Volume Quantification and Visualization for Spinal Bone Cement Injection

Xie Kai

A dissertation submitted to
the University of Hong Kong
in partial fulfillment of the requirements for
the degree of Master of Philosophy

August 2002

Acknowledgements

I would like to thank my supervisor, Dr. Wenping Wang, for his guidance through the course of my two-year studies in the HKU. He is kind and has inspired me a lot during these two years. He has also given me so many invaluable advices in the research. I have learned much from him. I could hardly finish my research without his help and encouragement.

I would also like to thank my co supervisor, Dr. W. W. Lu who has given me so many professional advices in medical disciplines. He has always tried his best to give me a help whenever I needed the cooperation with the Department of Orthopaedic Surgery.

I will thank my supervisors, Prof. Z. G. Wang and Prof. Q. Y. Ye in Shanghai Jiaotong University. I gained much under their supervision.

I would like to express my gratitude to the members of the graphics group. They are: Y. K. Choi, X. Y. Zhou, C. H. Tu, C. Hui, X. T. Wang, B. Chen, T. Wang, B. Wang, H. P. Yang, Kam Wong, Dominic Cheng and Kelvin Lee. The friendship with them is the most precious gift I received. I also thank my best friends: J. Wang, X. H. Lin, L. Wang, Z. Li, Alex Cheung, who have spent happy hours with me during two years. It could be difficult under pressure without their help and warm concern.

I will express deep appreciations to my parents, X. L. Pang and J. L. Xie, and my deceased grandmother, for their unfailing support. Finally, I would like to thank my girl friend, J. L. Lin, who is also studying in the field of computer science. Thanks for her technique advices and femininity.

To my grandmother

and

J. L. Lin

Declaration

I hereby declare that the dissertation, submitted in partial requirement for the degree of Master of Philosophy and entitled "Volume Quantification and Visualization for Spinal Bone Cement Injection", represents my own work and has not been previously submitted to this or any other institution for any degree, diploma, or other qualification.

Xie Kai

August 2002

Abstract of thesis entitled

"Volume Quantification and Visualization

for Spinal Bone Cement Injection"

submitted by Xie Kai

for the degree of Master of Philosophy

at the University of Hong Kong in August 2002

Imaging techniques, like X-ray, Magnetic Resonance Image (MRI) and Computer Tomography (CT), are common place in medicine nowadays. These instruments make it easier for surgeons and doctors to diagnose diseases and increase the ratio of success in surgery, as they enable internal human structures to be observed. Computer graphics, one of the branches of Computer Science, is commonly used in medical disciplines. It reconstructs the 2-dimension image slices obtained by CT or MRI to 3-dimensionel models, making it easier for surgeons and doctors to observe internal organs.

Applications of computer graphics can not only visualize medical data, but also quantify and evaluate it specified usages. A joint research project was conducted with medical specialists of the Department of Orthopaedic Surgery at the University of Hong Kong. A material called bone cement is often used with fracture patients in case where traditional medical methods do not provide satisfactory results, especially in the case of spinal fractures. Doctors and surgeons are very interested in the qualification of bone cement injection, but they must depend on their experience to distinguish bone cement from other tissues in 2-dimention CT slices. There is therefore a need for the development of a computational method which classifies the bone cement automatically. However, the complicated shape and the very similar densities of bone cement to parts of other tissues makes such a task very difficult. A comparison between the sample before the injection and after the injection of bone cement is needed in order to collect the characteristics of the shape of the bone cement in the spine.

In this thesis, a series of methods were developed to compare two such samples. Two major steps were involved in this procedure: Feature point detection and Point alignment. In the first step, groups of feature points were detected in both volume data sets by a statistical method which differs from the traditional feature point detection method. The second step aimed at aligning two point sets in 3-dimensionel space approximately. A method requiring only a few points among the point sets to be matched was designed to save as much running time as possible without significant loss of accuracy. Finally, two volume data sets were aligned by the matrix obtained by the second step before the comparison. Thus enables the shape of the bone cement to be easily classified by comparing the volumes voxel by voxel. The result can be more easily evaluated by the surgeons.

Total word 418

Signature____________ Date________________

Chapter 1

Introduction

Computer technology has continuously changed our lives. It is very hard to imagine what the world could be without computers nowadays. These amazing machines brought to us unbelievable processing powers. The possibility of its applications is explored in every aspect. Computer graphics, which is full of gimmicky geometrical theories, is one of the branches of Computer Science. Computer graphics deals with the problems related to graphics. It reconstructs the real world on the screen and helps us do something that is unable to do in the real world. The most common usage of computer graphics is computer games, which bring not only funs to the kids but also the troubles to their parents as well. However this complicated technology is not only limited in entertainment. Another field in which it can be widely used is the medical disciplines. Life was much easier when medical imaging came into place. Imaging techniques like X-ray, Magnetic Resonance Imaging (MRI) and Computer Tomography (CT) are widely used in these days. However, the 2D static images created by these techniques are not satisfactory enough. Ambitious scientists, researchers and doctors decided to go one step further. They tried to reconstruct these 2D images into 3D images. It is not an easy task. They have done it and have reason to do so. By looking at the 3D images, doctors now can check the magnificent structure inside the human body without cutting it off. Just as what a doctor has said in the National Geographic program, “If I can see it, I can fix it.”

3-dimensional image will be helpful to orthopaedic surgery. Orthopaedic surgery involves bone injuries. Surgeons usually use screws to adhere two parts of broken bones. This method works in most of cases, but they do sometimes fail. Some of the patients, especially the elderlies whose bones are not as strong as young persons’, cannot receive such operation. Calcium is lost day after day when human is growing up. Elderlies’ bones become looser because they are lack of calcium. Such bones are not strong enough to support the screws. Sometimes the screws in such bones may be moved by outside forces that make the situation even worse. Other patients’ bones are just chapped. Screw insertion will destroy too many normal structures. For these people, surgeons must find other ways for the operations.

A new method of orthopaedic surgery, called bone cement injection, was developed which can adhere and reinforce bone fracture. Bone cement is something like a glue which can be injected into the broken bones and pervades into the gaps of the bone. It has two functions. One is to pervade into the gaps inside the bone to adhere all parts of the broken bones. The other function is to reinforce the loosen bone. Surgeons inject bone cement into the bone first and insert screws after it is solidified.

Bone cement has a lot of prescriptions. Different prescriptions of bone cement have different characteristics. The fluidity is one of the most important characteristics, which determines where the bone cement goes in the bones. Although a lot of tests have been done, the surgeons still have no idea where exactly the bone cement will be in the bones after injection. By now, surgeons can only use their experience to examine the result of injection by studying the 2D slices obtained by computer tomography (CT). Each pixel in a CT slice is assigned a value relating to the density of the tissue at that pixel location. When looking at these CT slides, surgeons use their expertise, and at the same time incorporate also their imagination to stack the 2D images into a 3D volume. However, such a visualization technique is not clear enough. Bone cement looks like a group of cloud which mixed with other tissue in the CT image. Since the density is the only information obtained by CT, the parts for bone cement can easily be mixed up with bone cortices, which are the hardest parts in the bone and whose densities are very similar to the densities of bone cement. Surgeons, especially the students of orthorpaedic surgery, are sometimes unable to distinguish the bone cement from bone cortices.

On the other hand, researchers are developing methods to separate different tissues by mathematical theory in a volume data set. In order to check the result of their methods, the best way is to compare the same sample before bone cement injection and after bone cement injection. Currently available methods and software provide such comparisons between 2D images. Very few methods have been reported for 3D volumes.

The goal of this thesis is to find a series of approaches to compare a pair of volumes that are similar to each other. The main problem is not the comparison itself but the alignment of the two volumes. The position of the sample scanned before injection is not, in general, the same as its position after injection. Alignment must be done before the comparison. The limitation of discrete data space increases the difficulty (To be discussed in chapter 3). The method introduced here includes 3 steps to reach the goal:

- Feature points detection
- Matching the two points sets in 3-dimensional space
- Alignment and Comparing

The fundamental problems of volume comparison and its solutions are discussed in this thesis. Our method aims at reaching the best balance between speed and accuracy. In addition, other previous works in alignments of two point sets are also reviewed.

The remainder of this thesis is organized as follows. In chapter 2 we shall give a brief overview of the background of the research. In chapter 3 the feature point detection method in 3D space and its background will be introduced. This is the first step of our method. The second step, alignment, will be described in details in chapter 4. The comparison is introduced in chapter 5. The thesis is concluded in chapter 6, where some possible future developments of the research are discussed.

Chapter 2

Background

2.1 From Surfaces to Volumes

The most commonly used technique in 3D software, like, PC games Quake and Need for Speed, is called Surface Modeling. The objects created by surface modeling, like cars and monsters in the games, are made up of vertices, edges and faces. They are just 2D surface description of the topology or outlook of the 3D objects. They only have a beautiful exterior and are totally empty inside the surface. That means the Lamborghini visualized on the screen has nothing inside the shell. All the parts that cannot be seen from the outside have been ignored, no V12 engine, no gearbox, no suspensions, or even no seats in the old version of Need for Speed. Actually, most of the 3D models are 3D surface models. It is natural to omit the internal parts. Since they cannot be seen from the outside, there is no need for them to be rendered. As a consequence, computational time of perspective effects and lighting effects can be saved in order to increase display speed.

All graphics accelerators specialized in graphical display can handle the above mentioned geometric primitives, such as vertices, edges and polygons, efficiently. Manufacturers keep on improving their graphical hardware to meet the market's expectations. Nowadays graphical accelerators can process and render more than 30 million triangles per second. New graphics algorithms have been developed for fast manipulation of surfaces. Surface-based rending for surface modeling is quite successful and it has made those graphics-intensive applications possible even on low-end personal computers.

Although surface-based rendering is well-developed and well-supported by hardware, it has some limitations. Surface-based rendered images cannot deal with the internal structure of an object. This limits its usage in the medical discipline, because the internal structures of a human body are the most important thing to be observed in medicine. Besides, the medical data sets including internal structures of human body are usually organized in volume data sets, which contain information for each location in space. Such organization of data set is almost impossible to be visualized using surface-based rendering. To get around this difficulty, one must impose some surface-based primitives to the data sets. An efficient algorithm is suggested in [1] for defining surfaces passing through data points of the same scalar value, which can be rendered efficiently.

Another method for visualizing volume data sets is totally different from the surface-based approach. Geometric primitives like points, lines and polygons are abandoned but voxels are used instead. Voxels become the basic elements in the new approach called volume-based rendering.

2.2 Volume Visualization

Volume visualization is the technique of rendering and manipulating volumetric data sets. Medical visualization is one of the major categories among the various applications of volume visualization. First let us take a look at the basics for volume rendering before going into the details of the volumetric data evaluation.

2.2.1 Voxels

In 2D images, the basic element is pixel. A pixel is a unit square and a 2D image is a rectangle composed of pixels. A voxel is the basic element in 3D volume data sets analogous to a pixel in a 2D image. A voxel is a unit cube in general, and a volume is a cuboid composed of voxles. While each pixel can be addressed by an (x,y) couple indicating the coordinates of the pixel axes, a voxel is addressed by an (x,y,z) triple indicating the coordinates of the voxel axes.

Each voxel in a volume stores either a scalar value, e.g. density of a bone, or a vector value, e.g. gravity direction. A volume of scalar values is called a scalar field while that of vector values is called a vector field. In this thesis, the word “volume” will be used interchangeably with “scalar field” since the volumetric data set we focus on contains scalar values, which is bone density only.

A 3D scalar volume containing all the voxels can be considered as embedding a function $f : Z^3 \to R$. If the range of f is either 0 or 1, or each voxel of a volume only stores binary value, the volume is called a binary volume. Otherwise, it is called a gray-scaled volume.

2.2.2 Rendering

Volume rendering is to generate a display of a 2D projected image of the 3D volume from a given viewpoint. To reach this goal, a lot of algorithms have been developed over the last decades. Basically, these algorithms are separated into two kinds: Direct Volume Rendering (DVR) and Indirect Volume Rendering (IVR) [2]. IVR reconstructs a surface called the iso-surface and renders such a surface by traditional surface-based approach. The Marching Cubes (MC) [1] algorithm is the most famous approach in IVR. In contrast to IVR, DVR renders the volume data directly. Voxels are the fundamental elements in DVR approaches. The Ray Casting (RC) approach [3-4], the forward projection approach

[5-6], the splatting algorithm [7], and the 3D texture mapping based projection technique [8] all belong to DVR.

RC approach is most widely used among DVR approaches. RC is so far the only approach which enables adjustable sampling in all dimensions, perspective projections, and gray-level gradients [2]. Generally, a volume dataset is viewed from a viewpoint through a view plane. The RC algorithm casts a ray from the viewpoint through each pixel on the view plane into the volume. Within the volume, samples are taken by tri-linear interpolation from the voxel values of the volume. Each sample is classified using transfer functions (see section 2.5) for R (red), G (green), B (blue) and *alpha* (opacity) [2].

