VISUALIZATION OF LARGE MEDICAL VOLUME DATA

NGUYEN PHU BINH

(B.Eng., M.Sc., Hanoi University of Technology)

A THESIS SUBMITTED FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2012

Declaration

I hereby declare that the thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

[illegible]

Nguyen Phu Binh

27 June 2012

Acknowledgements

Many people, in one way or another, have helped to make this dissertation a reality. I can only mention a few of them here.

First and foremost, I wish to express my deep gratitude to my supervisor, Assoc. Prof. ONG Sim-Heng, for letting me be one of your students, and guiding me along my Ph.D. study. Without your insights and comments, this dissertation and other publications of mine would not have been possible. Thanks for your understanding, patience, and belief in me.

I would like to sincerely thank my co-supervisor, Dr. CHUI Chee-Kong, who first sparked my research interest in the field of scientific visualization. Thanks for your guidance, understanding, encouragement, and most importantly, your friendship during my study in Singapore. For everything you have done for me, I can say that I am very lucky to know you, to be your student, and to work with you.

My thanks also go to Dr. Stephen CHANG for providing me a number of medical image datasets for my research, and giving me useful advices from a senior surgeon's point of view. I have gained a lot of knowledge from discussing with you, not only in medicine, but also in other fields.

Special thanks to all group members, especially ZHANG Jing, QIN Jing, LI Bing Nan, YANG Tao and TAY Wei-Liang, for your friendship and assistances. Thank you, YANG Liangjing, for helping me a lot with my writings. Thanks to HAN Thanh Trung for giving me invaluable advices in doing research.

I would like to acknowledge the financial, academic and technical supports from the ASEAN University Network and the Southeast Asia Engineering Education Development Network Project (AUN/SEED-Net). I believe you will be successful in promoting human resource development in engineering in ASEAN.

During our time living in Singapore, I and my family receive a lot of help from our friends. I wish to express my appreciation to Dominic ANG, my first and best Singaporean friend, who have helped us since the first day we came to Singapore. I still remember the night you stayed with us in NUH when my son was sick. Thanks for helping and always being with us during our hard times.

I wish to thank my best friend since primary school, DO Quoc Anh, who is currently an Assist. Prof. in Singapore Management University, for your comradeship and support. I wish you and Kieu Trang all the success on your way ahead.

I and my family are grateful for the company of our neighbors in PGP and other Vietnamese friends in NUS. Special thanks to Mr & Mrs Tuan & Cuc, Dat & Ha, WANG Shuai & Lili, Thang & Quyen, and Phong & Huong for sharing memorable times with us.

Last but not least, I would like to thank all of our family members for their love, encouragement, and sacrifice. I am deeply thankful to my parents who raised me and supported me in all my pursuits, to my parents-in-law and my mother who spent time in Singapore to support us and help us look after our son; and to my younger brother, NGUYEN Viet Anh, for helping me take charge of all family matters when I am away from home.

This dissertation is dedicated to my loving wife, DO Thi Thu Trang, who always expresses her endless support, inspiration and faith in me, and my son, NGUYEN Minh Duc, who has gone abroad and shared a Ph.D. life with his parents since he was just 5-month-old. I love you both with all my heart.

Contents

Summary

Medical volume data has an important role in medical diagnosis. However, visualization of large medical volume data is very challenging due to their large memory storage requirement, constrained processing time, and other issues related to dynamic information management. In addition to using high performance visualization hardware, developing appropriate data structures and effective rendering algorithms are essential.

This dissertation addresses several issues related to the visualization of large medical volume data. Firstly, the dissertation describes an efficient compression method for fast rendering of dynamic medical volume data. The volumes are partitioned into a set of blocks and clustered using a BIRCH-based (Balanced Iterative Reducing and Clustering using Hierarchies) algorithm, which can find a high quality clustering with a single scan of the blocks. In each cluster of blocks, a *KeyBlock* is generated to represent the cluster, leading to a significant reduction of the storage space of the volumes. In addition, a dynamic memory management scheme is also implemented using the lifetime of each *KeyBlock* to further reduce the storage space. During rendering, each *KeyBlock* is rendered as a *KeyImage*, which can be reused if the view transformation and transfer function are not changed. This can help to increase the rendering speed significantly. Experimental results showed that the proposed method can achieve good performance in terms of both speed optimization and space reduction.

Secondly, the dissertation describes a new coding scheme for efficient compression of dynamic medical volume data. The scheme uses 3-D motion estimation to create homogenous preprocessed data to be compressed by a 3-D image compression algorithm using hierarchical vector quantization. A new block distortion measure, called variance of residual (VOR), and three 3-D fast block matching algorithms are used to improve the motion estimation process in terms of speed and data fidelity. The 3-D image compression process involves the application of two different encoding techniques based on the homogeneity of input data. The proposed method can achieve a higher fidelity and faster decompression time compared to other lossy compression methods producing similar compression ratios.

Thirdly, a clustering-based framework for the automatic generation of transfer functions for the visualization of medical volume data is introduced in this dissertation. The system first applies mean shift clustering to oversegment the volume boundaries according to their low-high (LH) values and their spatial coordinates, and then uses hierarchical clustering to group similar voxels. A transfer function is then automatically generated for each cluster such that the number of occlusions is reduced. The framework also allows for semi-automatic operation, where the user can vary the hierarchical clustering results or the transfer functions generated. The system improves the efficiency and effectiveness of visualizing medical images and is suitable for medical imaging applications.

Lastly, we describe in this dissertation a method for rendering flow particles in simulation of chemotherapy drug injection. In this method, the vessels are extracted from clinical CT images using 3-D region growing, and skeletonized using a 3-D thinning algorithm. The resultant skeleton is refined to be of unit pixel width by a post processing step. The vasculature is reconstructed using cubic b-splines and represented as generalized cylinders. The flow is modeled using Hagen-Poiseuille Flow and rendered using the quadrilaterals which are aligned along the viewing direction. This rendering method can be combined with a fast volume rendering algorithm to provide more context information of the scene. Our visualization method achieves a computational efficient and good visual approximation of the flow of particles inside the vessels under fluoroscopic imaging.

List of Tables

List of Figures

CHAPTER 1

Introduction

1.1 Medical Volume Data

Volume data are three-dimensional (3-D) entities that contain information inside them. In medical imaging, volume data often refers to a stack of two-dimensional (2-D) images. Each 2-D image is a 2-D grid of *pixels* (picture elements) representing a slice of the scanned object. Typically, the distance between two consecutive pixels is constant in each direction and identical in both horizontal (x) and vertical (y) directions for most medical image modalities. This distance is called the *pixel distance*. In volume data, individual images are combined and arranged on a 3-D grid. The data elements located on the grid points are called *voxels* (volume elements). In addition to the x and y dimensions, there is a dimension representing the depth (z). The distance between two neighboring slices, i.e., the distance between two grid points in z direction, is called the *slice distance*. The three distances in x, y, and z directions are known as the *voxel spacing*.

If the pixel and slice distances are identical, the volume data is classified as an isotropic dataset; otherwise, it is an anisotropic dataset. In most cases, medical

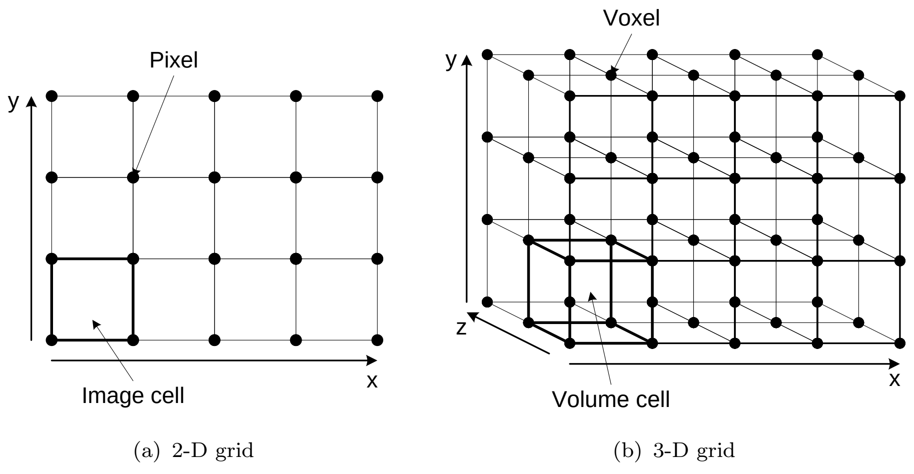


FIGURE 1.1: Image and volume cell. (a) All pixels of a slice image are arranged on a 2-D grid; (b) All voxels of a volume dataset are arranged on a 3-D grid.

volume data are anisotropic and the pixel distance is smaller than the slice distance. Four neighboring pixels in a slice image form an *image cell* and a cuboid formed by eight neighboring voxels in the grid is called a *volume cell* (Figure 1.1).

Static medical volume data. In this dissertation, *static* medical volume data refer to diagnostic 3-D images that are fixed in time. Different types of medical images can be made by using different type of energies and acquisition technologies. Common modalities, i.e., modes of producing medical images, include computed tomography (CT), magnetic resonance imaging (MRI), positron emission tomography (PET), and single photon emission computed tomography (SPECT).

Dynamic medical volume data. With advances in medical imaging technologies, the acquisition of high spatial resolution *static* medical image data is possible, allowing the assessment and analysis of the morphology of anatomic and pathological structures. However, a limitation of static image data is that they only provide snapshots of the organs of interest, and this may not be sufficient for diagnostic decisions and treatment planning. In contrast, *dynamic* or *time-varying* image data, which characterize functional processes, e.g., blood flow and metabolism, are often essential for discriminating pathologies with similar morphology and detecting diseases at an early stage. Common medical imaging modalities producing dynamic medical volume data are functional MRI (fMRI), dynamic PET (dPET), dynamic SPECT (dSPECT), and dynamic contrast enhanced (DCE) which is a modification of CT or MR imaging protocols.

1.2 Compression and Visualization of Medical Volume Data

Over the last two decades, modern technological advances in both precision and speed of medical image acquisition have led to an exploding storage requirement for medical volume data. The size of a typical medical volume dataset can range from hundreds of megabytes to hundreds of gigabytes. These datasets are often stored on servers and transmitted to clients when needed. Manipulation and visualization of huge datasets are challenging problems due to overwhelming data size, insufficient memory and I/O bandwidth, and heavy computational requirements. In addition to developing fast processing algorithms and using high performance hardware, *compression* would be extremely useful in such situations. The primary objective of medical image compression methods is to reduce the large amount of data to be stored, transmitted, or processed while preserving important diagnostic information. Compression algorithms applied to medical volume data can be generally classified into two types: *lossless* and *lossy*. Lossless algorithms allow exact reconstruction of the original data, while lossy algorithms introduce some errors or loss after the decompression process.

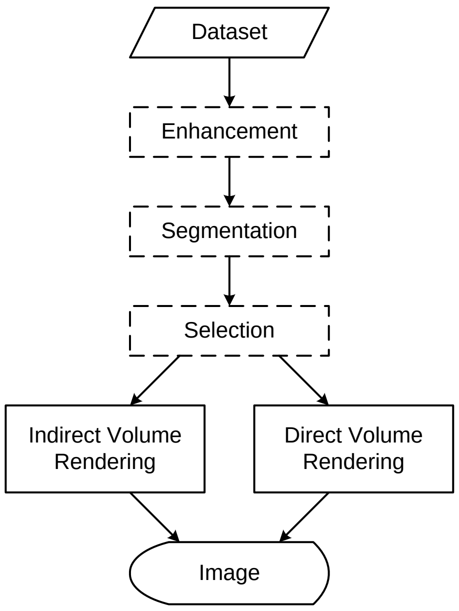


Figure 1.2: Volume visualization pipeline. Dashed boxes represent optional processes.

Since medical volume data are mainly for diagnosis, they need to be visualized to give meaningful information. The generation of a visual representation of volume data is termed *volume visualization*. Figure 1.2 shows the typical stages of a volume visualization process. After data acquisition, the quality of the dataset may need to be enhanced by filtering or applying other image processing techniques. Since the volume data contain a number of different anatomical structures, segmentation may need to be performed to separate the dataset into meaningful objects representing particular structures of interest. Subsequently, another possible step is to select a subrange of voxels by clipping or cropping the volume data. Finally, the voxels are rendered into an image using a *volume rendering* technique. There are two main categories of volume rendering: *direct* volume rendering and *indirect* volume rendering (otherwise known as *surface rendering* or *geometry rendering*). In most cases, volume rendering itself can be understood as direct volume rendering. Direct volume rendering does not use intermediate geometric primitives while surface rendering does.

1.3 Dissertation Objectives and Organization

The following research questions may be raised in processing medical volume data.

1. *How to compress a large medical volume data that minimizes the loss of important diagnostic information and concurrently supports a fast decompression for manipulation and visualization?* This is because of difficulties in manipulation and/or visualization of large medical datasets which are caused by the extremely large storage of the datasets, and the insufficiency of hardware resources and computational power. In such situations, compression would be useful besides other possible solutions, e.g., developing fast processing algorithms and using high performance hardware.

2. *How to combine efficient decompression closely with rendering to achieve the best overall performance of visualization?* This is because in most cases, compressed volume data need to be visualized. However, doing decompression and rendering in succession, especially using the central processing unit (CPU) in both processes, may not be a good solution since the interconnect system needs to transfer a very large amount of data from the CPU to the graphics processor.

3. *In direct volume rendering of medical volume data, how to automatically generate an appropriate mapping from data properties to optical properties that yields the desired visual information with as little intervention as possible from user?* Such a mapping is termed a *transfer function*, and it is an

important factor that affects the efficiency of volume rendering. However, finding appropriate transfer functions is not a trivial task since it requires an understanding of the transfer function domain and manually tweaking parameters on the part of the user. Thus, developing automatic transfer function design methods is essential.

4. *How to develop a hybrid rendering method that combines advantages of the two methods: surface and direct volume rendering?* This question arises from the fact that volume rendering is typically used for fast visualization in an overview of the image volume. It is not suitable for emphasizing specific objects or their parts because of difficulties in designing appropriate multi-dimensional transfer functions and the time consuming process to visualize small structures of interest within a large volume. In contrast, since surface rendering uses geometric primitives to represent parts of the volume data, it is capable of emphasizing objects using appropriate color and transparency settings. Hence, a hybrid rendering method to provide additional information in radiological diagnosis as well as to enable simulation and preoperative treatment planning is desirable.

This dissertation aims to address the above questions. Chapter 2 describes an efficient clustering method for fast compression and rendering of large dynamic medical volume data. In this method, the rendering is integrated tightly with the decompression process, leading to a good performance in terms of both rendering speed optimization and space reduction. Chapter 3 introduces a novel coding scheme for dynamic volume data using hierarchical vector quantization and motion compensation. This new method can achieve a higher fidelity and shorter decompression time compared to other lossy compression methods producing similar compression ratios. Chapter 4 is devoted to a clustering-based framework for the automatic generation of transfer functions for volume data visualization. The method uses multi-step clustering to incorporate both feature and spatial information to identify complex material boundaries in the dataset, then automatically produce transfer functions for a good visualization while preserving a high degree of freedom for the user to adjust the rendering results. Chapter 5 discusses an application of hybrid rendering, in which surface rendering is used to simulate the drug flow in the vascular system which has been modeled in advance, whereas volume rendering is employed to present the anatomical context. Lastly, the conclusions of the dissertation and future work are given in Chapter 6.

CHAPTER 2

Dynamic Medical Volume Data Rendering

With the advances in radiological science, dynamic scanning is increasingly popular in clinical applications. Fast rendering of these time-varying data, however, is a challenging task. With continuing improvements in spatial and temporal resolutions, these data usually comprise hundreds of megabytes or even gigabytes, thus dramatically increase the time and storage requirements for visualization. Preprocessing such as encoding and compression are usually necessary. Traditional encoding and compression algorithms based on spatial features applied to 3-D data are no longer affordable for dynamic data, as they should be analyzed and processed in joint feature-temporal space. On the other hand, we need to maintain a balance between the compression ratio and the quality of the rendered images to ensure that clinicians can obtain enough information for diagnostic decisions.

We introduce an efficient clustering method for fast rendering of these time-varying volumetric medical data (Wang et al., 2010). We divide the volumes into a set of blocks and cluster them by employing a modified BIRCH algorithm (Zhang et al., 1996) that considers both spatial and temporal coherence. In each cluster of blocks, a *KeyBlock* is generated to represent the cluster by considering the contributions of all blocks. Thus the storage space of the volumes is reduced greatly. In addition, we assign a lifetime to every *KeyBlock* and implement a dynamic memory management scheme to further reduce the storage space. During rendering, each *KeyBlock* is rendered as a *KeyImage*, which can be reused if the view transformation and transfer function are not changed. Extensive experiments have been conducted to evaluate the feasibility of the proposed method, in terms of compression speed, space savings and rendering speedup. Regression testing is also employed to analyze the impact of the compression scheme on the visual quality of rendering.

2.1 Related Methods

Among the relatively small number of published papers on time-varying volumetric medical images compression, methods which treat the data as a four-dimensional (4-D) field are dominant. In Wilhelms and Gelder (1994), a 4-D tree, which is an extension of the octree, is used for encoding time-varying data. Other methods often use the discrete wavelet transform (DWT), followed by quantization and/or a coefficient partitioning technique such as the embedded zerotree wavelet (EZW) (Shapiro, 1993) and set partitioning in hierarchical trees (SPIHT) (Said and Pearlman, 1996), and finally a symbol coding method. Zeng et al. (2002) used 4-D DWT and extended EZW to 4-D to encode the echocardiographic data. SPIHT is extended to 4-D and used with 4-D DWT in Lalgudi et al. (2005a) to compress fMRI and 4-D ultrasound images. Liu and Pearlman (2007) extended subband block hierarchical partitioning (SBHP), another coefficient partitioning technique originally described in Chrysafis et al. (2000), to 4-D and used it with 4-D wavelet decomposition for enabling progressive fidelity and resolution decompression of

4-D images. Generally, a method that relies on the 4-D wavelet transform can offer relatively high compression ratios with reasonable fidelity. However, it is not easy to achieve a fast decompression process due to the high complexity of the 4-D wavelet transform. In addition, a number of time steps (i.e., frames) may need to be decoded even if only one of them is to be manipulated or rendered.

The other approaches separate time dimension from spatial dimensions. In Shen et al. (1999), a 4-D volume rendering algorithm based on time-space partitioning (TSP) tree is proposed, and the algorithm is improved by using new color-based error metrics (Ellsworth et al., 2000). The TSP tree is effective in capturing spatial and temporal coherence of data and rendering performance is thus improved. However, the TSP tree is built as a supplementary data structure, and consequently, results in extra memory overhead and cannot reduce the space or I/O bandwidth effectively. Shen and Johnson (1994) proposed the differential volume rendering method, in which only the changed pixels are re-rendered in each time step. However, the process of determining the changed pixels may be long especially when the amount of changed pixels is large. In Liao et al. (2003) and Liao et al. (2004), this process is improved by using a two-level differential volume rendering method. The rendering performance of this method is improved since the time for determining the positions of changed pixels is reduced. However, this method cannot completely take advantage of the data coherence to further accelerate rendering.

Wang et al. (2006) proposed the dynamic linear level octree (DLLO) data structure for 4-D volume rendering. This method effectively resolves the I/O bandwidth problem and exhibits significant rendering speedup, but the employment of the octree restricts its flexibility to exploit more extensive data coherence while decomposing the volume data. In Schneider and Westermann (2003), high-pass coefficients from the Laplacian decomposition are encoded using vector quantization. During the rendering, the volume data is decompressed on-the-fly and rendered using hardware texture-mapping. This method can be applied to time-varying dataset since each volume frame can be encoded separately, producing an index texture and local codebooks for every time-step. Although the decompression speed is considerably fast due to the simple decoding, this approach does not exploit the dependency among voxels in different volumes. Furthermore, for a given resolution, the compression ratio is fixed and does not depend on the content of the volume. Several methods consider a 4-D image to be a 3-D video and extend the motion estimation and compensation techniques in video coding to 3-D for exploiting redundancies in all four dimensions (Kassim et al., 2005; Sanchez et al., 2008). However, it is difficult for these methods to achieve a good rendering performance since a number of prior frames need to be decoded before processing and rendering an intermediate frame.

2.2 Clustering-based Volume Rendering Method

This section describes our new clustering-based volume rendering method which consists of integrated clustering and rendering stages.

2.2.1 Clustering

A time-varying volumetric medical dataset usually contains a sequence of 3-D volumes, which are collections of voxels with density values. We first divide the dataset into a set of blocks (cubes). Given dataset Ψ including v volumes with n voxels in each dimension, we can uniformly divide these volumes into blocks with

m voxels in each dimension. Thus, Ψ can be divided into $v \times (n/m)^3$ blocks. A good choice of voxel number m can improve the quality of the clustering results. It usually depends on the voxel number n and the characteristics of the dataset. We adopt the BIRCH method to cluster these blocks for two reasons: (1) the I/O cost of the BIRCH algorithm is linear with the size of the dataset; and (2) the granularity of clusters can be adaptively adjusted by dynamically configuring threshold values.

2.2.1.1 BIRCH-based Clustering

BIRCH-based clustering technique is used to exploit the homogeneity of time-varying volumetric data. The blocks from all volumes are grouped into different clusters, and each cluster is represented by its centroid and radius. We denote block B_i as a vector: $\vec{B_i} = [x_1^i, x_2^i, ..., x_M^i]$, where x_j^i is the intensity value of j^{th} voxel and M is the number of voxels in block B_i, which equals to m^3. The distance between two blocks B_i and B_j is defined as follows:

$$D\left(\vec{B_i}, \vec{B_j}\right) = \frac{\left\|\vec{B_i} - \vec{B_j}\right\|}{M} = \frac{\sqrt{\sum_{k=1}^{M} \left(x_k^i - x_k^j\right)^2}}{M} \tag{2.1}$$

which is the *normalized Euclidean distance* of the two vectors representing the two blocks. We define the centroid and the radius of a cluster containing N blocks as follows:

$$\vec{C} = \frac{\sum_{i=1}^{N} \vec{B_i}}{N}, \tag{2.2}$$

$$R = \max_i \left(D\left(\vec{B_i}, \vec{C}\right)\right). \tag{2.3}$$

In our BIRCH-based implementation, all blocks are organized as a height-balanced tree, named CF tree, with two parameters, the branching factor F and the distance threshold D_{thres}. One of the advantages of the BIRCH algorithm is that it can usually find a high quality clustering with a single scan of the blocks. This is important for large time-varying volumetric medical data. Here we simply describe our implementation with some adaptations and improvements of the original BRICH algorithm. More details of BIRCH can be found in Zhang et al. (1996).

As a new block B is ready, we recursively descend the CF tree to compute the distance between the block and the centroids of existing clusters and find the cluster X_i having the smallest distance to B. This recursive operation can also be performed iteratively. Then we compute the value of new radius R_i' of X_i under the assumption that B is inserted into it. If $R_i' \leq D_{thres}$, we add the block to X_i and recompute its centroid C_i'. When there is no space for B, i.e., the number of block in X_i is greater than F, we must split the leaf node by choosing the farthest pair of blocks as seeds and redistributing the remaining blocks. If $R_i' > D_{thres}$, a new cluster is created containing only the block B. The pseudo code of our BIRCH-based clustering algorithm is presented in Algorithm 1.

In Algorithm 1, when a new block B is inserted to into the cluster X_i on trial, the centroid and radius of the cluster must be recomputed. The conventional method for computing the new centroid and radius of the cluster is to apply Equations (2.2) and (2.3). However, this method is computationally expensive since all the existing blocks in the cluster needed to be accessed repeatedly during the block insertion. To reduce the cost of computation, an approximate method is proposed for the cluster radius estimation. As illustrated in Figure 2.1, an existing cluster X_i has N_i blocks, centroid C_i and radius R_i. When a new block B is inserted into

Algorithm 1 BIRCH-based clustering algorithm

```
1: /* $RSet$: the set of blocks, initially including all the blocks.
   CF: the CF tree which is initially empty. */
2: while ($RSet \neq empty$) do
3:   select $B$ from $RSet$
4:   if (CF $\neq empty$) then
5:     /* Find the cluster with minimum distance to $B$ */
6:     select $X_i$: min(distance($B, X_i.centroid$))
7:     /* Try to insert the block into the cluster */
8:     compute $R'_i$
9:     /* Judge if the insertion is appropriate */
10:    if ($R'_i \leq D_{thres}$) then
11:      $X_i.radius \Leftarrow R'_i$
12:      compute $X_i.centroid$
13:      $X_i.size \Leftarrow X_i.size + 1$
14:      if ($X_i.size > F$) then
15:        split $X_i$
16:      end if
17:      continue
18:    end if
19:  end if
20:  create $cluster$
21:  $cluster \Leftarrow B$
22:  CF $\Leftarrow$ CF + $cluster$
23:  $RSet \Leftarrow RSet - B$
24: end while
25: return CF
```

X_i, the new centroid C'_i is computed as

$$C'_i = \frac{N_i \times C_i + B}{(N_i + 1)} \tag{2.4}$$

and the new radius R'_i is estimated based on the following formulas:

$$\begin{aligned} &r_1 = R_i + \tfrac{d}{N_i+1}, \\ &r_2 = \tfrac{d \times N_i}{N_i+1}, \\ &R'_i = \max(r_1, r_2), \end{aligned} \tag{2.5}$$

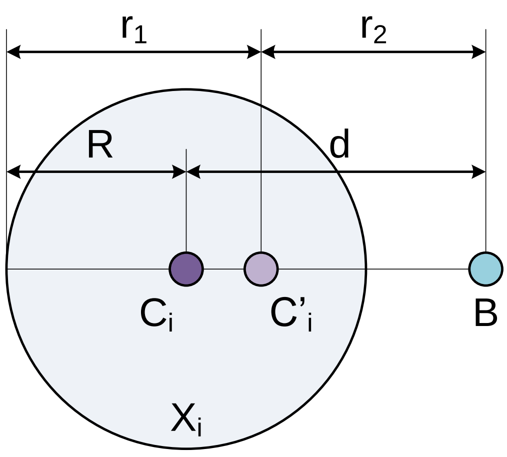


FIGURE 2.1: Estimation of the center and radius of a cluster for a trial insertion of a block.

where d is the distance between the trial block B and the centroid R_i. r_1 and r_2 are the two candidate radii, and the greater one will be chosen as the radius of the updated cluster. This mathematical model mimics the linear interpolation between two weighted points in the M-dimensional space. If a block is inserted into a cluster, the new center of the cluster will be pulled towards the inserted block, and the displacement is inversely proportional to the weights of the two M-dimensional points, which are the number of blocks represented, respectively.

It is easy to prove that Equations (2.2) and (2.4) produce the same results, meaning that there is no error introduced in the computation of the new centroid. However, Equation (2.5) tends to produce a value greater than that of Equation (2.3), i.e., the radius could be over estimated. The cluster is actually denser than that implied by the estimated radius. Therefore, this method is effective in producing clusters strictly under the pre-defined error-tolerance D_{thres}. Section 2.2.1.2 will present one technique to adjust D_{thres} for improving the quality of the clustering. The proposed method is also computationally efficient as it avoids accessing blocks that are already in the cluster. Furthermore, in contrast to Equation (2.3), the computation of the radius in Equation (2.5) is independent of the centroid. Thus, only the radius is evaluated when trying to insert a block into a cluster, and the centroid is updated only when the radius satisfies the cluster criterion. This implementation significantly improves the performance of the clustering process.

