are constants. Finally, the surface normal $N(X)$ is given by

$$N(x) = \frac{\nabla f(x)}{|\nabla f(x)|} \qquad (4.3)$$

where the gradient vector $\nabla f(x)$ is approximated using the operator:

$$\begin{aligned}\nabla f(x) &= \nabla f(x_i, y_j, z_k) \\ &\approx [0.5(f(x_{i+1}, y_j, z_k) - f(x_{i-1}, y_j, z_k)), 0.5(f(x_i, y_{j+1}, z_k) - f(x_i, y_{j-1}, z_k)), \\ &\quad 0.5(f(x_i, y_j, z_{k+1}) - f(x_i, y_j, z_{k-1}))]\end{aligned}$$

[1]$\lambda = r, g, b$

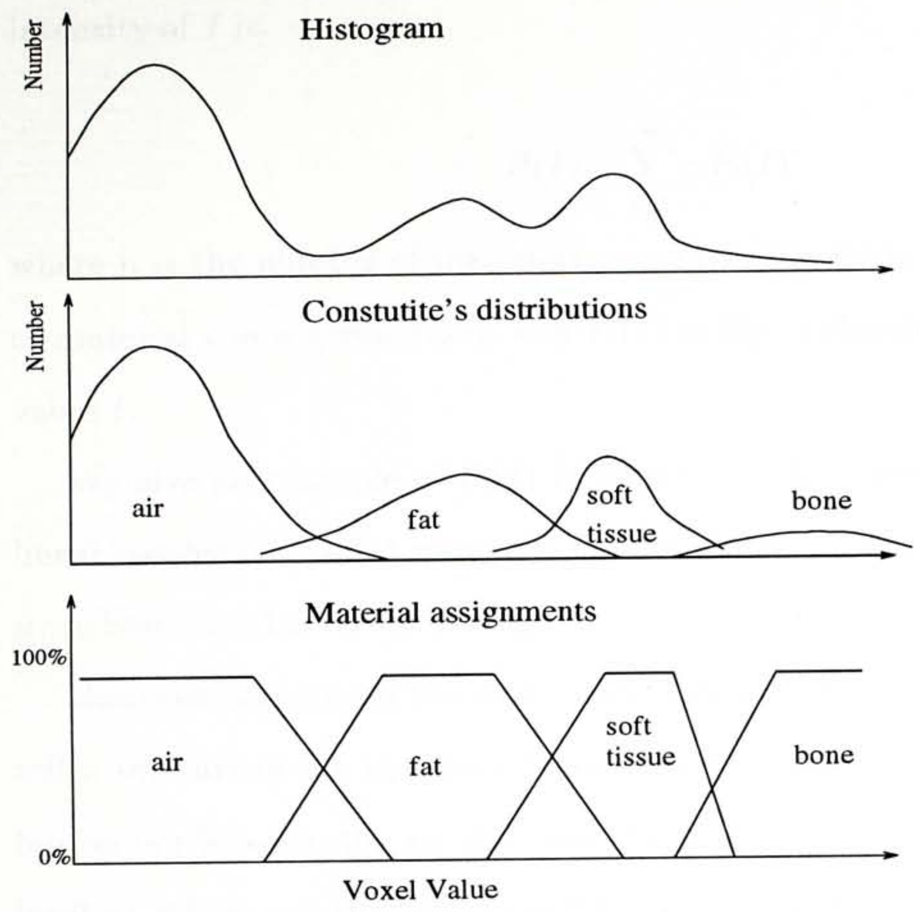

Figure 4.2: The Classification and Opacity assignment

4.4 Classification

To determine the opacities, a maximum-likelihood classifier for the classification of CT data is used. The probability that a material is presented can be used as an estimation of the percentage of the material presented in the voxel[4].

For example, in musculoskeletal CT systems, the value at each voxel represents the x-ray radiation absorption of air, bone, soft-tissue, fat or mixtures of them. The histogram of the x-ray absorption of the input volume is the sum of four overlapping distributions, corresponding, in increasing order of intensity, to air, fat, soft-tissue and bone. (Figure 4.2)

According to maximum-likelihood, the possibility $P(I)$ that a voxel has an intensity of I is

$$P(I) = \sum_{i=1}^{n} p_i P_i(I) \tag{4.4}$$

where n is the number of materials present in the volume, p_i is the percentage of material i in a given voxel, and $P_i(I)$ is the probability that material i has value I.

We give an example of $P_i(I)$ in figure 4.2 which assuming every voxel is a linear combination of at most two kind of materials. It is roughly the case here since bone and fat rarely overlap.

However, sometimes the maximum likelihood does not work well. It cannot tell a mixture of air and bone from an soft tissue since the soft tissue distribution lies between the air and bone distributions. Using the knowledge of the local neighborhood characteristics[24] can improve the performance. But in this thesis, we are only using this method to classify CT data.

Chapter 5

Fast data parallel rotation and resampling algorithms

5.1 Introduction

When colors and opacities of voxels are assigned after shading and classification, they have to be resampled according to the desired perspective. Resampling corresponds to an affine spatial transformation between the viewing space and the object space. For interactivity, the algorithms must efficiently exploit the parallelism within the transformation.

The concerned data are discrete so that the resampling algorithms shall map a sample p in the object space X to the viewing space Y sample p' through a mapping μ, shown as $p = \mu(p')$. Should the object space and the viewing space mesh exactly, the resampling becomes trivial. However, when the viewing space is obtained by spatially transforming the object space, the corresponding samples don't have such matches. Therefore, the algorithms hereby correspond not only to a mapping between the two spaces, but also to a resampling algorithm.

When a transformation is implemented in a data parallel manner, either the data are physically moved within the processor array, or the data are leaved in place with their coordinates manipulated. Leaving the data in place is preferred here since a transformation may incur irregular data movements[1] in the SIMD architecture. However, the result is that the data coordinates will no longer be constant. To have the final rendered scene laid in the processor array conform to either one of the virtualization model described in chapter 3 for display, a process to shuffle the data is used after the transformation and resampling. We will discuss this process in later chapter.

In this chapter, we will discuss the data parallel rotation and resampling algorithms. In particular, a new mapping algorithm is presented with optimal mapping properties and flexible cost and quality tradeoff for different requirements of speed and quality.

5.2 Affine Transformation

An affine transformation[6] is defined as a mapping μ between the viewing space Y and the object space X by:

$$Y = AX + V \tag{5.1}$$

where A is a 3×3 matrix and V is a vector in 3D space.

Some examples of affine maps according to this form are shown as follow:

[1]Examples of irregular data movement will be given in the next section.

Identity: given by $V = 0$, the zero vector, and by $A = I$, the identity matrix.

Translation: given by $A = I$, and a translation vector V.

Scaling: given by $V = 0$ and by a diagonal matrix A. The diagonal entries are defined by how much each component of X is to be scaled.

Rotation given by equaling A to the rotation matrix T,

$$T := \begin{pmatrix} 0 & 0 & 1 \\ -\sin\varphi & \cos\varphi & 0 \\ \cos\varphi & \sin\varphi & 0 \end{pmatrix} \times \begin{pmatrix} \sin\phi & 0 & \cos\phi \\ 0 & 1 & 0 \\ \cos\phi & 0 & -\sin\phi \end{pmatrix} \times \begin{pmatrix} \cos\theta & \sin\theta x & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{5.2}$$

and $V = 0$

Shear: given by $V = 0$ and

$$A := \begin{pmatrix} 1 & a & b \\ 0 & 1 & c \\ 0 & 0 & 1 \end{pmatrix} \tag{5.3}$$

Affine maps may be combined, and a complicated map may be decomposed into a sequence of simpler maps. It has been shown that every affine map is composed of translations, rotations, shears and scalings[6].

Transformations, such as translations, scalings, where the transformed lines and planes remain parallel to the original one, move the data in parallel. Rotation and shear, however, don't have the parallelism which means the physical movement of data in these transformations is very expensive in the SIMD model. Hence, we choose to leave the data in place. In the next section, we give data parallel rotation as an example of implementing affine transformations on the SIMD architecture.

5.3 Related works

5.3.1 Resampling in ray tracing

The direction of ray composition determines the direction of resampling. Backward methods, gathers the required information by shooting the ray from the screen pixel into the object space[17]. The resampling algorithm corresponding maps the 2D screen of the viewing space to the object space, requires the data be accessed by the order of the shooting ray. However, the data layout in the processors according to the object space may not align with the viewing space. Either a transformation is required to align the data in the processors[4] or the processors should wait for the shooting ray to arrive. The former leads to vast volume of the costly irregular data movement between the processors; the latter sacrifices parallelism, when in worst case, only N out of $N \times M$ processors in the processor array are working while other waiting for the shooting ray to arrive.

5.3.2 Direct Rotation

Forward method calculates the object space samples' contribution to the final image. Therefore, the resulting resampling algorithm should map the object space sample to the viewing space. Such mappings have the advantage of possible to be performed simultaneously and realized in the SIMD platform efficiently.

The most straightforward way for mapping the object space sample to the viewing space would be rotating a 3D coordinate at each processor, giving an address of viewing space which the given sample falls. However, the given address of the samples cannot match directly to the viewing space samples and

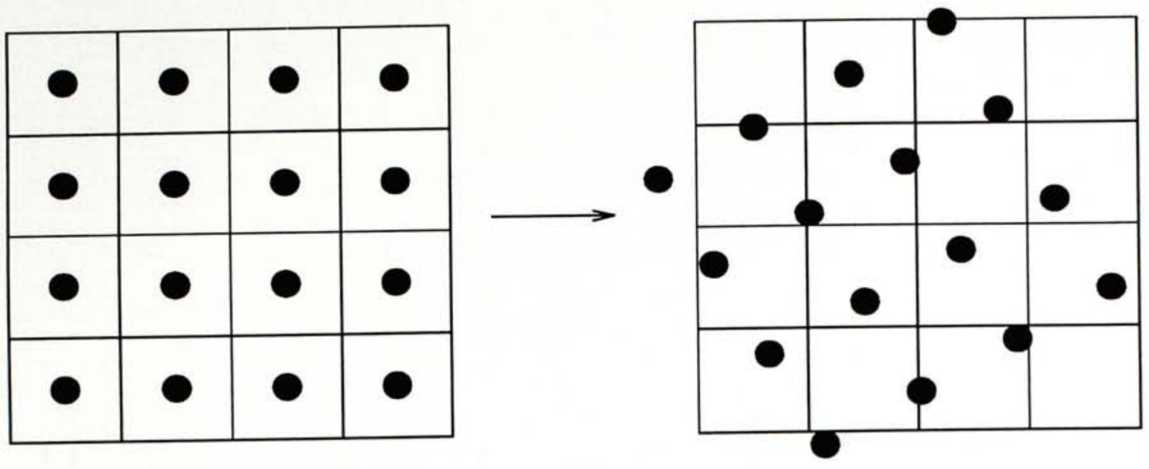
Figure 5.1: Examples of holes and doubles when directly rotating coordinates

a direct rounding will leave holes and doubles in the screen (See figure 5.1). Instead, an algorithm that transforms the integer lattice points in the original volume to the viewing space is needed.

5.3.3 General resampling approaches

Rotation transformation in the three dimensional space is defined by equation 5.2. Either T or the inverse transform T^{-1} may be used for the mapping. Westover[26] used direct convolution for the transformation T, Wolberg and Boult[28] used lookup table for mapping and reconstruction. Their methods integrate the process of reconstruction and mapping and provide general solutions to resampling. Feibush et al.[7] adopt the inverse transformation T^{-1} to map polygons between two different spaces.