A lot of optimized volume-rendering techniques that are based on fundamental rendering theories have been developed in recent years. Most of them sacrifice image quality for speed. Shear-warp rendering [9] is probably the currently fastest software method for volume rendering [10]. This method is based on a factorization of the viewing matrix into a 3D shear parallel to the slice of the volume data, a projection to form a distorted intermediate image, and a 2D warp to produce the final image. It combines the advantages of RC and the 3D texture mapping based projection technique. It achieves 1.1 Hz (1.1 frames/second) on a single 150MHz R4400 processor for 256*256*225 volume with 65 seconds of pre-processing time [11], or 10 Hz(10 frames/second) on a 16 processor SGI Challenge multiprocessor for 256*256*223 volume [9].

No matter how good or ingenious these algorithms are, they are based on software techniques that limit their performances. Some of the algorithms, like shear-warp rendering, need pre-processing, which prohibits immediate visual feedback when the parameter is changed. Some of the algorithms based on texture rendering need a large texture memory or special functions, like estimation of gradients that are required to identify surfaces for shading. On the other hand, the volume data itself is obviously large because it includes not only the outer shape of the objects but also the information about the interior as well. To implement these algorithms, a powerful computer is needed to achieve satisfactory performance. For example, VISBONE [12], which is one of the volume rendering systems developed at the University of Hong Kong in 1999, system requires an SGI Onyx2 workstation equipped with an Infinity Reality2 graphics board, 4 MIP R10000 CPUs, 512 Mbytes of main memory and 16Mbytes texture memory. This monster is as big as a low cabinet and worth HK$1,000,000 in 1998. Such a high price cannot be supported by general users. High requirements on hardware devices restrict the usage of volume rendering. For these reasons, VolumePro appears.

2.2.3 VolumePro

VolumePro is the world’s first single-chip real-time volume rendering system for consumer PCs which was developed by Mitsubishi Electronic Lab (see Figure 2.1). The first VolumePro PCI card was operational in April 1999, with 128Mbytes of volume memory and the vg500 rendering chip. It is based on the Cube-4 volume rendering architecture developed at SUNY Stony Brook [13]. “Cube-4 requires a large number of rendering and memory chips, many pins for inter-chip communication, and large one-chip storage for intermediate results.” [10]. Enhanced Memory Cube-4(EM-Cube) [14] was developed to reduce the price of this card. VolumePro supports 8-bit and 12-bit scalar volume. It can render 256*256*256 volume data at 30Hz (30 frames/second) without any pre-processing. It also implements several novel features, including gradient magnitude modulation, supersampling, supervolumes, slicing, and cropping. This system includes a software interface called Volume Library Interface (VLI), which collects C++ objects and provides the application programming interface to the VolumePro features.

VolumePro implements Ray Casting (RC) [4]. RC offers high image quality and is easy to parallelize. To achieve uniform data access, VolumePro uses a ray-casting technique with both object-order and image-order data traversal based on the shear-warp factorization of the viewing matrix. [15, 16, 9] (see Figure 2.2).

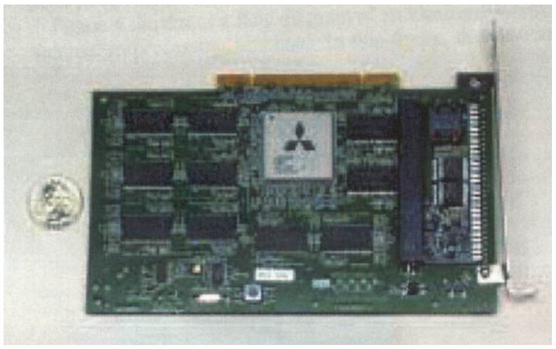


Figure 2.1
VolumePro, the world's first real-time volume rendering chip-set on PC platform.

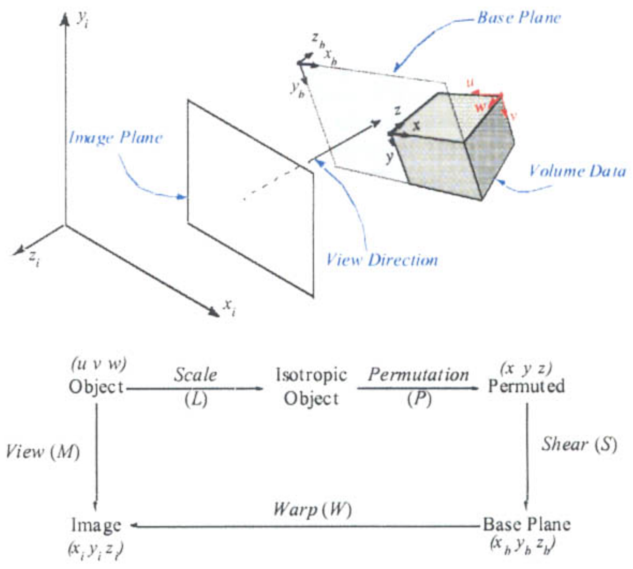


Figure 2.2
Shear-warp factorization of the viewing matrix.

Different to the shear-warp implemented by Lacroute and Levoy, VolumePro performs tri-linear interpolation to prevent view-dependent artifacts when switching between base planes and accommodates supersampling of the volume data [10]. (Supersampling improves the quality of the rendered image by sampling the volume data set at a higher frequency than the voxel spacing.) Tri-linear interpolation allows a sampled point which does not coincide with a voxel to obtain an estimation of the intensity and opacity from the voxels surrounding the sample point. It is an extension of the simple linear interpolation to the three dimensions. A linear interpolation of values v_0, v_1 at location p_0, p_1 respectively is given by:

$$v_x = \frac{x - p_0}{p_1 - p_0}(v_1 - v_0) + v_0$$

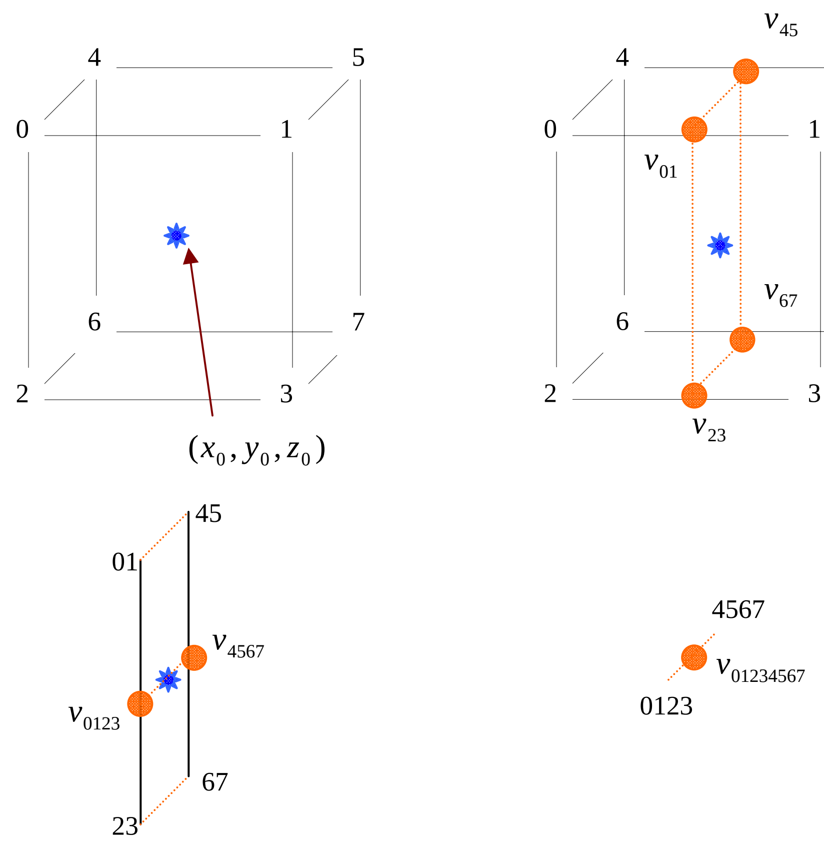


Figure 2.3
Tri-linear interpolation

where $x \in [p_0, p_1]$. Denote this linear interpolation by $v_{01} = \Lambda_{0,1}(x)$. A superscript x, y, or z will be used to represent an interpolation along a particular axis direction when 3D point locations are involved. 8 voxels that form a cube surrounding a sampled point (x_0, y_0, z_0) (see Figure 2.3(a)) are considered to obtain the interpolation voxel value of the sampled point. Three passes of linear interpolation will be performed, one for each of the three volume axes, x, y, and z. First linear interpolations along the x direction will be carried out to produce intermediate voxel values: $v_{01} = \Lambda^x_{0,1}(x_0), v_{23} = \Lambda^x_{2,3}(x_0), v_{45} = \Lambda^x_{4,5}(x_0), v_{67} = \Lambda^x_{6,7}(x_0)$ (see Figure 2.3(b)). Then it will be followed by two linear interpolations along the y direction: $v_{0123} = \Lambda^y_{01,23}(y_0), v_{4567} = \Lambda^y_{45,67}(y_0)$ (see Figure 2.3(c)). The final voxel value of the sampled point is got from the interpolation along the z direction: $v_{01234567} = \Lambda^z_{0123,4567}(z_0)$ (see Figure 2.3(d)).

2.3 Medical Data

Medical data sets are a very important category in all kinds of volume data sets. The internal structure of a human body for medical studies has been of interests for a very long time. The study of cadaver anatomy was began hundreds years ago. Scientists at that time sometimes took a great risk to study the internal structure of the human body and made anatomical atlas of human bodies by stealing bodies from tombs! Although some magnificent works of anatomical atlas have been achieved, there was still a need for sophisticated and clearly presented fine details rather than just primitive hand-sketching. Moreover, it would be impossible to cut off a living person. This situation has been changed since X-ray was discovered in the middle of 19th century. Since then, different medical imaging modalities had been developed. Computer Tomography (CT) is one of the most famous and widely used imaging modality. Magnetic Resonance Imaging (MRI) is another widely used image modality developed in the recent decade, which is specialized for soft tissues in human body for which CT cannot provide clearly imaging. Since this thesis focuses on the data of bone and bone cement, we will discuss CT system only.

2.3.1 X-ray and Computed Tomography

Computed Tomography (CT) depends on X-ray. Wilhelm Konrad Röntgen discovered X-ray involuntarily in 1895 when he was doing radio tube experimentation. He had no idea about this strange ray, so be called it X-ray. Penetration power of X-ray varies among different tissues. Tissues with higher density, like bones, absorb greater extent of X-ray than tissues with lower density, like muscles. When projecting a beam of X-ray through the human body to an X-ray sensitive photographic plate, shadows of different degrees will be formed. X-ray is widely used since it has been discovered. However it can only provide the projective images of an object.

The first CT scanner, which can provide the internal structures of an object slice by slice, was developed by Sir Godfrey Hounsfield in 1972. To produce a single CT slide, an X-ray tube rotates 360^o around the scanned object and casts X-ray through the object in different angles. A set of X-ray sensors around the object record the X-ray intensities penetrating the object. The computer performs a series of mathematical calculation called backward reconstruction on these data, and gives out an array of values. Each value represents the density at a particular location or pixel of a scanned slide. These values are called CT number or the Hounsfield numbers. A number of slides are piled up to reconstruct a complete volume data of the scanned object.

CT data is very useful to study bone tissues since the values got from CT are related to the densities of the scanned tissues themselves. The bone cement whose characteristics are very similar to bone tissues can be easily represented by CT scanners, too. In addition, CT data is a rich source for volumetric medical data. That is why CT data is chosen.

2.3.2 DICOM

CT and other digital medical imaging techniques, like MRI, have been developing quickly since 1970s. A lot of medical imaging systems have been developed by different manufacturers. These systems use different file formats and create a main obstacle in communicating, sharing and exchanging of data among themselves. Hence, a unified format for medical image data is needed.

In 1983, American College of Radiology (ACR) and National Electrical Manufacturers Association (NEMA) established a standard format for digital medical image data sets. It is called Digital Imaging and Communications in Medicine (DICOM). The goals of DICOM are to achieve compatibility and to improve workflow efficiency between imaging systems and other information systems in healthcare environments worldwide. The first version of DICOM was called ACR-NEMA 1.0 Standards Publications No. 300-1985, which was published in 1985. The second version was published three years later in 1988. This version created standardized terminology, an information structure, and unsanctioned file encoding. However most of the promise of a standard method for communicating digital image information has not been realized until the publication of the released version 3.0 in 1993. Manufacturers started to accept this standard and use it in their systems widely.

DICOM is a cooperative standard. Vendors cooperate in testing via scheduled public demonstrations, over the Internet, and during private test sessions. Every major diagnostic medical imaging vendor in the world has incorporated this standard into their product design and most are actively participating in the enhancement of the standard. Most of the professional societies throughout the world have supported and are participating in the enhancement of this standard as well.