2.2.1.2 Clustering Granularity

It is obvious that the threshold value D_{thres} greatly affects the size and quality of the clustering. For example, cluster centroids can be too close or cross each other if D_{thres} is too small. The number of clusters n_c also increases when D_{thres} decreases. On the other hand, if D_{thres} is too large, the intensity changes in the data may be lost during the rendering. Fortunately, BIRCH provides mechanisms to dynamically increase the threshold value when building the CF tree. Thus, we can set small initial value for D_{thres} and adjust it based on the CF tree that has been built. In the default implementation of BIRCH, D_{thres} is set as 0. In our implementation, we usually set the initial value based on the nature of the volumetric data, e.g., the intensity distribution of the data.

Suppose that D^i_{thres}, the current threshold value, turns out to be too small. A heuristic approach is used to estimate the next value D^{i+1}_{thres}. First, the total number of clusters n_c increases with the number of inserted blocks n_b. By preserving a relationship table of n_c and n_b during the CF tree building, we can estimate n_c^{i+1} using a least squares linear regression. Thus, we can approximately set

$$D^{i+1}_{thres} = D^i_{thres} \times \frac{n_c^{i+1}}{n_c^i}. \tag{2.6}$$

Second, if we want to decrease the number of clusters in current CF tree, it is reasonable that we increase D^i_{thres} by adding the distance of two closest clusters to it so that at least these two clusters can be merged. We set the new threshold D_{thres} according to

$$D_{thres}^{i+1} = \max\left(D_{thres}^{i} \times \frac{n_c^{i+1}}{n_c^{i}}, D_{min}\right). \tag{2.7}$$

2.2.1.3 Output Data

In the above descriptions, for simplicity, we assumed that each volume has n voxels in each dimension and each block has m voxels in each dimension. In practice, the number of voxels in each dimension of a volume can be different. For an adaptation to the size of the volume, each block dimension can have a different number of voxels. In addition, a dataset with a large number of time steps can be divided into multiple groups of frames in time order in which the proposed clustering algorithm is applied.

In our implementation, the centroid of a cluster is termed a *KeyBlock*. The output of the clustering step is a binary file containing three following sections:

1. *Header information* stores the resolution of the 3-D volumes, number of time steps, voxel format, data description and pointers to other sections.

2. *Volume-KeyBlock table* is a collection of lookup tables corresponding to all the 3-D volumes, one table for each volume at one time step. Each table can be considered as a 3-D array in which each element is a number representing the link between the corresponding block in the volume and the *KeyBlock* of the cluster it belongs to. This number actually is the index of the *KeyBlock* in the *KeyBlock data* section.

3. *KeyBlock data* contains all *KeyBlock* generated.

For efficient memory management, each *KeyBlock* is associated with a *last-volume number* (LVN), which is the index number of the last volume which contains blocks belonging to the cluster represented by the *KeyBlock*. The LVN indicates the life period during which a *KeyBlock* is used to reconstruct volume(s) from time to time and should reside in the memory. It also indicates the expiring time after which the *KeyBlock* should be released. The *KeyBlocks*, therefore, are not released one by one as the order they are loaded in. A dynamic memory management scheme should be employed during the implementation. In this way, *KeyBlocks* are stored so that they can be properly loaded and released as the sequence of volume being processed.

2.2.2 Rendering

In the rendering stage, each 3-D volume is reconstructed and rendered using any of various existing volume rendering techniques directly or with some optimizations. For instance, a ray casting-based rendering method using the proposed clustering algorithm can be described as follows.

Denote the index of the working volume as q. Initially, the volume at the first time step is used as the working volume ($q = 1$) and the following steps are executed:

1. *KeyBlocks* whose LVNs are less than q are released together with their associated partial-image buffers. The final image of the current time step is initialized.

2. *KeyBlocks* are read from the binary file in turn. Each *KeyBlock* is associated with a partial-image buffer, and *KeyImage*, the rendering result of each

KeyBlock, is saved into the partial-image buffer. After all the *KeyBlocks* in volume q are loaded, they are rendered according to the following two rules:

- *Rule 1.* If current volume is the first volume, all the *KeyBlocks* are rendered.
- *Rule 2.* If the current model-view transformation or transfer functions are changed as compared to that in the previous time step, all *KeyBlocks* are re-rendered; otherwise, only *KeyBlocks* that are newly loaded are rendered.

3. The *KeyImages* of the *KeyBlocks* are composed in 2-D space according to the *Volume-KeyBlock table* of volume q and the final image is constructed by the following rules:

   - *Rule 1.* According to the current viewing direction, blocks in volume q are accessed in front-to-back order. Using the information in the *Volume-KeyBlock table*, *KeyBlocks* can be easily located.
   - *Rule 2.* The *KeyImages* of the *KeyBlocks* are composed into the final image at the corresponding projection area.

   After all blocks of volume q are processed, the final image is produced and displayed.

4. To proceed the volume at the next time step, q is increased by one ($q = q + 1$).

The above steps are repeated until the entire sequence is processed. In this algorithm, once the *KeyImages* are produced, the final image is generated by composing their colors and opacities in front-to-back order based on the theory of partial ray composing. The final image can be composed from the *KeyImages* by using the *over* operator Porter and Duff (1984). 2-D re-sampling of the *KeyImages* may be required if the sampling rate of the *KeyImage* is different from that of the final image or when they are not sampled along the same set of rays. The early-ray-termination Levoy (1990) is still possible for both *KeyBlock* rendering and *KeyImage* composition, where samples in *KeyImages* can be safely skipped when pixels of the final image are already opaque.

In the rendering methods using the *over* operator directly (e.g., ray casting), *KeyImages* are used as the intermediate results for fastening the composition step; thus, improving the rendering speed. In other methods (e.g., texture mapping), the *KeyImage* may not be used since no *over* operation is directly performed. However, the rendering speed still improves in this clustering-based rendering algorithm since the use of *KeyBlocks* helps reduce the I/O bandwidth effectively.

2.3 Results and Discussion

In our experiments, three time-varying volume datasets named HAND, HEART, and ABDOMEN in DICOM format were used to evaluate the performance of the proposed algorithm (Table 2.1). They were all acquired at the National University Hospital, Singapore. The computing platform was a 2.52 GHz Intel Pentium IV desktop PC equipped with 1 GB RAM and a NVIDIA Quadro 4 700 XGL graphics card with 64 MB onboard memory.

The raw image data and their descriptions are extracted from the datasets to form the input data of the experiments. In the encoding phase, we used the proposed clustering algorithm with different initial distance threshold values D_{thres}

TABLE 2.1: Dataset specifications

Dataset	HAND	HEART	ABDOMEN
Bits allocated	8	8	8
Rows × Columns	512×512	192×156	256×256
Slices	136	27	12
Time step	5	20	39
Pixel size (mm)	0.39×0.39	1.67×1.67	1.02×1.02
Inter-slice spacing (mm)	4.0	8.0	5.0
Size (MB)	171.25	15.42	29.25
Modality	MRA	MRI	MRU

TABLE 2.2: Results of encoding the HAND dataset using different distance threshold values

Test name	Block size	D_{thres}	Time (s)	Size (MB)	SS (%)
HAND A	16×16×17	0.10	3953	36.27	78.7
HAND B	16×16×17	0.15	3044	26.29	84.5
HAND C	16×16×17	0.20	2525	20.94	87.7

TABLE 2.3: Results of encoding the HEART dataset using different distance threshold values

Test name	Block size	D_{thres}	Time (s)	Size (MB)	SS (%)
HEART A	12×13×27	0.10	22	3.53	77.1
HEART B	12×13×27	0.15	16	2.27	85.3
HEART C	12×13×27	0.20	13	1.61	89.6

to compress each dataset. The output of this phase are binary files in the format described in Section 2.2.1.3 with each file corresponding to a specific value of D_{thres}. The computing times and the space savings were measured. The parameters used and results of this phase are shown in Tables 2.2 to 2.4. In these tables, the space savings (SS) is defined as follows:

$$\text{SS} = \left(1 - \frac{\text{Compressed file size}}{\text{Uncompressed file size}}\right) \times 100\%. \tag{2.8}$$

Table 2.4: Results of encoding the ABDOMEN dataset using different distance threshold values

Test name	Block size	D_{thres}	Time (s)	Size (MB)	SS (%)
ABDOMEN A	16×16×12	0.10	305	20.76	29.0
ABDOMEN B	16×16×12	0.15	264	17.46	40.3
ABDOMEN C	16×16×12	0.20	236	14.93	49.0

The proposed cluster-based time-varying volume rendering algorithm was implemented using two underlying rendering techniques: 2-D texture mapping and 3-D texture mapping. If 3-D texture mapping is supported by the graphics card, the rendering speed can be improved due to the fast hardware accelerated 3-D interpolation. Otherwise, 2-D texture mapping can be used but software sampling may be required to create the texture images for the three major orientations. The experiments evaluate the improvement in terms of rendering speed of our algorithm compared to that of the 2-D texture mapping and 3-D texture mapping techniques.

After an encoded dataset is loaded into the system, it is rendered repeatedly 20 times with different preset viewing angles while the rendering timing of each time step is recorded. The execution times of the last 10 running times are then averaged and reported as the performance result of the dataset. The design of the experiment ensures that the recorded execution times are obtained when the system is stable and renderers can benefit from the I/O cache if possible. The speedup ratios obtained are presented in Table 2.5.

As seen from Table 2.5, depending on the specific dataset and the underlying rendering techniques used, the proposed algorithm yielded the rendering speedup of 1.5 to 9.4 compared to the corresponding regular algorithm. This is due to the fact that the data coherence in time-varying volume datasets is extensively

Table 2.5: Speedup ratios over the regular rendering techniques obtained when applying our method on different datasets

Test name	2-D texture mapping	3-D texture mapping
HAND A	7.14	1.56
HAND B	8.72	1.64
HAND C	9.44	1.69
HEART A	3.62	2.23
HEART B	4.71	2.39
HEART C	5.56	2.47
ABDOMEN A	3.98	1.59
ABDOMEN B	4.59	1.63
ABDOMEN C	5.14	1.74

exploited in both space and time dimensions through the employment of our clustering method. A number of works analyzed in Section 2.1 also considered both spatial and temporal coherence. However, they were mostly based on adjacent regions in the two dimensions. In our method, blocks that are grouped into a cluster may come from any portion of the dataset in both space and time dimensions, leading to an efficient saving of storage space and I/O bandwidth. Another advantage of the method is the simple decoding mechanism which is essential for fast volume rendering algorithms. Furthermore, similar regions can be represented by the same *KeyBlock* and are rendered only once if rendering parameters are unchanged. This also significantly reduces the time cost for rendering.

The proposed algorithm performs a lossy compression of the time-varying volume data. It is necessary to analyze the impact of the compression scheme on the visual quality. A regression testing method is employed for this purpose. The regression testing compares a test image that is produced with an algorithm being evaluated with a reference image that is assumed to be indeed correct. The comparison takes into account dithering and anti-aliasing effects, and creates an output image representing the difference between the test image and the reference image Schroeder and Martin (1998). The difference of the two images is also quantified in terms of *absolute error* (E_A) and *thresholded error* (E_T), which are calculated based on the following equations:

$$\begin{aligned} D_i &= \frac{\left|r_i^v - r_i^t\right| + \left|g_i^v - g_i^t\right| + \left|b_i^v - b_i^t\right|}{3}, \\ E_A &= \frac{\sum_i D_i}{L_c - 1}, \end{aligned} \tag{2.9}$$

$$\begin{aligned} H_i &= \begin{cases} D_i - T & if\, D_i - T > 0 \\ 0 & otherwise \end{cases}, \\ E_T &= \frac{\sum_i H_i}{L_c - 1} \end{aligned} \tag{2.10}$$

where (r_i^v, g_i^v, b_i^v) and (r_i^t, g_i^t, b_i^t) are the ith color pixel value (red, green and blue) of the reference image and the test image respectively; D_i is the difference between the ith pixel of the two images; L_c is the number of color levels of each channel; T is a threshold-tolerance for pixel differences. Thus, the *absolute error* is the total error in comparing the two images, and the *thresholded error* is the error for a given pixel minus the threshold and clamped at a minimum of zero. The latter will be more effective in representing the noticeable differences between two images.

In our implementation, all the images are generated in color with red, green and blue channels, and each channel has 8 bits, i.e., $L_c = 2^8 = 256$ levels. In order to avoid misinterpretation of images with the introduction of pseudo-colors, pixels are assigned with the same value for all three channels and the images thus appear in gray. A threshold-tolerance of 5 is used in the analysis of the image quality. It is less than 2% of the maximum pixel difference and normally is not noticeable by human eye. The images generated by the regular texture mapping method are employed as the reference images, and the image quality of cluster-based rendering

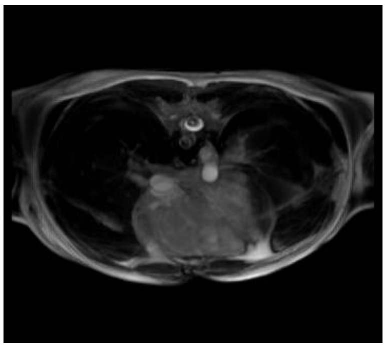
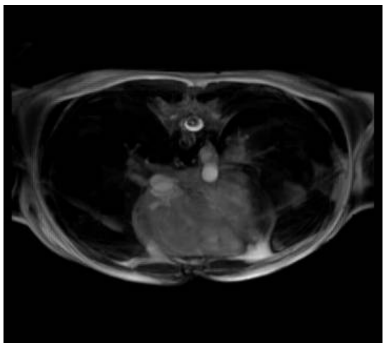
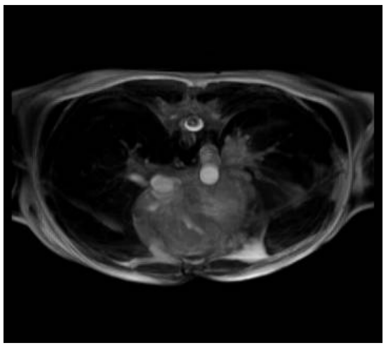
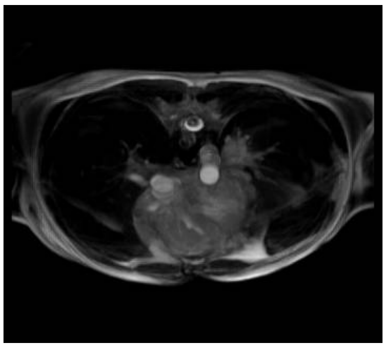
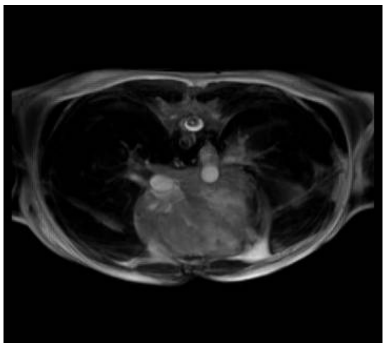
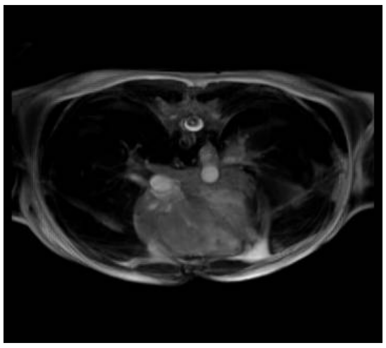
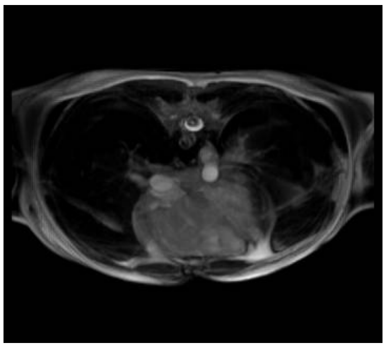
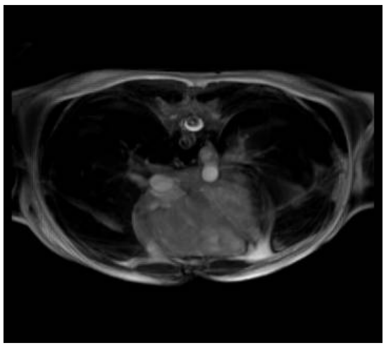
TABLE 2.6: Error analysis of cluster-based rendering algorithm

Test name	E_A	E_T
Inter-step	547.113	104.500
HAND A	15.796	0.009
HAND B	17.938	0.059
HAND C	30.066	0.541
Inter-step	360.980	52.322
HEART A	39.980	0.321
HEART B	54.321	1.182
HEART C	73.284	7.749
Inter-step	2761.917	1658.556
ABDOMEN A	206.086	17.966
ABDOMEN B	254.729	36.396
ABDOMEN C	343.661	70.943

is evaluated based on the following procedures.

For each dataset, the regression testing is applied to each pair of corresponding images at each time step. The *absolute error* (E_A) and *thresholded error* (E_T) of this time step are calculated accordingly. After the regression testing is finished for all time steps, the results are averaged and used to represent the error of the cluster-based rendering of this dataset. Based on the images generated by the regular texture mapping method, the regression testing is also applied between the images at successive time steps. The results are averaged and used to indicate the inter-step differences. It provides us an effective reference to evaluate the rendering quality. The inter-step errors also serve as a good measurement of the coherence of the dataset. Details of regression testing can be found in Schroeder and Martin (1998). Table 2.6 presents the results from the error analysis. It is clear that the cluster-based rendering algorithm achieves good rendering quality from the negligible error. Selected images are shown in Table 2.7, Figure 2.2, and Figure 2.3.

Table 2.7: Comparison of the image quality at different time steps between 2-D texture-mapped rendering and cluster-based rendering of HEART dataset ($D_{thres} = 0.15$)

Reference image	Rendered image	Error
		$E_A = 118.0$ $E_T = 11.50$ Step 1
		$E_A = 46.2$ $E_T = 0.30$ Step 7
		$E_A = 48.5$ $E_T = 0.18$ Step 13
		$E_A = 41.0$ $E_T = 0.04$ Step 20

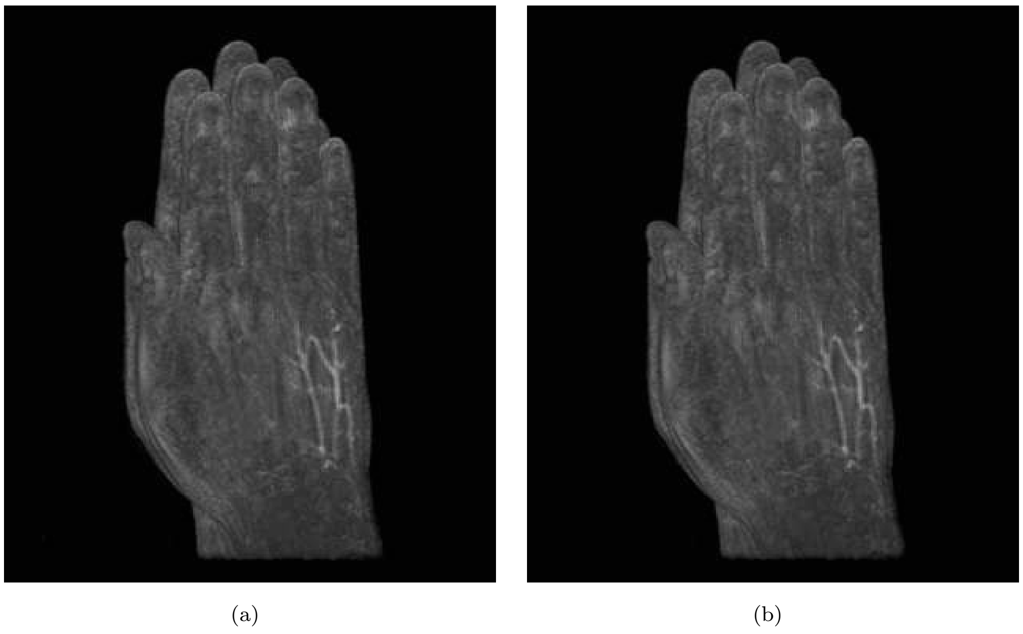


FIGURE 2.2: Comparison of the image quality between two rendering algorithms on the volume at the last time step in HAND dataset. (a) 2-D texture-mapped rendering and (b) cluster-based rendering ($D_{thres} = 0.20$).

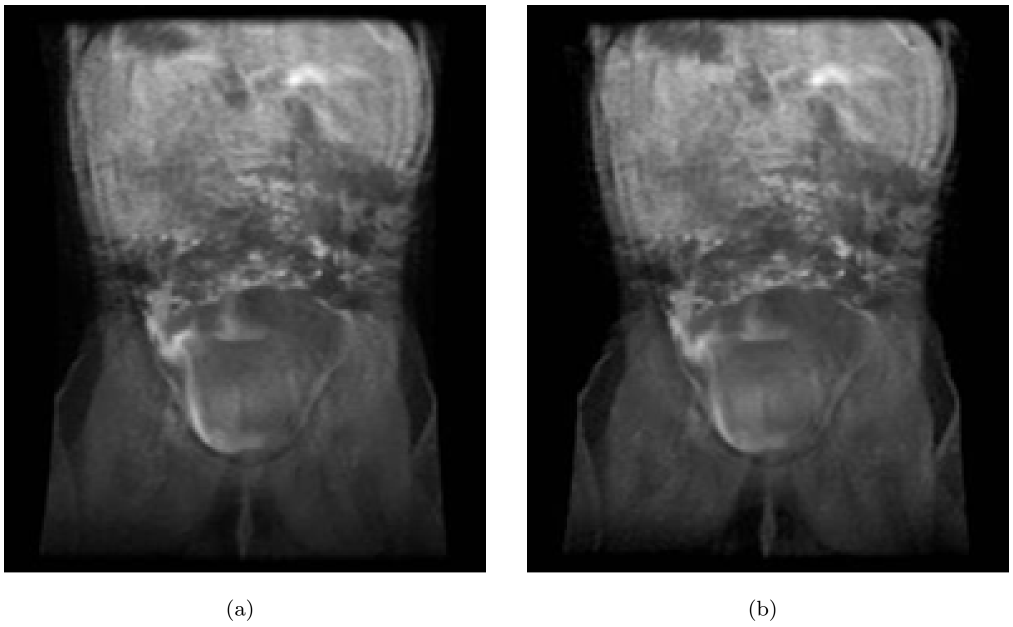


FIGURE 2.3: Comparison of the image quality between two rendering algorithms on the volume at the last time step in ABDOMEN dataset. (a) 2-D texture-mapped and (b) cluster-based rendering ($D_{thres} = 0.20$).

2.4 Summary

In this chapter, we have introduced a clustering-based volume rendering algorithm, which is a new method for fast visualization of time-varying volumetric medical images. The algorithm takes advantage of the inherent characteristics of time-varying volume data. Data coherence is exploited in both spatial and time dimensions through the employment of the clustering technique so that the rendering performance is enhanced. Since there is no restriction on the underlying type of renderers, the algorithm also provides flexibility for further extension. Extensive experiments were performed based on the texture-based implementations of this algorithm. A good performance was achieved in terms of both speed acceleration and space reduction. Results demonstrated the superiority of this method over regular algorithms for time-varying volume rendering. We could obtain over 89% of space savings and up to 9 times increase in rendering speed. Based on the analytical results of regression testing, errors introduced due to clustering-tolerance are quantitatively and visually small. Hence, high rendering fidelity can be achieved.

Chapter 3

Dynamic Medical Volume Data Compression for Visualization

In medical imaging, although static image data allow the assessment of the morphology of anatomic and pathological structures, many aspects relevant to diagnosis and treatment planning may be missed since static image data only provide snapshots of interested organs. This issue can be overcome by using dynamic image data, which characterize functional processes. However, in addition to issues related to dynamic information, such as motion correction, huge storage requirement is a major problem when visualizing dynamic image data. In this situation, compression is extremely useful to reduce the transmission bandwidth and the data loading time, thus fasten the rendering process. In this chapter, we describe a new compression scheme for dynamic medical volume data that can minimize the loss of important diagnostic information and concurrently support a fast decompression for manipulation and visualization (Nguyen et al., 2011a). The method combines 3-D motion estimation using a new block distortion measure and hierarchical vector quantization in an efficient way to achieve a higher fidelity and faster decompression time compared to other lossy compression methods producing similar compression ratios.

3.1 Related Methods

Compression techniques can be either lossless or lossy. Lossless algorithms allow exact reconstruction of the original data, while lossy algorithms introduce some error or loss after the decompression process. Lossless techniques may be used in cases where the datasets are not very large, but are not suitable for applications with limited transmission bandwidth or storage constraints, e.g., in picture archiving and communication systems (PACS), teleradiology, and remote dataset browsing systems. In these cases, lossy methods are more appropriate since they offer higher compression ratios. Popular lossy compression methods for medical images often use (1) an image transform, such as the discrete cosine transform (DCT) (Lee et al., 1993; Mohsenian et al., 1995; Lee et al., 1995; Wu and Tai, 2001) or the wavelet transform (Pratt et al., 1996; Menegaz and Thiran, 2003; Schelkens et al., 2003; Xiong et al., 2003; Miaou and Chen, 2004; Wu and Qiu, 2005), followed by (2) quantization and/or a coefficient partitioning techniques such as the embedded zerotree wavelet (EZW) (Shapiro, 1993) and set partitioning in hierarchical trees (SPIHT) (Said and Pearlman, 1996), and finally (3) a symbol coding method. The basic idea of applying an image transform to volume data is to obtain a different distribution in the transform domain that can make quantization and arithmetic coding more efficient.

In the case of manipulating or rendering time-varying medical volumetric images, which can be considered four-dimensional (4-D) images, data compression has become a very more important requirement. Among the relatively small number of published papers on 4-D medical image compression, methods that treat the data as a 4-D field are dominant. Wilhelms and Gelder (1994) proposed a 4-D tree, which is an extension of the octree, for compression and rendering of time-varying data. Each tree node contains a model of the data represented as a fixed number of basis functions, a measure of the modeling error, and a measure of the importance of the corresponding data. Although their method can provide flexibility in controlling the image quality and rendering speed by using user-defined tolerances of modeling error and data importance, the compression ratios for this method are not as good as those for other methods. Other methods often use the 4-D discrete wavelet transform (DWT) in a lossy compression scheme. Zeng et al. (2002) used the 4-D DWT and extended the EZW to 4-D to encode echocardiographic data. Although their method can handle arbitrarily sized input data and offers a wide range of compression ratio, the fidelity of decompressed data in terms of peak signal to noise ratio (PSNR) is rather low. Another limitation is that their experiments were conducted based on only one set of small size 4-D ultrasound data; no other image modality or large dataset was examined. In another work, Lalgudi et al. (2005b) studied the extension of JPEG2000 to 4-D and proposed a method to compress fMRI images using the DWT and JPEG2000. A 1-D DWT is applied along the time dimension of 4-D data first, then followed by another 1-D DWT along the z-axis, and finally the resulting 2-D wavelet coefficients are compressed using JPEG2000. The experimental results showed that the method was slightly better than other methods using 3-D JPEG2000. Another method was proposed by the same group (Lalgudi et al., 2005a), using the 4-D EZW and 4-D SPIHT with the 4-D DWT to compress fMRI and 4-D ultrasound images. Nevertheless, only an insignificant improvement was achieved compared to their previous work, i.e., the 4-D JPEG2000. Liu and Pearlman (2007) extended subband block hierarchical partitioning (SBHP), another coefficient partitioning technique originally described in Chrysafis et al. (2000), to 4-D and used it with 4-D wavelet decomposition for progressive fidelity and resolution decompression of 4-D images. In lossless mode, this method is comparable to the 4-D JPEG2000 (Lalgudi et al., 2005b), 4-D EZW and 4-D SPIHT (Lalgudi et al., 2005a). In lossy mode, although the method outperforms two other possible 3-D SBHP coding schemes, no comparison with other lossy methods is reported. Generally, a method that relies on the 4-D wavelet transform can offer relatively high compression ratios with reasonable fidelity. However, it is not easy to achieve fast decompression due to the complexity of the 4-D wavelet transform. In addition, a number of time-steps (i.e., frames) have to be decoded even if only one of them is to be manipulated or rendered.