5.3.4 Rotation by shear

Decomposing the rotation into shears provides the multipass solution to avoid the holes and doubles appear in direct rotation. The shear may be scale(stretch

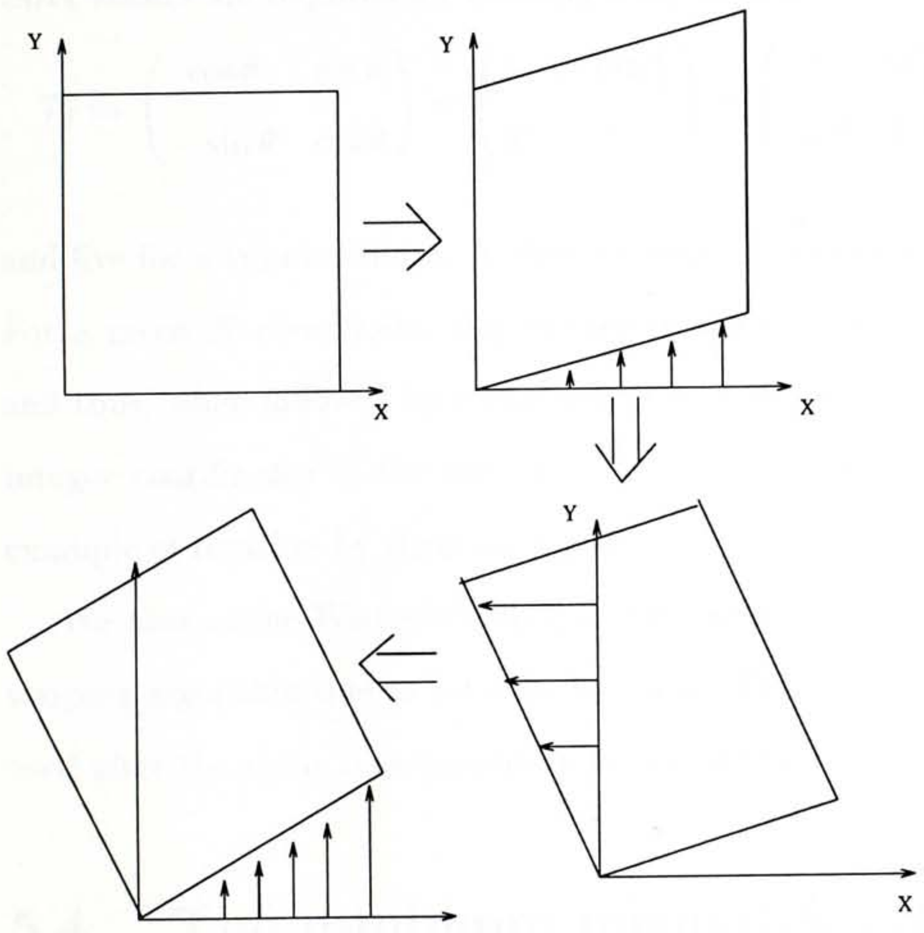

Figure 5.2: Rotation by shear on a plane

or shrink axes)[10] or not scale(distances preserved)[23]. Nonscaling shears have 1's along their diagonals as shown in equation 5.4.

Schröder and Salem[23] apply a scheme due to Paeth[22] to regularize the rotational transformation by decomposing it into multiple orthogonal shear transformations. Each shear is followed by rounding and interpolation. In general, three shears are required for rotating a 2D image:

$$T_\theta := \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} = \begin{pmatrix} 1 & -\tan\frac{\theta}{2} \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 \\ \sin\theta & 1 \end{pmatrix} \times \begin{pmatrix} 1 & -\tan\frac{\theta}{2} \\ 0 & 1 \end{pmatrix} \tag{5.4}$$

and five for a volume image. A shear operates on only one coordinate at a time. For a given X coordinate, the moving distance along the Y axis is constant, and thus, when followed by a rounding, the consequential mapping between the integer coordinates of the two space is indeed bijective. Figure 5.2 gives an example of rotation by shear on a plane.

We also notice Wittenbrink[27] at the same time of this thesis developed a warping algorithm due to rotation by shear. The inverse transformation T^{-1} is used after the shear transformation to provide more accurate results.

5.4 The minimum mismatch rotation

Supposed the volume data are distributed in the processor mesh according to the object space layout. Subsequently, each processor may be addressed by the groups of samples $p_G \in G$ that it maps to. Let H be the samples of viewing space Y obtained by rotating X by θ. To get H, we need a mapping $\mu : H \to G$ and another sample to processor mapping $\xi : G \to P$. Namely, processor $\xi(\mu(p_H))$ will be assigned the responsibility to resample the image at $p_H \in H$. Ideally,

(a) the size of $\mu(p_H)$ should be equal on each processor for high data-parallelism and (b) grid-point-to-processor mismatch should be small, namely, p_G should be close to the inverse mapping of p_H on X for proximity to processors holding neighboring pixels of p_H required for resampling by interpolation.

By minimizing the grid-point-to-processor mismatch, we derive a data-parallel rotation algorithm as follow: For $p_H \in H$, the mismatch-minimizing mapping is given simply by

$$\mu^*(p_H) = Floor(T_\theta \times p_H) \qquad p_H \in H \tag{5.5}$$

where T_θ is the inverse rotation matrix given by

$$T_\theta := \begin{pmatrix} 0 & 0 & 1 \\ -\sin\varphi & \cos\varphi & 0 \\ \cos\varphi & \sin\varphi & 0 \end{pmatrix} \times \begin{pmatrix} \sin\phi & 0 & \cos\phi \\ 0 & 1 & 0 \\ \cos\phi & 0 & -\sin\phi \end{pmatrix} \times \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{5.6}$$

That is, according to μ^*, resampling at the rotated grid point $p_H \in H$, or $T_\theta \times p_H \in G$ equivalently, is to be carried out by processor $\xi(\mu^*(p_H))$, which presumably does so by interpolating pixels neighboring $T_\theta \times p_H$ in G. Define the *grid neighborhood of point* $x \in X$, by

$$N_x(x,k) := \{p_G : \|x - p_G\|_\infty < 1, \quad p_G \in G\}, \tag{5.7}$$

and the *neighborhood of processor* $p \in p_P$, by

$$N_p(p,k) := \{p_P : \|p - p_P\|_\infty \leq 1, \quad p_P \in P\}. \tag{5.8}$$

The neighborhood of the rotated grid-point $T_\theta \times p_H$ is contained in the resampling processor $\xi(\mu^*(p_H))$, viz.

$$N_x(T_\theta \times p_H) \in N_p(\xi(\mu^*(p_H))) \tag{5.9}$$

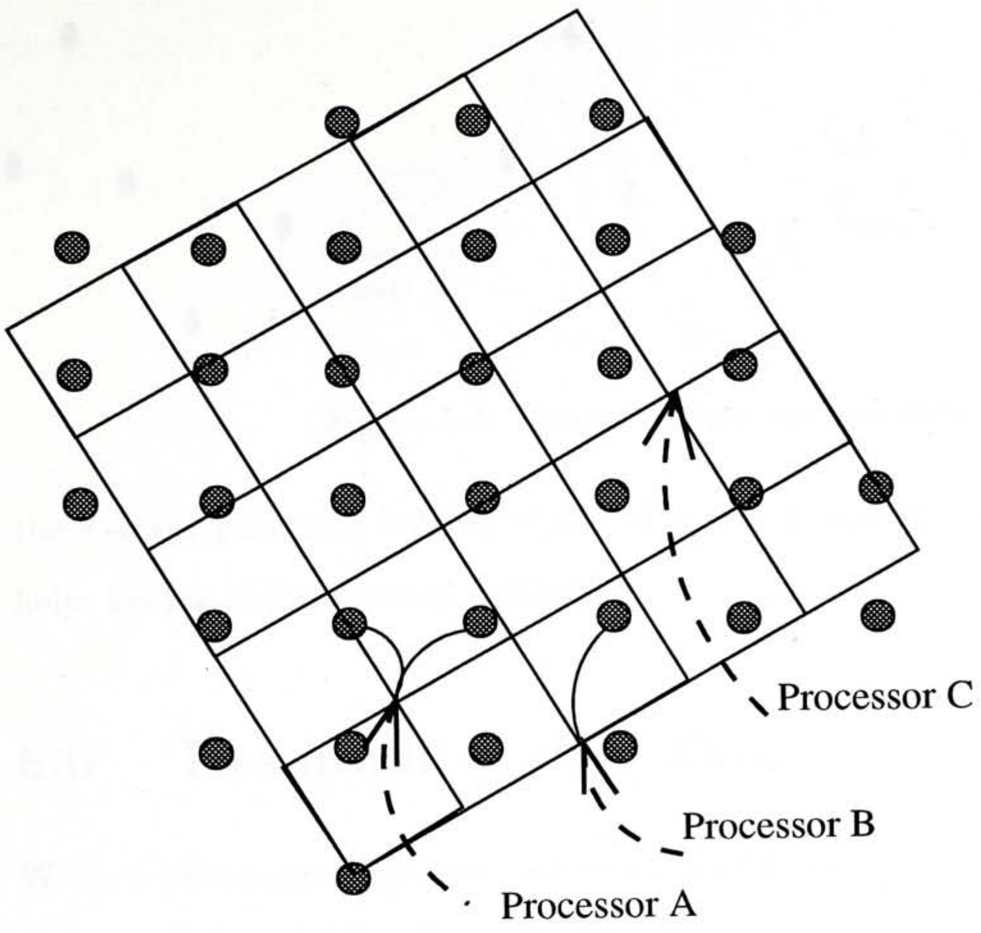

Figure 5.3: Assigning data points to processors

which ensures processor $\xi(\mu^*(p_H))$ has immediate access to neighboring pixels for resampling by interpolation.

Though this method minimizes the mismatch distance, it is not an one-to-one matching. For instance, in figure 5.3, two data points are assigned to Processor A while none is assigned to C. When loading is not equal in each processor, it will be hard to get high processor utilization.

5.5 Load balancing

As the ultimate concern is minimizing mismatch, load balancing will be done for high processor utilization while maintaining the minimum mismatch property.

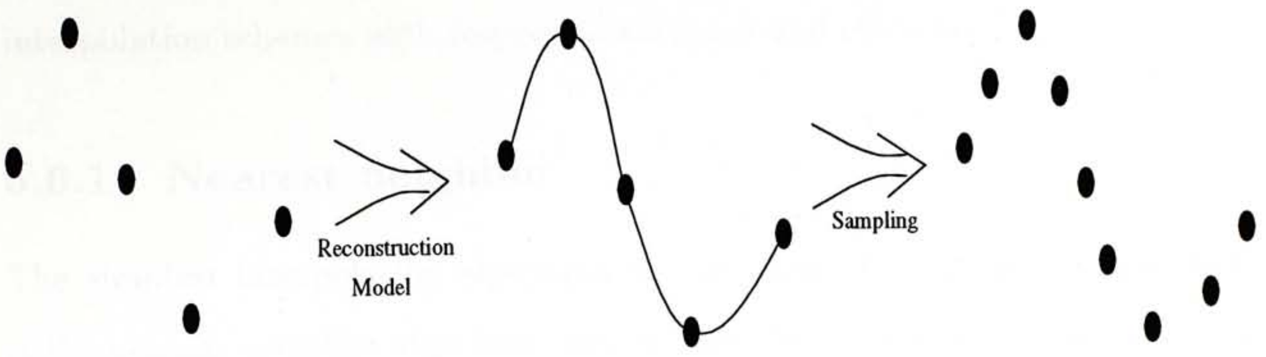

Figure 5.4: Reconstruction from samples

the average processor holding of $n \times m \times L$ if N and M is large enough. This helps keep a high processor utilization.