The complicated communication function of DICOM standard is not our main focus. The only requirement to DICOM standard is that the construction of data in DICOM files made by CT system should be known so that the volume data stored in DICOM files can be exported and translated into a format that is acceptable by VolumePro.

The DICOM file is composed of a DICOM File Meta Information (FMI) and a group of data elements. DICOM File Meta Information contains the identity of data elements. It begins with 128 empty bytes followed by 4 bytes long string “DICM”. These bytes are used to distinguish DICOM file from other file formats. FMI includes other useful information, too, such as, the transmission file format, the creation application of the file, etc.

The main part of a DICOM file is the data element. Data elements include not only the medical image itself, but also the information of the patient, such as the patient's name, age, gender, and a brief case history, etc. In the part of Object Definition in the DICOM standard, it defines the details of data elements where some of them must be included in the file and others are selectable. While constructing DICOM files, one must reference the data elements definition in DICOM standard documents and include the right elements into the files and in the right order. The file created according to DICOM standard is able to be read by other application supporting DICOM file format.

Volume data is stored voxel by voxel, line by line, and slice by slice in the data part of the DICOM files. Each voxel value is 16 bits long. The voxel value is not density, but the Hounsfield numbers. In some DICOM files, the 16 bits long data is stored by a method called endian-ness, which stores the lower 8 bits in front of the higher 8 bits. There is a flag in the Object Definition to determine the storage mode. To translate the Hounsfield numbers to densities, data should be flipped to reverse the lower 8 bits and higher 8 bits,

$$v = v_L \times 256 + v_H ,$$

where v_L is the low 8 bits in each 16-bit element in a DICOM file and v_H represents the higher 8 bits. An experiential function is then applied to v to translate a 16-bit long Hounsfield number into an 8 bits long density value,

$$d(v) = \begin{cases} 0 & v = v_P \\ 255 & v > 3000 \\ \left\lfloor \frac{v}{3000} \times 256 + 0.5 \right\rfloor & otherwise \end{cases} ,$$

where $\lfloor x \rfloor$ is an operation that gets the maximum integer smaller than *x*. v_P is called the padding value. Since the volume is a cube and the space scanned by the CT scanner is a cylinder, the gap between the cube and the cylinder is padded by the padding values. Hence the density of this part is set to 0.

2.4 Bone Cement

When bone fracture occurs, there are several methods for reconnecting the broken bone. The most widely used method is using plaster, which fits lightly injured patients. For those fractures which occur in the place where plaster cannot be used, such as injures in the spine or the hip joint, surgeons sometimes can do nothing. The patients have to lie down for several weeks or months to let the bones heal up themselves. During the time on the bed, patients cannot even move a bit. It is a torture more than a therapy to the patients. If one was heavily injured, surgeons might have to insert screws into the bones to join the broken pieces together. However screw insertion has a lot of problems. First, surgeons insert the screws based on anatomic landmarks and their experience. Since all these properties vary among different individuals, surgeons must examine the 2D CT sectional images of a patient carefully before an operation to determine screw insertion position and angle [12]. The failure rate of screw insertion without computer assistance varies between 28% and 39.9% [17]. This risk was reduced after computer-assistance surgery or training system, like ANALYZE [18], appeared. The second problem is that the screw insertion method can only be used in bones with high density. The bones of young people have enough firmness to support the screw inserted into the bone structure. However for the elderlies, their bones are much looser than those of the youngsters, and are unable to stabilize the screws. The movement of the screws inserted into such bones may cause more damages. Just like a screw inserted into a loose brick wall, it will dig a big hole in the wall while being jolted. Hence surgeons developed a material called bone cement to help curing such fractures.

Bone cement is not a new method to be used clinically. It was invented about 40 years ago. The bone cement usually has two components: the powder and liquid component (see Figure2.4a). They are mixed together before being injected into the patient's bone. The mixture looks just like sticky glue (see Figure 2.4b). Its concreting time varies with different chemical compositions and the quantity of liquid component. Surgeons dig a tiny hole on the surface of the broken bone, and inject the bone cement with suitable pressure. The bone cement goes into the bone, penetrates and fills the gaps in the bone. The concreted bone cement blocks not only stick the broken pieces together, but also reinforce the structure of the bone itself. For a light fracture in positions like spine, injecting bone cement is enough to help the patient to recover. For the heavy fracture of looser bones, surgeons will insert screws after injecting the bone cement. The bone cement which is much stronger than the bone surrounding the screws, fixes them in the right position and prevent them from moving.

To improve the characteristics of bone cement, surgeons and chemical scientists have been upgrading the chemical composition of the bone cement during the past 40 years. Different chemical compositions have been tested by many manufacturers and research organizations. For example, the chemical composition of Simplex bone cement is: 75% Methyl methacrylate-Styrene-copolymer containing residual benzoyl peroxide, 15% polymethylmethacrylate, and 10% barium sulfate [19]. Some bone cements use acrylic based composition to improved handling characteristics [20]. The goal of all these researches aimed at:

- Easy handling of concreting time;
- Easy handling of fluidity;
- Adequate stiffness [21]; Durability;
- Less toxicity;
- Bioactivity and low setting temperature [21];
- Radiopacity [21].

On the other hand, surgeons and researchers are also developing mixing techniques and injection techniques to allow:

- Fewer bubbles in the bone cement;

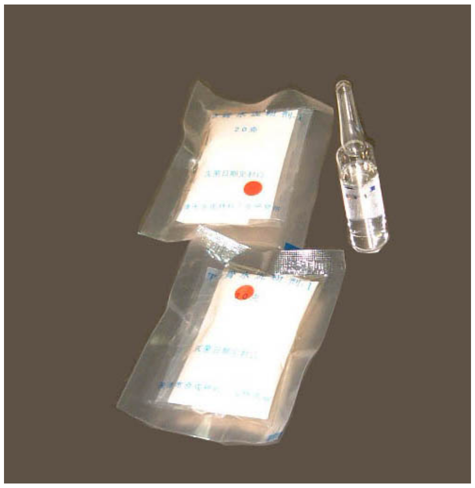
- Using minimal invasive techniques to inject bone cement for treating fractured or osteoporotic vertebral body [21];
- Easily controlled penetration.

To check the improvement of bone cement techniques, there are strong requirements to observe the shape of bone cement in the bone on clinical surgery and experimental surgery. Surgeons usually check the CT slides to observe the result of injection. It may be fast and sufficient while doing clinical operation. However it is not enough in experimental injection. Researchers need a more detailed and intuitive view of the shape of the bone cement, including its internal structure if possible. To achieve this goal, one must distinguish the part of bone cement from other tissues of the bone.

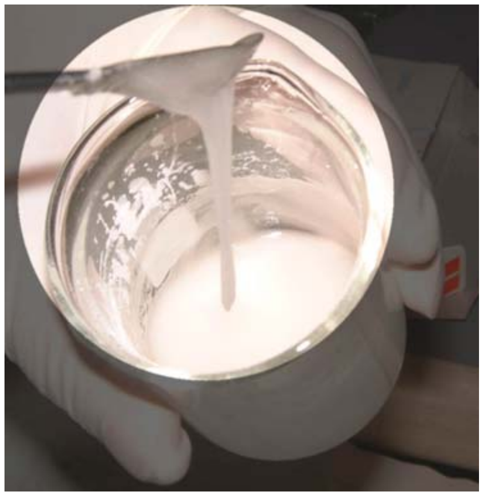
2.5 Previous Work on Tissue Classification

Conventional Tissue Classification

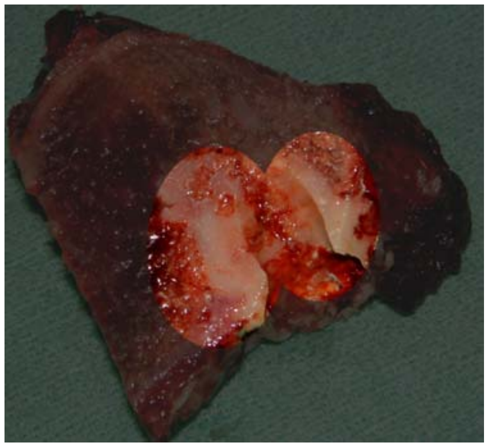
This is also called Single-Channel Tissue Classification [22]. As its name suggests, only one “channel” is used to classify the tissue in the volume data. The only “channel” is the original gray level. In the CT volume data, the original gray level is the density of the voxel. This is a very basic and obvious idea because different tissues have different densities. Bones must have higher densities than those of muscles. Even in bones themselves, the densities vary in different parts. VISBONE has already implemented this function by using specified transfer functions [12]. One can make the part with the specified density invisible. The hard part of the bone or the parenchyma surrounding the bone can be easily distinguished by densities. Figure 2.5 is a typical transfer function to classify the hard bone from parenchyma. With this function, the voxels with densities lower than a specified threshold are given a very low constant *alpha* value (opacity) in order to make them look more transparent. The voxels with densities higher than the threshold are given a linear transfer function to make them visible. A sample volume is displayed in Figure 2.6b using a specified transfer function, compared with the original image in Figure 2.6a.



(a)



(b)



(c)

Figure 2.4
Bone cement. (a) Before mixing: Powder and impregnant. (b) After mixing: glue-liked semiliquid. (c) After concreting, stone-liked solid.

Of course, users have more flexibility to adjust attributes of the transfer function. One can set more than one threshold to divide the density range into more than 2 parts. In each part of the range, different transfer functions, like linear or constant, can be set.

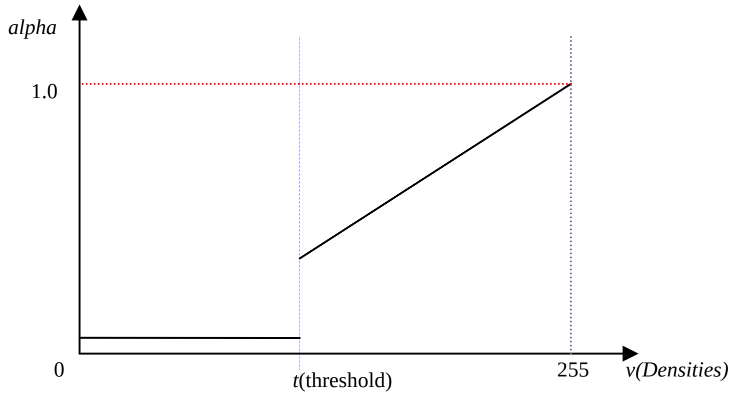


Figure 2.5

Transfer function.

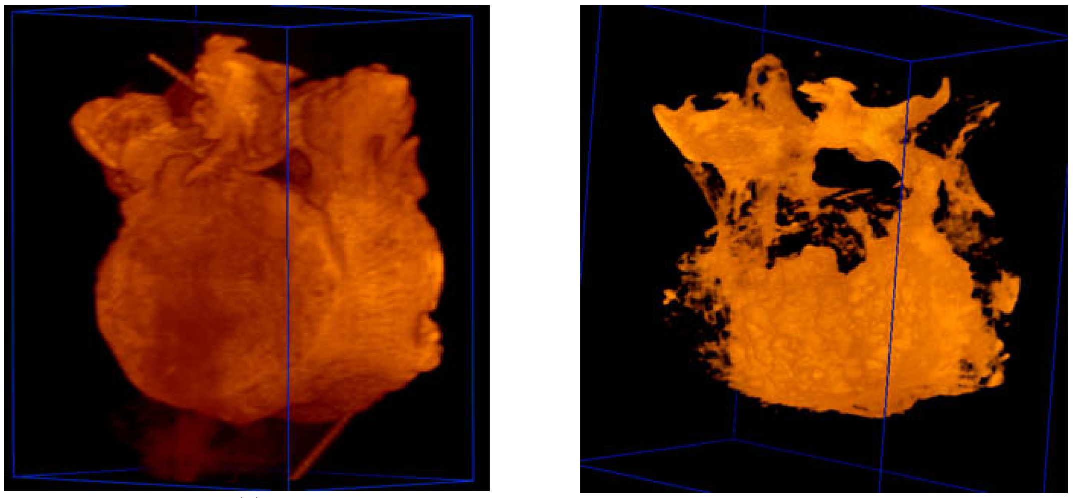


Figure 2.6
(a) Volume visualized in original way. (b) Volume visualized through specified transfer

Local Intensity Structure Tissue Classification

This approach belongs to Multi-Channel Tissue Classification. Multi-Channel Tissue Classification was first introduced to work on vector value volume sets, like MRI volumes [23][24]. The data in vector value itself has multiple values in each voxel. However for scalar value volume data, like CT data, the other "channel" other than the original value of each voxel (densities in CT data) should be "created" by mathematical approaches. Yoshinobu Sato et. al. introduced an approach to classify tissues for scalar value volume data sets [25]. The basic idea is to characterize each tissue based not only on its original intensity values, but also on its local intensity structures [26].