The other approaches process the time and spatial domain separately. It is obviously possible to compress 3-D volumes independently but this approach does not exploit the dependency of corresponding voxels in different volumes. A good method should exploit the high correlation between volumes in 4-D medical images. Several methods consider a 4-D image to be a 3-D video and extend the state-of-the-art methods in video coding to 3-D for exploiting redundancies in all four dimensions. Kassim et al. (2005) proposed a combination of the 3-D integer wavelet transform and 3-D motion compensation is used for lossy-to-lossless compression of 4-D medical images. Their experimental results showed that the lossless mode of this method can reduce the coding bit rates by approximately 25% compared with a method that applies a conventional 3-D compression technique for each time step, and the drop in image quality in the lossy mode is hardly noticeable when compared to other methods producing the same compression ratio. Sanchez et al. (2008) introduced a lossless compression method for 4-D medical images based on the most advanced features of the H.264/AVC standard, e.g., multi-frame motion compensation, variable block size and sub-pixel mode in motion estimation. The compression ratios obtained are superior to 3D-JPEG2000 when encoding fMRI images but similar to other lossless compression methods when applied to other types of medical images. This method was extended by Sanchez et al. (2009) with a new multi-frame motion compensation process that more effectively reduces the redundancy in both spatial and temporal dimensions by employing a 4-D search, variable-size block matching, and bidirectional prediction. The outputs of the motion compensation process, i.e., the residual and motion vector, are compressed by using a new context-based adaptive binary arithmetic coder designed based on the probability distribution of the data. Evaluation results showed that compared to 4-D JPEG2000 and H.264/AVC, this method archived an average improvement on compression ratio of 13% on real fMRI data. However, the outcome of the method when applied to other medical imaging modalities is still unknown.

We introduce a method that uses hierarchical vector quantization and 3-D motion compensation for the compression of 4-D medical images (Nguyen et al., 2011a). While this approach is motivated by the encoding technique presented in Schneider and Westermann (2003), we have introduced a number of modifications and extensions and include three novel 3-D motion estimation algorithms for improving the fidelity of the decompressed data. The 3-D motion estimation process is applied at the beginning of the compression scheme to create a homogenous preprocessed data to be compressed by a 3-D image compression algorithm using hierarchical vector quantization. A new block distortion measure, called *variance of residual*, and three 3-D fast block matching algorithms are used to improve the motion estimation process in terms of speed and data fidelity. The 3-D image compression process involves the application of two different encoding techniques based on the homogeneity of input data. Simulations confirm that we are able to obtain better fidelity and faster decompression compared to other compression methods while achieving similar compression ratios.

3.2 Compression Scheme

Video compression concepts are applied to 4-D medical images, which consist of sequences of 3-D image frames. Figure 3.1 provides an overview of the encoding process. In order to encode a group of 3-D image frames, the first frame (the key frame) F_0 is separately encoded and reconstructed by using a 3-D compression algorithm. The 3-D compression algorithm used in our work is based on HVQ, and is presented in Section 3.3. To encode the remaining frames (i.e., the intermediate frames), a new 3-D motion estimation algorithm, described in Section 3.4, is applied to produce the predicted frame P_i at time step i from the reconstructed frame F'_{i-1} so that P_i is matched with the frame at the same time step F_i as closely as possible (according to a matching criterion). F_i is then motion-compensated by subtracting P_i to produce a motion-compensated residual frame R_i. R_i is encoded and transmitted with a set of motion vectors v_i, which is the information required to recreate P_i from F'_{i-1}. Subsequently, the encoded version of R_i is decoded to produce R'_i, which is then added to P_i to generate the reconstructed frame F'_i to be used to encode the next frame. Optionally, the resulting bit stream can be arithmetic coded for further compression.

In the decoding phase, the key frame and residual frames are decoded by using the decompression process corresponding to the 3-D compression algorithm. These frames will be used to successively reconstruct the intermediate frames. The first predicted frame P_1 is reconstructed based on the reconstructed key frame F'_0 and

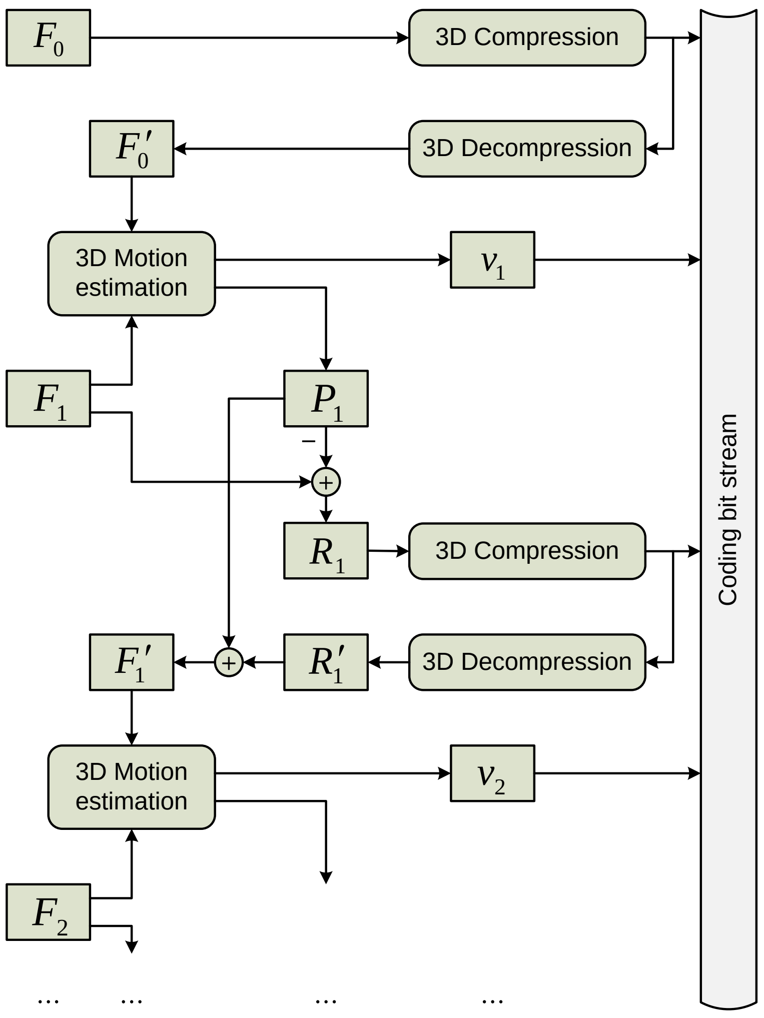
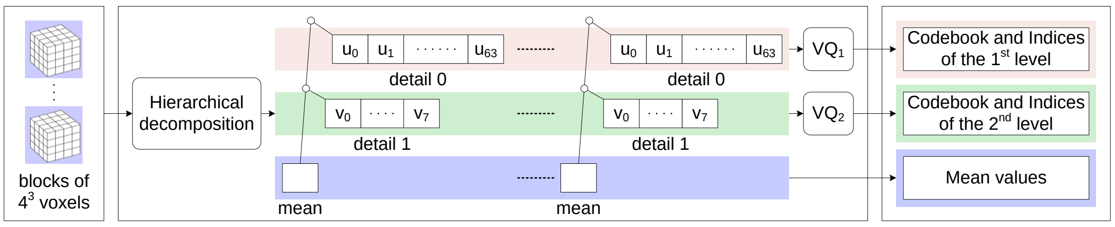


FIGURE 3.1: Overview of the encoding process.

the associated motion vectors v_1. Then P_1 is added to the decoded version of the first residual frame, R_1', to produce the reconstructed first intermediate frame F_1'. Subsequently, F_1' and the decoded version of the second residual frame, R_2', with its associated motion vectors v_2 are used to reconstruct the second intermediate frame F_2'. The process continues until all intermediate frames are reconstructed.

3.3 Three-Dimensional Image Compression using Hierarchical Vector Quantization

Part of our 3-D compression algorithm is based on the hierarchical vector quantization scheme proposed by Schneider and Westermann (2003), henceforth referred to as SW. In this method, the volume data is initially partitioned into disjoint cubes of $4 \times 4 \times 4$ voxels. Each cube is down-sampled by a factor of two by averaging disjoint sets of $2 \times 2 \times 2$ voxels. A 64-component vector (i.e., the first level data) is formed to store the difference between the original data samples and the respective down-sampled value. By applying the same process to the down-sampled version, a single value that represents the mean value of the entire $4 \times 4 \times 4$ cube and an 8-component vector (i.e., the second level data) carrying the residual data is obtained. The mean of the cube is stored in a 1-component vector (i.e., the third level data). In the next step, all the 64-component and 8-component vectors are separately sent to two vector quantizers to produce two codebooks containing 64- and 8-component codewords. This method is mainly applied to 8-bit volume datasets with the assumption that the length of each codebook is 256. Therefore, each cube is represented by three 8-bit values: one value represents the mean of the cube while the other two are indices into the respective codebooks representing the difference information. To decode a particular cube, its mean value and the difference information from the two codebooks are summed. The vector quantizer uses a principal component analysis (PCA) splitting scheme to find an initial codebook which is then refined by using the LBG algorithm (Linde et al., 1980). Some optimizations are applied to the refinement step, including restricted searching to a k-neighborhood of the original cell and partial searches for a speedup of the distortion calculation. The major steps of the method are shown in Figure 3.2.



blocks of 4^3 voxels
Hierarchical decomposition
u_0 u_1 ...... u_{63}
detail 0
v_0 .... v_7
detail 1
mean
VQ_1
VQ_2
Codebook and Indices of the 1st level
Codebook and Indices of the 2nd level
Mean values

Figure 3.2: Hierarchical decomposition and quantization of volumetric data.

The vector quantization scheme of the SW method can achieve compression rates and fidelity similar to that of wavelet based compression (Schneider and Westermann, 2003). Decompression speed is extremely fast due to the simple decoding. Furthermore, this compression scheme can perform decompression and rendering simultaneously on a graphics processing unit (GPU) to achieve the interactive frame rate for visualization applications. Our investigations have revealed that it is still possible to improve on the performance of SW by making several modifications. Firstly, regardless of the sizes of the two codebooks and other additional information, the CR for an 8-bit volume dataset using two 8-bit index codebooks is limited by a factor of $64 \div 3 \approx 21.3$. The CR in vector quantization is fixed for a given image resolution and does not depend on the content of the volume. For a $4 \times 4 \times 4$ cube having all voxels of the same intensity value, which is commonly found in medical images, this method uses 3 bytes to represent 64 bytes of voxels. However, this cube can be encoded in a more compact form using only one byte representing the mean value of all voxels in the cube, thus leading to an improvement in CR. Secondly, although this method is applied to 8-bit volumes using two codebooks of 256 bytes in length, it can be extended for encoding a volume with other bit-width formats using two codebooks with lengths other than 256. Nowadays, most imaging modalities store image data in a 16-bit format, and for encoding 16-bit volume data, the two codebooks should be wider for better

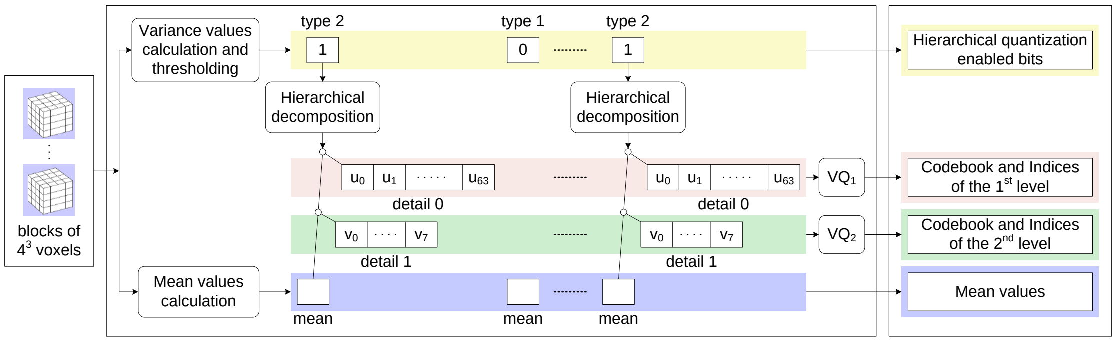

Variance values calculation and thresholding
type 2
type 1
type 2
1
0
1
Hierarchical decomposition
Hierarchical decomposition
u_0 u_1 u_{63}
detail 0
v_0 v_7
detail 1
VQ_1
VQ_2
blocks of 4^3 voxels
Mean values calculation
mean
mean
mean
Hierarchical quantization enabled bits
Codebook and Indices of the 1st level
Codebook and Indices of the 2nd level
Mean values

FIGURE 3.3: Enhanced 3-D image compression scheme.

fidelity.

The SW method is used in our 3-D compression algorithm with the following enhancements (Figure 3.3). In the first modification, we divide the partitioned cubes into two types based on their homogeneity. Type 1 refers to cubes with variance values smaller than a pre-defined threshold. A Type 1 cube is represented by only its mean value, which is more compact than decomposition and vector quantization. Type 2 cubes comprise the remaining cubes, which are encoded using the original scheme. For a cube, its type is denoted by a type bit (or HVQ-enabled bit) and encoded accordingly in the processing step. Since our new motion estimation method (Section 3.4) aims to produce cubes with minimum variance values, and due to the characteristics of 3-D medical images, there should be numerous Type 1 cubes in a volume dataset. This leads to an increase in CR. A second benefit is that only cubes having variance values greater than the threshold need to be quantized, thus considerably reducing the amount of computation and the quantization error (since the number of input vectors is decreased). This enhancement also has the potential to improve the fidelity of the reconstructed volume if an appropriate threshold is chosen. Furthermore, the decoding speed should be faster since no decompression is required for all Type 1 cubes; only the duplication of their mean values is performed.

The second modification to the SW method is that the lengths of the two codebooks are widened from 256 (or 8-bit index) up to 1024 (or 12-bit index) for a significant improvement in fidelity with a slight trade-off in CR. The experiments in Section 3.6 will demonstrate the efficiency of the enhanced compression algorithm.

It should be noted that the cube size of $4 \times 4 \times 4$ voxels was chosen for the balance between the CR, fidelity and processing time. There are possibilities of choosing other cube size, leading to the variation in these factors. However, this chapter does not cover analysis on the size of the partitioned cubes. In addition, although extra bits are needed to identify the two cube types, the overhead does not really affect the overall CR since the added storage is very small compared to the volume size (with the ratio of 1 bit to 64 or 128 bytes in most cases).

3.4 Three-Dimensional Motion Estimation and Compensation

In video coding, motion estimation and motion compensation are very important since they are used to reduce the temporal redundancy information between successive frames, thus improving CR. Among existing motion estimation methods, block matching algorithms (BMAs) are widely used because of their simplicity and effectiveness. Block matching aims to find, within a search window, the best-matched block from the previous frame based on a block distortion measure or other matching criteria. The displacement of the best-matched block is described as a motion vector relative to the block in the current frame. The algorithm that can find the best-matched block is the full search (FS) method, which evaluates all the candidate blocks within the search window. However, the high computational cost of FS limits its application in practice. In order to reduce the computational complexity, many fast BMAs have been proposed. Most fast BMAs are designed on the assumption that block distortion decreases monotonically when the search position moves toward the minimum distortion point. It is thus not necessary to check all the search points in the search window since the best-matched position can be found by following the changing trend of the distortion. Various search patterns have been used in BMAs to reduce the search points when finding the best-matched block, such as square patterns in the three-step search (TSS) (Koga et al., 1981), new three-step search (NTSS) (Li et al., 1994), four-step search (4SS) (Po and Ma, 1996), and block-based gradient descent search (BBGDS) (Liu and Feig, 1996); cross patterns in the cross search algorithm (CSA) (Ghanbari, 1990) and 2-D logarithmic search (TDLS) (Jain and Jain, 1981); diamond patterns in diamond search (DS) (Zhu and Ma, 1997); and hexagon patterns in hexagon-based search (HEXBS) (Zhu et al., 2002). More details of BMAs can be found in (Huang et al., 2006).

In 4-D image compression, motion estimation and compensation have been used for improving CR. In Guthe and Straβer (2001), the 3-D wavelet transform is used in combination with 3-D motion estimation and compensation to achieve high-rate compression of time-varying volumes. The 3-D extension of NTSS has been used with the 3-D integer wavelet transformation to compress 4-D images (Kassim et al., 2005). In Sanchez et al. (2008, 2009), 2-D motion estimation and compensation techniques of the most recent video coding standard H.264/AVC are adopted for the lossless compression of 4-D medical images. In this chapter, a new 3-D block matching motion estimation method named the *cross cube search* (CCS) is introduced. In this approach, we also describe a new block distortion measure named variance of residual (VOR) that can enhance the efficiency of the 3-D motion compensation step in our compression scheme. The experiments in Section 3.5 show that (1) our algorithm significantly reduces the computational complexity of the block matching motion estimation, and (2) using the proposed measure (VOR) instead of the widely used mean squared error (MSE) results in better peak signal to noise ratio (PSNR) performance.

3.4.1 Novel Block Distortion Measure

As implied from the compression scheme presented in Section 3.3, the distortion of the final volume after decompressing is mainly dependent on the distortion of the vector quantization step. Therefore, the fidelity of the final data is affected by the residual blocks (or cubes) that are the inputs to the vector quantizer. The residual blocks, which are the outputs of the motion estimation and compensation steps, are, in general, obtained from the block matching algorithm and the block distortion measure. In motion estimation for video coding, the most widely used block distortion measure is the MSE. However, this measure may not best suitable for the proposed system due to the fact that a small value of MSE calculated from the block matching step does not guarantee a small distortion of the vector quantization in the next step. We introduced a new distortion measure named the variance of residual (VOR) which can be used to obtain a smaller distortion of the vector quantization compared to using MSE. The proposed measure is the variance value of all voxel intensities in the residual cube. The VOR for an $L \times M \times N$-sample block is defined by

$$\text{VOR} = \frac{1}{LMN} \sum_{i=0}^{L-1} \sum_{j=0}^{M-1} \sum_{k=0}^{N-1} (P_{ijk} - Q_{ijk})^2 - \mu^2 \tag{3.1}$$

where P_{ijk} is a sample of the current block, Q_{ijk} is a sample of the reference area, and μ, the mean of all voxel intensities in the residual block, is calculated by

$$\mu = \frac{1}{LMN} \sum_{i=0}^{L-1} \sum_{j=0}^{M-1} \sum_{k=0}^{N-1} (P_{ijk} - Q_{ijk}). \tag{3.2}$$

VOR is clearly a measure of the homogeneity of the residual block. A small value of VOR means that the corresponding residual block has small changes in voxel intensity. This results in the nearly identical elements of the input vector of the quantizer. Quantizing multi-dimensional vectors in which the vector elements are nearly identical often produces a small distortion. Other widely used measures, e.g., MSE, are not suitable for this compression scheme since they do not assure the small changes in voxel intensity in residual blocks. This argument will be demonstrated in Table 3.2 and Figure 3.11 in Section 3.5.

3.4.2 Novel 3-D Motion Estimation Algorithms

To study the characteristics of motion vector probability (MVP) distribution, we applied motion estimation using FS on four 4-D medical image datasets, described in Section 3.5, to obtain the optimal motion vectors. VOR is employed as the block distortion measure with block size of $4 \times 4 \times 4$ and the search window size of $15 \times 15 \times 15$ voxels. This means that each component of the motion vector ranges from -7 to $+7$. Figure 3.4 depicts the 3-D graphs of the MVP distribution corresponding to difference values of the z-component, d_z. The graphs associated with $|d_z| > 2$ are not shown since the MVPs are relatively small. As seen in Figure 3.4, the motion vectors are center-biased. Furthermore, the probability of the motion vector in the axis-aligned directions is larger than the other directions. Based on this cross center-biased characteristic of the MVP distribution, the CCS algorithm uses four search patterns: large cube search pattern (LCSP), medium cube search pattern (MCSP), small cube search pattern (SCSP) and cross search pattern (CSP) (Figure 3.5).

The CCS algorithm comprises the following steps:

- Step 1: Apply LCSP and CSP at the center of the search window. Evaluate the matching distortion of all search points in these two search patterns. If the minimum matching distortion point (MMDP) occurs at the center, the search process stops and the motion vector is found at the center. Otherwise, put the center of the search pattern on the current MMDP and go to Step 2a if MMDP is found in LCSP, or Step 2b if MMDP is found in CSP.

- Step 2a: Use MCSP to find the new MMDP. Move the center of the search window to this MMDP, and then go to Step 3.

- Step 2b: Repeat using CSP to find the new MMDP until the MMDP occurs at the center point of the search pattern. The search process stops when the motion vector is found at the last MMDP.

- Step 3: Apply SCSP to find the new MMDP which is the position of the motion vector.

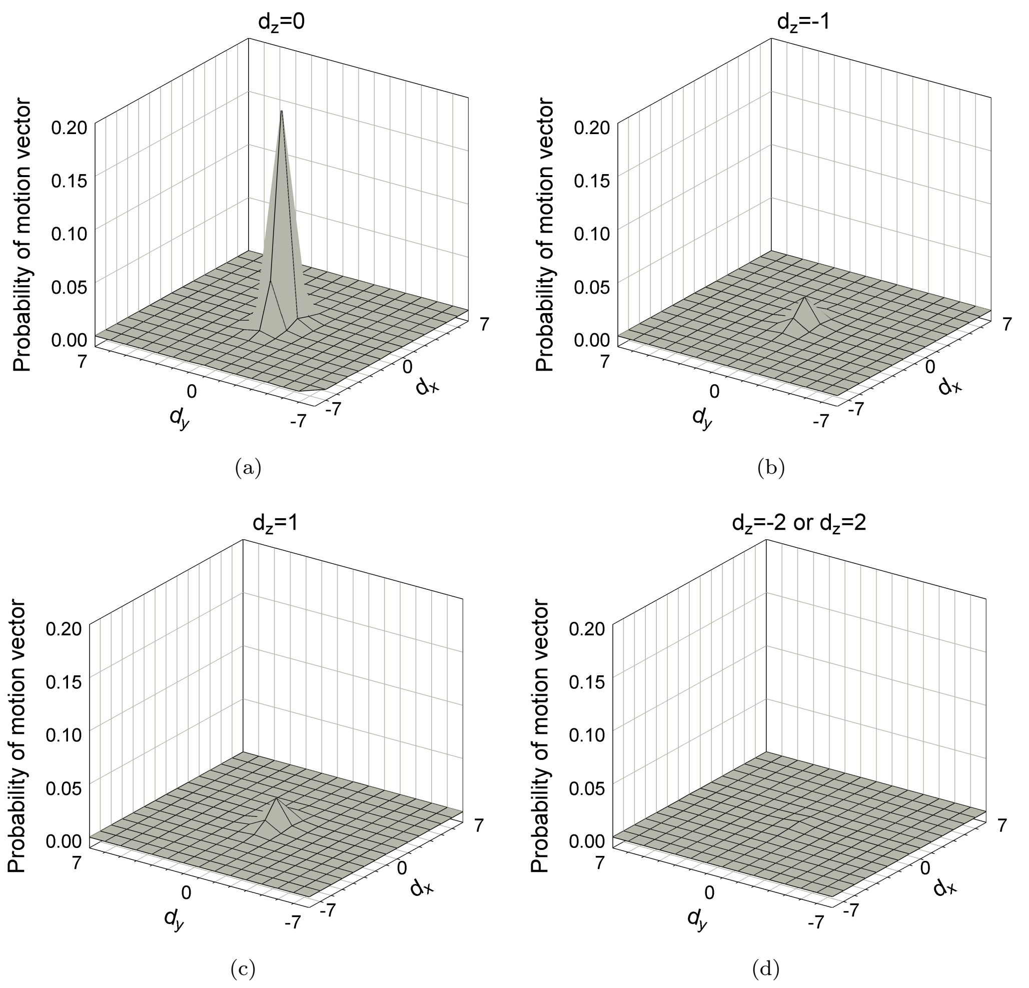
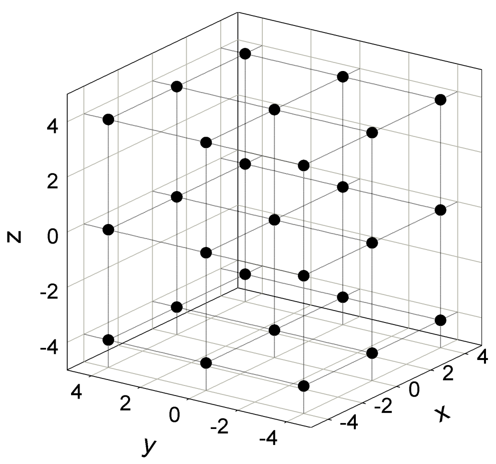
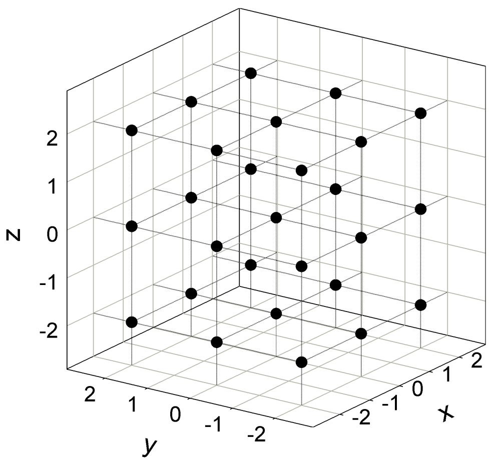


FIGURE 3.4: Motion vector probability distribution with different values of d_z. (a) $d_z = 0$, (b) $d_z = -1$, (c) $d_z = +1$, and (d) $|d_z| = 2$.