5.6 Resampling algorithm

When a processor is assigned the responsibility of the viewing space samples, a resampling algorithm is used. This resampling algorithm is identical to a interpolation problem. Generally, the resampling quality is better if more input samples are used. However, computation complexity will increase as the result.

The interpolation process is depicted in figure 5.4 for one dimensional case. There are two steps in the process, namely, the reconstruction and the sampling process. The reconstruction is performed by convolving the discrete input signal with a continuous interpolating function to model a continuous function; the consequent sampling process samples the continuous function in the desired position. In this section, the analysis is applied to one dimensional cases and the interpolation in three dimension is a simple extension to the results of one dimension.

Polynomial interpolation is the most fundamental of all interpolation concepts and the simplest to be implemented. We will review different polynomial interpolation schemes with respect to accuracy and efficiency.

5.6.1 Nearest neighbor

The simplest interpolation algorithm in the sense of computation complexity is the nearest neighbor algorithm, where each desired interpolation point is assigned the value of its nearest neighbor. The reconstructed continuous function is described by:

$$f(x) = f(x_k) \qquad \frac{x_{k-1} + x_k}{2} < x \leq \frac{x_k + x_{k+1}}{2} \tag{5.10}$$

This may be achieved by convolving the discrete signal with a one pixel width rectangle in the spatial domain. The convolution kernel in frequency domain is the *sinc* function(figure 5.5). The convolution of the kernel in spatial domain is equivalent to multiplication of the *sinc* function in the frequency domain.

According to the sampling theory, the ideal reconstruction of the band limited signal is the idea low pass filter. However, due to the prominent side lobes and infinite extent, a *sinc* function makes a poor low pass filter. Therefore, the alais effect introduced by the nearest neighbor may be large.

5.6.2 Linear Interpolation

Linear interpolation passes a straight line through every two consecutive points of the input signal. Given two discrete point x_1 and x_2 in an interval (x_1,x_2) and the discrete inputs f_1 and f_2 at x_1 and x_2, the reconstructed continuous function is defined by:

$$f(x) = f_0(x) + \frac{x - x_0}{x_1 - x_0}(f_1 - f_0) \tag{5.11}$$

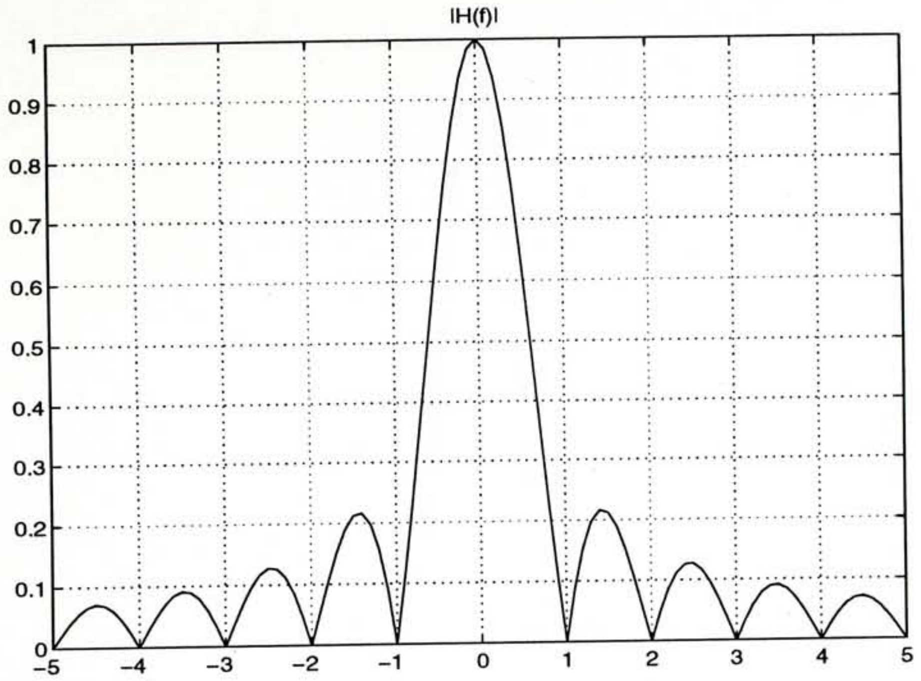

Figure 5.5: The magnitude of *Sinc* function

The interpolation kernel corresponds to linear interpolation is

$$h(x) = \begin{cases} 1 - |x| & -1 < x < 1 \\ 0 & otherwise \end{cases} \tag{5.12}$$

The frequency domain of this kernel is in the form of $sinc^2$(See figure 5.6). The energy is more concentrate on low frequency area and the side lobes are far less prominent which indicates improved performance in the stop band. It is a much better low pass filter in the frequency domain than the one used in the nearest neighbor method.

The linear interpolation method is widely used for the reasonable result it produces at moderate cost. However, when high fidelity is required, more sophisticated algorithms have been formulated.

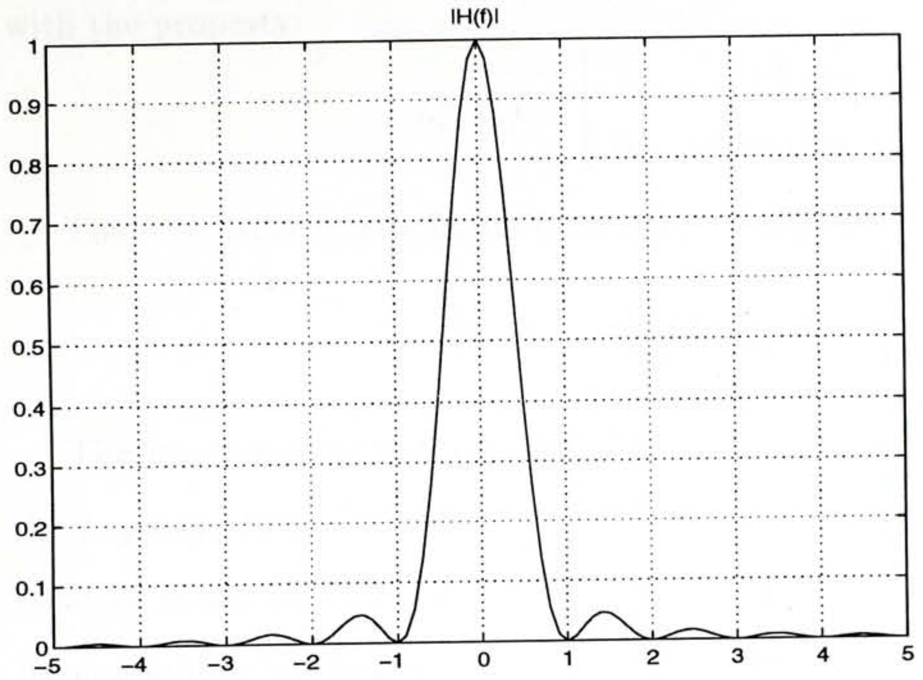

Figure 5.6: The magnitude of Sinc function

5.6.3 Aitken's Algorithm

In the nearest neighbor and the linear interpolation method, the number of discrete inputs is one and two respectively. When more inputs are available, more accurate results are possible. For our resampling algorithm, not only the first order neighbor but also the higher order neighbors may be used.

We extend the interpolation method to the case of n inputs. Suppose the discrete inputs at $x_0, x_1 \ldots x_n$ are with value $f_0, f_1 \ldots f_n$ respectively. The reconstructed continuous function shall have the same value as the discrete inputs at $x_0, x_1 \ldots x_n$.

Define the *Lagrange Polynomials* as:

$$L_i^n(x) = \frac{\prod_{j=0,j\neq i}^{n}(x - x_j)}{\prod_{j=0,j\neq i}^{n}(x_i - x_j)} \tag{5.13}$$

with the property:

$$L_i^n(x_k) = \begin{cases} 1 & k = i \\ 0 & otherwise \end{cases} \tag{5.14}$$

The $n'th$ order interpolation function $f(x)$ is given by

$$f_{0,1\ldots n}(x) = \sum_{i=0}^{n}(f_i \times L_i^n(x)) \tag{5.15}$$

The interpolation method derived from this reconstruction function is call the *Lagrange Polynomials.* In this method, if the accuracy of a low order interpolation is not enough and requires higher order interpolation, the continuous reconstruction function has to be built from the beginning.

Aitken's algorithms overcome this by observing the following:

$$\begin{aligned} f_{0,1\ldots n}(x) &= \sum_{i=0}^{n} f_i \times \frac{\prod_{j=0,j\neq i}^{n}(x-x_j)}{\prod_{j=0,j\neq i}^{n}(x_i-x_j)} \\ &= \frac{x-x_n}{x_0-x_n}\sum_{i=0}^{n-1} f_i \frac{\prod_{j=0,j\neq i}^{n-1}(x-x_j)}{\prod_{j=0,j\neq i}^{n-1}(x_i-x_j)} + \frac{x-x_0}{x_n-x_0}\sum_{i=1}^{n} f_i \frac{\prod_{j=1,j\neq i}^{n-1}(x-x_j)}{\prod_{j=1,j\neq i}^{n}(x_i-x_j)} \\ &= f_{0,1\ldots n-1} + \frac{x-x_0}{x_n-x_0}(f_{0,1\ldots n-1} - f_{1,2\ldots n}) \end{aligned} \tag{5.16}$$

In *Aitken's algorithms*, the approximation is made in a progressive manner. The zero order interpolation functions $f_{x_0}^0, f_{x_1}^0 \ldots, f_{x_n}^0$ (The nearest neighbor) are calculated first, then the first order interpolation function(The linear interpolation) is calculated by applying equation 5.17 to the zero order functions. Recursively, higher order interpolations are achieved as follow:

$$\begin{array}{lllll} f^0(x_0) & \searrow & & & \\ f^0(x_1) & f_{x_0,x_1}^1 & \searrow & & \\ \vdots & \vdots & \vdots & & \\ f^0(x_n) & f_{x_{n-1},x_n}^1 & f_{x_{n-2},x_{n-1},x_n}^2 & \cdots & f_{x_0,x_1\ldots x_n}^n \end{array}$$

The higher order *Lagrange Polynomials*, whose corresponding low pass filter in the frequency domain is closer to the ideal low pass filter, have better performance at the cost of more computation. We will discuss this trade-off later.

5.6.4 Polynomial resampling in 3D

Resampling in 3D is an extension to the case of one dimension by applying equation 5.17 in each each dimension respectively. For the example of linear interpolation in three dimension, we first interpolate on one dimension to give four interpolated point out of the eight neighbors of the desired sample point, then, two interpolated points out of the four in the next dimension and finally the result is given.

The interpolated results of the nearest neighbor method, the linear interpolation method and the second order interpolation method in two dimensional image are given in figure 5.7, 5.8 and 5.9.

5.7 A comparison between the resampling algorithms

This section presents a comparison between our minimum mismatch algorithm and the multi-pass methods. Quality and cost are used as criterions. In addition, the progressiveness of the algorithm, which is the ability of progressively getting high quality results from the low quality results at no additional cost, is very important for a fair trade-off between the speed and quality.