In [25], tissues in volume, like blood vessels, bone cortices and nodules, are characterized by line-like, sheet-like and blob-like structures. To classify these structures, 3D filters are designed, which are based on the gradient vector and the Hessian matrix of the volume density function combined with isotropic Gaussian blurring to enhance these specific 3D local density structure. Different from traditional filters, the outputs of these filters are vectors. The outputs are used as multi-channel information to classify the tissues. More complicated transfer functions are designed to suit the multi-channel outputs.

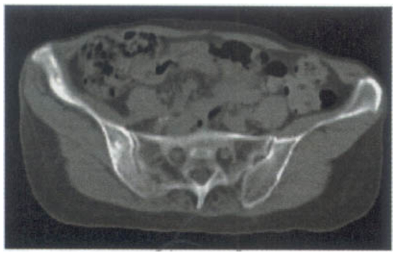
[25] shows the results of tissue classification (see Figure 2.7).

However, when conventional and local intensity structure tissue classification methods are used on bone cement, both of them fail. Bone cement has special characteristics that make neither of the classification methods work well. The density of the bone cement is almost the same as bone cortices, which is the hardest part of the bone and usually lies on the surface of the bone. Single-channel classification based on density value will obviously fail under this circumstance. On the other hand, local intensity structure based classification method also fails because the shape of the bone cement is not line-like, sheet-like or blob-like. The bone cement looks like nothing but a group of mass. Even some day a new filter is invented to be able to classify the mass-like object, there is still no way to handle the part where the bone cement joins to the bone cortices. In fact, such knds of situations frequently occurred, and the part where two materials joint together is indeed the past most interesting to the surgeons.

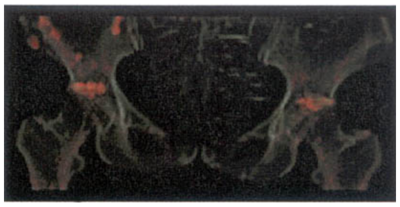
To our knowledge, there is no feasible method for separating bone cement from other tissues, and the requirement of studying the shape of bone cement injected into the bone is necessary and urgent. In order to classify the accurate shape of bone cement, CT volume data of a bone before and after the injection can be compared. Although this simple and feasible method cannot classify a sample which has no corresponding volume data before the injection, it is also helpful to the surgeons and researchers:

- To check the result of injection;
- To improve the mixing techniques in order to decrease the number of bubbles in the bone cement;
- To study the damage to the bone structure after injection;
- Help the surgery students learn how to distinguish the bone cement and other tissue;
- Help the computer science researchers develop a classification method to classify the bone cement automatically and evaluate the result of this method.

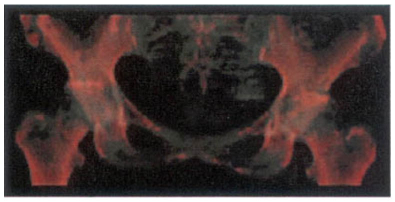
The main problems of this simple approach is that the position of the bone during the first scanning before the injection is different from its position during the second scanning after the injection. That means it is difficult to compare the two volumes directly because the relative positions of the bone in the volumes are different. The result of direct comparing of the two volumes is useless. The challenge here is therefore to align the bone in the two volumes before comparison. This is not an easy job. In the following chapters, we will focus on the difficulties of this problem and describe the steps of volume data alignment.



(a)



(b)



(c)

Figure 2.7
Visualization of pelvic bone tumors from CT data. (a) Original CT image. (b) Classification with 3D local structures. (c) Classification with original densities.

Chapter 3

Feature Points Detection in Volume Data

In this project, volume data of spines are used. A typical raw volume data contains a knot of the spine with the size of 512×512 pixels in each slice, and 30 to 40 slices in each volume. Each voxel value is 8-bit long which represents 256 scale levels. A naive way to align two volumes is by comparing voxels directly. Alternatively, some special points, called feature points, can be detected and used to guide the alignment. Feature points are those points that can point out the locations of some special features in the volume. Such locations indicate to the same positions on the object represented in the volume. Two volumes can be aligned if these feature points are aligned first. Since the two volumes represent the same bone, the only difference between these two volumes is the internal structure where bone cement has been injected. The external of the bone, and the structure of the bone cortices have not been changed after the injection. Detecting feature points in the samples are possible and most of these points should be assured to lie on the same position of both samples.

3.1 Discrete Data Space

Volume data occupies a discrete data space, which is very different form the continuous data space. Hence, it is necessary to discuss the characteristics of a discrete space first.

3.1.1 Connection in Discrete Data Space

The first thing that should be declared is that a voxel is treated as a space-occupying rectangular block in this thesis, not a point which does not occupy any space. Under this assumption, the definition of the connection in 3D discrete space is given out as follows.

A volume data set is a 3D discrete space Z^3 with a set of voxels. A voxel p is defined by its coordinates (x, y, z) in the volume. The neighborhood (u, v, w) of p is a set of voxels that can be defined by one of the following criteria:

a. Share a face with p.
b. Share a face or an edge with p.
c. Share a face, an edge, or a vertex with p.

The definition a. is called 6-connection because each voxel has 6 neighborhood voxels around it (see Figure 3.1a). The definition b. is called 18-connection because each voxel has 18 neighborhood voxels around it (see Figure 3.1b). The definition c. is called 26-connection because each voxel has all 26 neighborhood voxels around it (see Figure 3.1c). In some papers, the 6, 18, or 26-connection is called 6, 18, or 26-adjacent [27] or 6, 18, or 26-neighbors [28].

Two voxels p and q is 6, 18, or 26-connected if there is a sequence of voxels $(v_1, v_2, v_3 ... v_n)$ in the volume, such that v_i and v_{i+1} are neighbors to each other, p is the neighbor of v_1, and q is the neighbor of v_n under the definition of 6, 18, or 26-connection.

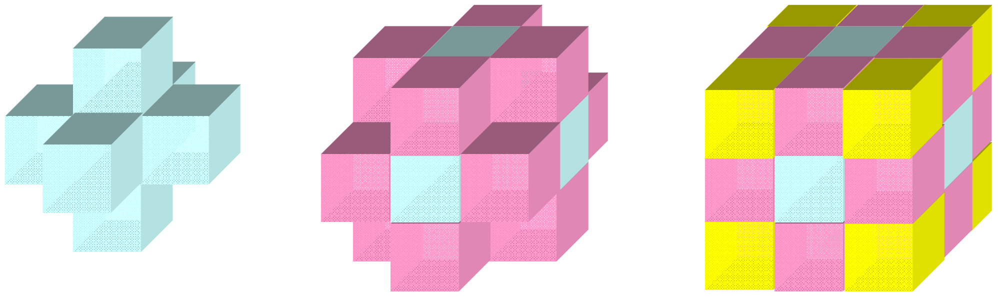

(a) (b) (c)

Figure 3.1
(a) 6-connection. (b) 18-connection. (c) 26-connection.

3.1.2 Rotation Problems in Discrete Space

The coordinates in discrete space are all integers. If the coordinates of a voxel are not integers after some types of transformations, like rotation, approximate integer coordinates will replace the decimal fractions. Problems occur because of such approximate replacement. A simple example can illustrate this problem.

Assume there is a very short line *L* in 3D discrete space, which is composed of two voxels, $l_1 = (0,0,w)$ and $l_2 = (1,0,w)$. According to their coordinates, this line is parallel to the x-axis, and orthogonal to the y- and z-axis (see Figure 3.2a). First represent them in homogeneous coordinates:

$$\vec{l}_1 = (0,0,w,1)$$
$$\vec{l}_2 = (1,0,w,1)$$

Assume it is rotated by the rotation matrix *M*, where

$$M = \begin{vmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix},$$

which rotates the line in anti-clockwise direction around the z axis.

The new line *L'* after rotated by M is:

$$\vec{l}'_1 = \vec{l}_1 M$$
$$\vec{l}'_2 = \vec{l}_2 M$$

Assume $\theta = \frac{\pi}{6}$, then $\vec{l}'_1 = \vec{l}_1 = (0,0,w,1), \vec{l}'_2 = (\frac{\sqrt{3}}{2}, \frac{1}{2}, w, 1)$. The coordinates of $\vec{l}'_2$ are obviously not integer. They will be represented as integer coordinates approximately in discrete space. The $\vec{l}'_2$ is approximately represented as $(1,1,w,1)$ here (see Figure 3.2b). Assume $\theta = \frac{\pi}{4}$ this time, then $\vec{l}'_1 = \vec{l}_1 = (0,0,w,1), \vec{l}'_2 = (\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}, w, 1)$. $\vec{l}'_2$ is supposed to be represented as $(1,1,w,1)$ again. The two different lines in continuous space now are the same in discrete space. In fact, a line like this that has only two voxels can only be represented by 26 kinds of lines after the any application of rotation matrix under the 26-connection definition. Under 6, or 18-connection definitions, the kinds of representations of the voxels with decimal-fraction coordinates are even less. Lines such as *L'* is not a line any more in 6-connection. They are only two discrete voxels.

Rotation destroys and changes the connection activity of tiny structures in discrete data space. Short lines are divided; voxels are overlapped after such transformation. All these make the feature point detection method based on measuring tiny structures in the volume fail.

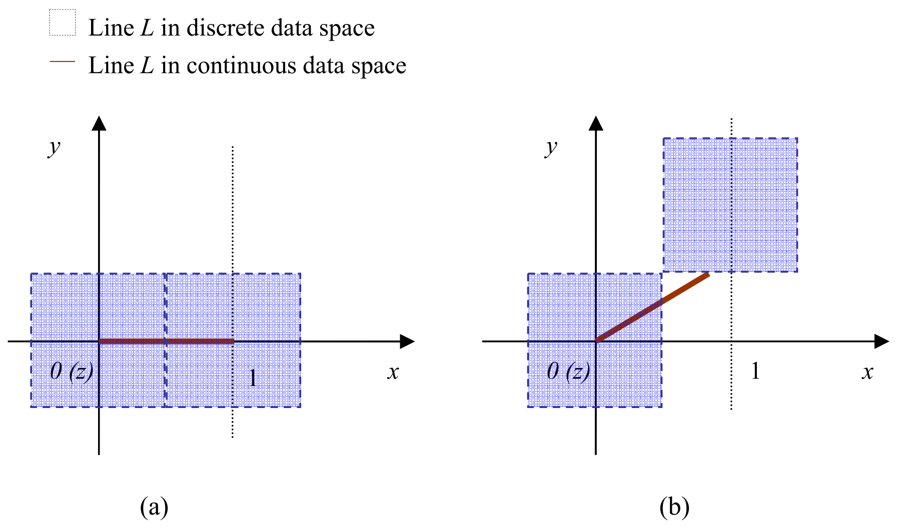

Line *L* in discrete data space
Line *L* in continuous data space
y
0 (z)
1
x
(a)
y
0 (z)
1
x
(b)

Figure 3.2
(a) Line *l*. (b) Line *l* has been rotated 30 degrees around the z axis.

3.2 Interpolation Along *z*-axis

The CT system stores the data in DICOM files. The raw data is not suitable for analysis because the volume data is not isotropic. The resolution in *x*-*y* plane, is 0.28mm per pixel. However the clearance between slices is 1.4mm minimal. It means that the resolution along the *z* axis is much lower than that along *x*- and *y*-axis. Although VolumePro is able to deal with anisotropic data and eliminates distortions automatically, anisotropic data are still not convenient to be analyzed. Four additional slices must be interpolated between each two slices in the original volumes.

Linear interpolation is used to add these additional slices for it may be the most natural interpolation. In addition, it can create a smooth gradient along the z axis, which is very important for the later works. Other interpolation, like quadric interpolation, may produce great gradient change along the z-axis, which may confuse the feature point detector confuse.

Assume that there are S_z slices in the volume, $P_0, P_1, ... P_{S_z - 1}$, consider the i^{th} and $i+1^{th}$ slices P_i, P_{i+1}, where $i = 0,1,2,..., S_z - 2$. The new slices interpolated are defined as I_1, I_2, I_3, I_4. Each pixel on I_j slice is represented by $V'_j(x, y)$ and each pixel on P_i is represented by $V_i(x, y)$. The pixels on each new slice are computed as follows,

$$V'_j(x, y) = V_i(x, y) + \frac{V_{i+1}(x, y) - V_i(x, y)}{5} \cdot j, j \in \{1,2,3,4\}$$

After interpolating new slices into the volume, the data space becomes isotropic.

3.3 Detecting Feature Points in Discrete Data Space

The structure of human bone is complicated and erose. Some features in the bone should be found as a reference to alignment, and those features should not be changed after injection of the bone cement all. The bone cortices are a good choice because they are the hardest part of the bone. They also may be the most regular parts of the bone. They are sheet-like [25], but not always in the spines. The bone cortices are not changed after injection of bone cement. The points on the sharp corners of the cortices are the most wanted feature points.