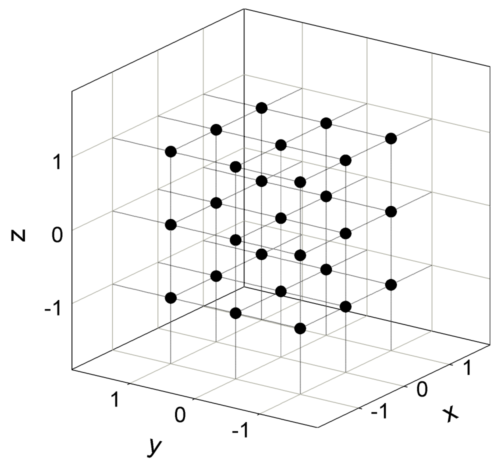
In addition, two well-known block-matching algorithms in video coding, DS and HEXBS, are extended from 2-D to 3-D for comparing their performance with the proposed method. These two algorithms are based on the MVP distribution and are noted for their computational efficiency. The 3-D extension of the DS algorithm, which is called *octahedron search* (OS), uses two search patterns (Figure 3.6). The large octahedron search pattern (LOSP) contains 19 check points while the small octahedron search pattern (SOSP) comprises 7 check points. During the search process, LOSP is repeatedly used until MMDP occurs at the center



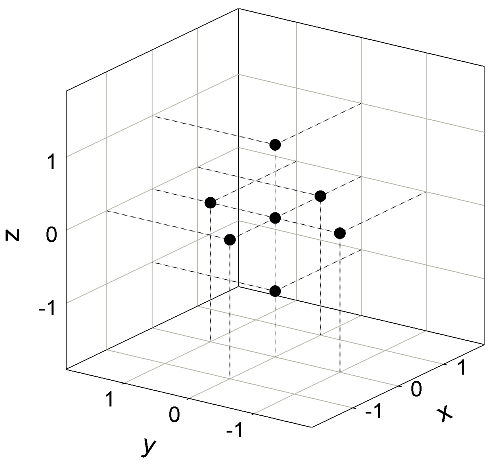
(a) LCSP



(b) MCSP



(c) SCSP

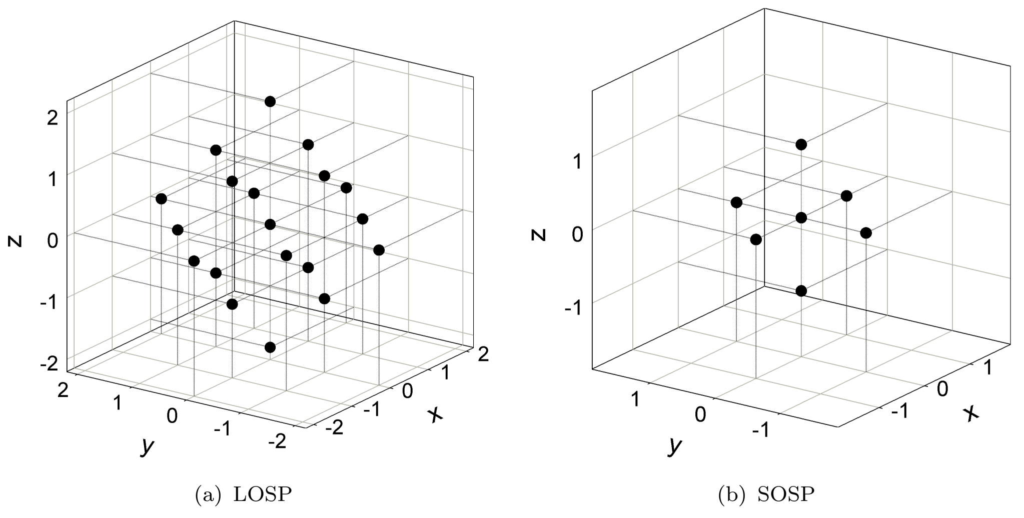


(d) CSP

FIGURE 3.5: CCS search patterns. (a) LCSP, (b) MCSP, (c) SCSP, and (d) CSP.

point. The search pattern is then switched from LOSP to SOSP. The motion vector is formed based on the position of the point yielding the minimum matching distortion among the seven check points in SOSP.

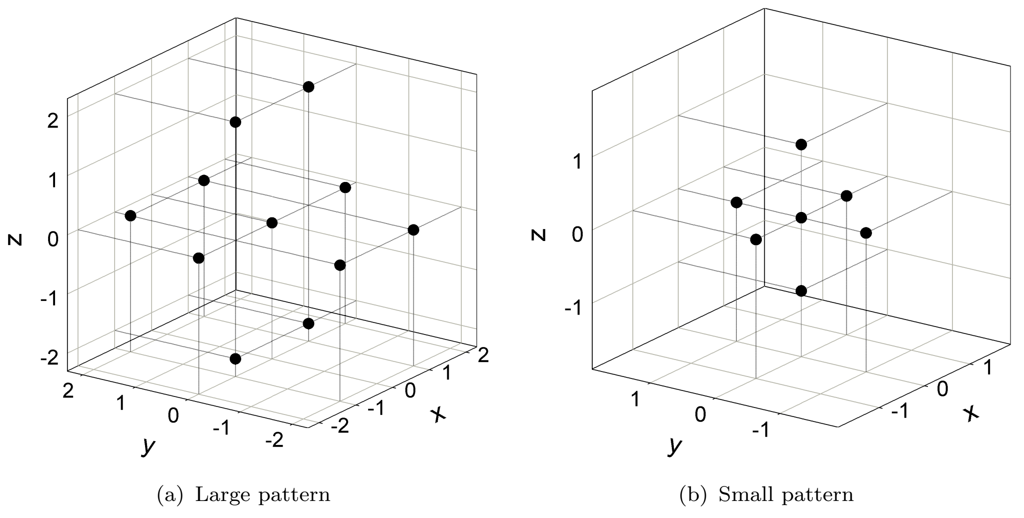
3-D HEXBS, the 3-D version of HEXBS, uses the two search patterns shown in Figure 3.7. The large search pattern (LSP) consists of 11 check points, while the small search pattern (SSP) is actually SOSP in OS. In the first step, LSP is used to find the MMDP. If the MMDP is not located at the center point of LSP, the search



(a) LOSP

(b) SOSP

FIGURE 3.6: OS search patterns. (a) LOSP and (b) SOSP.



(a) Large pattern

(b) Small pattern

FIGURE 3.7: 3-D HEXSB search patterns. (a) large pattern and (b) small pattern.

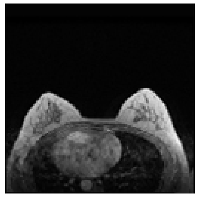
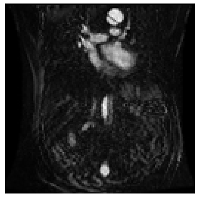
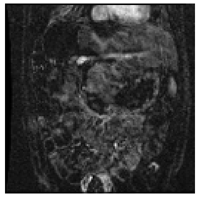
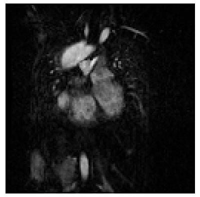
process is repeated by centering LSP at the position of the MMDP. Otherwise, it goes to the last step, where SSP is applied to find the last MMDP, which is the position of the motion vector.

3.5 Experiments

In this section, we describe the experiments used to evaluate the performance of the proposed method, which we term the enhanced Schneider and Westermann method with motion compensation (ESW-MC). The test data comprise one magnetic resonance (MR) 8-bit and three 16-bit time-varying volume datasets which were acquired at the National University Hospital, Singapore, and one 12-bit computed tomography (CT) time-varying volume dataset from the University Hospital of Geneva, Switzerland (Table 3.1).

The raw image data and their descriptions are extracted from the DICOM datasets to form the input data of the system. In the encoding process, the number of bytes allocated for representing a cube depends on the bit-width of the dataset, the type of the cube and the existence of motion estimation in the corresponding frame. The size of the search window in our experiments is $15 \times 15 \times 15$ voxels. Each component of a motion vector ranges from -7 to $+7$ and is represented as a signed 4-bit integer, forming the 12-bit motion vector. For Type 1 cubes, their mean values are represented by 8-, 12-, or 16-bit voxels corresponding to dataset bit-widths of 8, 12, or 16, respectively. For Type 2 cubes, the sizes of the two codebooks used to quantize level 1 and level 2 data depend on the dataset bit-width and whether motion estimation is applied in the corresponding frame. Figure 3.8 summarizes the representation format of a cube. The codebook sizes and the cube representation format are chosen so that the bits allocated to represent a cube are byte-aligned. This leads to a fast and relatively straight forward implementation of data decompression. Other possible configurations may lead to the variation in CR, fidelity and decompression time of the compressed data. However, the analysis on alternate configuration is not within the scope of this chapter.

TABLE 3.1: Dataset specifications

Dataset	BREAST	AORTA	ABDOMEN	THORAX	HEART
					
Bits allocated	8	16	16	16	12
Rows × Columns	256 × 256	512 × 512	512 × 512	512 × 512	512 × 512
Slices	26	160	120	136	188
Time steps	5	14	12	16	20
Pixel size (mm)	1.25 × 1.25	0.9375 × 0.9375	0.4688 × 0.4688	0.6445 × 0.6445	0.4883 × 0.4883
Inter-slice spacing (mm)	4.2	1.1	0.8	0.9	0.75
Size (MB)	8	1120	720	1088	1880
Modality	MRI	MRA	MRA	MRA	CT

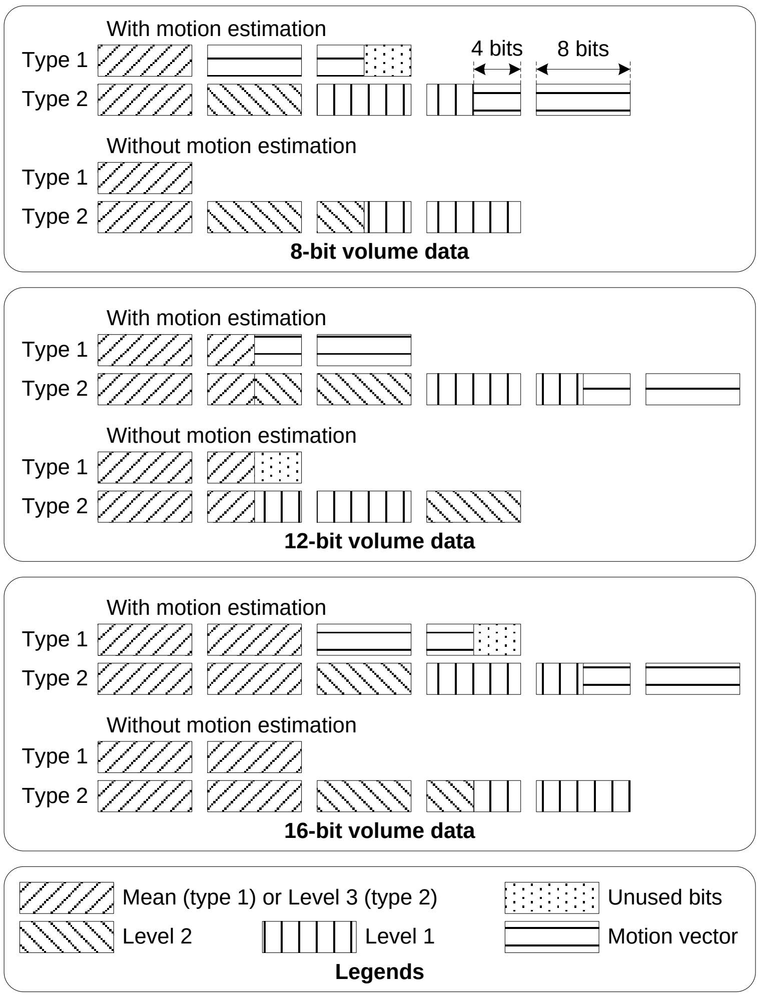


FIGURE 3.8: Bits allocated to represent a cube.

Five experiments were performed to evaluate the efficiency of the method. The computing platform was a 2.5 GHz Intel Core 2 Duo laptop equipped with 4 GB RAM. Each test was run at least three times and the execution times averaged. The fidelity of the decompressed data was measured by their PSNRs. The refinement step in the vector quantization process was included only in the last experiment so that we could evaluate its effect on algorithm performance separately from the use of motion compensation and cube classification.

Experiment 1

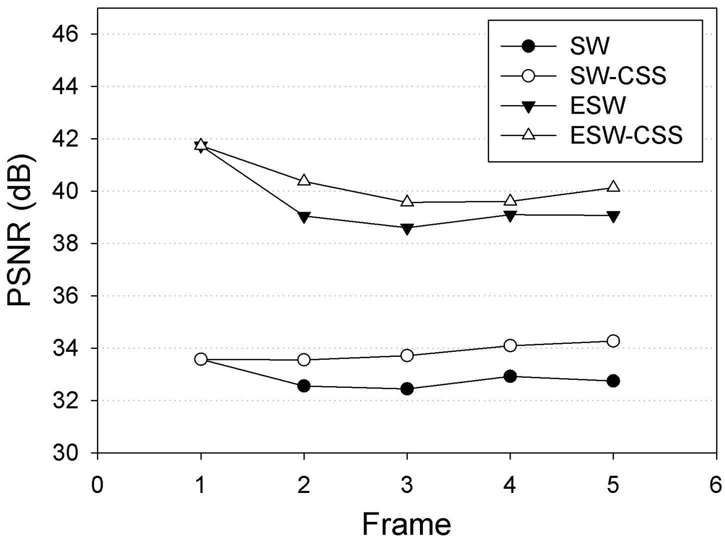
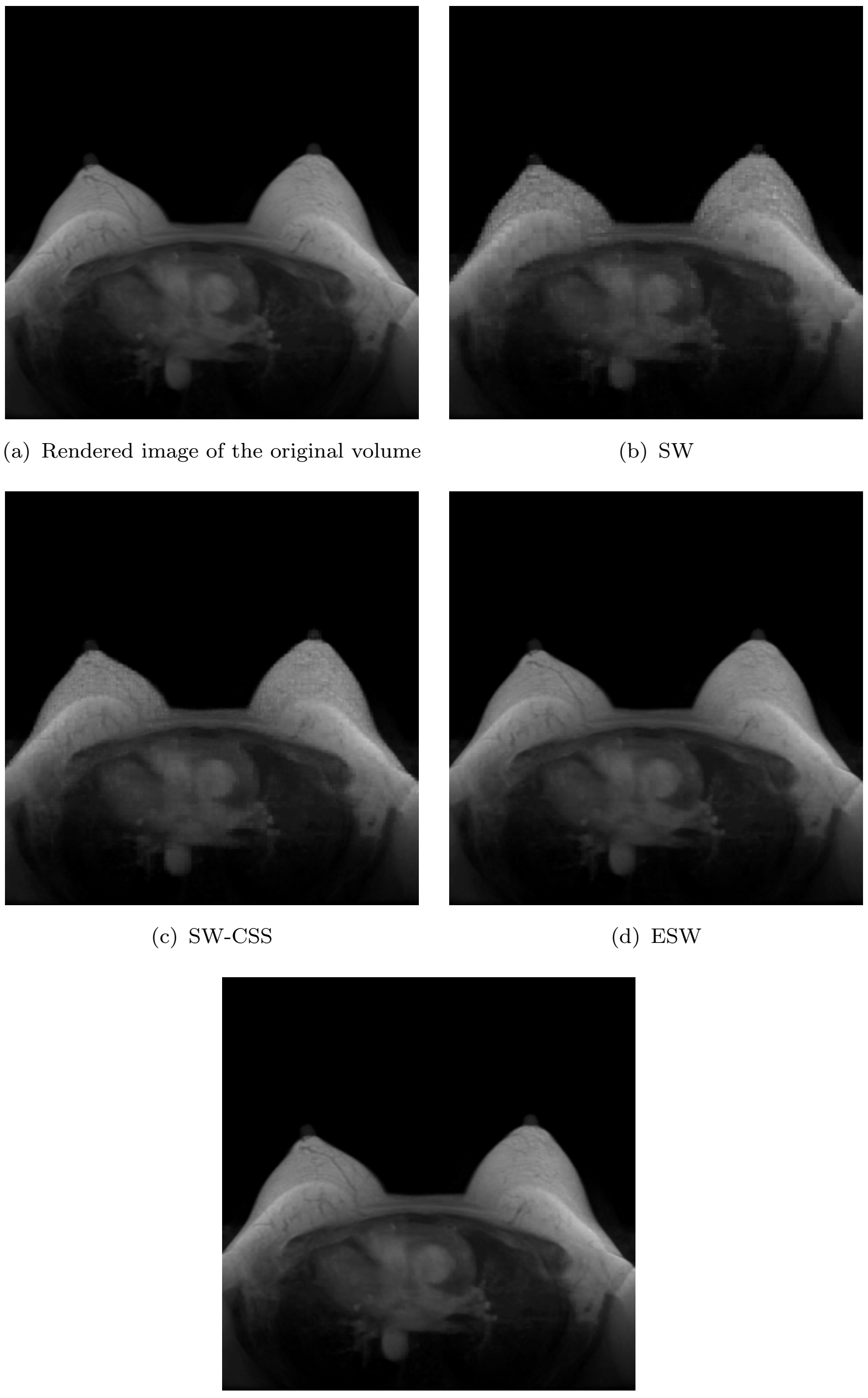


FIGURE 3.9: Test result on BREAST dataset. The compression ratios of the respective methods are 21.33, 14.22, 16.00, and 12.80. The corresponding average processing times per frame are 8, 9, 14, and 15 s, respectively.

The objective was to compare our method with the SW method. Since the latter is designed for 8-bit data, we used the BREAST dataset. The dataset was encoded using four compression schemes: (1) SW and (2) SW-CSS, which refer to the SW method without and with motion compensation using CSS, respectively; and (3) ESW and (4) ESW-CSS, which refer to our compression method without and with motion compensation using CSS, respectively. In addition, the vector quantization step in schemes (1) and (2) used two 8-bit codebooks and a 5-iteration refinement step to encode the data. The VOR measurement is used in the motion estimation process in schemes (2) and (4). The results of this experiment are presented in Figures 3.9 and 3.10.

Experiment 2

The aim was to demonstrate of the effectiveness of using motion compensation in the proposed compression scheme and the effectiveness of the new block distortion measure, VOR. The datasets used were AORTA, ABDOMEN, THORAX, and


(a) Rendered image of the original volume

(b) SW

(c) SW-CSS

(d) ESW

(e) ESW-CSS

FIGURE 3.10: 2-D texture mapping rendering of the second frame in BREAST dataset encoded using different methods.

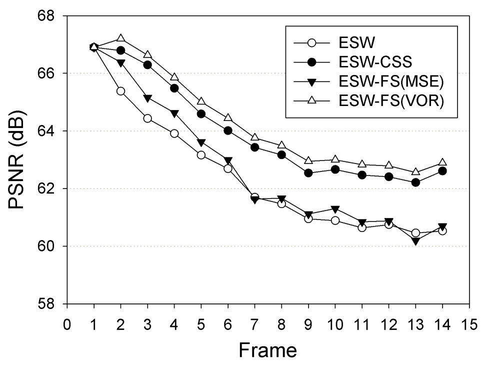
TABLE 3.2: Experiment 2 results: compression ratio, processing time and average PSNR

Dataset	Method	Ratio	Time (s)	Avg. PSNR (dB)
AORTA	ESW	23.58	48	62.42
	ESW-CSS	18.66	63	63.97
	ESW-FS(MSE)	18.66	793	62.72
	ESW-FS(VOR)	18.66	889	64.31
ABDOMEN	ESW	20.24	47	60.34
	ESW-CSS	17.95	51	61.80
	ESW-FS(MSE)	17.95	594	60.35
	ESW-FS(VOR)	17.95	656	61.94
THORAX	ESW	22.22	49	61.17
	ESW-CSS	18.25	64	62.17
	ESW-FS(MSE)	18.25	643	61.03
	ESW-FS(VOR)	18.25	767	62.14
HEART	ESW	26.60	62	40.68
	ESW-CSS	18.97	84	44.35
	ESW-FS(MSE)	18.97	907	44.14
	ESW-FS(VOR)	18.97	1157	44.68

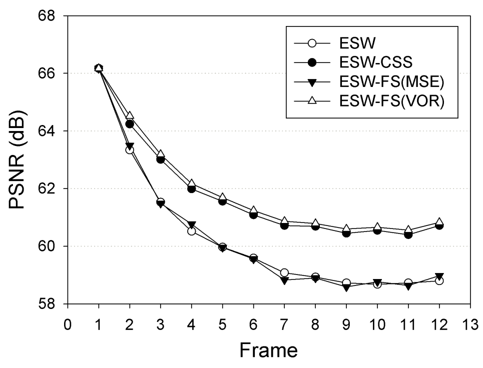
HEART. This experiment comprised four compression schemes: (1) ESW and (2) ESW-CSS, which are the corresponding compression schemes in Experiment 1; and (3) ESW-FS(MSE) and (4) ESW-FS(VOR), which refer to the use of motion estimation and compensation using FS with MSE and VOR, respectively. Only the first frame in each dataset was chosen as the key frame and no threshold was applied in the 3-D compression phase, meaning that all the partitioned cubes were Type 2 cubes. The results of this experiment are shown in Table 3.2 and Figure 3.11.

Experiment 3

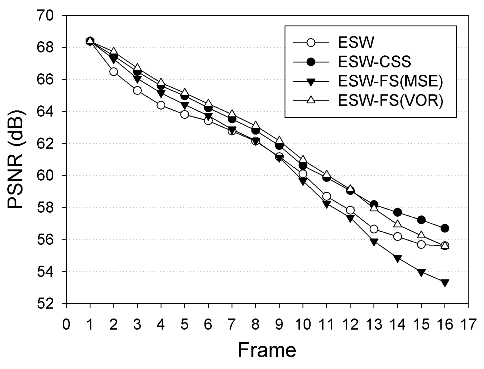
This experiment aimed to compare the three proposed 3-D motion estimation algorithms, CCS, OS, and 3-D HEXBS, with FS and the 3-D NTSS method (Kassim et al., 2005) in terms of speed and fidelity of the decompressed data. The new



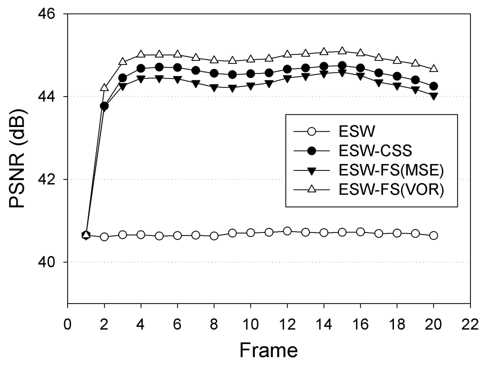
(a) AORTA



(b) ABDOMEN



(c) THORAX



(d) HEART

FIGURE 3.11: Variation in PSNR *versus* time step when applying the four compression schemes on various datasets. (a) AORTA, (b) ABDOMEN, (c) THORAX, and (d) HEART.

block distortion measure, VOR, was used in all these motion estimation methods. The results of this experiment on our 12- and 16-bit datasets are presented in Table 3.3.

Experiment 4

Our 3-D compression algorithm uses a threshold to classify a partitioned cube into two types associated with two different coding processes. This experiment analyzes the effect of this threshold value on the processing speed, the CR, and the fidelity of the compressed data using our 12- and 16-bit datasets. Since the homogeneity of a key frame may be different from that of an intermediate frame, there should

TABLE 3.3: Comparison of motion estimation methods

Dataset	Method	Avg. number of search points	Speed-up	Avg. PSNR (dB)
AORTA	FS	3194.89	1	64.31
	3-D NTSS	60.67	52.66	63.94
	3-D HEXBS	19.57	163.29	63.59
	OS	31.57	101.19	63.74
	CCS	43.42	73.58	63.97
ABDOMEN	FS	3158.16	1	61.94
	3-D NTSS	59.18	53.36	61.75
	3-D HEXBS	18.59	169.88	61.56
	OS	29.83	105.86	61.61
	CCS	40.76	77.49	61.80
THORAX	FS	3175.44	1	62.14
	3-D NTSS	60.06	52.87	62.12
	3-D HEXBS	19.10	166.29	62.05
	OS	30.69	103.46	62.14
	CCS	42.47	74.77	62.17
HEART	FS	3211.29	1	44.68
	3-D NTSS	59.08	54.35	44.32
	3-D HEXBS	18.58	172.84	44.07
	OS	29.37	109.34	44.10
	CCS	41.41	77.55	44.35

be two different threshold values, θ_1 for key frames and θ_2 for intermediate frames; thus, two tests were executed. In the first test, for each value of θ_1, the first four frames of each dataset were encoded using our 3-D compression algorithm without motion estimation (ESW) and the average PSNR was calculated. The results of this test are shown in Figure 3.12. The value of θ_1 yielding the highest average PSNR was chosen for the second test when compressing the first frame as a key frame and the next three frames as intermediate frames. These intermediate frames were encoded using CCS motion estimation with the varying threshold value θ_2. The processing time, CR and PSNR of these three frames were accumulated and averaged. Figure 3.13 presents the results of the second test.

Experiment 5

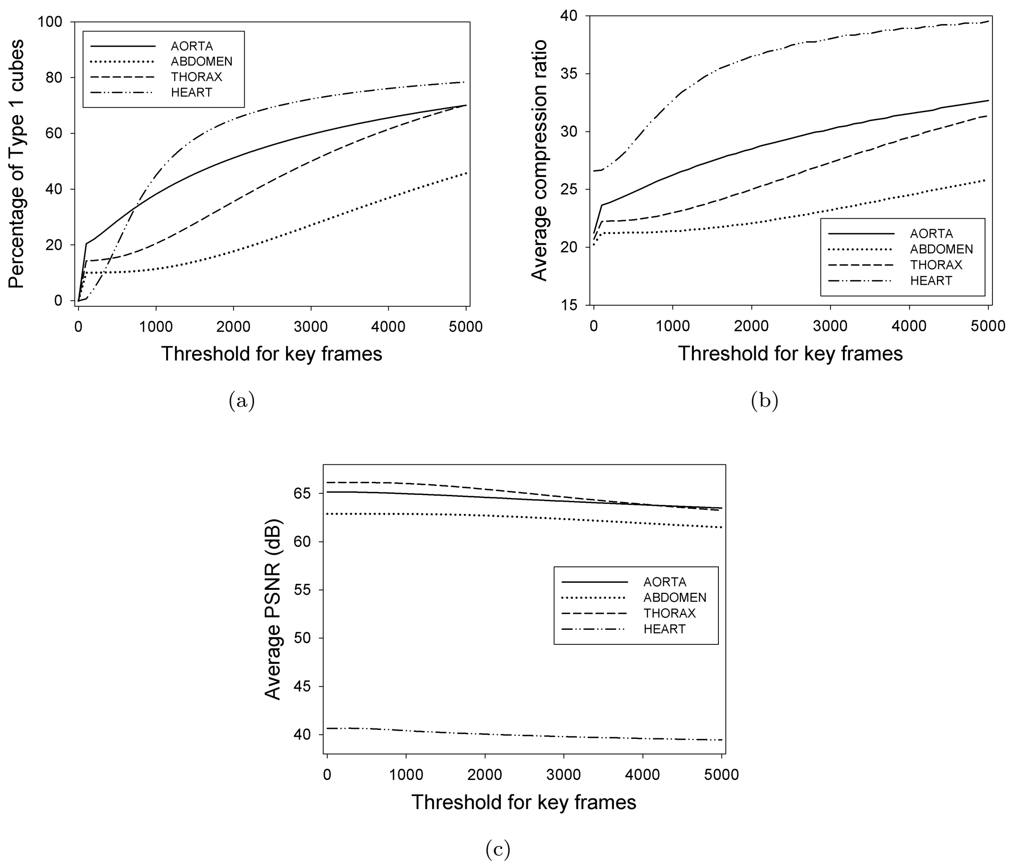
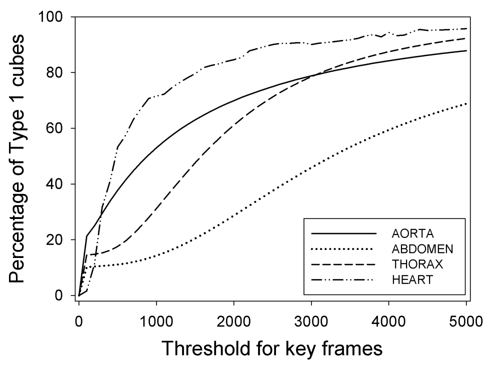


FIGURE 3.12: Effect of threshold value θ_1 on the average of: (a) percentage of type 1-cubes, (b) compression ratio, and (c) average PSNR of the first 4 key frames in our 12-bit and 16-bit datasets.