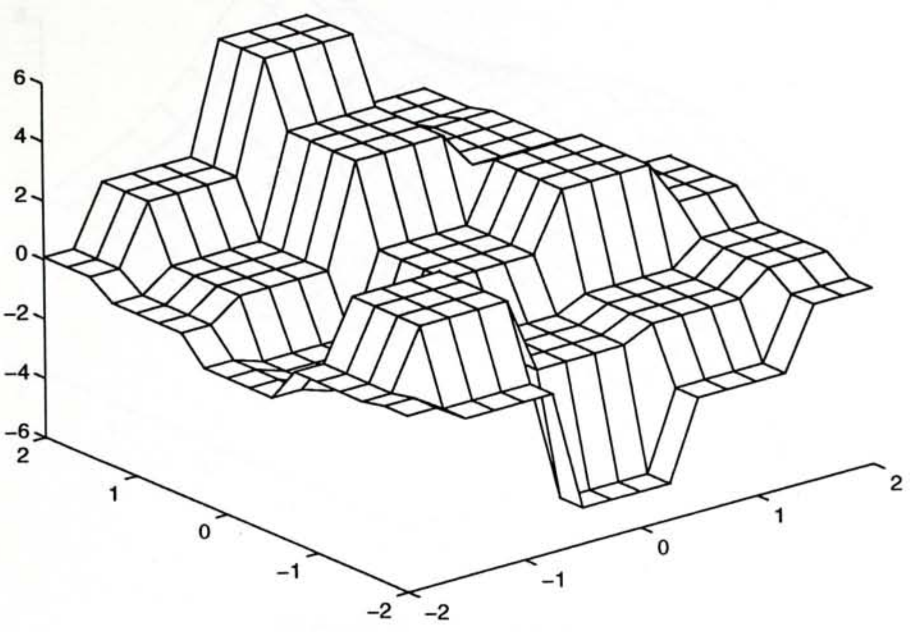

Figure 5.7: Nearest neighbor interpolation

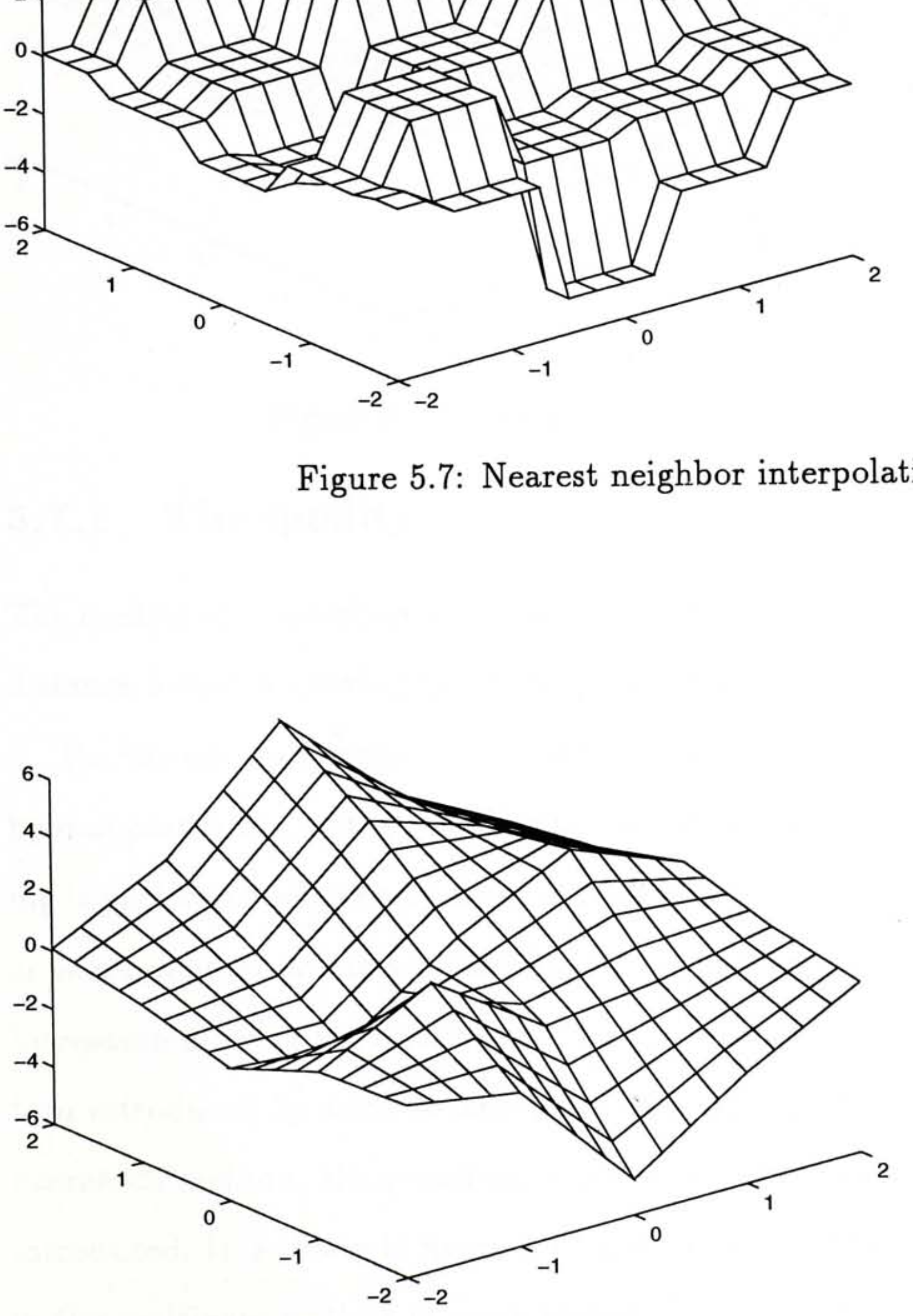

Figure 5.8: Linear interpolation

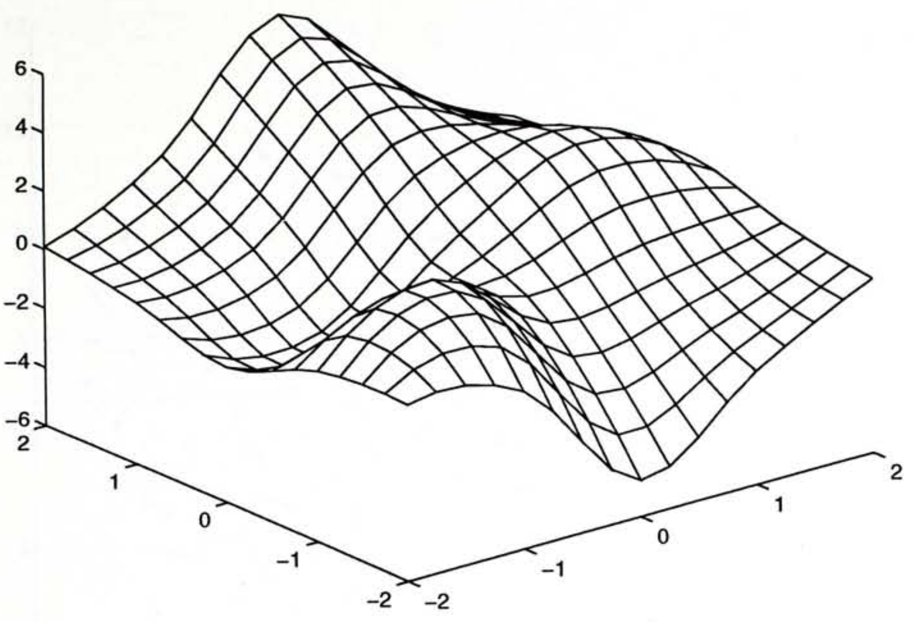

Figure 5.9: Second order(Cubic) interpolation

5.7.1 The quality

The quality of resampling is evaluated by two factors: the filter quality and the distance between viewing space samples and the object space samples.

For the example of zero-order hold reconstruction filter, the distance incurred by mapping is due to the rounding operation. In the multipass mapping, rounding is performed after every shear operation and the resulting image is degraded as non-proximal points are used for reconstruction. The mis-match introduced by resampling may accumulate and there maybe artifacts due to the regular pattern introduced by such rounding and resampling. However, in the minimum mismatch method, the rounding is performed only once and less error will be introduced. It is shown in figure 5.10 and figure 5.11 that the mismatch distance in the multipass method is much higher.

Wittenbrink[27] reduced the mismatch distance by introducing a inverse

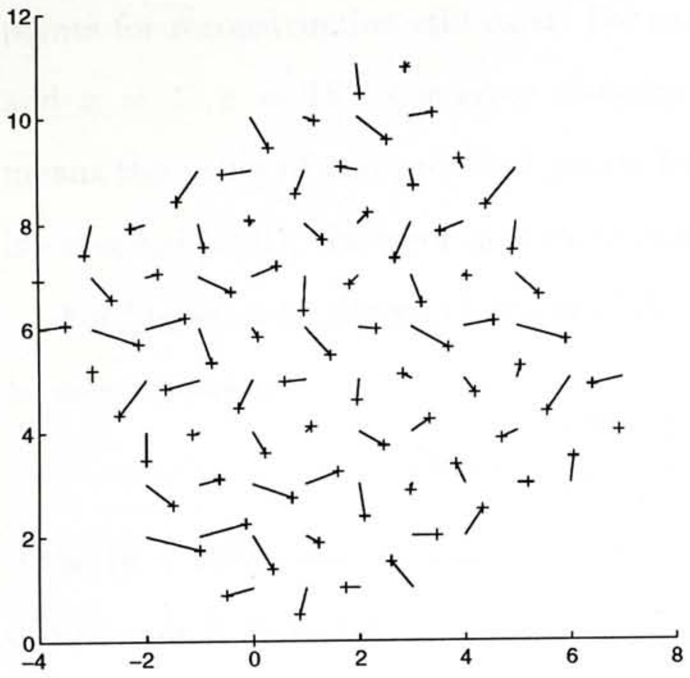

Figure 5.10: The mismatch in the multi-pass method

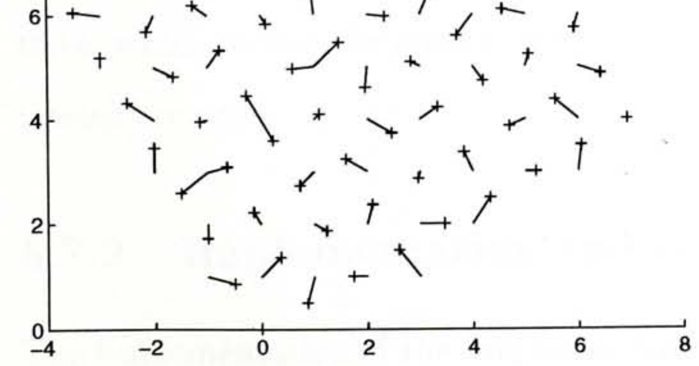

Figure 5.11: The mismatch in the minimum mismatch method

transformation after the rounding. However, the problem of using non-proximal points for reconstruction still exist. For example in equation 5.4, when $\theta = 30^\circ$ and $x = 47, y = 181$, the error distance in coordinate x will be 1.04 which means the using of non-proximal points for reconstruction. When the rotation is extended to 3D, the error may more easily accumulate.