3.3.1 Previous Work on Feature Point Detection

A lot of researches on feature point detection in discrete data space have been done in the late few decades, but most of them were implemented in 2D space. Most of the researches on feature point detection are the previous step of matching two images. Researchers focused on matching 2D images rather than 3D volume or surface based data because volume visualization itself is a new branch in computer graphics, and very few papers can be found that describe the approaches to detect feature points in 3D volume data sets. Nonetheless, the approaches detecting feature points in 2D images are relevant to the detection of feature points in 3D volume.

In most of the approaches of feature detection, edges and corners are the most interested features. The traditional way of feature point detection is based on the gradient of the values of the pixels. These approaches usually get the differential coefficient or partial derivatives according to the *x* direction and the *y* direction pixel by pixel. The results are then selected by a threshold. To increase the calculation speed and predigest the procedure for programming, "filters" or "operators" are designed to implement these approaches. Filter or operator is a matrix, usually 3 by 3 or 2 by 2 large in 2D images. It is a representation of the differential coefficient or partial derivatives. Figure 3.3 shows a very simple and typical filter in 2D space that is used to detect the edge parallel to the *x*-axis.

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

Figure 3.3
A typical edge detector.

It comes from the gradient magnitude and direction of *f*:

$$grad \quad f \equiv \nabla f = (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y})$$

Consider now that only the edge parallel to the *x*-axis. The values of the pixels should change very quickly along the *y*-direction on such edges. Therefore, only the partial derivatives in the *y*-direction need to be calculated. In discrete space, it is represented by:

$$\Delta_y f(x,y) = f(x,y) - f(x,y-1)$$

The corresponding filter is:

$$\begin{bmatrix} 1 \\ -1 \end{bmatrix}.$$

Note the $\Delta_y f$ is symmetric about $(x, y - \frac{1}{2})$, so $\Delta_y f(x,y) = f(x,y+1) - f(x,y-1)$ instead. The filter becomes

$$\begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}.$$

The pixels near the target pixel should also be considered. So another two columns are added into the filter,

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}.$$

When this filter is applied to a pixel (u,v), the value is computed as follows:

$$P(u,v) = \left| \sum_{i=0}^{i=2} \sum_{j=0}^{j=2} p(u+i-1, v+j-1) f(i,j) \right|,$$

where $p(x,y)$ is the value of the pixel (u,v), and $f(x,y)$ is the value of the element at the x^{th} row and y^{th} column of the filter. A threshold is set for comparing with $P(u,v)$. Any pixel (u,v) whose $P(u,v)$ is larger than the threshold would be selected.

To measure all the edges, parallel to the *x*-axis or *y*-axis, or the edges making an angle 45 degrees to the axes, we use 4 filters:

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -1 \end{bmatrix}$$

All four filters should be applied to each pixel if of all directions need to be detected, and the maximum value of these results would be the final result.

The previous method is simple but not good for edges which are not abrupt. Some improvement methods have been developed, like Laplacian filter. The Laplacian filter depends on second order derivatives:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

The filter is:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

The Laplacian filter is much easier to use, and its effect is better too. This filter is widely used in detecting edges in 2D space. (see Figure 3.4)

There are 3D filters working in 3D discrete data space, too. [30] described an operator (filter) to detect edges in 3D discrete data space. 3D edge is an extension of the 2D edge. In 3D space, edges are considered as a plane instead of a line as in 2D space.

Two sizes of filters were mentioned in this paper. They are used in the same way as the filters in 2D space.

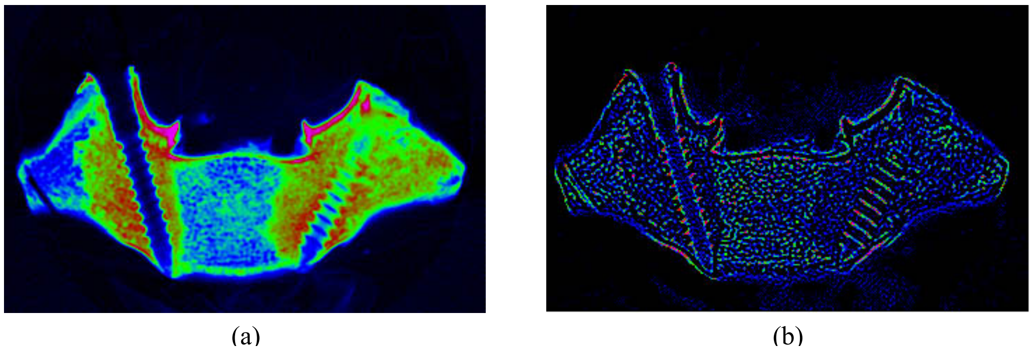


Figure 3.4
(a) Original CT image (One slice). (b) Filtered by Laplacian filter. (Just in 2D slice)

Although these filters can detect edges in 2D or 3D discrete space, the results are far from suitable for meeting the feature points which are available for matching. All the pixels or voxels selected by the filter are the group of pixels or voxels belonging to the edge. The number of them is still too large for matching. The special pixels or voxels need to be re-detected again among the group of points.

Chen introduced an approach to detect feature points in 2D images after the edges are measured [29]. Just like lots of feature points detection approaches, it was a preprocessing stage for matching two 2D images. This algorithm first find feature points belonging to the same edge by some edge detectors (filters), such as the Canny edge detector [31] that was used by Chen. The points with high-curvature are then detected. Generally, the high-curvature points are the points at the corners.

Chen first connected the pixels on the edge as a chain. Chen defined c_i to be:

$$c_i = \max_{1 \le j \le l} \{ \max \{ \delta_{i,j}, \Delta_{i,j} \} \},$$

where

$$\delta_{i,j} = \angle(p_{i-j+1} - p_{i-j}, p_{i+j+1} - p_{i+j})$$
$$\Delta_{i,j} = \angle(p_{i-j+1} - p_{i-j}, p_{i+j} - p_{i+j-1})$$

where p_i was the i^{th} point in the chain, and the l was a length parameter, which was usually set to 4 or 5.

Second, another quantity d_i was introduced, which recorded the least *j* which gave the maximum in the expression for c_i at the i^{th} point:

$d_i = j$.

When c_i attained its maximum value in [*i-l,i+l*], the i^{th} point was taken as the feature point. If there were more than one such maximum point in [*i-l,i+l*], the i^{th} maximum point with a minimum value of d_i was taken. (see Figure 3.5)

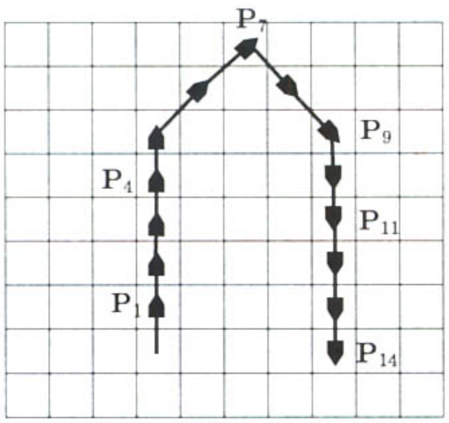


Figure 3.5
Finding point of highest curvature.

Denote $l = 5$. Points p_4 to p_9 have the same curvature $c_i = \pi$. But p_7 is selected as the point of highest curvature because it has the least value of d_i.

In 3D space, an edge is defined as a plane instead of a line. The points on the plane cannot be linked as a chain. Hence we would consider them as a net. Finding a high-curvature point through all directions on the net is not as easy as along a the line.

Due to the characteristics of 3D discrete data space, most of the traditional methods for detecting feature points cannot work well. 3D discrete data space has one more dimension than 2D discrete space. There are 3 degrees of freedom in rotation in 3D space, 2 more than that of 2D space. The problems occur because of the rotations. The main reason is that the relationship among voxels may be totally different after the rotation. The voxels originally connected together may be separated after the rotation. Some of the voxels may overlap each other and become one voxel after the rotation. Traditional approaches may detect several edges at some particular locations of the object. However they might not be able to find those edges at the same locations after the object rotates in the volume. All these reasons make the traditional 'filters' in 2D data space or 3D data space useless or inaccurate. Even the edges have been detected successfully in both volumes, how to measure feature points on the edge is another problem. The edge in 3D space is no longer a curve, but a surface. More complicated mathematical approached should be used to handle it. Unfortunately, the result is not satisfactory enough.

A new method is developed to find the feature points at the sharp corner of the bone cortices in the volume data sets. Different from traditional feature point detection methods that are based on gradient, this method is based on a statistical method. In addition, no edge detectors are needed to measure the edge first. The feature points can be detected in one step. This new approach can avoid the structure changing of the voxels after rotation. It is insensitive with connection definitions, and produces the same result no matter 6, 18, or 26-connection definition is used. It works well with the complicated and erose structure in discrete data space, such as human bones. This approach can also be modified to detect other structures, such as detecting edges, lines or blocks.

3.3.2 Detecting Feature Points by Statistical Method

Feature points should lie on the bone cortices which are the hardest part of the bone, and the point itself must be a sharp corner of the cortices. Hence a threshold *t* is first set, and only those voxels whose values are greater than *t* will be considered. The basic idea of this method is to conder a cube whose center is the current voxel being considered, called the center-voxel. All the voxels in this cube with values greater than *t* are considered as a part of a single object. They all have a uniform value instead of the gray-level values. The other voxels in the cube are ignored as if they do not exist. The center of gravity (CG) of the object is calculated. By comparing the coordinates of the CG with the coordinates of the center-voxel, it is determined if the center-voxel lies on the corner of the object, on the edge of the object, on the surface of the object or in the middle of the object. If the center-voxel is a point on the corner, the CG point must be far from the center axes. Otherwise, the center-voxel must be a voxel in the middle of the object if it is very near to the CG, or the object in the cube is cloud-like, in which the voxels are distributed throughout the capacity. Range *r* is set to limit the size of the cube centered at the specified voxels.

The CG point of a given object is calculated by the following function in continuous space,

$$\overrightarrow{CG} = \frac{\iiint \vec{v}\rho(x,y,z)dxdydz}{\iiint \rho(x,y,z)dxdydz}$$

where $\vec{v}$ is a vector in 3D space, $\rho(x,y,z)$ is the density function corresponding to the coordinates. In discrete data space, this equation is represented by,

$$\overrightarrow{CG} = \frac{\sum_x \sum_y \sum_z \vec{v}\rho(x,y,z)}{\sum_x \sum_y \sum_z \rho(x,y,z)}$$

To each voxel (u,v,w) that is treated as a center-voxel, the feature point detector function is defined as:

$$F_\phi^{r,t}(u,v,w,V(u,v,w)) = \begin{cases} 0 & V(u,v,w) < t \\ \dfrac{|L_\phi^{r,t}(u,v,w) - P_\phi^{MIN\_VALUE}(u,v,w,V(u,v,w))|}{r} & otherwise \end{cases}$$

where $V(u,v,w)$ is the value of the voxel *(u,v,w)*. ϕ corresponds to the *x*-, *y*-, or *z*-axis. The *L(x,y,z)* is defined as:

$$L_\phi^{r,t}(u,v,w) = \frac{\sum_{i=u-r}^{u+r+1} \sum_{j=v-r}^{v+r+1} \sum_{k=w-r}^{w+r+1} P_\phi^t(i,j,k,V(i,j,k))}{N(t)},$$

where $P_\phi^t(u,v,w,d)$ returns one of the coordinates according to ϕ if *d* is greater than *t*, otherwise, $P_\phi^t(u,v,w,d)$ returns zero. *MIN_VALUE* is a constant that corresponds to the minimum value of the volume data set. Since CT data is being considered, each voxel value in the volume data is 8 bits long, and is from 0 to 255. $P_\phi^{MIN\_VALUE}(u,v,w,V(u,v,w))$ simply gets one of the coordinates of the center-voxel *v* according to ϕ. $N(t)$ is a counter that records the number of voxels whose values are greater than *t* in the current cube.