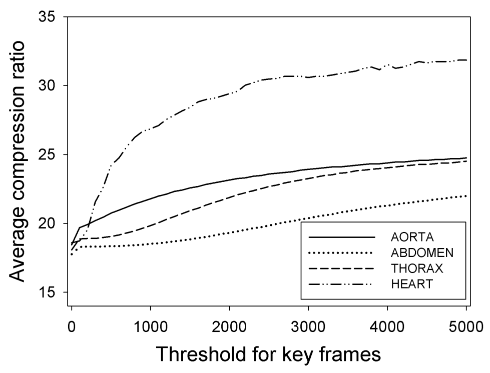
The last experiment was performed to evaluate the use of the refinement step in vector quantization in our method. Several tests without refinement and with the number of refinement iterations ranging from 1 to 3 were executed. In these tests, θ_1 and θ_2 were set at 100 and 300, respectively. The results of this experiment are shown in Figure 3.14.

3.6 Results and Discussion

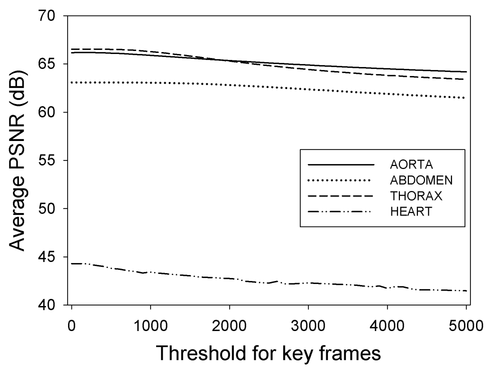
Experiment 1



(a)



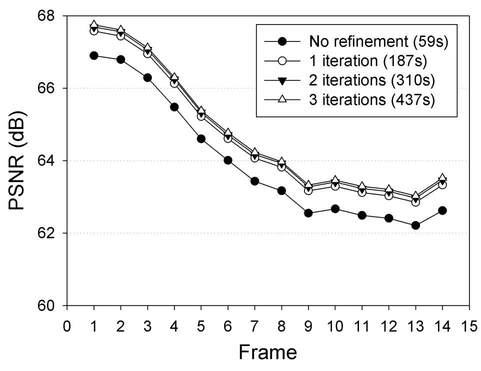
(b)



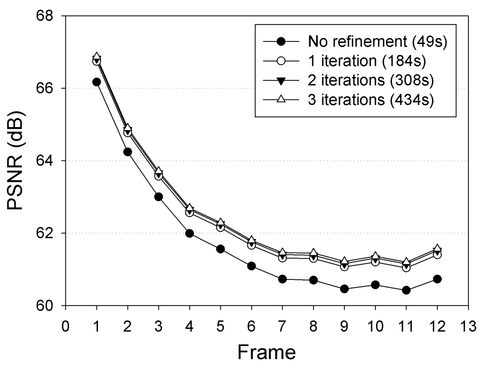
(c)

FIGURE 3.13: Effect of threshold value θ_2 on the average of: (a) percentage of type 2-cubes, (b) compression ratio, and (c) average PSNR of the first 3 intermediate frames in our 12-bit and 16-bit datasets. The threshold for the first (key) frame was set at 100 for all the datasets.

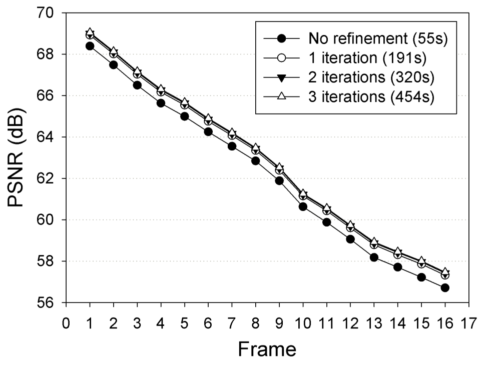
In the first experiment, the first frame was selected as a key frame for compression schemes using motion estimation. Figure 3.9 shows the line graph of the PSNR values between the encoded frame and the corresponding original raw frame. Due to the relatively small size of the BREAST dataset, the compressed file size used to calculate CR in this experiment did not include the sizes of the two codebooks and other information in the file header. The results show that widening the bit-width of the codebook index from 8 bits in SW to 12 bits in ESW significantly improved the fidelity of the encoded frame (by 6.1–8.2 dB) with a slight trade-off in processing speed and CR. Furthermore, the motion compensation process also



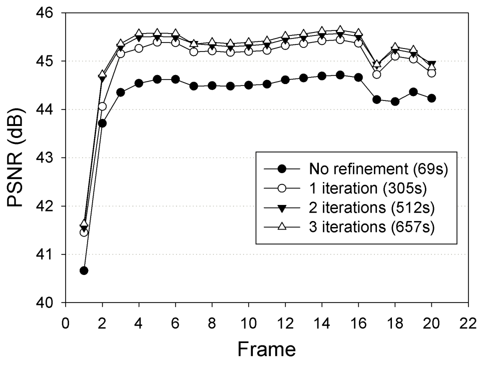
(a) AORTA



(b) ABDOMEN



(c) THORAX



(d) HEART

FIGURE 3.14: Variation in PSNR *versus* time step when applying the different number of refinement iterations on: (a) AORTA, (b) ABDOMEN, (c) THORAX, and (d) HEART dataset. The average encoding times per frame are indicated in the legend.

appreciably contributed to the quality improvement of each compressed frame (an increase of up to 1.5 dB for SW and 1.3 dB for ESW), as can be clearly seen in Figure 3.10, which presents the rendered images of the second frame in the BREAST dataset with different compression schemes.

Experiment 2

From Figure 3.11, it is clear that using motion estimation and compensation with the new block distortion measure, VOR, can significantly improve the fidelity of the encoded frame. Using VOR with FS, the improvement in PSNR ranged from

0.97 dB to 4.00 dB, depending on the dataset (Table 3.2). However, using FS dramatically slowed down the processing speed (from about 14 to 18 times). It took about 8–19 minutes to encode a frame using motion estimation with FS. In this case, replacing FS with CSS was a better choice since it still produced a large improvement in fidelity (1.00–3.67 dB) with a slight trade-off in speed (only 1.08–1.35 times slower). Using MSE as the block distortion measure was not suitable for the proposed compression scheme since fidelity was not improved even though the FS algorithm was used.

Experiment 3

It can be seen from Table 3.3 that the proposed 3-D motion estimation algorithm CCS produced a fidelity almost equivalent to FS. However, CSS ran faster than FS by a factor of almost 80. CSS was approximately 1.5 times faster while still maintaining a higher PSNR than 3-D NTSS. In addition, OS and 3-D HEXBS, our extensions of the DS and HEXBS algorithms, are good choices for fast motion estimation process since they could run from 2 to 3 times faster with hardly noticeable fidelity loss compared to 3-D NTSS.

Experiment 4

As shown in Figure 3.12, when the threshold value θ_1 increased, the average CR noticeably increased due to the increase of Type 1 cubes in each dataset. The average PSNR was almost unchanged for $\theta_1 \leq 500$ before decreasing. This trend is also observed in Figure 3.13 with the variation of θ_2. When the threshold increased, in addition to the improvement of CR, the encoding and decoding process should be faster since the number of Type 1 cubes increased (and processing Type 1 cubes is faster than processing Type 2). Particularly, when $\theta_1 = 100$, the encoding process for key frames was 12.06–24.64% faster, the CR increased with the largest increment from 19.65:1 to 23.36:1 while the PSNR remained at the same level, as compared to the setting with no threshold. When θ_2 was set at 300, the fidelity was still optimal, but the encoding speed increased by up to 22.82% and CR also increased with the largest change from 17.39:1 to 19.34:1. Ignoring the time taken for loading data to the memory and storing data, it took about 770 ms for decompressing one key frame of AORTA, our largest 16-bit dataset, when θ_1 was 0. When θ_1 was set at 200 and 500, the average decoding time was 670 and 630 ms, respectively. Decompressing intermediate frames was approximately 1.6 times slower since motion compensation needed to be executed. This is significantly faster than that of other compression methods.

Experiment 5

From Figure 3.14, we see that the fidelity of the encoded data increased considerably when the refinement step was included. The most significant increase in fidelity occurred after the first refinement iteration, with average PSNR increasing from about 0.24 dB to 0.80 dB. From the second iteration onwards, the increment sharply decreased and stabilized after several iterations. The average refinement processing times for one frame in the AORTA, ABDOMEN, THORAX, and HEART datasets were 128, 135, 136, and 260 s, respectively. It took from 2 to 3 times longer than the time spent to encode one frame. The number of iterations to be used with the refinement step is based on the dataset size and the expected processing time. In most cases, encoding a dataset using the 1-iteration refinement step is a good choice.

Discussion

Since level 3 data can be considered shrank representation of the original volume, redundancy would be found in the resulting bit stream. Therefore, an extra gain to CR can be achieved by applying any lossless data compression method to the resulting bit stream. In order to analyze this, all the compressed frames using ESW-CSS in Experiment 2 were encoded using the arithmetic coding method introduced in Said (2003). The new average CR obtained for the AORTA, ABDOMEN, THORAX, and HEART datasets were 23.33, 22.26, 22.37, and 23.65, respectively. The average of extra compression and decompression time due to the arithmetic coding step ranged from 100 to 200 ms per frame, depending on the dataset. These overheads are relatively small.

Table 3.4 lists a cursory comparison of the proposed scheme with existing 4-D lossy compression methods. It should be noted that CR, image fidelity and processing time are not measured under the same conditions. However, based on the experimental results and the information from Table 3.4, it is seen that good image fidelity and fast decompression are two advantages of the proposed method. While the CR of our method is not the best compared to that of methods based on wavelet transform, e.g., 3-D JPEG2000, it can be considered to be at a high level. At the same level of CR, our method may achieve a comparable or even better PSNR. Especially, the method offers a very fast decompression speed thanks to the simple decoding. Therefore, the proposed method is suitable for numerous applications, particularly time-critical applications dealing with massive data.

The proposed method has two other interesting features. First, the classification of the partitioned cubes into two types can increase CR and the encoding and decoding speed with a slight loss in fidelity. Currently, this process is automatically done using the two thresholds θ_1 and θ_2. However, we can manually classify the cubes based on their homogeneity and input regions of interest (ROIs). Cubes with low homogeneity belonging to ROIs should be encoded using HVQ. Each of the remaining cubes containing less important information or having high homogeneity

Table 3.4: Comparison of lossy 4-D medical image compression methods

Method	Computing platform	Typical dataset	Performance	Processing time
ESW-MC	2.5GHz Intel Core 2 Duo, 4GB RAM	$512 \times 512 \times 160 \times 14$, 16-bit MRA, 1.1 GB (AORTA)	PSNR = 64 to 65 dB (CR = 18 to 32)	Encoding: 60s/frame, decoding (no I/O access): 670 to 900ms/frame
4-D DWT and 4-D EZW (Zeng et al., 2002)	Pentium III 550 MHz	$240 \times 126 \times 32 \times 32$, float, 210 MB	PSNR = 37.87 dB (CR = 32), PSNR = 37.49 dB (CR = 64)	Encoding: 266.1 s, decoding: 230.5 s
4-D JPEG2000 (Lalgudi et al., 2005b)	not reported	$64 \times 64 \times 21 \times 100$, 13-bit fMRI, 16 MB (mb01)	PSNR ≈ 57.5 dB @ 0.5 bpp (CR = 26), PSNR ≈ 60.0 dB @ 1.0 bpp (CR = 13)	not reported
3-D JPEG2000 (xyt) (Lalgudi et al., 2005b)	not reported	$64 \times 64 \times 21 \times 100$, 13-bit fMRI, 16 MB (mb01)	PSNR ≈ 57.0 dB @ 0.5 bpp (CR = 26), PSNR ≈ 59.5 dB @ 1.0 bpp (CR = 13)	not reported
3-D JPEG2000 (xyz) (Lalgudi et al., 2005b)	not reported	$64 \times 64 \times 21 \times 100$, 13-bit fMRI, 16 MB (mb01)	PSNR ≈ 52.0 dB @ 0.5 bpp (CR = 13)	not reported
4-D SPIHT (Lalgudi et al., 2005a)	not reported	$64 \times 64 \times 21 \times 100$, 13-bit fMRI, 16 MB (mb01)	PSNR ≈ 57 dB @ 0.5 bpp (CR = 26), PSNR ≈ 59 dB @ 1.0 bpp (CR = 13)	not reported
4-D SBHP (Liu and Pearlman, 2007)	not reported	$64 \times 64 \times 21 \times 100$, 13-bit fMRI, 16 MB (mb01)	SNR ≈ 22 dB @ 0.5 bpp (CR = 26), SNR ≈ 27 dB @ 1.0 bpp (CR = 13)	not reported
3-D IWT and 3-D NTSS (Kassim et al., 2005)	Pentium 4 2.4 GHz	$128 \times 128 \times 107 \times 15$, 16-bit CT, 50 MB (Sequence A)	PSNR = 40 to 42 dB @ 1.0 bpp (CR = 16)	8 s for encoding-decoding each frame

bbp: bits per pixel

is represented by the mean value of the its voxels. This strategy can significantly improve CR and processing speed while preserving information of interest. Second, this method is a block-based coding method. In order to decode one block, we only need the two codebooks of the corresponding frame and the content of one block in the previous frame together with a motion vector in the event that the corresponding frame is an intermediate frame, and that the previous frame has already been decoded. Therefore, we can decompress any slice or any block in one frame without decoding the frame. This will be useful when we wish to obtain an overview of an encoded dataset.

The proposed method has two disadvantages. First, due to the complexity of vector quantization based methods, the proposed scheme may require a relatively long time for compressing a large dataset. Nevertheless, this drawback should not be an issue since the encoding phase is to be done only once, and using a powerful computer will help to hasten the process. Second, although using motion compensation in the proposed method can significantly improve the fidelity of the encoded data, it slows down the decoding process in the random frame access mode. This is due to the fact that to decompress one frame, a number of previous frames from the nearest key frame have to be decoded if their contents are not ready. However, this drawback does not appear in the data browsing mode. This limitation can also be reduced by setting a small number of intermediate frames to be encoded in one group and using cache memory to store neighboring frames which have already been decoded. Another possible solution is to use parallel processing to concurrentize the data preparation and data decompression processes (Nagayasu et al., 2008).

3.7 Summary

In this chapter, we have introduced an efficient compression scheme, which can be applied to numerous applications, particularly the compression of 4-D medical images. The scheme uses 3-D motion estimation to create a homogenous preprocessed data to be effectively compressed by a 3-D image compression algorithm using hierarchical vector quantization. A new block distortion measure, the *variance of residual*, and three 3-D fast block matching algorithms are developed for improving the motion estimation process in terms of speed and homogeneity of the preprocessed data. The 3-D image compression algorithm is designed to be suitable with the input data by using two different encoding techniques: representing high homogeneity partitioned cubes by the mean values of their voxels, and using hierarchical vector quantization to compress homogenous cubes. These two techniques can increase the processing speed and CR while maintaining the fidelity of the compressed data. The experimental results on typical 4-D medical images showed that our method can achieve a higher fidelity and faster decompression time compared to other lossy compression methods producing similar CRs. The combination of motion compensation and hierarchical vector quantization is the key factor for the good performance.

Chapter 4

Transfer Function Design for Medical Visualization

Direct volume rendering is a powerful technique in scientific visualization, especially in diagnostic imaging where images are volumetric data generated by CT and MR scanners. Volume rendering is a method that could overcome the lack of spatial perception in the traditional 2D slice-by-slice viewing. However, the efficiency of volume rendering strongly depends on the transfer function (TF), which is a mapping from data properties (e.g., scalar value and gradient magnitude) to optical properties (e.g., opacity and color). Although a good TF can reveal the important structures in the data, finding an appropriate mapping that yields the desired visual appearance is not a trivial task. It requires an understanding of the TF domain and manually tweaking parameters on the part of the user.

LH space, which describes voxels based on the two materials forming the boundary, has recently been proposed as a feature domain in TF design (Šereda et al., 2006a,b). Compared to the more conventional TF domains of scalar value and its derivatives, the LH histogram, a 2-D representation of the LH space, represents boundaries more compactly and robustly. Moreover, LH space can provide an unambiguous classification of boundaries with distinct LH values. Therefore, applying a clustering technique to the LH space is more suitable than the traditional intensity-gradient space in which boundaries appear as arches that tend to overlap.

The system introduced in this chapter incorporates multiple clustering steps to automatically identify the material boundaries in volumetric data (Nguyen et al., 2011b, 2012). TFs can then be applied to each material boundary using an automated TF design module. The system allows users to interactively select the number of clusters and to modify the TFs assigned to each cluster. Further changes to the visualization output can also be achieved by changing the clustering parameters. The proposed system can significantly reduce the time and effort required to obtain good TFs for volume rendering and enable visualizations with quality approaching that of existing methods to be automatically generated.

4.1 Related Methods

Interactive TF design has been addressed through many different studies. Due to the ease of implementation and parallelization, most approaches employ derivatives to design TFs. Kindlmann and Durkin (1998) used the first derivative (i.e., gradient) as an attribute to generate multi-dimensional TFs. In the 2-D TF domain, which incorporates the intensity and gradient magnitude, material boundaries can be interpreted as arches. Thus, they can be selected and visualized by manipulating certain TF widgets to approximate the arches. However, these arches often overlap, which prevents proper isolation of a material from others. One possible approach to overcome this drawback is to include the second directional derivative along with the gradient direction (Kniss et al., 2001, 2002).

Nevertheless, these methods cannot fully solve the blur effect in the intensity-derivative histogram which is caused by noise. Lum and Ma (2004) used the two intensity values on both sides of the border to set up a TF with the assumption that the width of the border represented by the distance between these two sample positions varies with the amount of blur in the volume. Šereda et al. (2006a) proposed another method to represent boundaries by searching for low and high intensity values in both the negative and positive gradient directions of the voxels in a boundary. The representation of those low and high values in a 2-D plane is called the *LH histogram*. An important advantage of LH histograms over the 2-D intensity-gradient magnitude TF is that boundaries appear as blobs rather than arches. Blobs are easier to parameterize for clustering and are less likely to overlap in complicated datasets than arches; thus LH histograms allow for boundaries to be more easily separated either manually or automatically through clustering. Another advantage is that LH histograms have greater robustness to noise, bias and partial volume effects than intensity-gradient magnitude histograms. Recently, a semi-automatic generation of LH TFs using a fast generation of LH values has been introduced by Praßni et al. (2009).

Apart from gradient-based methods, another popular approach to designing TFs is to employ curvature-based information. Bajaj et al. (1997) introduced the *contour spectrum* which consists of a variety of scalar data and contour attributes to describe isosurfaces. They also provided an interactive user inferface for the selection of relevant isovalues. Another method for isosurface visualization that uses a cumulative Laplacian-weighted intensity histogram was proposed by Pekar et al. (2001). Inspired by the work of Hladuvka et al. (2000), Kindlmann et al. (2003) proposed a methodology for computing high quality curvature measurements, and the application of curvature-based TFs in volume rendering.

Since derivatives or curvature measurements only represent local information, a number of methods have attempted to extend the traditional 1-D or 2-D histograms with spatial information. Lundström et al. (2005, 2006a) introduced *local histograms*, which utilize the distribution of intensity values in a given neighborhood to differentiate between different tissues. They also introduced another method to incorporate spatial coherence into an enhanced histogram, the α-*histogram* (Lundström et al., 2006b). The local histograms of disjoint local regions are calculated, and then raised to the power of $\alpha > 1$ before being summed and normalized to produce the global histogram. Thus, spatially concentrated value ranges are amplified to enlarge peaks corresponding to different materials in the global histogram. Röttger et al. (2005) also applied the idea of using spatial features of the dataset for TF design. They compute the mean and variance values of all voxels belonging to one single bin in the 2-D histogram, and then use these measures with a maximum feature radius to classify the histogram. Tappenbeck et al. (2006) employed a distance-based TF, which allows the optical properties of structures to be modified based on their distance to a reference structure.

Recently, *feature size* has been used in several approaches as a parameter to design TFs for separating structures with similar intensity values. Correa and Ma (2008) used scale fields for continuous representation of feature size then assigned to each voxel an additional parameter depending on the local scale of the feature containing the voxel. Size-based TFs then map the opacity and color based on the relative size of features. Hadwiger et al. (2008) used region growing to derive feature size information. Instead of using region growing, in Wesarg and Kirschner (2009); Wesarg et al. (2010), information of structure size is estimated by searching voxels with intensity satisfying an user specified tolerance value along 26 directions connecting each voxel with its neighbors. The image generated from the structure size data is used as the second property of a *structure size enhanced* (SSE) histogram. In our method, we also use the size of regions, estimated using a new method, to automatically assign visual parameters to minimize occlusions.

One difficulty of the TF design is the lack of a measure to quantify the quality of TFs. Research works by Correa and Ma (2009b, 2011) use *visibility*, the contribution of a sample in terms of opacity to the final image, as a primitive to guide TF design in both manual and automatic modes. With the help of *visibility histograms*, which is a multi-dimensional representations of the distribution of visibility in a rendered image, users can produce TFs that maximize the visibility of the intervals of interest. In the automatic mode, TFs are generated in order to minimize the mismatch between the opacity TF defined by the user and the computed one, and maximize the visibility of important structures. Visibility was also used in Chan et al. (2009) together with shape and transparency to evaluate and enhance the perceptual quality of transparent structures in the rendered image. Correa and Ma (2009a) introduced the *occlusion spectrum*, which is the distribution of weighted averages of the intensities in a spherical neighborhood of each voxel. With this spectrum, better 2-D TFs that can help classify complex datasets in terms of the spatial relationships among features can be produced.

In addition to finding new presentations of features, algorithms to separate different regions in the feature domain have been investigated as well. Tzeng and Ma (2004) presented a method to create TFs based on material classes extracted from the cluster space using the ISODATA technique. Šereda et al. (2006b) applied hierarchical clustering to LH space to group voxels based on their LH values. Maciejewski et al. (2009) used non-parametric clustering to extract patterns from TF feature space and guide the generation of TFs. Zhou et al. (2010) developed a parallel mean shift technique to assign visual parameters to different regions of the volume by clustering voxels based on their scalar values and spatial locations. In our work, we apply a multi-stage clustering process including mean shift and hierarchical clustering that incorporates LH and spatial information to cluster and identify complex material boundaries. This multi-stage clustering is non-parametric, robust, and preserves a large degree of freedom for user manipulation of results.

4.2 Transfer Function Design System

Volumetric data consists of multiple material boundaries where the boundary voxels have similar LH values and spatial location. Our system offers three modes of operation, two automatic and one semi-automatic, to visualize volumetric data. Figure 4.1 presents an overview of our system. The brief descriptions on the three operational modes are given below.

4.2.1 Automatic TF Design using Two-step Clustering

Mean shift clustering (Fukunaga and Larry, 1975) is used to oversegment the LH space into multiple groups. Hierarchical clustering using a similarity measure based on neighborhood relations (Šereda et al., 2006b) then merges these groups into clusters. An automatic TF design module is then used to assign colors and opacities to each cluster such that the number of occlusions is reduced. If the resulting visualization is unsatisfactory, the user can modify the visualization results using the bounding polygon widget in the user interaction module. The mean shift algorithm in our system is implemented in CUDA and uses the GPU (graphics processing unit) to reduce the clustering time.

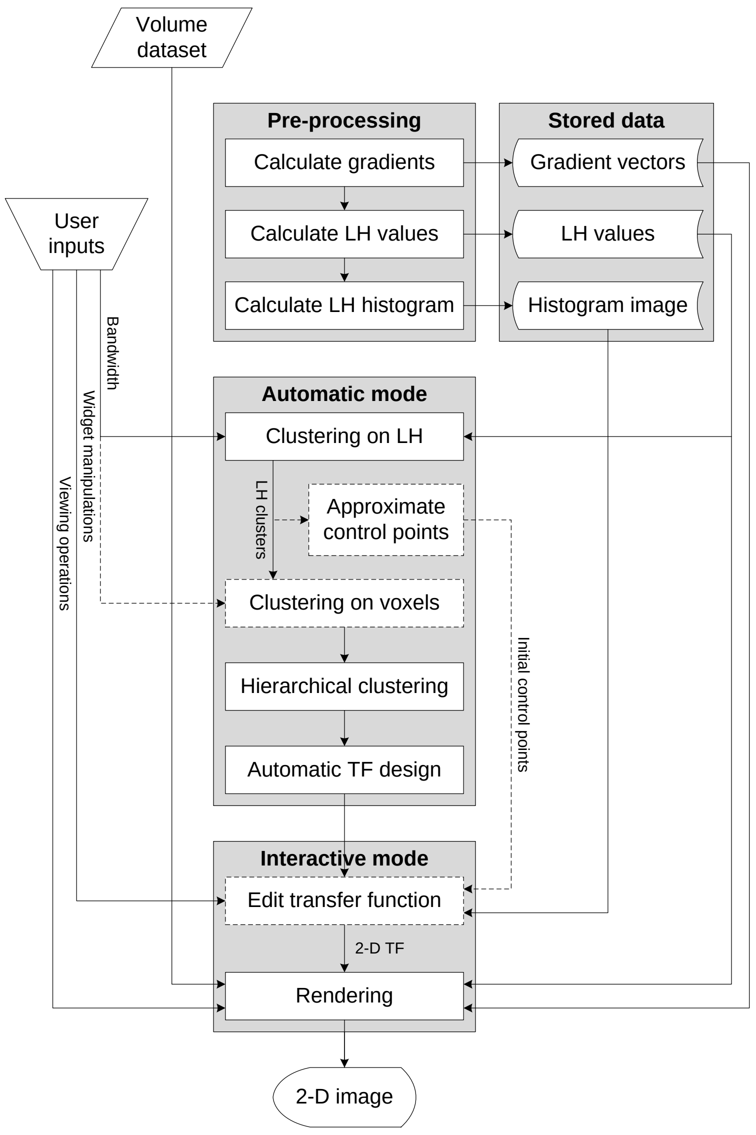


FIGURE 4.1: Overview of the system. Dotted boxes represent optional modes.