For higher order filters, the error of the *n'th order Lagrange Polynomials* can be estimated by:

$$E_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{j=0}^{n} (x - x_j) \tag{5.17}$$

if the $(n+1)'th$ order derivative $f^{(n+1)}(x)$ of $f(x)$ exists, and the upper bound of the error is given by :

$$E_n(x) \leq \frac{Max(f^{(n+1)}(\xi))}{(n+1)!} \prod_{j=0}^{n} (x - x_j) \tag{5.18}$$

When higher order filters are used, the error is not only related to the maximum value of $(n+1)'th$ order derivatives, but also related to the value of the function $\prod_{j=0}^{n}(x - x_j)$. The mis-matching distance of two samples is proportional to the $\prod_{j=0}^{n}(x - x_j)$ and can also be used as an estimation for the error introduced.

For a band limit signal, there will be a upper bound in the $n'th$ order derivatives, which means the error will decrease at a factor of $\frac{1}{n}$ as $n'th$ order interpolations are used.

5.7.2 Implementation and cost

The implementation of the minimum mismatch algorithm comprises four steps, namely the forward transformation to estimate the volume where the processor shall hold the responsibility for resampling, the inverse transformation to find the matching point, the load balancing which only involves local communications and the interpolation.

The cost when implemented in a parallel machine comprises the cost of computation and communication within the processors. The computation complexity is denoted by the cost of multiplication(M), addition(A) and rounding(R).

Our method does not need the shear operation. Instead the cost depends heavily on the cost of reconstruction scheme. The mapping algorithm only requires a forward rotation and an inverse rotation, each is composed of 9 multiplications and 6 additions. If a processor hold a cube of $N \times M \times L$, only 8 forward transformations of the vertices is needed to estimate the volume in the viewing space. The volume of the estimated viewing space may vary according to the viewing perspective and is proportional to the average number of data each processor holds. Generally, the number of inverse rotation is $9M + 3A + R$ per sample. If the nearest neighbor interpolation scheme is used, the cost may be estimated as $9M + 3A + R$. If the linear interpolation is used, the cost of the linear interpolation in 3D is $14M + 7A$ and the total cost become $23M + 9R + 10A$

In Schröder's method, every shear operation is followed by a rounding and a resampling process. The shear operation requires an addition and a multiplication, or one addition and two multiplication in 3D if the shear operation involves the other two coordinate. The resampling process requires 2 multiplication and an addition and 6 multiplication and 3 addition when the following resampling is according to two coordinate. When 4 shears in one coordinate and 1 shear in two coordinate is used, the computation cost in their method is $20M + 9R + 11A$.

The communication complexity is estimated by the local communication used. In Schröder's method, one of the time consuming process is to keep track of neighbors after each shear operation. Also, the local communication is incurred every time the resampling after the shear operation. The minimum mismatch only requires the local communication in the final interpolation scheme as well as the load balancing process. If the nearest neighbor scheme is chosen, there is no need for communication.

Chapter 6

Data reordering using binary swap

The result of the resampling process is that data in the processors are assigned viewing space coordinates and values; the actual volume of each processor represented is unchanged. The remaining job for rendering is to have the data along the rays composited and the result displayed. To efficiently composite the rays, it is desirable to have all the data in one shooting ray laid in one processor; to efficiently display the result, the final result should be laid in the processor array according to a certain virtualization model.

The visualization pipeline of figure 1.2 meets the above requirement by composition in three steps: the local composition, the reordering process and the global composition. The local composition forms a polygon in each processor by compositing the data within each processor; the reordering process re-arranges the data in the processor array to the viewing space layout where data in the same ray are grouped in one processor; and the global composition composites the data into the rendered result. The global composition and re-ordering may be combined as global composition may be done while reordering.

We will discuss the reordering in this chapter and the composition in chapter 7. Suppose the local composition is complete, the footprints are held in the processors as polygons. The polygons are laid in the processor array according to object space layout and may overlap if projected on the screen in the viewing space. The reordering algorithm groups the overlapping part of the polygon in one processor for efficient composition and lays the final image according to the Cut and stack virtualization model with minimum communication overhead. As the local x-net communication is much cheaper than the global communication, we choose it as the communication mechanism of the reordering algorithms.

6.1 The sorting algorithm

Since original data layout and the desirable data layout of the final image in the processor is according to object space and viewing space respectively, the reordering could also be described as a sorting process while the overlapping parts are composited.(Figure 6.1)

Assuming P and D are the binary representations of the processor coordinate and data coordinate respectively:

$$P = P_{N-1}P_{N-2}\cdots P_0 \tag{6.1}$$

$$D = D_{n-1}D_{n-2}\cdots D_0 \tag{6.2}$$

when there are 2^N processors in $x(y)$ direction and 2^n pixels in $x(y)$ direction and $h : D \rightarrow P$ is the mapping of voxels with data coordinate D to processor

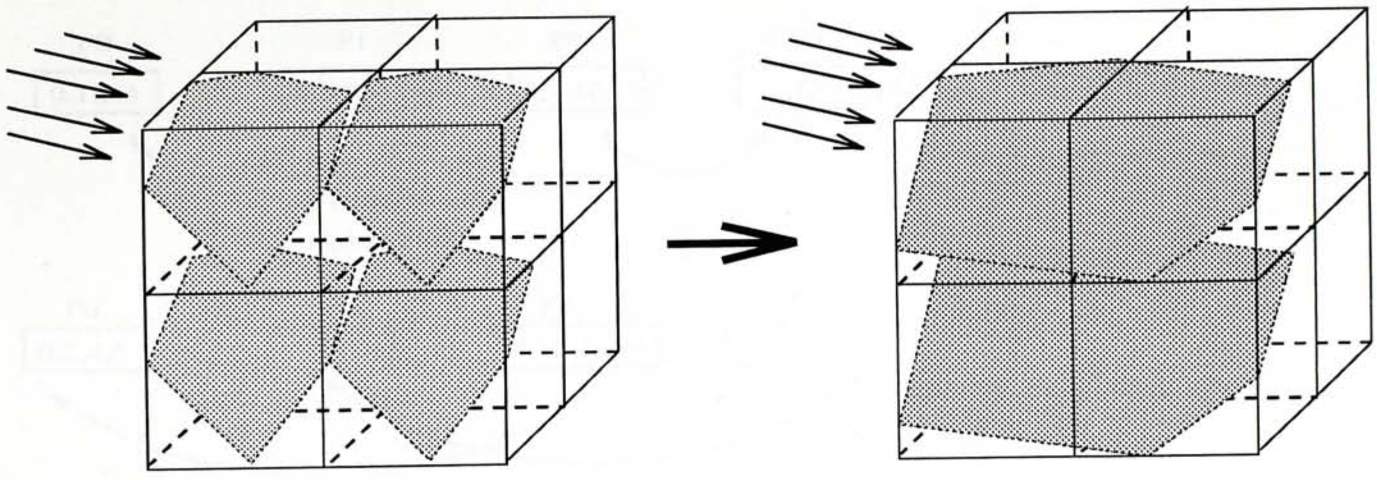

Figure 6.1: The process of volume merging

grid P:

$$h = D_{\sigma_{N-1}} D_{\sigma_{N-2}} \cdots D_{\sigma_0} \tag{6.3}$$

In the cases of Cut and stack model, we have

$$h : \sigma_k = k (k = 0 \ldots N-1) \tag{6.4}$$

From the sorting point of view, to reorder the locally rendered volume means to find an algorithm that changes the h of original data layout to the h of the Cut and stack model.

We design our algorithm as follow: Suppose the resulting size of image will be 2^n in $x(y)$ direction and the size of processor mesh is 2^N in $x(y)$ direction. We shall do the reordering both in X and Y.

- Let the reordering distance be 1. Define the *reordering bit* to be the non-zero bit of the binary representation of the reordering distance.
- Data on the processor shall exchange in a way that those data with $x(y)$ coordinate i and processor coordinate I should have their *reordering bit* in common to achieve the mapping of the Cut and stack model. For example,

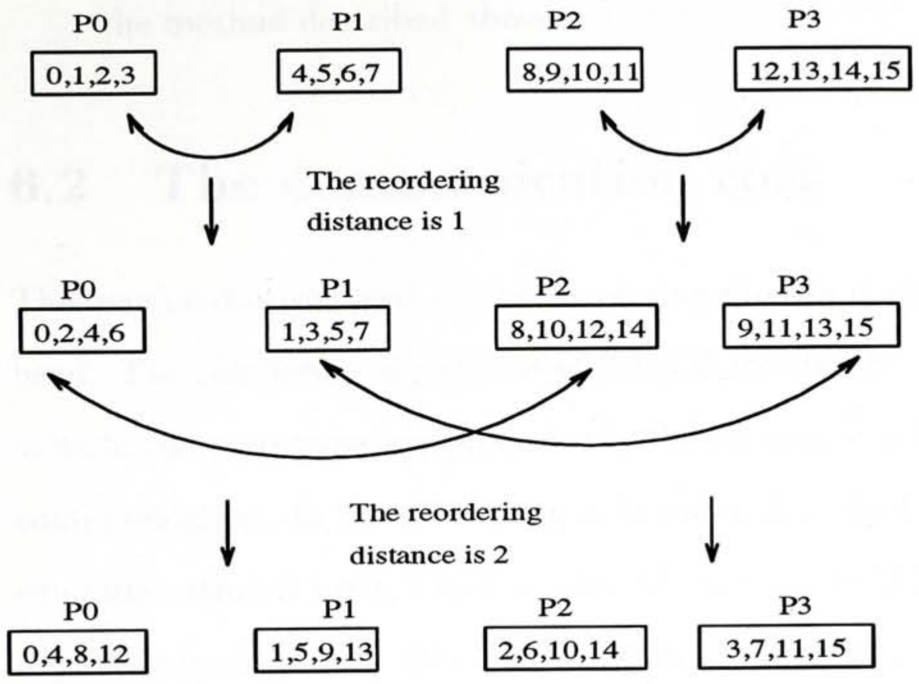

Figure 6.2: Example of binary swap with 16 data samples in 4 processors

if the reordering distance is 1, the *reordering bit* is the last bit, the data exchange should be done that the data with even coordinate are on even $X(Y)$ coordinate processor, data with odd $x(y)$ coordinate are on odd coordinate processor.

The processor pairs in distance of the reordering distance exchange their data: those data have the same *reordering bit* as the processor coordinates remain and others swap out. Figure 6.2 give an example of binary swap of 16 data laid in 4 processors.

- For each processor, those data with the same coordinate, which is the overlapping part of the two polygon should be composited using method described in chapter 7.
- If the distance of reordering is less then 2^{N-1}, then double the distance of reordering and exchange the data according to the reordering bit using

the method described above.

6.2 The communication cost

The overhead introduced by the reordering process is the communication overhead. The reordering algorithms of [23][13] are simply by sending the local results to the corresponding processor in the viewing space, which involves global communication. In the reordering algorithms described aboved, only the x-net communication is used, which is very efficient in the SIMD parallel machine as described in chapter 3. Below is an estimation of the communication overhead.

Suppose data coordinate is $x(y)$ and the processor coordinate is $X(Y)$. From the properties of swapping, after M times of reordering, the M'th least significant bits of x is equal to M'th least significant bits of X:

$$x \ mod \ 2^M = X \ \ mod \ 2^M \tag{6.5}$$

This means after the M'th reordering in X and N'th reordering in Y , the Cut and stack model exists in the sub-processor mesh $2^M \times 2^N$, data are equally distributed in the sub-processor mesh.