This equation is identical to the previous equation for all the voxels are treated as either 1 or 0. Hence *ρ* is uniformly equals 1 if the value of the voxel is greater than *t*. Therefore,

$$\rho(u,v,w) = \begin{cases} 0 & V(u,v,w) \le t \\ 1 & V(u,y,z) > t \end{cases},$$

$$\vec{v}\rho(u,v,w) = P_\phi^t(u,v,w,V(u,v,w)), \quad \sum_x \sum_y \sum_z \rho(u,v,w) = N(t), \text{where } \phi = x,y,z.$$

The gray-level volume CT data first is considered as binary a volume. The voxels with values greater than t are set to 1, otherwise, they are set to 0. Voxels with the values greater than t are renamed as 1-voxels and those with values smaller than or equal to t are renamed as 0-voxels. All the soft parts in the bone whose densities are low are ignored. To each 1-voxel, a cube centered at it is considered. The current 1-voxel is called the center-voxel. The x, y, and z coordinates of all the 1-voxels that are inside the cube centered at the center-voxel are added respectively. The size of the cube is $(2r+1)\times(2r+1)\times(2r+1)$. The sums corresponding to x, y, or z coordinates, respectively, are then divided by r, and this gives $F_x^{r,t}$, $F_y^{r,t}$, and $F_z^{r,t}$. These results show the average allocation of 1-voxels inside the cube. Another threshold α is set and used to compare with $F_x^{r,t}$, $F_y^{r,t}$, and $F_z^{r,t}$, where $\alpha \in [0,1]$. If all these three values approximate zero, it means the 1-voxels inside the cube are evenly distributed. There should be several situations. The voxels in the cube may be a cloud-like object. The cube may fill with 1-voxels everywhere. Or the 1-voxels in the cube crowd to a blob-like shape and the center-voxel is just the voxel at the center of this blob. All these situations show that the center-voxel cannot be a point at the corner. (see Figure 3.6) Otherwise, the center-voxel should lie at a corner, on an edge or on a surface.

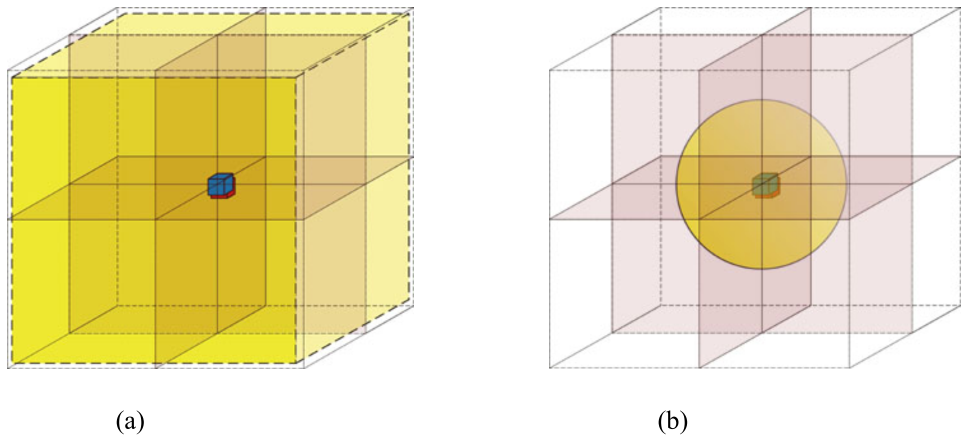


Figure 3.6
All Fs are smaller than α

If all three Fs are greater than α, the 1-voxles lean to one side of each axis. If the cube is subdivided into 8 equally-sized quadrants, the 1-voxels must crowd in one quadiant. The center-voxel should be a point at the corner. Rarely, the center-voxel may lie on a surface or an edge too. (see Figure 3.7)

If only two of the Fs are greater than α, the center-voxels could be a point at the corner too (see Figure 3.8a), but the body of 1-voxel group can be a symmetric construction. The symmetric plane is contains the axes corresponding to those Fs that are greater than α. On the other hand, it could also be a point on a surface or an edge. (see Figure 3.8b,c)

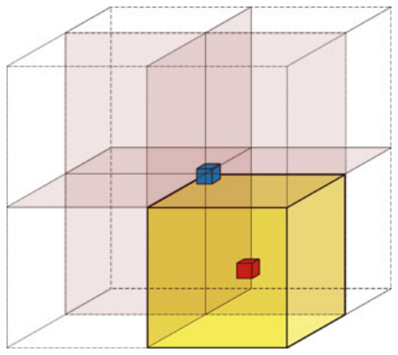


Figure 3.7
Three Fs are greater than α.

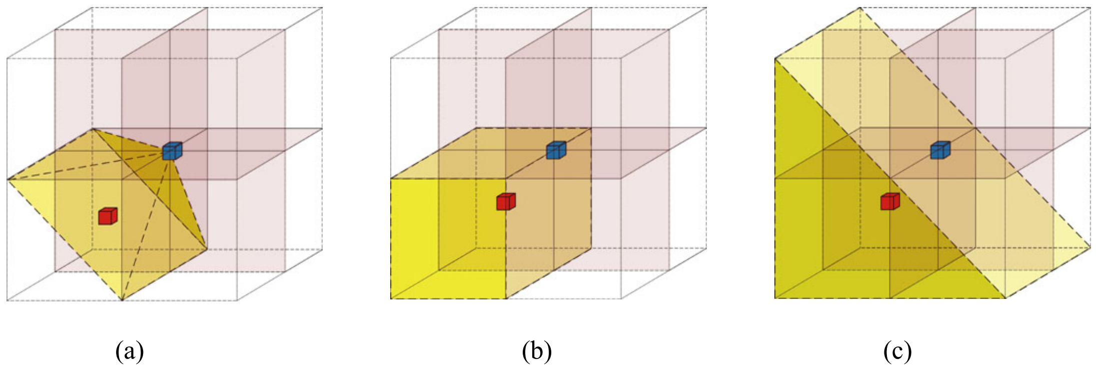
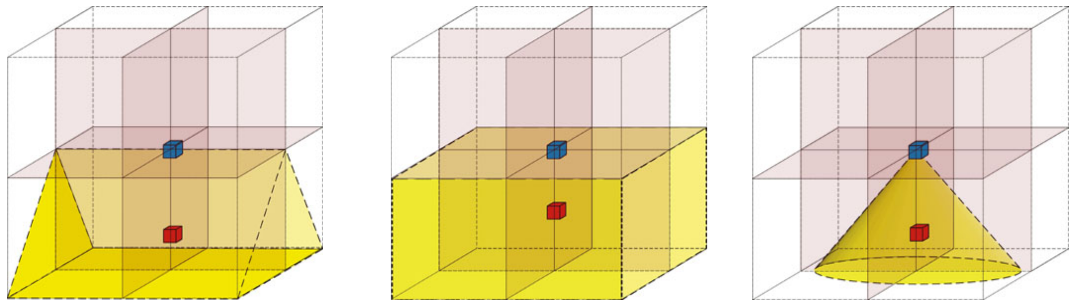


Figure 3.8
Two Fs are greater than α.

If only one of the Fs is greater than α, the center-voxel may lie on any kinds of constructions. Having only one F greater than α, means that the body of the 1-voxel group is symmetric about one of the axes. The center-voxel could be one point on an edge (see Figure 3.9a). It could also lie on a plane. (see Figure 3.9b) It could be a point at a corner (see Figure 3.9c), too.

Hence it is not sufficient to determine whether the center-voxel is a point at a corner or not by considering Fs only. A new threshold is introduced to solve the problem,

$$c = r^3 + \lambda .$$



(a) (b) (c)

Figure 3.9
One F is greater than α.

If $N(t) > c$, there are too many 1-voxels in the cube. If there are more than one Fs greater than α, the $N(t)$ need to be compared with c. The center-voxel can be a point on an edge or on a plane if $N(t) > c$ because the number of 1-voxels is large in such situations. Otherwise, $N(t)$ will not be more than a quarter of the total number of voxels in the cube. The λ introduced here is a parameter to adjust the sharpness of the corner. The greater the λ is, the more 1-voxels are allowed inside the cube, and the blunter the corner is. However λ must be less than r^3, otherwise the center-voxels on a plane or an edge will be considered as corner points too.

In practice, λ is set to about 20. To avoid too many feature points lying on blunt corners, this setting also ensures a reasonable number of feature points and is compatible enough in different situations. r is set to 7. That means the size of the cube centered at each 1-voxle is 15×15×15 large, much larger than traditional filters. If r is too small, there are too few voxels inside the cube. For the statistical method is based on a large number of samples, it cannot work well in a small number of sample sets. Those problems similar to the traditional filters will occur when r is too small. On the other hand, the speed decreases if r is too large, because there are too many voxels that need to be measured for each 1-voxels. Hence r=7 is acceptable. Another parameter t is set to 198. This is an experimental constant value, and values greater than 198 in an 8-bit long data set usually corresponds to the harder part of the human bone. Figure 3.10 shows a 2D slice extracted from the 3D volume data. Notice the red crosses in the image, which are the feature points detected by this algorithm. Most of them are exactly at the corner of the bone cortices. Points lying on edges are rare. Points on the surface are almost impossible. The number of feature points is limited between 100 to 300. Too many feature points will decrease the speed of matching. On the other hand, too few feature points will decrease the accuracy of alignment.

3.3.3 Eliminate Unsuitable Feature Points

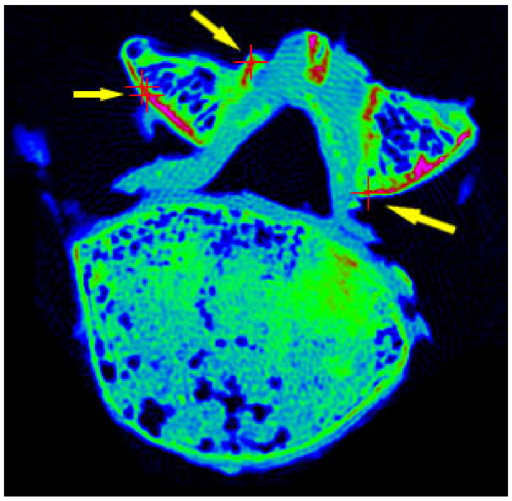
The feature point detection algorithm is applied to the two samples, the one before the bone cement injection and the one after bone cement injection. Since these two samples are in general different, the feature points that are detected by the algorithm are different too. The number of feature points detected in the sample after injection is much more than that of the one before the injection. Since bone cement is a radiopacity substance, it has very high densities whose values are definitely higher than 198. The previous algorithm will consider them as parts of the bone cortices and detect feature points in it.

If the part belonging to the bone cement were know, eliminating the feature points on it would be very easy. However, what exactly needs to be done is separating the bone cement from other tissue. Nonetheless there are still ways to eliminate these feature points on the bone cement according to the location of the bone cement. The bone cement is always in the middle part of the bone. Only very little bone cement floods out of the entrance where the cement is injected. Besides, bone cement cannot change the structures of bone cortices, though it may change the internal structure of the bone. That means the sample scanned after injection has the difference only inside the bone, and the outside of the bone has the same structure. Hence we have developed a simple algorithm to ignore the feature points that are located inside the bone.

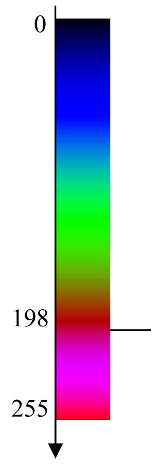
To each feature point, $f(u,v,w)$ (it is the voxel whose coordinates are (u,v,w) actually), the distance to the bone surface is calculated. If the distance is larger than a threshold *d*, this feature point is eliminated in the feature point list. The surface of the bone, which is called isosurface in the volume data sets, should be extracted first before calculating the distance from a voxel to the surface. However isosurface extraction is a hard work. Marching Cubes method was introduced in [1] to extract isosurface in volume data sets according to a threshold *t*. Although it is a dependable algorithm for extracting the isosuface, Marching Cubes is very slow and is not worthy to be used here. A much easier way is used here to evaluate the distance from the voxel to the surface. A new function is defined here:

$$E(u,v,w) = \min_{\phi=x,y,z}(D_{\phi}^{t'}(u,v,w)).$$

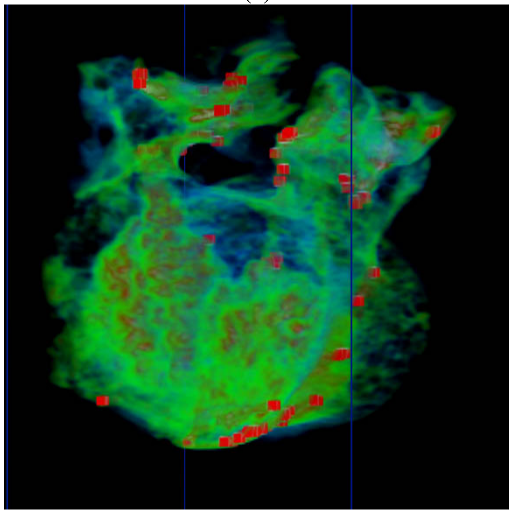
The formula *D* is a counter that records the number of the consecutive voxels whose values are greater than *t'* along ϕ-axis direction from *f*. In shot, imagine there are 6 radials that start from the *f* and go along the negative and positive directions of *x*-, *y*-, or *z*-axis. The radial stops if it touches a voxel whose value is lower than the threshold. The function *D* just returns the minimum at the lengths of these 6 lines.



(a)



(b)



(c)

Figure 3.10
(a) One 2D slice in a volume. The red crosses point out the feature points detected by this algorithm. (b) Density reference. (c) The same volume in 3D view. Tiny red cubes point out the feature points detected by this algorithm. Note the voxels with low densities have been eliminated by a transfer function.