4.2.2 Automatic TF Design using Three-step Clustering

The automatic mode with three-step clustering uses a more complex clustering scheme that incorporates spatial information, and is more suitable for complex data compared to that of the automatic mode described above. The automatic TF design module is responsible for generating the boundary voxel clusters, and for the assignment of TFs to each cluster. To generate the boundary voxel clusters, a three-stage clustering process is performed. First, mean shift clustering is applied twice in series. The first mean shift clustering is performed to segment the LH space into multiple LH clusters, while the second mean shift clustering further segments voxels in each LH cluster based on their spatial proximity. A third clustering step using hierarchical clustering then generates a cluster hierarchy. The cluster hierarchy is used to obtain a final set of voxel clusters, each of which represents a single material-material boundary in the volume. All clusters are then automatically assigned colors and opacities based on the distribution of voxels in each cluster so that this minimizes the occlusion and maximizes the discrimination of the boundaries. If the rendering result is not satisfactory, the user can adjust the visualization results using the user interaction module.

4.2.3 Semi-automatic User Interaction

The user interaction module operates in parallel with the automatic TF design module and allows the user to adjust the system parameters to obtain more desirable visualizations. The user can influence the result at various stages in the system to achieve varying effects on the visualization output. First, the linking threshold can be interactively modified to vary the number of material boundaries. Second, the automatically generated transfer function assigned to a cluster can also be interactively adjusted to change the opacity or color of the target cluster. Third, if a greater degree of adjustment is desired, the mean shift bandwidths for the LH and spatial clustering steps can be changed to refine the boundaries, or the results of the LH clustering step can be adjusted using an interaction widget; however these changes are more extensive and may require more time to compute.

4.3 Transfer Function Design Processes

This section illustrates the important processes of our TF design system.

4.3.1 Pre-processing

In all the operating modes, the gradient vector and LH values corresponding to each voxel are first computed in a pre-processing step. To calculate the gradients, the method proposed by Hong et al. (2003) is used to determine a second-degree polynomial function to approximate the density function in a local neighborhood. The coefficients of the polynomial function can be solved by minimizing the error of the approximation. Once these coefficients are known, the first partial derivatives, i.e., the gradient, and the second directional derivatives corresponding to each voxel can be determined from its intensity and its position. The advantages of this approximation method are (1) we can take into account the difference between the pixel spacing and the spacing between slices which often exists in medical datasets; (2) we do not have to use any computationally expensive interpolation method to estimate the gradient vector of an arbitrary sampling point between voxels; (3) this method is robust to noise since it does not interpolate the curve passing through all the given data points.

Based on the computed gradients, the lower (L) intensity and higher (H) intensity values of each voxel can be determined by tracking the boundary path using gradient integration in both directions. We use Heun's method, which is a modified Euler's method, to integrate the gradient field:

$$u_{i+1} = u_i + \frac{1}{2}d\left(\nabla f\left(u_i\right) + \nabla f\left(u_i + d\nabla f\left(u_i\right)\right)\right), \tag{4.1}$$

where u_i and u_{i+1} are positions of the current and the next sampling voxels, respectively, ∇f denotes normalized gradient vector when tracking H or the inverse one when tracking L, and d is the step size of the integration. In our experiments, a step size of one voxel is chosen as a good balance between accuracy and computation speed. The integration stops when a local extremum or an inflexion point is reached. In order to emphasize voxels on the boundary or near the middle layer between two materials, each pair [L, H] is weighted by a factor w when being accumulated to create the LH histogram. The weight w is determined from

$$w = 1 - \frac{|d_L - d_H|}{d_L + d_H}, \tag{4.2}$$

where d_L and d_H are the accumulated distances along the boundary path from the current voxel to the sampling voxels corresponding to L and H, respectively. Preliminary experiments suggest that the new interpolation method based on approximation is superior to conventional gradient estimation methods. Combined with the distance weighting factor, it can reduce the noise in the LH histogram and improve the resulting visual quality.

The LH histogram in our method is represented as an image of $N \times N$ pixels. To balance between the memory needed and the visual quality, N is chosen as 512 pixels. The image is constructed by determining the correct bin for each [L, H] pair, scaling the sum of all corresponding weight factors taking the logarithm, and then mapping the resulting value to a color band, e.g., the cold-to-hot spectrum (Figure 4.2). At the end of this pre-processing step, all the gradient vectors, the LH values, and the histogram image are stored in an intermediate data file for further processing.

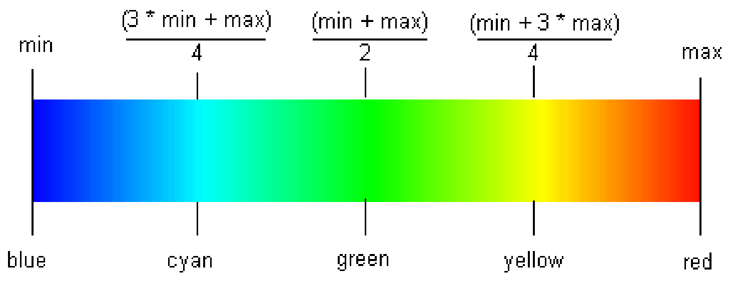


FIGURE 4.2: Cold-to-hot color ramp.

4.3.2 Mean Shift Clustering in LH Space

Mean shift is a popular non-parametric clustering algorithm that seeks the modes of the given sample space. Mean shift clustering offers two main advantages over other clustering methods: (1) no limitations or assumptions on the structure or distribution of the data are made, (2) and the number of clusters does not need to be specified a priori. For the first clustering step, each voxel is clustered according to its LH value. A bandwidth parameter B_{LH} controls the sensitivity of the mean shift clustering. From our experiments, a good B_{LH} lies between 7–9% of the maximum LH value, $\max_{LH} = \max(\max_L, \max_H)$. As a further performance optimization, mean shift clustering is computed over discrete values in the LH histogram, and each LH point is weighted during the mean computation step by the number of times it occurs in the volume. Since all voxels can only have discrete LH values and the LH histogram is relatively sparse, this reduces the time and memory required for mean shift clustering. The procedure of mean shift clustering is summarized in the following algorithm:

1. Define a clustering parameter, window bandwidth B_{LH}.
2. For a point in the LH histogram, find all points that have LH values within the bandwidth B_{LH}.

3. Find the mean μ_n of the set of neighboring points, with each point weighted by its voxel frequency.

4. Shift the window center to the new mean, and continue steps 2-4 until convergence. A cluster is deemed to have converged if the distance between successive means is less than ρB_{LH} where ρ is a threshold preset as 0.001 in our experiments.

5. Repeat steps 2-4 for each point in the LH histogram.

6. Points that converge to the same modes (the converged cluster mean) are grouped as a single cluster, and clusters that have modes within $B_{LH}/2$ of each other are also grouped as one cluster.

After this clustering step, the data will be grouped into a few hundred clusters, each containing a large number of voxels with similar LH values.

In our implementation of mean shift clustering using CUDA, we use two 1-D textures to store the LH points and their corresponding weights. The cache mechanism of texture memory can recur memory traffic when the LH points and their weights are read by the kernel. The kernel in each thread is programmed to process a set of LH points by finding their means according to the algorithm above. After all the means are determined, the LH points are grouped into different clusters depending on the distance between their means. This GPU-based implementation is about 10 times faster than the CPU-based version in our previous work (Nguyen et al., 2011c), making the proposed multi-stage clustering algorithm more practical.

4.3.3 Mean Shift Clustering on Spatial Domain

At the end of previous step, each cluster contains voxels with similar LH values. However, it cannot be assumed that these voxels belong to the same boundary, unless they are also located close together. In this step, mean shift clustering is applied to cluster each LH cluster based on their spatial coordinates. As a non-parametric clustering method, mean shift clustering is particularly suitable for this task because the material boundaries may have complex shapes. A spatial bandwidth parameter B_{XYZ} controls the sensitivity of the clustering process. After all clusters have been processed, the volume voxels will be grouped into clusters, where each group has similar LH values and are spatially close.

The mean shift clustering algorithm used to cluster the voxels is the same algorithm used in the previous step, with the appropriate change of feature domain and bandwidth parameter.

4.3.4 Hierarchical Clustering of All Clusters

Hierarchical clustering is a clustering method which builds a hierarchy of clusters by incrementally merging pairs of clusters together (Duda et al., 2001). By stopping the merging based on some external criteria, for example a threshold distance between two clusters, the number of final clusters can be controlled as desired. Here, hierarchical clustering is initialized using each output cluster from the previous step. Then, the two clusters with the lowest pairwise distance are merged together, and this merging process continues until all voxels are grouped under one single cluster. The user is then able to vary the number of boundary clusters by reversing the agglomerative process and breaking clusters in the reverse merge order. The agglomerative hierarchical clustering algorithm is summarized below:

1. Begin with a set of clusters from the previous clustering step.
2. Compute the pairwise distance between all pairs of clusters in C.
3. Let the pair of clusters with the lowest pairwise distance be C_a, C_b. Merge C_a, C_b. Also record the clusters that have been merged, the order of merging, and the distance between C_a and C_b.
4. Update the pairwise distance between the newly merged cluster and all other clusters.
5. Repeat steps 3 and 4 until only one cluster remains.
6. Cluster links are repeatedly cut from the top of the hierarchy until N_C clusters remain. N_C is an input parameter defined by the user.

For the pairwise distance between clusters, the metric is based on the volume similarity measure introduced by Šereda et al. (2006b). This volume similarity measure evaluates the number of neighborhood relations between the two clusters. First, for each cluster C_i, the number of neighborhood relations is

$$\mathrm{NR}(C_i) = \sum_j \mathrm{NR}(C_i, C_j), \tag{4.3}$$

whereas the neighborbood relations between C_i and C_j is computed by counting the number of 26-neighbors belong to C_j for each voxel v_i in C_i:

$$\mathrm{NR}(C_i, C_j) = \sum_{v_i \in C_i} \sum_{v_j \in C_j} N_{26}(v_i, v_j); C_i \neq C_j. \tag{4.4}$$

The similarity measure is then computed by taking the maximum of the normalized sum of neighborhood relations between the two clusters:

$$s(C_i, C_j) = \max\left\{\frac{\mathrm{NR}(C_i, C_j)}{\mathrm{NR}(C_i)}, \frac{\mathrm{NR}(C_i, C_j)}{\mathrm{NR}(C_j)}\right\}. \tag{4.5}$$

This similarity measure is updated after each merging of clusters. The number of relations in the merged cluster $C_{i\cup j}$ is updated with the following approximation:

$$\mathrm{NR}(C_{i\cup j}) = \mathrm{NR}(C_i) + \mathrm{NR}(C_j) - 2\mathrm{NR}(C_i, C_j) \tag{4.6}$$

while the number of relations and the similarity measure with all other clusters C_k are respectively recalculated as

$$\mathrm{NR}(C_{i\cup j}, C_k) = \mathrm{NR}(C_i, C_k) + \mathrm{NR}(C_j, C_k) \tag{4.7}$$

$$s(C_{i\cup j}, C_k) = \max\left\{\frac{\mathrm{NR}(C_{i\cup j}, C_k)}{\mathrm{NR}(C_{i\cup j})}, \frac{\mathrm{NR}(C_{i\cup j}, C_k)}{\mathrm{NR}(C_k)}\right\}. \tag{4.8}$$

4.3.5 Assignment of Visual Parameters

The TF design module is built to satisfy four objectives that describe good visualizations. First, each cluster and material boundary must be visually distinct from all other clusters and material boundaries. Second, within each individual cluster and material boundary, voxels closer to the boundary interface are more important than voxels farther away from the boundary interface. Third, within each individual cluster and material boundary, voxels with different scalar values should have slightly different visual appearances. Fourth, internal structures should not be occluded by larger or outer structures. Therefore, the TF design module first assigns a different opacity and color to each separate cluster, depending on how much the cluster occludes other clusters. Each voxel in a cluster then inherits visual properties from its cluster, modulated by a scaling factor dependent on the voxel's gradient magnitude relative to the gradient magnitude of other voxels in the cluster.

A region is more likely to occlude other regions when it is large and has several close neighboring regions. Hence, the degree by which a region occludes other regions can be estimated by the size of the region in the volume and the relative distance between that region and its neighbors. The *size* of the region R_i corresponding to the cluster C_i is coarsely estimated by the standard deviation σ_i of the positions of all the voxels $v_j = \left(v_x^j, v_y^j, v_z^j\right) \in R_i$

$$\sigma_i = \sqrt{\frac{1}{N_i} \sum_{v_j \in R_i} \left(v_j - \mu_i\right)^2}, \tag{4.9}$$

where N_i is the number of voxels in R_i, and μ_i is the mean of the positions of all voxels in R_i:

$$\mu_i = \frac{1}{N_i} \sum_{v_j \in R_i} v_j. \tag{4.10}$$

The *distance* between two regions R_i and R_j is defined as the Euclidean distance between the two corresponding mean values:

$$D\left(R_i, R_j\right) = \sqrt{\left(\mu_x^i - \mu_x^j\right)^2 + \left(\mu_y^i - \mu_y^j\right)^2 + \left(\mu_z^i - \mu_z^j\right)^2} \tag{4.11}$$

A region R_i *occludes* region R_j if

$$\begin{cases} \sigma_i > \sigma_j \\ \sigma_i > k_d D\left(R_i, R_j\right) \end{cases}, \tag{4.12}$$

where $k_d \geq 1$ is a pre-defined value. The opacity α_i assigned to region R_i is calculated by

$$\alpha_i = \frac{\alpha_i^*}{k_s\left(S_i + 1\right)}, \tag{4.13}$$

where k_s is an adjustable factor, S_i is the number of regions occluded by R_i, and α_i^* is the value corresponding to σ_i in the linear mapping of $\left[\min_j \sigma_j, \max_j \sigma_j\right]$ to a predefined opacity range $[\alpha_{\min}, \alpha_{\max}]$:

$$\alpha_i^* = \frac{\max_j \sigma_j - \sigma_i}{\max_j \sigma_j - \min_j \sigma_j}\left(\alpha_{\max} - \alpha_{\min}\right) + \alpha_{\min}. \tag{4.14}$$

Each voxel in a cluster inherits the cluster opacity scaled to the enhance voxels nearer to the boundaries. The voxel opacity α_v^i corresponding to the voxel v in the region R_i is individually modulated by the ratio of its gradient magnitude and the maximum gradient magnitude of all the voxels in the region:

$$\alpha_v^i = \alpha_i \frac{\|\nabla v\|}{\max_{u \in R_i} \|\nabla u\|}. \tag{4.15}$$

The true colors of the materials cannot be determined from the data volumes alone, hence colors in TFs are typically assigned according to some external criteria or with external knowledge. Here, the color of each region is assigned according to the relative size of the structure, mapped onto a cold-to-hot spectrum (Figure 4.2). Hence, small structures will be mapped to hot colors (red) while large regions will be mapped to cool colors (blue). Alternatively, a pre-defined color array can be applied for the color mapping as the number of regions tends be small. The color c_v^i of each voxel within a region R_i is further modulated by the ratio of its intensity value f_v and the maximum intensity of all voxels in the region:

$$c_v^i = c_i \frac{f_v}{\max\limits_{u \in R_i} f_u}. \tag{4.16}$$

This scaling is only applied to the brightness value of the corresponding color in the HSV color space, which means that voxels within the same region have the same hue and saturation values and can hence be differentiated from voxels belonging to other regions.

4.3.6 Interaction Widget for Modifying LH Clusters

While mean-shift clustering automatically assigns labels to each voxel in the volume, the results may not be immediately satisfactory to the user. Minor adjustments made by the user will improve the quality and relevance of the visualization. To facilitate easy modification of the automatically extracted clusters, the voxel cluster labels are used to generate a set of cluster-bounding polygons. The advantage of cluster polygons is that they are easy to manipulate and modify via polygon and vertex operations. Entire clusters or individual vertices can thus be edited on the LH histogram.

Ideally, each cluster polygon should only contain all voxels assigned to that cluster, but this requires computing concave bounding polygons which is computationally expensive. Furthermore, concave polygons tend to be more complicated than convex polygons. To simplify the computation we assume that the bounding polygons are convex polygons. Then, the bounding polygon can be computed in $\Omega(n \log(n))$ time by fast convex hull algorithms such as Andrew's monotone chain algorithm (Andrew, 1979). To resolve overlaps between bounding polygons we

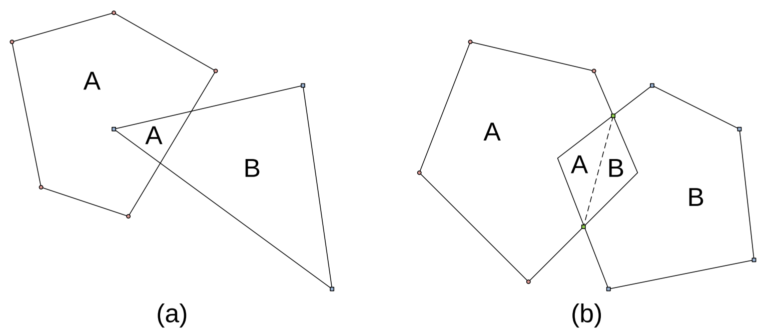


FIGURE 4.3: Two demonstrations of the overlap disambiguation scheme.

perform collision detection for each pair of polygons. For each overlap, there are two intersections. A dividing line is drawn between the two intersections and each partitioned area is assigned to the cluster it is nearest to.

The regions along the main diagonal of the LH histogram belong to voxels lying within the same material, i.e., material not lying on the material interfaces (Šereda et al., 2006a). These clusters may not be important for visualization and can be discarded or rendered with a low opacity value. After the cluster bounding polygons are generated, a simple check is rendered to detect and discard such clusters. All polygons with at least one vertex within a diagonal window of the main diagonal of the LH histogram are treated as clusters of non-boundary material. The diagonal window is experimentally defined to have a width of 2% of the range of LH values. The procedure for computing the cluster bounding polygons is summarized in the following algorithm:

1. For each cluster C_i obtained from the mean shift algorithm, obtain the set of points P_i and compute a convex hull H_i containing all the points in P_i.

2. Construct a convex polygon H_{diag} using the following 6 coordinates: $[0, 0]$, $[0, 0.01 \times \max_H]$, $[0.99 \times \max_L, \max_H]$, $[\max_L, \max_H]$, $[\max_L, 0.99 \times \max_H]$, $[0.01 \times \max_L, \max_H]$, where $\max_L$ and $\max_H$ are the maximum values in the

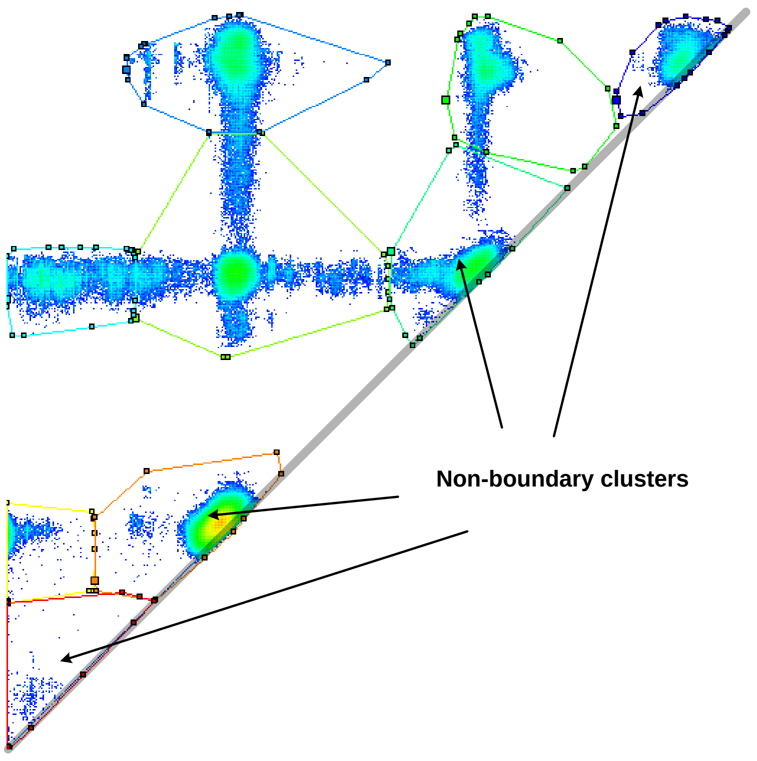


FIGURE 4.4: Demonstration of non-boundary cluster removal.

LH histogram. For each convex hull H_i, if any vertex in H_i lies in H_{diag}, the cluster C_i is treated as a non-boundary cluster and is removed or rendered with low opacity.

3. For each pair of remaining convex hulls H_a and H_b, compute the intersection, if any, between each combination of hull segments. If there are intersections denote them as I_a and I_b. Add both I_a and I_b to both hulls H_a and H_b, and remove all hull points interior to the line segment created by H_a and H_b.

Cluster bounding polygons are a tool to facilitate the manipulation of clusters generated in LH space. This functionality was retained in our system as an interaction widget in the user interaction module. If the user wishes to modify the results of the LH clustering, a convex hull algorithm is first applied to each LH cluster generated using the clustering algorithm to obtain a bounding polygon that describes

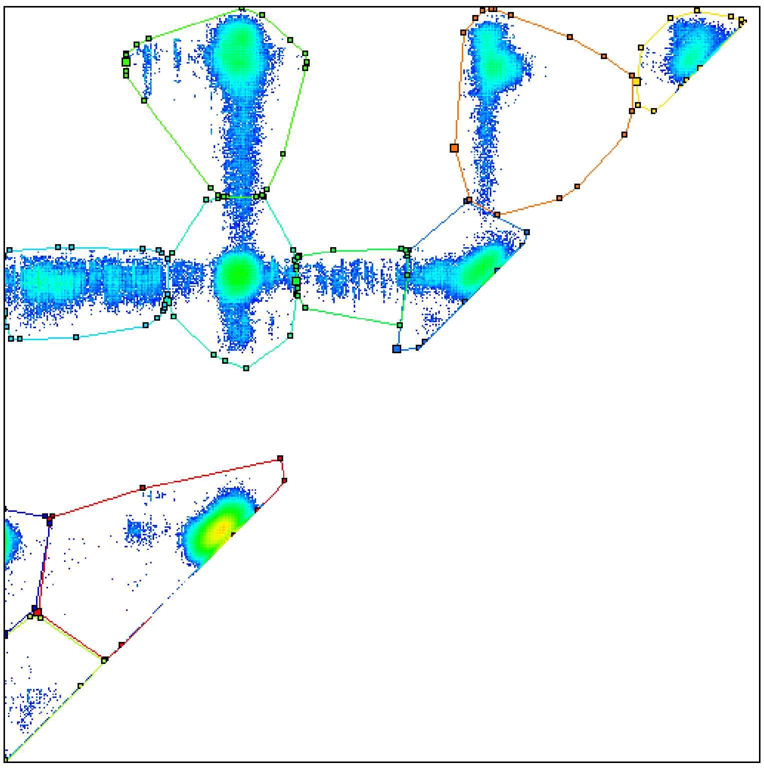


FIGURE 4.5: Demonstration of cluster bounding polygons.

each cluster. Then, a disambiguation scheme is run on the bounding polygons to resolve any overlaps between pairs of bounding polygons. Each automatically generated cluster is now represented as a convex polygon, and the user can manipulate the clusters by performing vertex or polygon operations. Figure 4.5 shows the cluster bounding polygons for a sample dataset.

4.4 Results and Discussion

Three 16-bit CT volumes were used in our experiments: the Feet ($256 \times 256 \times 125$), Head ($128 \times 256 \times 156$), and Pig ($256 \times 256 \times 128$) datasets. The Feet dataset is from University Hospital of Geneva, Switzerland, and the Head dataset is from National Library of Medicine, National Institutes of Health, USA. The Pig dataset is from

TABLE 4.1: Evaluation parameters

Figure	Dataset	Method	B_{LH}	Clusters
4.6(b)	Feet	Two-step	9%	15
4.6(c)	Feet	Two-step	8%	18
4.7(b)	Head	Two-step	9%	17
4.7(c)	Head	Two-step	9%	6
4.8	Pig	Three-step	7%	32

our own surgical planning experiments. Results were obtained on a 2.66 GHz Intel i5-750 system equipped with 4 GB RAM and a NVIDIA Quadro FX 3800 graphics card using C++ and CUDA. The preprocessing time was less than 3 minutes for all datasets used. A GPU-based renderer employing ray marching through a 3-D texture was used for rendering the results and achieved real-time frame rates for all datasets used. The parameter k_d controlling the effect of occlusions in the automatic TF design algorithm was set to 1 in all our trials.

Table 4.1 contains the system parameters used to generate Figures 4.6 to 4.8. The "Method" column describes the clustering process (two-step or three-step) used for each trial. All figures were generated using the automatic mode with the stated B_{LH} parameters for the LH clustering step and the stated number of clusters for the hierarchical clustering step.

For the Feet dataset (Figure 4.6), the automatic mode was run with two-step clustering and a B_{LH} of 9% of $\max_{LH}$ to generate the initial LH clusters (Figure 4.6(a)). The number of clusters generated with hierarchical clustering was varied to obtain Figure 4.6(b) (15 clusters) and Figure 4.6(c) (18 clusters). The clustering process took around 10 s to complete, with the majority of the time (90%) being spent on the hierarchical clustering operation. All colors and opacities were automatically assigned using the automatic TF design module. The visualizations clearly show the bones within the feet, with the metatarsal bones

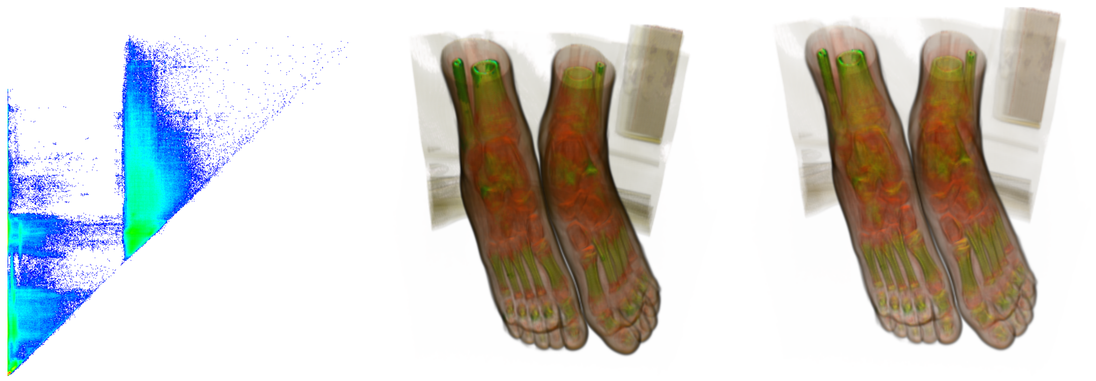

(a) (b) (c)

FIGURE 4.6: Volume rendering of the Feet dataset. (a) LH histogram, (b) Rendered image with 15 clusters, and (c) Rendered image with 18 clusters.