We argue that in every time of the swap, about half of the data in a processor are swapped out. Noting that in the beginning the data in each processor represent a polygon, which means the both the x and y coordinates on the processor are continuous. After M'th swap in X and N'th swap in Y, a bigger polygon is virtualized in the $2^M \times 2^N$ sub-processor mesh, which means that continuity exists in $[\frac{x}{2^M}]$ and $[\frac{y}{2^N}]$[1]. With this continuity, half of $[\frac{x}{2^M}]([\frac{y}{2^N}])$ is

[1] $[x]$ represent the largest integer number less than x

odd; the data coordinate's *reordering bit* is half one and half zero.

To estimate the amount of the data in the processor, we first estimate the area of the biggest cross section polygon of a cube. It is easy to prove that in a cube of volume $a \times b \times c$, the biggest cross section polygon's area is less than $\frac{1}{2}(a^2 + b^2 + c^2)$, which may be used to estimated as the maximum number of data a processor could ever have. After several swaps, the overlapping part of the polygons are composited. Since the data are evenly distributed in the processor mesh, the amount of the data in each processor are decreasing. Since there will be M and N swaps in X and Y respectively, the total volume of the x-net communication is no greater than $M \times N \times \frac{1}{2}(a^2 + b^2 + c^2)$.

Another advantages of the re-ordering algorithm is its balance of loading among the processors. Since all locally rendered polygons are identical, both in shape and orientation with the only difference is in their locations in the viewing space, we expect the workload on each processor is about the same.

Chapter 7

Ray composition

7.1 Introduction

The final step of volume rendering is the ray composition. By composition, the image is reduced from 3D to 2D according to a lighting model. In the mean time, fast speed of composition is desirable.

Different operations according to different lighting models have been proposed. They are classified as commutative operations and non-commutative operations. One example of commutative operations [11][12] is averaging the intensities of voxels along parallel rays from the volume to the image plane. Commutative operations are good for their simplicity and easy for calculation and parallelization. Non-commutative operations, where compositions are performed in the order of the shooting ray, model precisely the light transmission through the volume. However, the computation complexity is increased; the parallelism of the operation is lowered. One example of the non-commutative operation is the Kaijiya[14]'s visualization model with radiation transport equations.

The most commonly used equation for composition [13][23][1] is simply the transparency formula, which is simply a linear interpolation[17]:

$$C_{out} = C_{in}(1-\alpha) + \alpha C \tag{7.1}$$

where C and α are color vector and opacity of the voxel respectively. Using this formula, every voxel has to be counted from the back to front or front to back order. Therefore the computation overhead in this step is very high considering the number of voxels. Although improvements such as adaptive termination[18] and hierarchical splatting[16] have been designed to simplify the calculation, it seems inevitable to heavily use multiplication operations and count in the voxels with little contribution to the final image.

In this chapter, an algorithm is presented for efficient ray composition through a Monte-Carlo method. Voxels are sampled according to the opacity distribution in the ray. It provides a way for progressive composition and allows easy tradeoff between speed and quality. We will first look into the question of ray composition along a ray; then extend it to an associative solution and finally the implementation on an SIMD machine.

7.2 Ray Composition by Monte Carlo Method

The composition of a ray with N voxels using (7.1) is:

$$\overline{C} = \sum_{i=0}^{N-1} C_i \cdot \alpha_i \cdot \prod_{j=0}^{i-1}(1-\alpha_j) + C_N \prod_{j=0}^{N-1}(1-\alpha_j) \tag{7.2}$$

where C_i and α_i are color vectors and opacities of the N voxels from front to back respectively.

The coefficients of C_i in equation (7.2) sum up to 1 and $\overline{C}$ may be considered as the expectation of C_I where I is a random variable with distribution

$$p(i) \triangleq Prob(I = i) = \begin{cases} \alpha_i \cdot \prod_{j=0}^{i-1}(1-\alpha_j) & \text{if } i < N \\ \prod_{j=0}^{N-1}(1-\alpha_j) & \text{if } i = N \end{cases} \tag{7.3}$$

From the *Law of Large Numbers*, we have:

$$\overline{C} = \lim_{N\to\infty} \frac{1}{N} \sum_{m=1}^{N} C_{R_m} \tag{7.4}$$

where R_m are samples of I.

Samples of I may be generated as follow:

Suppose x_k is a series of independent random variable uniformly distributed on [0,1]

```
(1) $k = 0$
(2) if $x_k < \alpha_k$
        *sample* $R = k$
        else if $k < N$
            k=k+1; repeat (2)
        else *sample* $R = N$
```

The estimation is a process of sampling, where voxels with high contribution to the final image are sampled heavily.

7.3 The Associative Color Model

According to equation (7.1), operation $\circ(X, Y, Z, ..)$ for compositing voxel $X, Y, Z, ...$ is defined as follow:

$$\circ(X,Y) \triangleq C_X(1-\alpha_Y) + \alpha_Y C_Y \tag{7.5}$$

$$\circ(X,Y,Z) \triangleq \circ(\circ(X,Y),Z) \triangleq (C_X(1-\alpha_Y)+\alpha_Y)(1-\alpha_Z)+\alpha_Z C_Z \tag{7.6}$$

where operation $\circ$ implies no associativity; composition must be in the order of X, Y, Z.

A more efficient operation is possible by introducing a new attribute of voxel, *associative color* C', defined as

$$C' \triangleq \alpha \times C \tag{7.7}$$

and new operator $\diamond$ of composition on associative color C' and opacity α:

$$\diamond(X,Y) \triangleq [C'_{new}, \alpha_{new}] \tag{7.8}$$

where

$$C_{new} \triangleq (1-\alpha_Y)C'_X + C'_Y \tag{7.9}$$

$$\alpha_{new} \triangleq (1-\alpha_Y)\alpha_X + \alpha_Y \tag{7.10}$$

It is easy to show that operation $\diamond$ is associative, namely,

$$\diamond(X,Y,Z) = \diamond(\diamond(X,Y),Z) = \diamond(X,\diamond(Y,Z)) \tag{7.11}$$

Associativity is helpful for parallelism since different parts of the ray may be processed independently. Blinn[1] shows more advantages of associative color in image interpolation and filtering.

The associative color can be regarded as a regular color vector composited onto black background. We simply add a black voxel as voxel $N+1$ to the result of equation (7.2) to estimate the composited associative color with Monte Carlo method:

$$\overline{C'}(x,y) = \sum_{i=0}^{N} C_i(x,y) \cdot \alpha_i(x,y) \cdot \prod_{j=0}^{i}(1-\alpha_j(x,y)) \tag{7.12}$$

The composited opacity can be obtained by extending equation (7.10) to N voxel:

$$\overline{\alpha} = 1 - \prod_{j=0}^{N}(1-\alpha_j) \tag{7.13}$$

According to equation (7.12), the generation of samples is slightly different: The sample of I may reach the new voxel $N+1$ with probability $\prod_{j=0}^{N}(1-\alpha_j)$ where the color vector is zero. Opacity is estimated by the number of samples not reaching $N+1$. The estimation of color is decreased by a factor of $\prod_{j=0}^{N}(1-\alpha_j)$ comparative to the estimation in section I.

Compared to the conventional method, our method requires multiplication only in the random number generation. Large amount of multiplication between the colors and opacities is substituted by comparison between opacities and random numbers and averaging of the samples. Different rays may share the same random number for sample generation and different components of the color vector may share the same sample since they are independent.

Consider a volume with N^3 voxels. Direct computation requires $N^3 \times 3$ times of multiplications and additions for compositing the color vectors. Adaptive termination method would give a saving of a factor t, which is the ratio of voxels in a ray being traced before terminated by the accumulating opacity. However, it introduces the composition of opacity α as well as comparison of α with the threshold. It requires $N^3 \times 4 \times t$ multiplications,$N^3 \times 4 \times t$ additions as well as

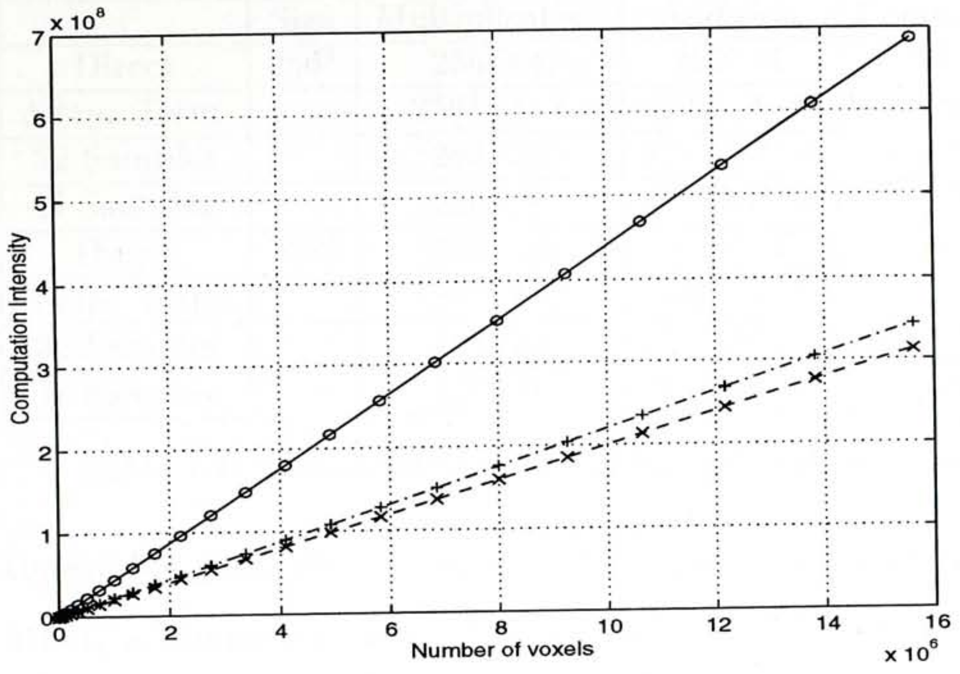

Figure 7.1: Computation vs. number of voxels in different methods[1]

$N^3 \times t$ comparisons. The factor t is also used here as an estimator for the number of comparisons in our method since most of the comparisons will stop before the termination. In Monte Carlo estimation with n samples, we require $N \times n$ times of multiplications and $N^3 \times n$ times of comparisons for sample generation, plus $N^2 \times n \times 3$ additions for averaging the color vectors. Table 7.1 compares the complexity of ray composition for direct computation, adaptive termination and the Monte Carlo method with different volume size. Image with 32 samples[2] are given as example for direct comparison with others. Generally, the computation cycle for addition and comparison is about the same, so they are counted with one timing value A. Figure 7.1 shows the trend of computation intensity versus the number of voxels. It can be shown that the number of multiplication is far less in our method. The number of comparison in our method is larger(10

[1](o) for the direct calculation; (+) for adaptive termination and (x) for Monte Carlo Method with 32 samples.

[2]We will show later that 32 sample is sufficient for most compositions.