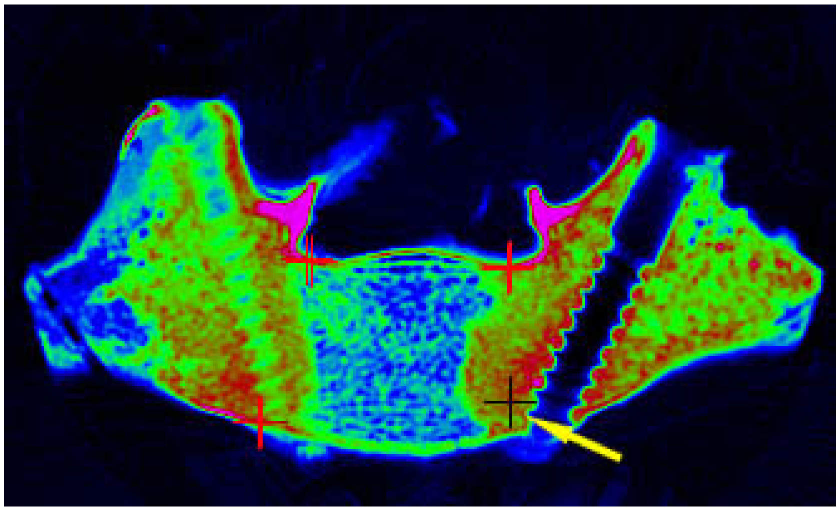


Figure 3.11

The feature points (black cross) deep inside the bone will be eliminated.

While using this function in practice, a problem was found. When a feature point lies in the middle of the bone and at the same time there are some “holes” or “bubbles” very close to it, then this approach may not eliminate this point in the feature point list if one of the 6 radials hits the hole or bubble. “Holes” and “bubbles” are the voxels in the volume whose values are very low. Some parts in the real bones are spongy, and the bone cement also contains bubbles, which are created during mixing and injecting. Therefore, bubbles and spongy parts cause the density as low as the outter space whose density is almost zero. In the volume data set, these bubbles and holes are represented by independent voxels with very low density surrounded by the group of voxels with much higher values. The above algorithm regards that the radial goes into the outer space when it touches the hole or bubble (see Figure 3.12).

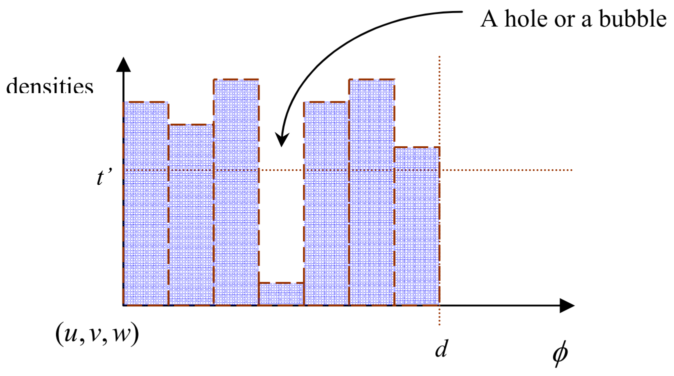


Figure 3.12

Holes and bubbles in the volume disturb the correctness of eliminating unqualified feature points.

Hence we improve the original algorithm to avoid such situation. All the voxels whose values suddenly decrease for only one voxel will be ignored. Only when the radial hits two or more consecutive voxels whose values are lower than *t'* will it be considered as going into the outter space. The presentation of the algorithm in pseudocode as follows:

```
for (each feature point  $f(u,v,w)$) do
{
  for (x=u) to (u+r) do
  //Shoot a radial to positive direction of x axis.
  {
    if ( $v(x,v,w) < t$ )
    {
      if ( $v(x+1,v,w) < t$  and  $v(x+2,v,w) < t$)
      {
        Erase(v);
      }
    }
  }

  for (x=u-r) to (u) do
  //Shoot a radial to negative direction of x axis.
  {
    if ( $v(x,v,w) < t$ )
    {
      if ( $v(x-1,v,w) < t$  and  $v(x-2,v,w) < t$)
      {
        Erase(v);
      }
    }
  }

  ...//do the same steps on y axis
  ...//do the same steps on z axis
}
```

3.4 Conclusion of Feature Point Detection

Feature point detection produces two sets of points with their coordinates only, one set contains the feature points extracted in the volume data scanned before the bone cement injection; the other set contains the feature points extracted in the volume data scanned after the bone cement injection. The numbers of feature points in the two sets are different but not much. The specific location in the object in one volume where there is a feature point detected may not have its corresponding feature point in the other volume. Even there are pairs of feature points that are extracted on the relevant location in the object in both volumes, their geometry characteristics, like distance or angles between each other, may not be exactly equal. These problems are not introduced by the detection method, but the limitation of the discrete data space and the complicated structure of human bone and bone cement. There is a robust matching method to be introduced in the next chapter, with tolerance for such inaccurate coordinates of feature points.

Chapter 4

Approximate Point Pattern Matching

An approach for matching and aligning a pair of sets of points according to their coordinates information only, is introduced in this chapter.

There is no doubt that a number of researches have been done on this field because of its urgent and wide applications in practice. The large amount of digital information gathered needs to be compared, recognized and registered automatically. The matching and analysis of geometric patterns and shapes have become very important in various application areas. For examples computer vision, pattern recognition, cartography, molecular biology and computer animation, all these fields need the technique to match groups of points in 2D or 3D space.

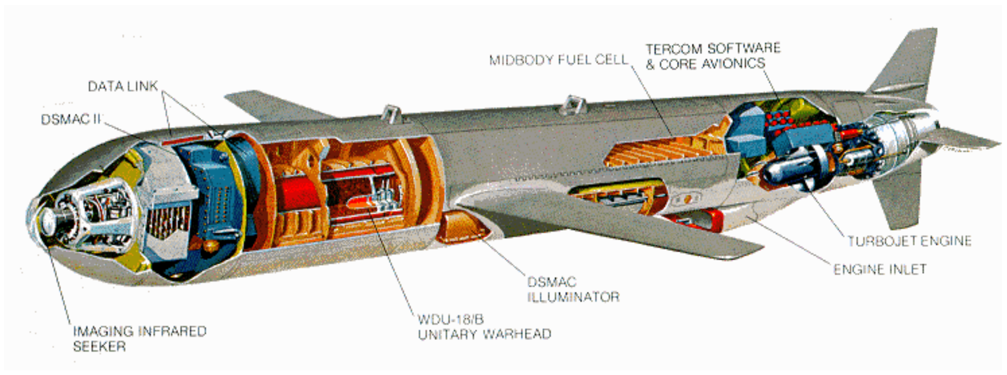
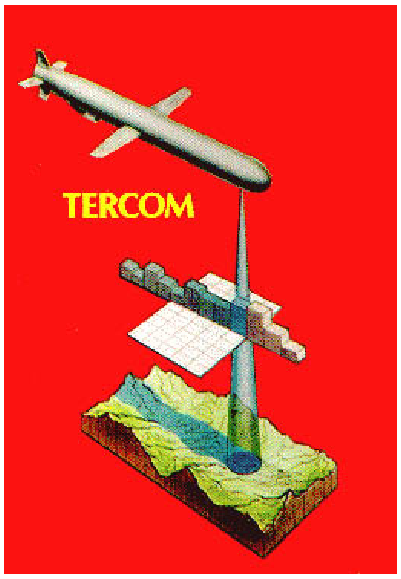
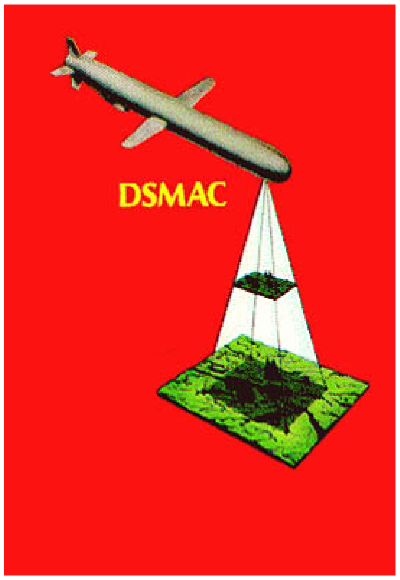


Figure 4.1
BGM-109 Tomahawk cruise missile



(a)



(b)

Figure 4.2
(a) TERCOM. (b) DSMAC

BGM-109 Tomahawk (see Figure 4.1), the ship or submarine-launched land-attack cruise missile equipped by the U.S. Navy, is a typical application of matching techniques. The shocking videos broadcast a decade ago during the Gulf War in 1991 showed how such a deadly weapon flew over 1,000 kilometers, steered clear of several civilian buildings and then hit the target through the window. Such an accurate navigation system depends on the radar guidance system called Terrain Contour Matching (TERCOM). The TERCOM radar compares a stored reference map with the actual terrain to determine the position of the missile. In the last few minutes when the missile reaches an area very close to the target and is able to get a visual contact with the target, it uses another system to recognize the target. It is called Digital Scene Matching Area Correlation (DSMAC). It uses a digital camera fixed on the head of the missile and compares a stored image of the target (it may have been taken by a satellite or spies) with the actual target image (see Figure 4.2). TERCOM navigates the missile flying over 1,000 kilometers and just leads it to an area near the target. DSMAC recognizes the target to prevent mistaken-attack and finally guides the missile to hit the target.

The U.S military, of course, would not describe the details of those two systems except the sketches shown in their website. However it must be based on some matching methods. It is impossible to compare a whole image with the stored map or target image because it is very slow and the computer in the missile is not very powerful. These systems must detect some feature points just like what has been presented in the previous chapter, and then compare the feature point with the stored data. A literature review is first presented next, before we present our matching approach.

4.1 Previews Works

Unfortunately, most of the literatures present methods and theories of point matching and alignment working in 2D space. It is easy to understand that most of the practical works are developed for 2D images, just like the situation in feature point detection. Despite some of the approaches are suitable for 3D space, they also project the 3D models onto the 2D planes before matching. Although there are few papers we can reference, some of the theoretic literatures can be found that describe the basic approaches and analyses the speed of matching algorithms.

[32] gives an overview in this field. The general description of the problem is that there are two given object ***A***, ***B*** and we want to know how much they resemble each other. One of them may be transformed, like rotated, translated, or scaled in order to be matched with the other one as good as possible. The matching could be a partial matching, when ***A*** resembles only some part of ***B***, or find the most similar one for a given A in a fixed preprocessed set of objects, like character or traffic sign recognition. It also could be a simplification of objects, that given an object ***A*** find the most simple object ***A'*** resembling ***A*** within a given tolerance, like finding a polygonal line with as few edges as possible that resembles a curve. The transformation that is allowed to match objects ***A*** and ***B*** may be a very simple translation in some of the applications. Nonetheless the matching problem usually is not as simple as that, for rotation is allowed at the same time that translation is also allowed. Such a transformation is called rigid motion or Euclidean transformation. Scaling is another transformation that could be considered, which stretches an object by a certain factor λ about the origin in any the dimensions. The transformation which includes scaling is not a rigid motion, and would not be considered in this thesis. We will discuss the case where both of the finite sets ***A*** and ***B*** consist of points (those detected by the feature point detection algorithm) and the transformation is composed of translations and rotations only.

The simplest situation in point pattern matching is Exact Point Pattern Matching. Namely, given two finite sets of n points each$\boldsymbol{A},\boldsymbol{B} \subset \mathbf{R}^d$, they can be matched exactly. It should be stressed that each point in ***A*** must have a counterpart in ***B*** which can be strictly matched to it.

The algorithm for exact point pattern matching can easily be reduced to string matching by sorting polar coordinates of the points. Atkinson introduced his algorithm in [33],

1. Determine the centroids c_A, c_B of the set ***A*** and ***B***, respectively.
2. Determine the polar coordinates of all points in ***A*** using c_A as the origin. Then sort ***A*** lexicographically with respect to these polar coordinates, angle first, length second, and obtain a sequence $(\phi_1, r_1), ..., (\phi_n, r_n)$. Let *u* be the sequence $(\psi_1, r_1), ..., (\psi_n, r_n)$ where $\psi_i = \phi_i - \phi_{(i+1) \bmod n}$. Compute in the same way the corresponding sequence *v* for ***B***.
3. Determine whether *v* is a cyclic shift of *u*, i.e. a substring of *uu* by some fast string-matching algorithm.

The algorithm gives a positive answer if ***A*** and ***B*** are exactly congruent. The running time is O(*n*log*n*) because the fastest sorting algorithm is O(*n*log*n*), and the other steps are linear time.

Alt also introduced his algorithm for 3D exact point pattern matching in [34]. [32] declared that the algorithm takes O(*n*log*n*). It also states that such as exact pattern point matching could be solved in time O(*n*log*n*) in 2D and $O(n^{d-2} \log n)$ in 3D or higher dimensions.