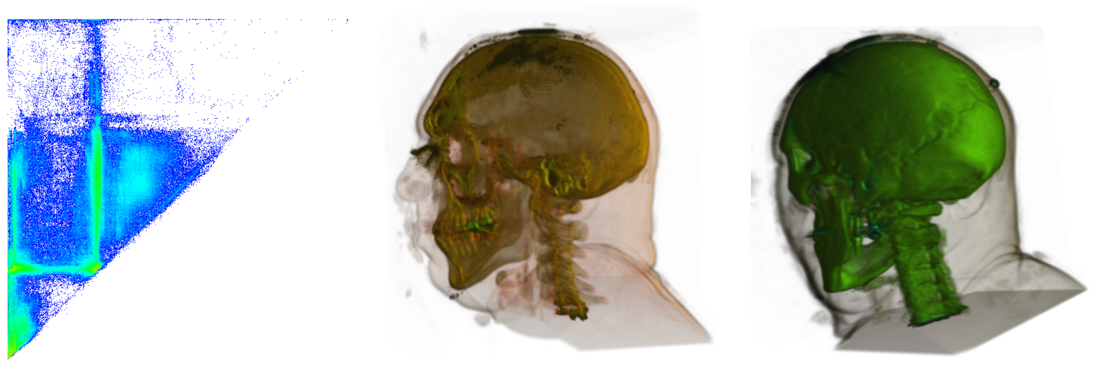

(a) (b) (c)

FIGURE 4.7: Volume rendering of the Head dataset. (a) LH histogram, (b) Rendered image with 17 clusters, and (c) Rendered image with 6 clusters with a different viewing angle.

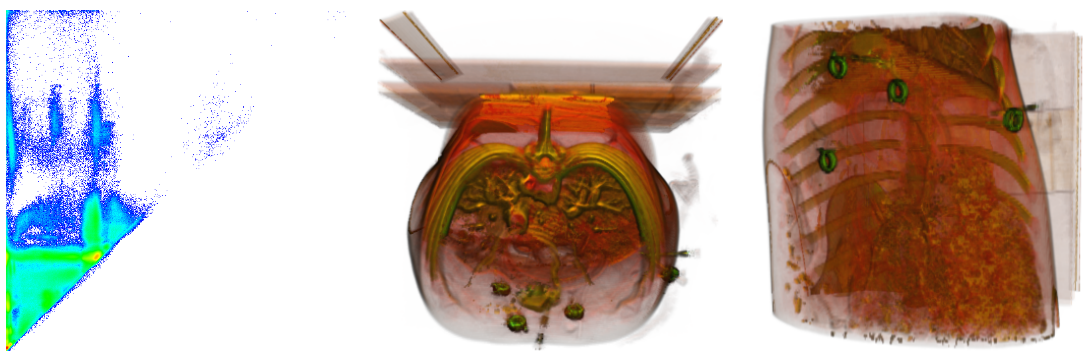

(a) (b) (c)

FIGURE 4.8: Volume rendering of the Pig dataset. (a) LH histogram, (b) and (c) Rendered image with 32 clusters from two different viewing angles.

being colored differently from the other tarsal bones. The results demonstrates the potential of the TF design system for identifying and visualizing structures with medical volumes.

For the Head dataset (Figure 4.7), two-step clustering was used again with a B_{LH} of 9% to generate the initial LH clusters (Figure 4.7(a). Figures 4.7(b) and 4.7(c) were generated using hierarchical clustering with 17 and 6 clusters respectively. The clustering time was also around 10s. The results show that adjustment of the number of clusters influences the number of distinct structures visible in the region. By setting the system to output more clusters, more objects are visible in Figure 4.7(b). Conversely, the number of unique structures can be reduced by lowering the cluster number, such as in Figure 4.7(c) where the main focus was on capturing the crack in the skull region.

The Pig dataset (Figure 4.8) is a complex volume with a number of similar structures within the volume. The CT scans were obtained from a robot assisted surgical experiment conducted on a live pig. It is difficult to select, whether manually or automatically using clustering algorithms, the clusters from the LH histogram. Furthermore, it is not possible to separating the structures from the LH values alone. Figures 4.8(b) and 4.8(c) were generated using the automatic mode with three-step clustering. The system was capable of producing a visualization that differentiates between the surgically important structures (surgical markers, bones, major blood vessels) within the volume. With minor operator intervention to further improve the visualization, the TF design system is useful for medical and surgical visualization, and it enables the surgeon to accurately and easily plan the surgical intervention during operations.

4.5 Summary

We have developed a system for the automatic and semi-automatic generation of TFs for medical volume visualization. The multi-step clustering process incorporates LH and spatial information to cluster and identify complex material boundaries, while the automatic TF design module is able to assign good TFs such that boundaries are not occluded. The proposed system automatically generates good visualizations while preserving a high degree of freedom for the user to adjust the rendering results. The visualizations generated by the proposed method are comparable to existing state-of-the-art approaches.

Chapter 5

Vasculature and Flow Rendering for Medical Simulation

Visualization of vasculature and flow is essential in medical simulation. This chapter introduces a physics-based hybrid rendering method for interactive simulation of drug injection into vasculature (Nguyen et al., 2009). The method is motivated by the mechanism of *chemoembolization*, which is an important therapeutic method to treat cancer, most often of the liver.

Chemoembolization is a combination of local delivery of chemotherapy and a procedure called *embolization*. In chemoembolization, a catheter is used as a conduit to inject anti-cancer drugs directly into a cancerous tumor to place thrombotic agents inside the vessels that supply blood to the tumor. While the liver continues to receive blood from the portal vein, oxygenated blood is no longer supplied to the tumor since the artery is blocked. This also traps the anti-cancer drugs in the tumor, allowing the drugs to stay within the tumor for a longer period of time. Real-time X-ray fluoroscopy is used in chemoembolization to monitor the passage of catheter through the artery, as well as the injection of drugs into the tumor. Several groups have been working on interactive simulation systems on catheter navigation for training and surgical planning (Chui et al., 2002; Alderliesten et al., 2004). The injection of chemotherapy drugs is just as important compared to the manipulation of catheter since it is a complex image guided procedure that requires a high degree of hand-eye coordination skills on the part of the physician. In addition, there is always a risk that the thrombotic agents can lodge in the wrong place and deprive normal tissue of its blood supply.

In order to simulate the flow of particles in vasculature, the method firstly reconstructs the vessels from medical images, and then models the movement of the particles using fluid dynamics. Subsequently, a computationally efficient rendering method is applied to produce good visual approximation of the flow of particles inside the blood vessels. This rendering method can be combined with a fast volume rendering algorithm to show the vasculature within other organs. Although the proposed method can be used to visualize the flow in the vessels of an arbitrary organ, we use chemoembolization as an example to illustrate the simulation method. In this simulation, we assume that the catheter has already been inserted and navigated to the artery that feeds the tumor. Details of the method are described as below.

5.1 Vascular Reconstruction

We hypothesize that a hepatic vessel can be represented by one or more finite element beam elements. Such an element has a circular cross section and can be visualized as a generalized cylinder. This is a valid assumption of the hepatic vessels since they are typically small vessels. When filled with blood, they generally have a circular cross section. In order to obtain a geometric model of hepatic vascular, a 3-D region growing algorithm is first applied to extract the raw regions

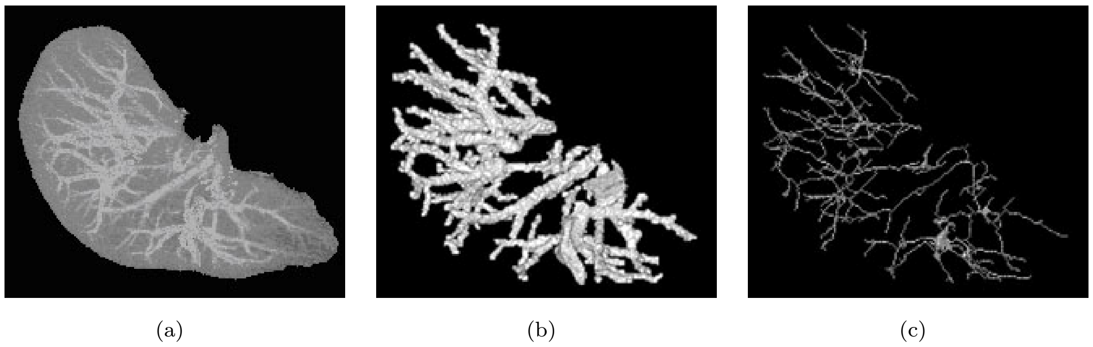


FIGURE 5.1: Hepatic vessels reconstruction. (a) The maximum-intensity projection images for all slides, (b) The raw regions of interest, and (c) The skeleton of the vessel tree.

of interest from a 3-D volume constructed based on CT images of patient's liver organ (Figure 5.1). Next, a 3-D thinning algorithm is used to generate the hepatic vasculature skeleton from the raw regions. The number of voxels in the skeleton is reduced in a post-processing step; the vessel skeleton thus becomes a unit-width curve. After that, each branch in the vessel trees is approximated by a cubic b-spline. Finally, a set of control points forming the splines and the branching structure of the trees will be used in modeling the hepatic vasculature. Details of the method are presented below.

5.1.1 3-D Region Growing

Before processing, the user defines the intensity range for valid seeds and the allowed intensity for voxels in resultant regions. The 3-D region growing algorithm involves the following steps:

1. Find the first unvisited seed which is in the allowed intensity range. Assign the next available region number to this voxel and push it into a waiting queue.

2. Pop a voxel (x, y, z) from the queue and examine all its unvisited neighbors. Assign the current region number to the neighbor (x', y', z') and push (x', y', z') into a queue if and only if: both (x, y, z) and (x', y', z') are valid seeds, or (x', y', z') is in the allowed intensity range and the intensity difference between it and (x, y, z) is small enough. Repeat this step until the queue is empty.

3. Return to step (1) if more seeds in the volume still need to be examined.

4. Remove all extracted regions which have the total number of voxels and/or the total number of seeds less than the pre-specified numbers.

5.1.2 Thinning and Skeletonization

An efficient 3-D thinning algorithm (Palágyi and Kuba, 1999) is applied to all the remaining regions from 3D region growing to extract their skeletons. This algorithm preserves the topology of each region and guarantees that the skeleton will be close to the medial axis of the region (Figure 5.1(c)). However, like most of other 3-D thinning algorithms, this algorithm cannot be guaranteed to generate unit-width curve skeletons in some cases. In order to improve the accuracy of subsequent processing steps, a post-processing procedure is applied to reduce the number of voxels in the skeleton, thus causing it to be a unit-width curve.

We define the *degree* of a voxel as the number of object voxels in its 26 neighbors. We also define a *crowded joint voxel* as a voxel that has degree > 2 and at least one of its neighbors has degree > 2. A crowded region is a region formed by 26-connected crowded joint voxels.

The post-processing step begins with finding all the crowded regions. Next, all object voxels having degree > 2 and 26-adjacent to each crowded region are located and marked as entry voxels. Then an object voxel in a crowded region which is closest to the region centroid is determined and marked as a target voxel. In each crowded region, the Dijkstra algorithm is applied to find the shortest path between each entry voxels and the target voxel. Finally, all crowded voxels that are not on any of the shortest paths are removed. As the result, the skeleton becomes one unit wide because all the crowded regions are eliminated.

5.1.3 Generalized Cylinder Vessel Modeling

A tree-like branching structure is dynamically established by a recursive traverse procedure through all unit-width vessel skeletons. In order to reduce the computational load and for ease of representing the skeleton in the form of a vessel, a finite number of control points and a knot sequence are assigned to each branch of the skeleton. Subsequently, the new branch is obtained by finding a cubic b-spline curve passing through all those control points (Farin, 1999). Each b-spline can be sampled by specifying a number of points between two successive control points. Therefore, each branch is represented by the set of lines connecting the sample points in the corresponding b-spline. These lines will be the central lines of the generalized cylinders representing the vessel branch. The radius of each cylinder can be derived from the raw region of interest and the corresponding skeleton of each vessel branch obtained after previous step. A circle in the plane of the cross-section is continuously grown until it reaches a relevant local maximum of the allowed intensity, thus giving estimated radius along the central lines. The use of control points in this process is computationally faster compared to conventional methods and hence more suitable for interactive simulation, particularly augmented reality applications.

5.2 Flow Model

In fluid dynamics, the Hagen-Poiseuille equation is a physical law that describes slow viscous incompressible flow through a constant circular cross-section. Neglecting effect of gravity, the differential equation of fluid flow is given by:

$$\mu \frac{du}{dr} = \frac{r}{2}\frac{dp}{dx}. \tag{5.1}$$

Assuming the viscosity is constant and the flow is steady, we can derive the following formula (Sutera and Skalak, 1993):

$$-\Delta P = \frac{8\mu L}{\pi R^4} Q. \tag{5.2}$$

In the above equations, μ is the dynamic fluid viscosity, $u(r)$ is the axial velocity at radial distance r from the tube centre line, x is a distance in direction of the flow, Q denotes the volumetric flow rate, L is the length of the tube, R is the radius of the tube, and ΔP is the pressure drop across the tube.

The standard Hagen-Poiseuille Equation (5.2) applies for a tube section with constant radius. However, it can be modified to apply to a conical tube with a linearly varying radius. Obviously, a conical tube with radii R_1 and R_2 at its ends can be obtained by rotating a line segment 360° about the x-axis (Figure 5.2). Hence, it

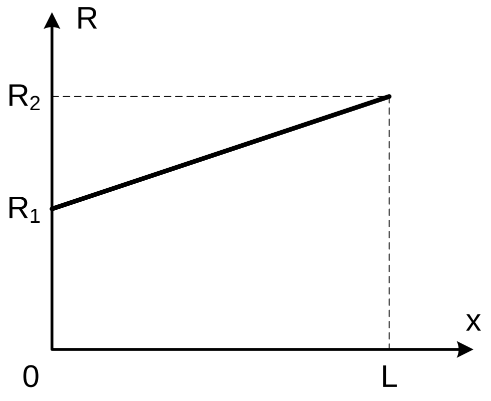


FIGURE 5.2: A line segment can be rotated 360° about the x-axis to generate a conical pipe.

can be considered as being made up of numerous thin sections. Each of the sections can be approximated as a constant radius tube section where the Hagen-Poiseuille relations apply. As a result, we can derive the pressure difference between the two ends as follows:

$$-\Delta P = \int_0^L \left(\frac{8\mu Q}{\pi R^4}\right) dx. \tag{5.3}$$

The equation of the line segment can be represented as a function $R(x)$:

$$R = \frac{R_2 - R_1}{L} x + R_1. \tag{5.4}$$

From (5.4), we have:

$$dx = \frac{L}{R_2 - R_1} dR. \tag{5.5}$$

Substituting (5.5) into (5.3) and changing the limits of integration accordingly,

$$-\Delta P = \int_{R_1}^{R_2} \frac{8\mu QL}{\pi R^4 (R_2 - R_1)} dR = \frac{8\mu QL}{\pi}\left[\frac{1}{3(R_2 - R_1)}\left(\frac{1}{R_1^3} - \frac{1}{R_2^3}\right)\right]. \tag{5.6}$$

Hence, we can derive the following formula for ΔP:

$$-\Delta P = \frac{8\mu L Q}{\pi}\left[\frac{1}{3}\left(\frac{1}{R_1 R_2^3} + \frac{1}{R_1^2 R_2^2} + \frac{1}{R_1^3 R_2}\right)\right]. \tag{5.7}$$

For a tube subjected to a flow rate Q and pressure drop $-\Delta P$, the conductance to flow is

$$C = \frac{Q}{-\Delta P}. \tag{5.8}$$

Assuming that L and μ are similar for all subsidiary branches at points of branching, the minimum conductance of a filled subsidiary branch is essentially proportional to the fourth power of its minimum radius, i.e., $R_{\min}^4$. For a partially filled subsidiary branch, $R_{\min}$ is given by the minimum radius of cross sections where the fluid has flowed past. This will be updated as branch filling continues. This dynamic update of network conductance to alter the flow distribution is clearly more realistical as the network is filled progressively. Hence, at a branching point with N subsidiary branches and a total input flow rate of Q_0, the flow rate through a subsidiary branch i, with corresponding $R_{\min} = R_i$, is given by

$$Q_i = \frac{R_i^4}{\sum\limits_{j=1}^{N} R_j^4} Q_0. \tag{5.9}$$

The above branching distribution relation also ensures continuity is maintained because

$$Q_0 = \sum_{i=1}^{N} Q_i. \tag{5.10}$$

Similarity, the distribution of injected volume V_0 through a subsidiary branch i is given by

$$V_i = \frac{R_i^4}{\sum\limits_{j=1}^{N} R_j^4} V_0. \tag{5.11}$$

After vascular reconstruction step, the two radii of each tube and its length are known. Therefore, with the input values of injected volume V_0 and flow rate Q_0, based on these above equations, we can calculate the partial length of each segment that is filled by fluid at each time step for rendering purposes.

5.3 Rendering Method

There is a variety of methods for three-dimensional flow visualization, for example the "virtual tubelets" (Schirski et al., 2004). This method is based on cylindrical billboards. They are basically quadrilaterals, which are aligned to face the viewer, and which are drawn or textured appropriately to create tubes. The head of the list of quadrilaterals is positioned at the current position of the corresponding particle with its tails passing through the particle's recent positions (Figure 5.3). Preliminary investigation on visualization of flow particles in chemoembolization reported in (Chui et al., 2005) was also based upon cylindrical billboards. An ellipse was briefly considered in place of quadrilateral. However, the quadrilateral has advantages over ellipse in terms of rendering speed.

In order to simulate the fluoroscopy, we rendered the flow as overlaying and semi-transparent quadrilaterals representing the particles' trails. The quadrilaterals might be drawn in decreasing grayness from the head to the last quadrilateral to depict the washing out effect during injection. This method is efficient in terms of

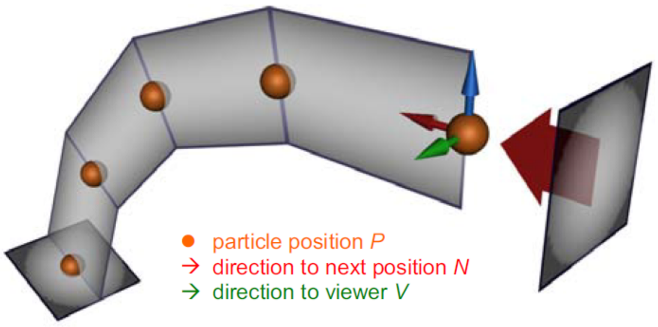


FIGURE 5.3: Anatomy of a virtual tubelet (Schirski et al., 2004).

speed since for each tube segment, there are only one or two quadrilaterals drawn instead of numerous polygons.

This rendering of the flow can be combined with a fast volume rendering algorithm, e.g., the maximum-intensity projection (MIP), to provide more context information of the scene. With a specific viewing direction, the volume rendering is employed first to provide the anatomic context (Figure 5.1(a)). Subsequently, the quadrilaterals are rendered to simulate the flow.

5.4 Results and Discussion

We implemented the reconstruction method of hepatic vessels, and physics-based visualization of particles flow on an Intel Core 2 Duo based notebook computer. Figure 5.4 is a snap shot of the rendered flow in various discrete time steps. Figure 5.5 shows a region of fluoroscopy image of a hepatic artery and the corresponding simulation image generated. The vessel in this test consists of 384 conical tube segments created from 75 control points. The number of tube segments can be changed by changing the point density, i.e., the number of interpolated points between 2 consecutive control points, leading to the trade-off between image quality

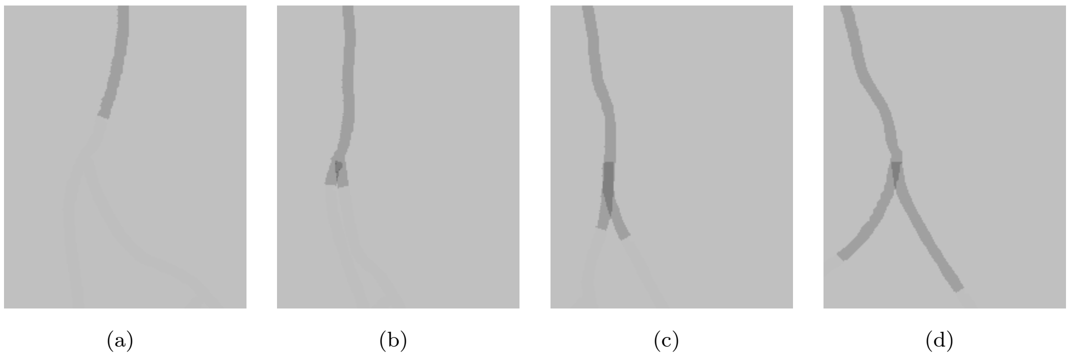


FIGURE 5.4: Visualization of drug injection into a vessel during 4 consecutive time-frames with difference viewing angles. (a) t, (b) $t+1$, (c) $t+2$, and (d) $t+3$.

and speed. The average rendering time including the calculation when the viewing angle is changed approximately is 47 ms, meaning a frame rate of 21 frames per second. It takes approximately 3 s for modeling the vascular tree from a volume dataset of $256 \times 256 \times 120$ voxels. The timings are measured on a 2.5 GHz Intel Core 2 Duo notebook. This un-optimized simulation program uses only Graphics Device Interface (GDI) functions to render the image. Although the frame rate in this test can be considered as an interactive frame rate, it may be significantly improved by shifting computational load to the GPU on an accelerated graphics card. Modern programmable graphics hardware has mechanisms to execute a small assembly program for every vertex which is sent to the graphics system. Hence, several procedures, e.g., computing the orientation of the billboards, can be moved from the CPU to the GPU for increasing the responsiveness of the visualization system.

The quadrilaterals which are aligned along the viewing direction are rendered to visualize the movement of particles through the flow modeled using Hagen-Poiseuille Flow. The physics-based flow model is unique and is important for an accurate simulation of drug flow. While a prior work by Wu et al. (2007) assumes that

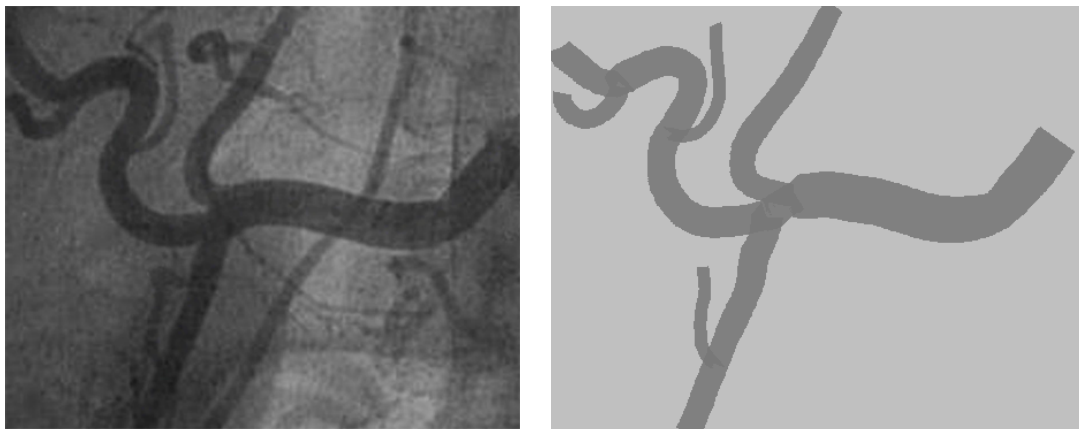


FIGURE 5.5: Fluoroscopic imaging of a hepatic vessel. (a) A part of the image in reality, (b) Corresponding generated image based on the proposed method.

the vascular resistance is invariant in time and the vessel's radius in one segment generally is a constant, the Hagen-Poiseuille equation in this chapter is modified to apply to a conical tube with a linearly varying radius. This approach has capability to create simple but realistic motions of blood flow. Our visualization method is computationally efficient and has achieved good visual approximation of the flow of particles inside the vessels under fluoroscopic imaging. This method is applicable to simulation of chemotherapy drug injection as well as contrast dye injection for angiography.

The quadrilaterals may eventually be replaced with ellipses. Although the quadrilaterals can be rendered faster than that of ellipses, the later may be a realistic representation. This realism will be important when we have a close up view of the flow of particles within the tumor. The flow model has to be improved to more accurately represent the flow within the tumor. The representation of a hepatic vessel using one or more finite element beam elements is important. It provides means to simulate the deformation of the small vessels due to the injection of the particles. The liver organ which is just below the heart is also subjected to periodic deformation from regular heart beats.

Realistic rendering of the tissues including tumor surrounding the vessels can be done by a volume rendering method. Combined with the proposed physics-based geometrical rendering of particles flow, we have a hybrid rendering method that exploits the advantages of both surface and volume rendering.

5.5 Summary

We have described a physics-based method for rendering the flow particles in the simulation of chemotherapy drug injection. A 3-D region growing technique is used to extract hepatic vessels from clinical CT images. These vessels are skeletonized using a 3-D thinning algorithm. An additional post processing step is introduced to ensure that the resultant skeleton is of unit pixel width. The vascular geometries are reconstructed using cubic b-splines. The cubic b-splines method enhances the smoothness of the rendered vessels and is more computational efficient compared to conventional methods. Quadrilaterals which are aligned along the viewing direction are rendered to visualize the movement of particles through the flow modeled using Hagen-Poiseuille Flow. This rendering method can be combined with a fast volume rendering algorithm to provide more context information of the scene. Our visualization method achieves a computational efficient and good visual approximation of the flow of particles inside the vessels under fluoroscopic imaging.

Chapter 6

Conclusions and Future Work

Medical volume data, both static and dynamic types, are playing important roles in medicine. However, a typical characteristic that may limit the use of medical volume datasets in practice is their very large storage requirements. This dissertation has addressed several challenging issues related to visualization of large medical volume data.

6.1 Compression for Visualization of Large Medical Volume Data

In Chapter 2, we proposed an efficient compression method for fast rendering of large dynamic volume data. The algorithm deeply exploits the inherent characteristics of dynamic volume data in both spatial and time dimensions through the employment of a fast and efficient clustering technique. By maintaining a *KeyImage* corresponding to each *KeyBlock*, the rendering process is integrated with the decompression, leading to a significant increase of the rendering speed. To further validate the method, more experiments on larger datasets should be performed.

Importance analysis techniques can be integrated into our clustering method to ensure dynamic features of the time-varying volumetric medical data to be correctly visualized after compression. The relationship between some parameters, such as the initial threshold values and the characteristics of the data, is important, and can be studied in depth in future.

Since there is no restriction on the underlying renderer, the algorithm in Chapter 2 provides flexibilities for further extensions. The rendering technique described in the method can be implemented using a graphics processing unit (GPU) for further improvement in terms of rendering speed. In this case, the *Volume-KeyBlock* table and the *KeyBlocks* are loaded into the GPU's texture memory instead of the main memory. Since GPU is capable of traversing data stored in the texture memory in parallel by using fragment shader, the rendering time is significantly reduced compared to other CPU-based implementations.