	Size	Multiplication	Addition	Comparison	Total
Direct	256^3	$256^3 \cdot 4$	$256^3 \cdot 4$	0	$6.68e7M + 6.68e7A$
Adap. Term.		$256^3 \cdot 4 \cdot t$	$256^3 \cdot 4 \cdot t$	0	$(6.68e7M + 6.68e7A)t$
32 Samples		$256 \cdot 32$	$32 \cdot 256^2 \cdot 4$	$32 \cdot 256^3 \cdot t$	$8.2e3M + 6.24e8A \cdot t$
N Samples		$256 \cdot N$	$N \cdot 256^2 \cdot 4$	$N \cdot 256^3 \cdot t$	$N(256M + 2.6e5A \cdot t)$
Direct	128^3	$128^3 \cdot 4$	$128^3 \cdot 4$	0	8.3e6M+8.3e6A
Adap. Term.		$128^3 \cdot 4 \cdot t$	$128^3 \cdot 4 \cdot t$	0	(8.3e6M+8.3e6A)t
32 Samples		$128 \cdot 32$	$32 \cdot 128^2 \cdot 4$	$32 \cdot 128^3 \cdot t$	$4.1e3M + 6.7e7A \cdot t$
N Samples		$128 \cdot N$	$N \cdot 128^2 \cdot 4$	$N \cdot 128^3 \cdot t$	$N(128M + 2.1e6A \cdot t)$

Table 7.1: Complexity estimation for different ray composition model

times), but generally, a comparison will cost less than multiplication (e.g. DEC MP-1, a comparison needs 24 timing clocks while a multiplication needs 239 timing clocks) The increase in comparison can be easily compensated by the decrease in multiplication. In addition, the advantage of Monte Carlo method increases as the number of voxels increases. The sample variance may be used to measure the accuracy of the estimation:

$$var[\overline{C}] = var[\frac{1}{N}\sum_{i=1}^{N} C_{R_i}] = \frac{1}{N} \cdot var[C_{R_i}] \tag{7.14}$$

The probability that a voxel's color vector C_i being sampled is determined by its opacity and the opacities of the voxels that lie before it in the viewing ray. The bigger its opacity value, the higher the probability of it being sampled. The probability decreases exponentially after the first few opaque voxels and the composited result is often determined by only a few voxels. In our experiment, a volume image of $256 \times 256 \times 256$ is composited using the Monte Carlo method (figure 7.7) and the direct computation(figure 7.4 and 7.5) respectively. Figure 7.2 shows the trend when we increase the number of samples. Peak to peak Signal to Noise Ratio (PSNR) measures the difference to the result of direct computation, which is the expectation of our estimation. Generally, a image

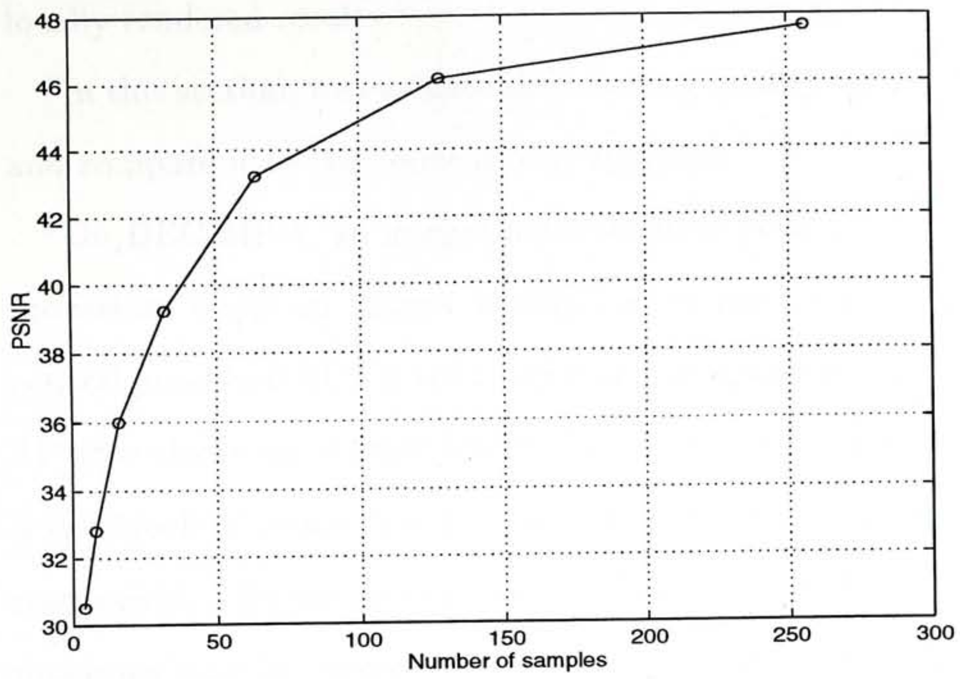

Figure 7.2: Error decrease when number of sample increase

with a PSNR higher than 38dB will be visibly identical to the original one. In our experiment, when samples are more than 32, there are little visible difference from the direct computation (figure 7.6). So, number 32 is a threshold, after which no further collection of the sample is need.

7.4 Parallel Implementation

The parallel implementation of the composition as described in the previous chapter involves three parts: the local composition, the re-ordering process and the global composition. As we have discussed the reordering and global composition process in the previous chapter, this chapter will focus on how to composite the data in each processor to get the locally composited results.

By simply applying the Monte Carlo process on the sub-ray in each processor, we get the results of associated colors as well as the associated opacities as the locally rendered results.

In this section, we evaluate the resulting quality and cost of the global result, and compare it to the conventional approach.

On DEC MP-1, an integer multiplication need 239 timing clocks on parallel processors while an integer multiplication need only 37 timing clocks on the central processor(ACU); the adding and comparison is much cheaper, with only 24 time clocks for integer and 6 clocks for character. Every processor process a sub-block of resampled data for composition of color R, G, B and opacity α respectively. As the random number used in the Monte Carlo process in each processor may be shared, we locate the generation of the random number in the ACU, where the cost of multiplication is less.

The number of rays and the length of the rays in each processor may vary for different perspectives. For simplicity, we assume the case of compositing a sub-block of $8 \times 8 \times 8$ which is perpendicularly projected on the viewing space, where the number of rays in each processor is 8×8 with 8 samples each. To composited the results of R, G, B and opacity α, the direct method need one multiplication on every sample of the ray in each component, which is $8 \times 8 \times 8 \times 4 = 2048$; the Monte Carlo method with 32 samples each ray need to generate at most 8×32 random numbers. The number of addition is $8 \times 8 \times 32$ and the maximum number of comparison is $8 \times 8 \times 8 \times 32$. Table 7.2 gives a comparison of the implementation cost to get the locally rendered results. We can see that the computation time is significantly reduced in our Monte Carlo method.

To evaluate the result, we compare the final composited image using our parallel implementation to the result of direct computation. PSNR is again used as the measurement(Figure 7.3). According to equation (7.2):

	Multiplication		Adding		Comparison		Total time
	num.	time	num.	time	num.	time	(timing clocks)
Direct computation	2048	489472	2048	49152	0	0	538624
32 Sample	256	9472	2048	49152	16384	98034	156658
64 Sample	512	19944	4096	98304	32768	196608	314856

Table 7.2: Time for ray composition with different methods

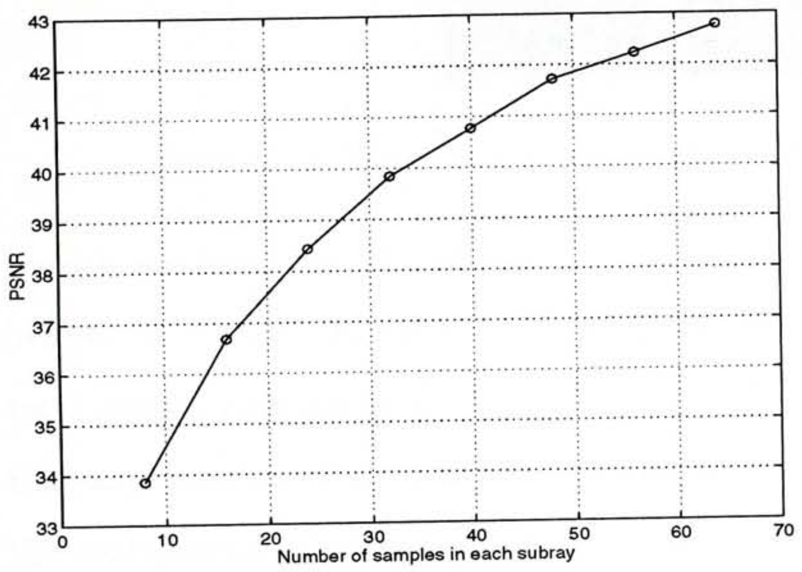

Figure 7.3: Error decrease when number of sample in each subray increase

$$\overline{C}(x,y) = \sum_{i=0}^{N-1} \overline{C}_i(x,y) \cdot \overline{\alpha}_i(x,y) \cdot \prod_{j=0}^{i-1}(1-\overline{\alpha}_j(x,y)) + \overline{C}_N \prod_{j=0}^{N-1}(1-\overline{\alpha}_j(x,y)) \quad (7.15)$$

where the voxel color vector and opacity are substituted for Monte Carlo estimated color vector $\overline{C}_i$ and opacity $\overline{\alpha}$ for sub-rays computed by the processors in a distributed manner.

Since the variance of voxel color is much smaller in the sub-ray according to the low-pass characteristic of the image, the estimation in the sub-ray should be more accurate. However, only a small error in the estimation of $\overline{\alpha}$ will be accumulated and amplified in the final result(figure 7.8 and 7.9 and 7.10). Also, some $\overline{C}_i$ may have little contribution and waste the computation resource. As

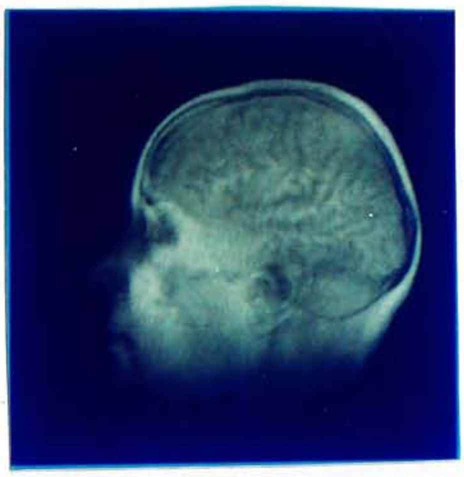

Figure 7.4: The composition result using direct computation

we can see from our experiment, the result is not as good as the one we got before. It requires 32 samples in each sub ray to get the similar result when total composition method only need 32 samples alone. It is because by using the same number of samples on each sub-ray is not close to the distribution of the contribution of samples to the final ray.

We have seen from the results of the Monte Carlo composition on the entire ray(figure 7.6 and figure 7.7) as well as the on the sub-ray(figure 7.8 and 7.9 and 7.10) that some artifact may be introduced. The concentric artifacts seen in figure 7.6 and figure 7.7 are due to the same random number used for each ray to generate the samples. The quality of figure 7.8, figure 7.9 and figure 7.10 is not good because of the error in the estimation of opacities.

7.5 Discussion and further improvement

Further improvements may be given if we choose to calculate the composited opacities instead of using Monte Carlo estimation. For some sub-rays with higher composited opacities, an adaptive algorithm may be developed to take

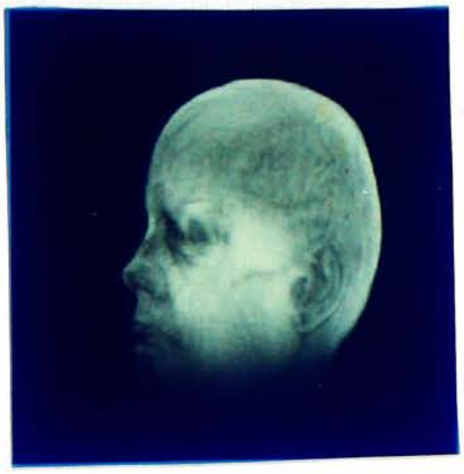

Figure 7.5: The composition result using direct computation with resampling

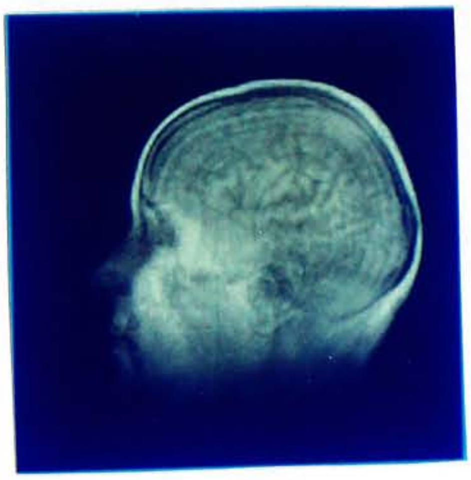

Figure 7.6: The composition result using Monte Carlo estimation(32 samples)

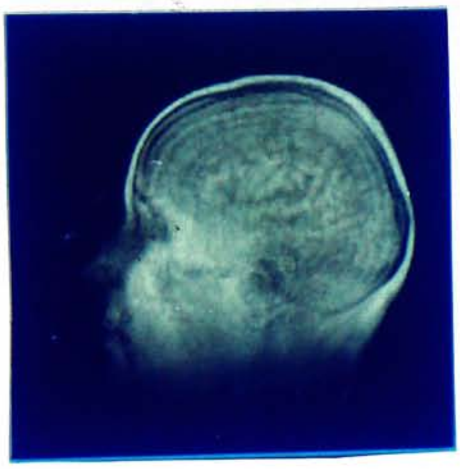

Figure 7.7: The composition result using Monte Carlo estimation (64 samples)

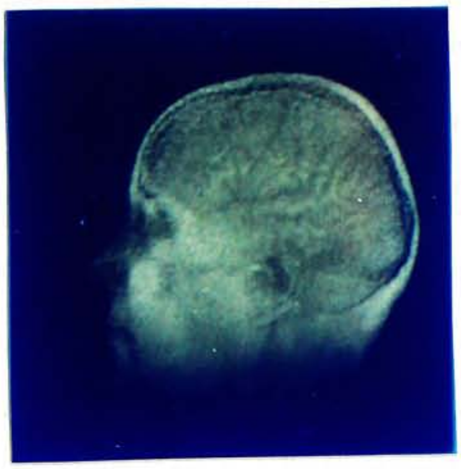

Figure 7.8: The composition result using 32 samples in one sub-ray each

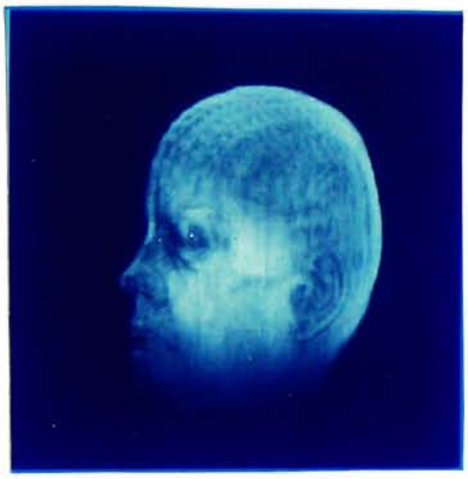

Figure 7.9: The composition result using 32 samples in one sub-ray each with resampling

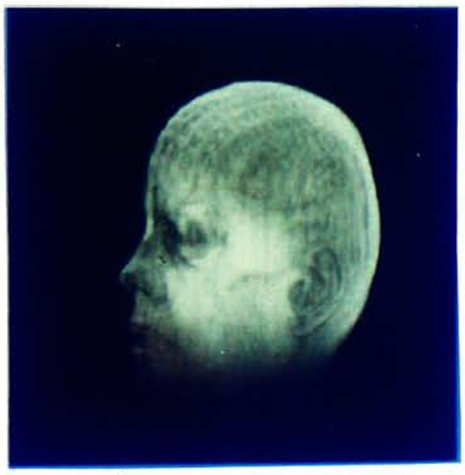

Figure 7.10: The composition result using 64 samples in one sub-ray each with resampling

more samples out. This will make the sample distribution more conform to its contribution to the final result.

In this method, the quality of the output image may be easily predicted by the variance and the number of samples. This is good for the time and quality tradeoff. Another advantage is progressiveness, which is very useful for interactivity. Sometimes, only a few samples will roughly give an acceptable rendering effect when time is important. Refinement of it does not require additional computation.

Nowadays computers may have more floating point power and the cost of multiplication is reduced a lot. This will make the advantage of Monte Carlo method lesser. However, the implementation cost for Monte Carlo composition may be reduced by using precomputed comparison table for specified opacities, and hardware implementation of random number generater.

The error caused by this method is equal to composite a ray with different opacities, which in most of the applications are assigned by user. If the difference in opacity is only moderate, the difference will be insignificant. As a result, the perceptual error would also be insignificant.

Like in hierarchical splatting[16], some large volumes of data with similar opacities and color vector can be associated first. With no chances of being sampled, voxels with transparent opacity may be dropped first.

Chapter 8

Conclusion and further work

In this thesis, we emphasize on the new techniques in parallel volume rendering. A prototype system implementing our algorithms has been developed on the Maspar System.

We develop a rendering pipeline for parallel volume rendering on the SIMD machine shown in figure 1.2. The resampling, reordering and composition part of the pipeline are given particular interest.

In the resampling part, we provide a resampling algorithms that minimize the mismatch distance between the viewing space sample and the object space sample. The mapping is more flexible for different filtering techniques. With only one mapping, different order filters can be built from the lower order one at no additional cost using the *Aitken's algorithms.* Comparing to the conventional resampling by multipass method, our method provides more flexibility and more accuracy. The trade-off is that the workload in each processor may be uneven, but this can be solved by the load balancing algorithms provided.

The reordering algorithm using binary swap is the only algorithm that uses only the local communication mechanism to align the object space samples to the viewing space layout when writing this thesis. Since the local communication is much more efficient in the SIMD machine, our algorithm is superior in performance. In addition, the workload on each processor is even and regardless of the perspectives. This provides a stable rendering speed, while the speed of the general global communication may vary in different perspectives since collision may be high in certain perspectives.

The Monte Carlo composition method proposed in this thesis is a new method for efficient composition. By sampling the data in each ray according to their contribution to the final results, the composition may be efficient. However, we have seen that the resulting quality is not satisfactory in the parallel implementation. This is due to the corresponding sampling distribution is not relating to the overall ray distribution. Further work will be done to adjust the distribution and improve the performance. The performance of the Monte Carlo method is slightly better than the adaptive termination in our model. However, modern machine often have the same operation cycle for both multiplication and addition, which may make the advantage of the Monte Carlo method lesser. However, the Monte Carlo method also has rooms for improvement, which includes hardware implementation of comparison. Also, a pre-computed comparison table could also trade the speed for more memory.

As a volume rendering system, works should also be done on a friendly user interface.

Bibliography

[1] James F. Blinn. Compositing, part i. *IEEE Computer Graphics and Applications*, pages 83–88, September 1994.

[2] B.T.Phong. Illumination for computer-generated pictures. *Communications of the ACM*, 18(5):311–317, June 1975.

[3] Digital Equitment Crop. *DECmpp 1200/Sx System Overview Manual, Programming Language Reference Manual, Programming Language User's Guide, Commands Reference Manual.*

[4] Robert A. Drebin, L. Carpenter, and P. Hanrahan. Volume rendering. *Computer Graphics*, 22(4):65–74, August 1988.

[5] Robert A. Drebin, Elliot K. Fishman, and Donna Magid. Volumetric three dimensional image rendering: Thresholding vs. non-thresholding techniques. *Radiology*, 165:131, 1987.

[6] Gerald Farin. *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide.* Academic Press Inc., 1990.

[7] E. Feibush, M. Levoy, and R. Cook. Synthetic texturing using digital filters. *Computer Graphics*, 14(3):294–301, July 1980.

[8] Michael J. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, 21:948–960, 1972.

[9] Stuart Green. *Parallel Processing for Computer Graphics*, chapter 3, page 23. The MIT Press, 1991.

[10] Pat Hanrahan. Three-pass affine transforms for volume rendering. *Computer Graphics*, 25(5):71–78, November 1990.

[11] Loowell D. Harris, R. A. Robb, T. S. Yuen, and E. E. Ritman. Non-invasive numerical dissection and display of anatomic structure using computerized x-ray tomography. *Proceedings SPIE*, 152:10–18, 1978.

[12] Karl H. Hoehne, R. L. Delapaz, R. Bernstein, and R. C. Taylor. Combined surface display and reformatting for the three-dimensional analysis of tomographic data. *Investigative Radiology*, 22(7):658–664, July 1987.

[13] William M Hsu. Segmented ray casting for data parallel volume rendering. *Parallel Rendering Symposium Proceedings*, pages 7–13, 1993.

[14] James T. Kaijiya and Brian P.Von Herzen. Ray tracing volume densities. *Computer Graphics*, 18(3), 1984.

[15] Philippe Lacroute and Marc Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. *Computer Graphics Proceedings, Annual Conference Series*, pages 451–458, 1994.

[16] David Laur and Pat Hanrahan. Hierachical splatting: A progressive refinement algorithm for volume rendering. *Computer Graphics*, 25(4):285–288, July 1991.

[17] Mark Levoy. Display of surface from volume data. *IEEE Computer and Graphics and Applications*, 8(5), 1988.

[18] Mark Levoy. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, 9(3):245–261, July 1990.

[19] Kwan-Liu Ma, James S. Painter, Charles D. Hansen, and Michael F. Krogh. Parallel volume rendering using binary-swap compositing. *IEEE Computer Graphics and Applications*, pages 59–68, 1994.

[20] Shigeru Muraki. Volume data and wavelet transforms. *IEEE Computer Graphics & Applications*, pages 50–56, 1993.

[21] I. Page. *Parallel Processing for computer vision ad display*, chapter 6, page 89. Addison-Wesley, 1989.

[22] A. W. Peath. A fast algorithm for general raster rotation. *Proceedings Graphics Interface*, pages 77–81, May 1986.

[23] P. Schröder and J.B. Salem. Fast rotation of volume data on data parallel architectures. Technical report, Thinking Machines Corporation, 1991.

[24] Victor T. Tom. Adaptive filter techniques of digital image enhancement. *SPIE Digital Image Processing: Critical Review of Technology*, 528, 1985.

[25] Takashi Totsuka and Marc Levoy. Frequency domain volume rendering. *Computer Graphics Proceedings, Annual Conference Series*, pages 271–278, 1993.

[26] L. Westover. Footprint evaluation for volume rendering. *Computer Graphics*, 24(4):367–376, August 1990.

[27] C. Wittenbrink and A.Somani. Permutation warping. *Parallel rendering symposium proceedings*, pages 57–60, 1993.

[28] G. Wolberg and T.E. Boult. Separable image warping with spatial lookup tables. *Computer Gaphics*, 23(3):369–378, July 1989.

CUHK Libraries
003598778