The compression scheme described in Chapter 3 is highly appropriate when compressing large volume datasets in which fidelity and decompression time are critical. The key feature of the method is the novel block distortion measure, *variance of residual (VOR)*, which successfully links the hierarchical vector quantization and the motion compensation modules together so that the distortion of the decompressed data is significantly reduced. Due to the simple decompression, the proposed compression scheme is suitable for visualization of large volume data. The compressed data can be loaded to the GPU's memory, and then decompressed and rendered using a GPU-based volume rendering algorithm. Since this is a block-based compression scheme, applying a rendering method that exploits the pre-rendered images corresponding to the partitioned cubes, which is similar to that of Chapter 2, is also possible.

Using GPU to improve the compression speed for the method in Chapter 3 is another direction for future work. Currently, on a low-end computing platform, it takes about 1 minute to compress a typical medical volume without refinement, and about 3 minutes in the case a 1-step refinement is used. Although this is generally acceptable, the compression speed can be significantly improved by performing the vector quantization using the LBG algorithm, the most time-consuming step in the scheme, using GPU. In addition, implementing the 3-D motion estimation module using GPU is possible due to the high parallelism in data access.

6.2 Transfer Function Design for Visualization of Medical Volume Data

Although being a powerful technique in medical diagnosis, the efficiency of direct volume rendering strongly depends on the transfer function. Finding appropriate transfer functions that yields the desired visual information is difficult, non-intuitive, and time-consuming. The clustering-based framework introduced in Chapter 4 is suitable for medical imaging applications since it supports both automatic and semi-automatic generation of transfer functions with the comparable visualization results to existing state-of-the-art approaches. The use of LH values in this method provides an unambiguous classification of boundaries. The multi-step clustering process incorporates LH and spatial information to cluster and identify complex material boundaries, while the automatic transfer function design module is able to assign good transfer functions such that boundaries are not occluded. The proposed system also preserves a high degree of freedom for the user to adjust the rendering results; thus, it is suitable for both experienced and non-experienced users.

The visual quality of the proposed system can be improved if we introduce an *importance* value to each cluster and enhance the rendering process so that important parts are emphasized or focused in the image space. The assignment of importance values is done manually only once for each given medical task and can be reused in future. By using importance values, the optical properties can be *locally* manipulated, leading to possibilities to render important parts with more details and higher contrast than the rest. This manipulation can be done by reducing the opacities of sample points having low importance values, darkening the outline of more important parts, and using no shading for unimportant parts and more complex shading models for important parts, respectively.

The application of LH values is not limited to static medical volume data. A research topic that may be interesting is the application of the LH feature domain in automatic design of transfer functions for visualization of dynamic medical volume data. This may be helpful in diagnostic of chemotherapy drugs injection, in which the blood vessels and the organs of interested are captured through multiple time-steps. Since LH values can represent the material boundaries well, the movement of the drugs flow can be modeled by examining the changes of the LH histogram along the time.

6.3 Vasculature and Flow Rendering for Medical Simulation

In Chapter 5, we proposed a physics-based method for rendering the flow particles in the simulation of chemotherapy drugs injection. The hepatic vessels are extracted from clinical CT images using 3-D region growing, and skeletonized using a 3-D thinning algorithm. The resultant skeleton is refined to be of unit pixel width by a post processing step. The vasculature are reconstructed using cubic b-splines and represented as generalized cylinders. The cubic b-splines method enhances the smoothness of the rendered vessels and is more computational efficient compared to conventional methods. The flow is modeled using Hagen-Poiseuille Flow and simulated by rendering the quadrilaterals which are aligned along the viewing direction. This approach has capability to create simple but realistic motions of blood flow. It can also be combined with a fast volume rendering algorithm to provide more context information of the scene.

Three-D region growing algorithm is used in our method due to its simplicity and computational efficiency. However, it requires a number of pre-defined parameters and the segmentation result may be sensitive to these parameters, especially in noisy datasets. More advanced image segmentation algorithms, for example level set methods (Li et al., 2012), could be used to improve the segmentation result. In addition, blood flow dynamics is highly complex. In future, we will integrate a more accurate blood flow model in our simulation system.

Bibliography

Alderliesten, T., Konings, M. K., Niessen, W. J., 2004. Simulation of minimally invasive vascular interventions for training purposes. Computer Aided Surgery 9 (1), 3–15.

Andrew, A., 1979. Another efficient algorithm for convex hulls in two dimensions. Information Processing Letters 9 (5), 216–219.

Bajaj, C. L., Pascucci, V., Schikore, D. R., 1997. The contour spectrum. In: Proceedings of IEEE Visualization. pp. 167–173.

Chan, M.-Y., Wu, Y., Mak, W.-H., Chen, W., Qu, H., 2009. Perception-based transparency optimization for direct volume rendering. IEEE Transactions on Visualization and Computer Graphics 15 (6), 1077–2626.

Chrysafis, C., Said, A., Drukarev, A., Islam, A., Pearlman, W. A., Jun 2000. SBHP-a low complexity wavelet coder. In: Proceedings of the 2000 IEEE International Conference on Acoustics, Speech, and Signal Processing. Vol. 4 of ICASSP '00. pp. 2035–2038.

Chui, C.-K., Cai, Y., Wang, Z., Ye, X., Anderson, J. H., Teoh1, S.-H., Sakuma, I., 2005. Flow visualization for interactive simulation of drugs injection during chemoembolization. In: Westwood, J. D., Haluck, R. S., Hoffman, H. M., Mogel,

G. T., Phillips, R., Robb, R. A., Vosburgh, K. G. (Eds.), Medicine Meets Virtual Reality 14: Accelerating Change in Healthcare: Next Medical Toolkit. Vol. 119 of Studies in Health Technology and Informatics. IOS Press, pp. 99–101.

Chui, C.-K., Li, Z., Anderson, J. H., Murphy, K., Venbrux, A., Ma, X., Wang, Z., Gailloud, P., Cai, Y., Wang, Y., Nowinski, W., 2002. Training and pretreatment planning of interventional neuroradiology procedures–initial clinical validation. Vol. 85 of Studies in Health Technology and Informatics. pp. 96–102.

Correa, C. D., Ma, K.-L., 2008. Size-based transfer functions: A new volume exploration technique. IEEE Transactions on Visualization and Computer Graphics 14 (6), 1380–1387.

Correa, C. D., Ma, K.-L., 2009a. The occlusion spectrum for volume visualization and classification. IEEE Transactions on Visualization and Computer Graphics 15 (6), 1465–1472.

Correa, C. D., Ma, K.-L., 2009b. Visibility driven transfer functions. In: IEEE Pacific Visualization Symposium. pp. 177–184.

Correa, C. D., Ma, K.-L., 2011. Visibility histograms and visibility-driven transfer functions. IEEE Transactions on Visualization and Computer Graphics 17 (2), 1077–2626.

Duda, R. O., E.Hart, P., Stork, D. G., 2001. Pattern Classification, 2nd Edition. Wiley-Interscience.

Ellsworth, D., Chiang, L.-J., Shen, H.-W., 2000. Accelerating time-varying hardware volume rendering using TSP trees and color-based error metrics. In: Proceedings of the IEEE Symposium on Volume Visualization 2000 (VVS'00). pp. 119–128.

Farin, G. E., 1999. Curves and Surfaces for Computer-Aided Geometric Design: A Practical Guide, 4th Edition. Academic Press.

Fukunaga, K., Larry, H. D., 1975. The estimation of the gradient of a density function, with applications in pattern recognition. IEEE Transactions on Information Theory 21 (1), 32–40.

Ghanbari, M., Jul 1990. The cross-search algorithm for motion estimation. IEEE Transactions on Communications 38 (7), 950–953.

Guthe, S., Straβer, W., Oct 2001. Real-time decompression and visualization of animated volume data. In: Proceedings of the 2001 IEEE International Conference on Visualization. VIS '01. pp. 349–356.

Hadwiger, M., Fritz, L., Rezk-Salama, C., Höllt, T., Geier, G., Pabel, T., 2008. Interactive volume exploration for feature detection and quantification in industrial CT data. IEEE Transactions on Visualization and Computer Graphics 14 (6), 1507–1514.

Hladuvka, J., König, A., Gröller, E., 2000. Curvature-based transfer functions for direct volume rendering. In: Proceedings of Spring Conference on Computer Graphics. pp. 58–65.

Hong, D., Ning, G., Zhao, T., Zhang, M., Zheng, X., 2003. Method of normal estimation based on approximation for visualization. Journal of Electronic Imaging 12 (3), 470–477.

Huang, Y.-W., Chen, C.-Y., Tsai, C.-H., Shen, C.-F., Chen, L.-G., Mar 2006. Survey on block matching motion estimation algorithms and architectures with new results. The Journal of VLSI Signal Processing 42 (3), 297–320.

Jain, J. R., Jain, A. K., Dec 1981. Displacement measurement and its application in interframe image coding. IEEE Transactions on Communications 29 (12), 1799–1808.

Kassim, A. A., Yan, P., Lee, W. S., Sengupta, K., Mar 2005. Motion compensated lossy-to-lossless compression of 4-D medical images using integer wavelet transforms. IEEE Transactions on Information Technology in Biomedicine 9 (1), 132–138.

Kindlmann, G., Durkin, J. W., 1998. Semi-automatic generation of transfer functions for direct volume rendering. In: Proceedings of IEEE Symposium on Volume Visualization. pp. 79–86.

Kindlmann, G., Whitaker, R., Tasdizen, T., Möller, T., 2003. Curvature-based transfer functions for direct volume rendering: Methods and applications. In: Proceedings of IEEE Symposium on Volume Visualization. pp. 513–520.

Kniss, J., Kindlmann, G., Hansen, C., 2001. Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In: Proceedings of IEEE Symposium on Volume Visualization. pp. 255–262.

Kniss, J., Kindlmann, G., Hansen, C., 2002. Multi-dimensional transfer functions for interactive volume rendering. IEEE Transactions on Visualization and Computer Graphics 8 (3), 270–285.

Koga, T., Iinuma, K., Hirano, A., Iijima, Y., Ishiguro, T., Nov 1981. Motion compensated interframe coding for video conferencing. In: Proceedings of the National Telecommunications Conference. Vol. 4 of NTC '81. pp. G5.3.1–G5.3.5.

Lalgudi, H. G., Bilgin, A., Marcellin, M. W., Nadar, M. S., Sep 2005a. Compression of fMRI and ultrasound images using 4D SPIHT. In: Proceedings of the 2005

IEEE International Conference on Image Processing. Vol. 2 of ICIP 2005. pp. 746–749.

Lalgudi, H. G., Bilgin, A., Marcellin, M. W., Tabesh, A., Nadar, M. S., Trouard, T. P., Feb 2005b. Four-dimensional compression of fMRI using JPEG2000. In: SPIE International Symposium on Medical Imaging. pp. 1028–1037.

Lee, H., Kim, Y., Riskin, E. A., Rowberg, A. H., Frank, M. S., Jun 1995. A predictive classified vector quantizer and its subjective quality evaluation for X-ray CT images. IEEE Transactions on Medical Imaging 14 (2), 397–406.

Lee, H., Kim, Y., Rowberg, A. H., Riskin, E. A., Sep 1993. Statistical distributions of DCT coefficients and their application to an interframe compression algorithm for 3-D medical images. IEEE Transactions on Medical Imaging 12 (3), 478–485.

Levoy, M., Mar 1990. A hybrid ray tracer for rendering polygon and volume data. Computer Graphics and Applications 10 (2), 33–40.

Li, B. N., Chui, C.-K., Chang, S., Ong, S.-H., Aug 2012. A new unified level set method for semi-automatic liver tumor segmentation on contrast-enhanced CT images. Expert Systems with Applications 39 (10), 9661–9668.

Li, R., Zeng, B., Liou, M. L., Aug 1994. A new three-step search algorithm for block motion estimation. IEEE Transactions on Circuits and Systems for Video Technology 4 (4), 438–442.

Liao, S.-K., Lai, J. Z. C., Chung, Y.-C., Apr 2004. Time-critical rendering for time-varying volume data. Computers & Graphics 28 (2), 279–288.

Liao, S.-K., Lin, C.-F., Chung, Y.-C., Lai, J. Z., Jun 2003. A differential volume rendering method with second-order difference for time-varying volume data. Journal of Visual Languages & Computing 14 (3), 233–254.

Linde, Y., Buzo, A., Gray, R. M., Jan 1980. An algorithm for vector quantizer design. IEEE Transactions on Communications 28 (1), 84–95.

Liu, L.-K., Feig, E., Aug 1996. A block-based gradient descent search algorithm for block motion estimation in video coding. IEEE Transactions on Circuits and Systems for Video Technology 6 (4), 419–422.

Liu, Y., Pearlman, W. A., Mar 2007. Four-dimensional wavelet compression of 4-D medical images using scalable 4-D SBHP. In: Proceedings of the 2007 Data Compression Conference. DCC '07. pp. 233–242.

Lum, E. B., Ma, K.-L., 2004. Lighting transfer functions using gradient aligned sampling. In: Proceedings of IEEE Visualization. pp. 289–296.

Lundström, C., Ljung, P., Ynnerman, A., 2005. Extending and simplifying transfer function design in medical volume rendering using local histograms. In: Proceedings of IEEE/Eurographics Symposium on Visualization. pp. 263–270.

Lundström, C., Ljung, P., Ynnerman, A., 2006a. Local histograms for design of transfer functions in direct volume rendering. IEEE Transactions on Visualization and Computer Graphics 12 (6), 1570–1579.

Lundström, C., Ynnerman, A., Ljung, P., Persson, A., Knutsson, H., 2006b. The α-histogram: Using spatial coherence to enhance histograms and transfer function design. In: Proceedings of IEEE/Eurographics Symposium on Visualization. pp. 227–234.

Maciejewski, R., Chen, W., Woo, I., Ebert, D. S., 2009. Structuring feature space - a non-parametric method for volumetric transfer function generation. IEEE Transactions on Visualization and Computer Graphics 15 (6), 1473–1480.

Menegaz, G., Thiran, J.-P., Mar 2003. Three-dimensional encoding/two-dimensional decoding of medical data. IEEE Transactions on Medical Imaging 22 (3), 424–440.

Miaou, S.-G., Chen, S.-T., Nov 2004. Automatic quality control for wavelet-based compression of volumetric medical images using distortion-constrained adaptive vector quantization. IEEE Transactions on Medical Imaging 23 (11), 1417–1429.

Mohsenian, N., Nosratinia, A., Liu, B., Orchard, M. T., Dec 1995. Adaptive entropy constrained transform coding of magnetic resonance image sequences. IEEE Transactions on Nuclear Science 42 (6), 2309–2316.

Nagayasu, D., Ino, F., Hagihara, K., 2008. A decompression pipeline for accelerating out-of-core volume rendering of time-varying data. Computers & Graphics 32 (3), 350–362.

Nguyen, B. P., Chui, C.-K., Ong, S.-H., Chang, S., Nov 2009. Vascular flow rendering for interactive simulation of contrast and drugs injection. In: Proceedings of the IEEE Region 10 Conference 2009 (TENCON 2009). pp. 1–5.

Nguyen, B. P., Chui, C.-K., Ong, S.-H., Chang, S., Sep 2011a. An efficient compression scheme for 4-D medical images using hierarchical vector quantization and motion compensation. Computers in Biology and Medicine 41 (9), 843–856.

Nguyen, B. P., Tay, W.-L., Chui, C.-K., Ong, S.-H., Jun 2011b. Automatic transfer function design for volumetric data visualization using clustering on LH space. In: Proceedings of Computer Graphics International 2011 (CGI 2011). pp. 1–10.

Nguyen, B. P., Tay, W.-L., Chui, C.-K., Ong, S.-H., 2011c. Automatic transfer function design for volumetric data visualization using clustering on LH space. In: Proceedings of Computer Graphics International. pp. 1–10.

Nguyen, B. P., Tay, W.-L., Chui, C.-K., Ong, S.-H., Feb 2012. A clustering-based system to automate transfer function design for medical image visualization. The Visual Computer 28 (2), 181–191.

Palágyi, K., Kuba, A., Jul 1999. A parallel 3D 12-subiteration thinning algorithm. Graphical Models and Image Processing 61 (4), 199–221.

Pekar, V., Wiemker, R., Hempel, D., 2001. Fast detection of meaningful isosurfaces for volume data visualization. In: Proceedings of IEEE Visualization. pp. 223–230.

Po, L.-M., Ma, W.-C., Jun 1996. A novel four-step search algorithm for fast block motion estimation. IEEE Transactions on Circuits and Systems for Video Technology 6 (3), 313–317.

Porter, T., Duff, T., Jul 1984. Compositing digital images. Computer Graphics 18 (3), 253–259.

Praßni, J.-S., Ropinski, T., Hinrichs, K. H., 2009. Efficient boundary detection and transfer function generation in direct volume rendering. In: Proceedings of the 14th International Fall Workshop on Vision, Modeling, and Visualization (VMV09). pp. 285–294.

Pratt, M. A., Chu, C. H., Wong, S., Aug 1996. Volume compression of MRI data using zerotrees of wavelet coefficients. In: Wavelet Applications in Signal and Image Processing IV. Vol. 2825 of Proceedings of SPIE. Denver, CO, USA, pp. 752–763.

Röttger, S., Bauer, M., Stamminger, M., 2005. Spatialized transfer functions. In: Proceedings of IEEE/Eurographics Symposium on Visualization. pp. 271–278.

Said, A., 2003. Arithmetic coding. In: Sayood, K. (Ed.), Lossless Compression Handbook. Academic Press, San Diego, CA, USA, Ch. 5, pp. 101–152.

Said, A., Pearlman, W. A., Jun 1996. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. IEEE Transactions on Circuits and Systems for Video Technology 6 (3), 243–250.

Sanchez, V., Nasiopoulos, P., Abugharbieh, R., Jul 2008. Efficient lossless compression of 4-D medical images based on the advanced video coding scheme. IEEE Transactions on Information Technology in Biomedicine 12 (4), 442–446.

Sanchez, V., Nasiopoulos, P., Abugharbieh, R., Jul 2009. Novel lossless fMRI image compression based on motion compensation and customized entropy coding. IEEE Transactions on Information Technology in Biomedicine 13 (4), 645–655.

Schelkens, P., Munteanu, A., Barbarien, J., Galca, M., Giro-Nieto, X., Cornelis, J., Mar 2003. Wavelet coding of volumetric medical datasets. IEEE Transactions on Medical Imaging 22 (3), 441–458.

Schirski, M., Kuhlen, T., Hopp, M., Adomeit, P., Pischinger, S., Bischof, C., 2004. Efficient visualization of large amounts of particle trajectories in virtual environments using virtual tubelets. In: Proceedings of the 2004 ACM SIGGRAPH International Conference on Virtual Reality Continuum and Its Applications in Industry (VRCAI'04). pp. 141–147.

Schneider, J., Westermann, R., Oct 2003. Compression domain volume rendering. In: Proceedings of the 2003 IEEE International Conference on Visualization. VIS '03. pp. 293–300.

Schroeder, W., Martin, K., 1998. The Visualization Toolkit: An Object-Oriented Approach to 3-D Graphics, 2nd Edition. Prentice-Hall, Inc.

Shapiro, J. M., Dec 1993. Embedded image coding using zerotrees of wavelet coefficients. IEEE Transactions on Signal Processing 41 (12), 3445–3462.

Shen, H.-W., Chiang, L.-J., Ma, K.-L., 1999. A fast volume rendering algorithm for time-varying fields using a time-space partitioning (tsp) tree. In: Proceedings of the IEEE International Conference on Visualization 1999 (VIS'99). pp. 371–545.

Shen, H.-W., Johnson, C. R., 1994. Differential volume rendering: a fast volume visualization technique for flow animation. In: Proceedings of the IEEE International Conference on Visualization 1994 (VIS'94). pp. 180–187.

Sutera, S. P., Skalak, R., 1993. The history of Poiseuille's law. Annual Review of Fluid Mechanics 25, 1–19.

Tappenbeck, A., Preim, B., Dicken, V., 2006. Distance-based transfer function design: Specification methods and applications. In: Proceedings of Simulation and Visualization. pp. 259–274.

Tzeng, F.-Y., Ma, K.-L., 2004. A cluster-space visual interface for arbitrary dimensional classification of volume data. In: Proceedings of IEEE/Eurographics Symposium on Visualization. pp. 17–24.

Šereda, P., Bartroli, A. V., Serlie, I. W. O., Gerritsen, F. A., 2006a. Visualization of boundaries in volumetric data sets using LH histograms. IEEE Transactions on Visualization and Computer Graphics 12, 208–218.

Šereda, P., Vilanova, A., Gerritsen, F. A., 2006b. Automating transfer function design for volume rendering using hierarchical clustering of material boundaries. In: Proceedings of IEEE/Eurographics Symposium on Visualization. pp. 243–250.

Wang, Z., Chui, C.-K., Cai, Y., Ang, C.-H., Teoh, S.-H., Apr 2006. Dynamic linear level octree-based volume rendering methods for interactive microsurgical simulation. International Journal of Image and Graphics 6 (2), 155–172.

Wang, Z., Nguyen, B. P., Chui, C.-K., Qin, J., Ang, C.-H., Ong, S.-H., Jun 2010. An efficient clustering method for fast rendering of time-varying volumetric medical data. The Visual Computer 26 (6–8), 1061–1070.

Wesarg, S., Kirschner, M., 2009. Structure size enhanced histogram. In: Brauer, W., Meinzer, H.-P., Deserno, T. M., Handels, H., Tolxdorff, T. (Eds.), Bildverarbeitung für die Medizin 2009. Informatik aktuell. Springer Berlin Heidelberg, pp. 16–20.

Wesarg, S., Kirschner, M., Khan, M. F., 2010. 2D histogram based volume visualization: combining intensity and size of anatomical structures. International Journal of Computer Assisted Radiology and Surgery, 1–12.

Wilhelms, J., Gelder, A. V., 1994. Multi-dimensional trees for controlled volume rendering and compression. In: Proceedings of the 1994 Symposium on Volume Visualization. VVS '94. pp. 27–34.

Wu, X., Allard, J., Cotin, S., Oct 2007. Real-time modeling of vascular flow for angiography simulation. In: Ayache, N., Sébastien Ourselin, A. J. M. (Eds.), Proceedings of the 10th International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI 2007). Vol. 4791 of Lecture Notes in Computer Science. Springer, pp. 557–565.

Wu, X., Qiu, T., Jun 2005. Wavelet coding of volumetric medical images for high throughput and operability. IEEE Transactions on Medical Imaging 24 (6), 719–727.

Wu, Y.-G., Tai, S.-C., Sep 2001. Medical image compression by discrete cosine transform spectral similarity strategy. IEEE Transactions on Information Technology in Biomedicine 5 (3), 236–243.

Xiong, Z., Wu, X., Cheng, S., Hua, J., Mar 2003. Lossy-to-lossless compression of medical volumetric data using three-dimensional integer wavelet transforms. IEEE Transactions on Medical Imaging 22 (3), 459–470.

Zeng, L., Jansen, C. P. J., Marsch, S., Unser, M., Hunziker, P. R., Sep 2002. Four-dimensional wavelet compression of arbitrarily sized echocardiographic data. IEEE Transactions on Medical Imaging 21 (9), 1179–1187.

Zhang, T., Ramakrishnan, R., Livny, M., 1996. BIRCH: an efficient data clustering method for very large databases. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'96). pp. 103–114.

Zhou, F., Zhao, Y., Ma, K.-L., Sep 2010. Parallel mean shift for interactive volume segmentation. In: Wang, F., Yan, P., Suzuki, K., Shen, D. (Eds.), Proceedings of the First International Workshop on Machine Learning in Medical Imaging (MLMI 2010). Vol. 6357 of Lecture Notes in Computer Science. Springer, pp. 67–75.

Zhu, C., Lin, X., Chau, L.-P., May 2002. Hexagon-based search pattern for fast block motion estimation. IEEE Transactions on Circuits and Systems for Video Technology 12 (5), 349–355.

Zhu, S., Ma, K.-K., Sep 1997. A new diamond search algorithm for fast block matching motion estimation. In: Proceedings of the International Conference on Information, Communications and Signal Processing. Vol. 1 of ICICS '97. pp. 292–296.

List of Publications

Journal Papers

1. Binh P. Nguyen, Wei-Liang Tay, Chee-Kong Chui, and Sim-Heng Ong. A Clustering-Based System to Automate Transfer Function Design for Medical Image Visualization. *The Visual Computer*, 28(2):181-191, Feb 2012.

2. Binh P. Nguyen, Chee-Kong Chui, Sim-Heng Ong, and Stephen Chang. An Efficient Compression Scheme for 4-D Medical Images using Hierarchical Vector Quantization and Motion Compensation. *Computers in Biology and Medicine*, 41(9):843-856, Sep 2011.

3. Zhenlan Wang, Binh P. Nguyen, Chee-Kong Chui, Jing Qin, Chuan-Heng Ang, and Sim-Heng Ong. An Efficient Clustering Method for Fast Rendering of Time-varying Volumetric Medical Data. *The Visual Computer*, 26(6-8):1061-1070, Jun 2010.

Conference Papers

1. Binh P. Nguyen, Wei-Liang Tay, Chee-Kong Chui, and Sim-Heng Ong. Automatic Transfer Function Design for Volumetric Data Visualization using

Clustering on LH Space. In *Proceedings of Computer Graphics International 2011 (CGI 2011)*, pages 1-10, Ottawa, Ontario, Canada, 12-15 Jun 2011.

2. Binh P. Nguyen, Trang T. T. Do, Chee-Kong Chui, and Sim-Heng Ong. Prediction-based Directional Search for Fast Block-Matching Motion Estimation. In *Proceedings of the Symposium on Information and Communication Technology (SoICT 2010)*, ACM ICPS 449, pages 86-91, Hanoi, Vietnam, 27-28 Aug 2010.

3. Jing Qin, Wai-Man Pang, Binh P. Nguyen, Dong Ni, and Chee-Kong Chui. Particle-Based Simulation of Blood Flow and Vessel Wall Interactions in Virtual Surgery. In *Proceedings of the Symposium on Information and Communication Technology (SoICT 2010)*, ACM ICPS 449, pages 128-133, Hanoi, Vietnam, 27-28 Aug 2010.

4. Binh P. Nguyen, C. K. Chui, S. H. Ong, and Stephen Chang. Vascular Flow Rendering for Interactive Simulation of Contrast and Drugs Injection. In *Proceedings of the IEEE Region 10 Conference 2009 (TENCON 2009)*, pages 1-5, Singapore, 23-26 Nov 2009.

5. Bing Nan Li, Phu Binh Nguyen, Jing Qin, Liang Jing Yang, S.H. Ong, Stephen Chang, and Chee-Kong Chui. Image Processing and Modeling for Active Needle Steering in Liver Surgery. In *Proceedings of the International Asia Conference on Informatics in Control, Automation and Robotics 2009 (CAR 2009)*, pages 306-310, Bangkok, Thailand, 1-2 Feb 2009.

6. Phu Binh Nguyen, Tao Yang, Florence Leong, Stephen Chang, S.H. Ong, and Chee-Kong Chui. Patient Specific Biomechanical Modeling of Hepatic

Vasculature for Augmented Reality Surgery. In *Proceedings of the 4th International Workshop on Medical Imaging and Augmented Reality (MIAR 2008)*, pages 1-9, Tokyo, Japan, 1-2 Aug 2008.