Now consider two sets of points ***A*** and ***B*** with the same number of points *n*, find a transformation matching each points $b \in \boldsymbol{B}$ into the *ε*-neighborhood (where ε is a fixed error and $\varepsilon \geq 0$) of some point $a \in \boldsymbol{A}$. This is called approximate one-to-one point pattern matching. There are more variants to this problem:

- Different types of transformations that are allowed.
- Solving either the decision problem: given *ε*, is there a matching? Or the optimization problem that finds the minimal *ε* allowing a matching.
- A fixed one-to-one-mapping between ***A*** and ***B*** is either already given or one should be found.
- Different metrics, a concept generalized by Arkin et al. [35] to arbitrary “noise regions” around the points.

[32] only demonstrated the algorithms solving the decision problem, which found a rigid one-to-one matching between the point sets $\boldsymbol{A} = \{a_1, ..., a_n\}, \boldsymbol{B} = \{b_1, ..., b_n\}$ with a given *ε* as a Euclidean tolerance. It gave out a very fundamental algorithm first. If there is a transformation that is a valid matching between ***A*** and ***B***, there must be two points b_i, b_j in ***B*** being matched to the boundaries of the *ε*-neighborhoods $U_\varepsilon(a_k), U_\varepsilon(a_l)$ of two points in ***A***. Mapping b_i, b_j onto the boundaries of $U_\varepsilon(a_k), U_\varepsilon(a_l)$ respectively will leave only one degree of freedom which can be

parameterized by the angle $\phi \in [0,2\pi)$ between the vector $b_i - a_k$ and the horizontal line. For all possible values of ϕ, any other point in ***B***, $b_m \in B, m \neq i, j$, will trace an algebraic curve C_m. This curve intersects the boundary of any $U_\varepsilon(a_r)$ at most a constant number of times. Hence there must be at most a constant number of intervals of the parameter ϕ where the image of b_m lies inside $U_\varepsilon(a_r)$. For example, the curve intersects $U_\varepsilon(a_r)$ for 4 times (see Figure 4.3), and there are two intervals of the ϕ where b_m lies inside $U_\varepsilon(a_r)$. In fact the number of intervals must be half of the number of intersections because each intersection changes the inside or outside state of the curve. All intervals like this are collected. It gives a positive answer when for all ϕ in one interval the same points of B are mapped into the same neighborhoods of points of A, and vice versa. This is checked by finding subinterval in $[0,2\pi)$. This algorithm is obviously only suitable for 2D points sets for it only seeks 3 pairs of points. It was declared that the running time is $O(n^8)$. Besides, determination of intersection of the curve with the boundary circle was another big problem for it might cause nontrivial numerical problems. However, simpler and faster algorithms were developed for this problem. Efrat and Itai [36] found a method to speed such one-to-one matching problem in which only translation was allowed. The running time is $O(n^{1.5} \log n)$. On the other hand, Arkin et al. [35] gave numerous efficient algorithms mostly assuming that the ε-neighborhoods or other noise-regions of the points did not intersect each other. One of these algorithms is carried out in $O(n^4 \log n)$ times under this assumption. Heffernan and Schirra [37] gave an alternative method that was, approximate decision algorithms, which only gave a correct answer if the given tolerance ε was not too close to the optimal solution. The running time was reduced to $O(n^{2.5})$. Others, like Behrends [38], also developed approximate decision method whose running time was $O(n^2 \log n)$ under some assumptions.

Now consider two sets of points, $\boldsymbol{A} = \{a_1, ..., a_n\}, \boldsymbol{B} = \{b_1, ..., b_m\}$, where $n \neq m$. Algorithms for optimally matching such two sets are given by Huttenlocher, Kedem, and Sharir [39]. They used a theory called Voronoi-surface to determine the matching under translations only. The Voronoi-surface of the set ***A*** is defined as follows:

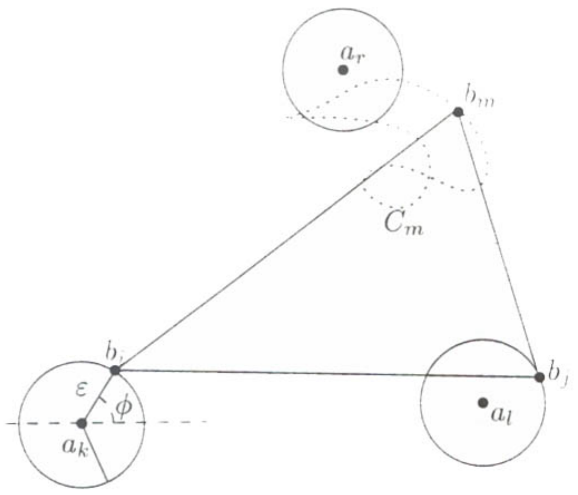


Figure 4.3

Curve of point b_m when b_i, b_j are moved on the boundaries of $U_\varepsilon(a_k), U_\varepsilon(a_e)$

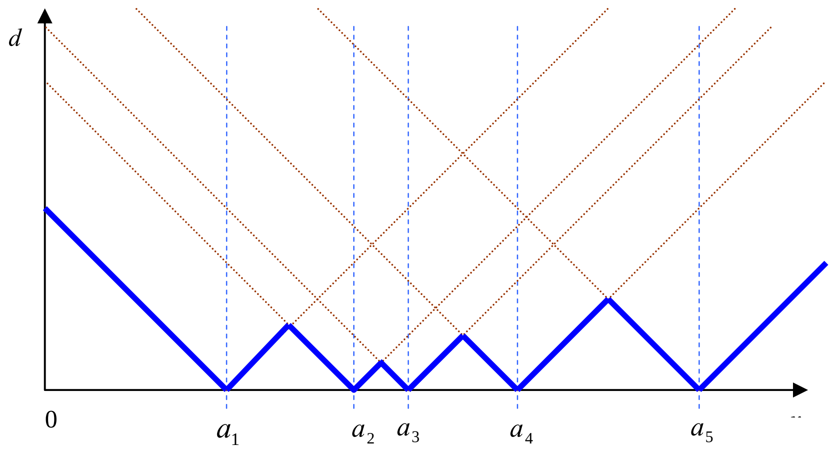


Figure 4.4

Voronoi-surface in 1D space. $d(x)$ is represented by bold lines. Purple dash lines represent

$d_{a_i}(x)$.

$$d(x) = \min_{a \in A} \|x - a\|$$

The function assigns to each point x the distance to the nearest point in A, which is the lower envelope of all $d_a(x) = \|x - a\|$. Figure 4.4 shows the graph of $d_a(x)$ and Voronoi-surface in 1D space. They are presented by serials of lines. In 2D space, the graph of $d_a(x)$ is an infinite cone in 3D space whose apex is at a and the Voronoi-surface ($d(x)$) is piecewise composed of these cones and the projection of the boundaries of these pieces. (see Figure 4.5)

Finally, the $\min_t h(t)$ is searched for the optimal translation vector. It declared the running time is $O(nm(n+m)\log nm)$ for L_1- and $L_\infty-$metric in 2D space and it spends more time for L_r-metric, where r=2,3,…

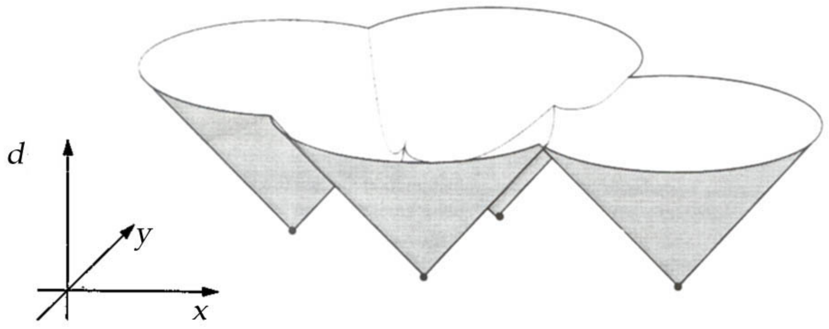


Figure 4.5
Voronoi-surface in 2D space.

In addition, Huttenlocher et al. introduced an algorithm that is called dynamic Voronoi diagrams [40], which was able to deal with the matching between two point sets allowing not only translations but also rotations under the assumption that the point set consists of *k* rigid subsets of *n* points each. The running time for an optimal match of two point sets in 2D space under arbitrary rigid motions is $O((m+n)^6\log(mn))$.

According to the previous works that have been done by others, it is found that most of the algorithms were able to work well and fast in 2D space or when only translations are allowed. However algorithms designed for matching in 3D space are not good enough except those algorithms for exact point pattern matching. Besides, the algorithms for approximate point pattern matching usually have been developed for specified problems. Specified assumptions were made in these algorithms, which made these algorithms only suitable for limited problems. Some of them limited the ε-neighborhoods such that they could not intersect each other, like the method given by Arkin et al. Some of them allowed translations only, like the method introduced by Huttenlocher, Kedem, and Sharir. In addition, they did not discuss how to find the minimum *t* as well.

The balance between speed and accuracy is another problem. Algorithms usually sacrificed the speed to reach the optimal matching. The speed of the algorithms that finds the optimal result is very slow even in 2D space or allowing translations only. It will be much slower if a similar method is used in 3D space and combining translations and rotations together. Hence the goal of the matching method introduced in this chapter aims at balancing between the speed and the accuracy of the matching result as good as possible.

4.2 Approximate Point Pattern Matching

The characteristics of the two point sets to be matched are as follows:

- The two point sets are extracted by feature point detection algorithm.
- The cardinalities of the two sets are different.
- Most of the points in one set can lie in the ε-neighborhood of their corresponding points in the other set.
- Some of the points cannot lie in the ε-neighborhood of any points in the other set. That means they have no corresponding points in the other set.
- Both translations and rotations are allowed.

The goal of the algorithm is to find a transformation matrix that transforms one point set in order to match good enough to the other point set. First, two Center of Geometry (COG) of both sample volumes are obtained. By overlapping two COGs, translation can be ignored. Second, points in each set are ordered by their distance to COG, which can increase the speed of searching for candidates in the point set. Third, three key points are selected in A. These three points and COG must be noncoplanar, and their distance to each other should be far enough. Then, their corresponding points in B are searched. They are selected according to their distances to COG and their relative distances too. After three pairs of points have been decided, the transformation matrix *M* is also determined. All points in B are transformed by this matrix. After that, a method, called Average Hausdorff Distance (AHD), is used to measure how good the matching is. If the AHD is large, it means the matching may not be good enough. Another three pairs of points will be selected until the AHD is acceptable. When selecting point in the sets, speeding-up techniques are used to increase the speed. The balance between accuracy and speed is the most important thing.

The remaining part of this chapter is organized as followings: the traditional Huasdorff Distance and the modified Huasdorff Distance (AHD) is discussed first; then we will discuss how to determine the transformation matrix *M* by three pairs of points; after that, the matching method is introduced step by step; speed-up techniques are introduced finally.

4.2.1 Hausdorff Distance

In order to measure how good the matching is, Hausdorff distance is introduced.

For two point sets ***A*** and ***B*** in *d*-dimensional space $\mathbf{R}^d$, one-side Hausdorff distance from ***A*** to ***B*** is defined as follows,

$$\tilde{\delta}_H(\boldsymbol{A},\boldsymbol{B}) = \max_{a\in\boldsymbol{A}} \min_{b\in\boldsymbol{B}} \|a-b\|,$$

where $\|.\|$ is the Euclidean distance in $\mathbf{R}^d$. The Hausdorff distance between ***A*** and ***B*** is defined as

$$\delta_H(\boldsymbol{A},\boldsymbol{B}) = \max(\tilde{\delta}_H(\boldsymbol{A},\boldsymbol{B}), \tilde{\delta}_H(\boldsymbol{B},\boldsymbol{A})).$$

The Hausdorff distance assigns to each point in one set the distance to its nearest point in the other set and takes the maximum value of all these values. It works well in most cases but may fail in cases where there is noise in the images. It also fails in our case because the points in one set that have no corresponding points in the other set can be treated as noise in the set. The minimum value of the Hausdorff distance measured from a matching doesn't mean the matching is the best one. For example (see Figure 4.6) in 2D space, $\boldsymbol{A}=\{a_0,a_1,a_2,a_3\}, \boldsymbol{B}=\{b_1,b_2,b_3\}$, a_0 has no corresponding points in ***B*** that can be matched to it, nd a_i can be matched to $b_i (i=1,2,3)$. Assume there is a transformation that transforms $b_i (i=1,2,3)$ to the *ε*-neighborhood of its corresponding point in ***A*** (see Figure 4.6a), and the Hausdorff distance is calculated. $\|a_0-b_1\|$ is the maximum distance among all the minimum distances from the points in ***A*** to the points in ***B***, so is it from ***B*** to ***A***. The Hausdorff distance is:

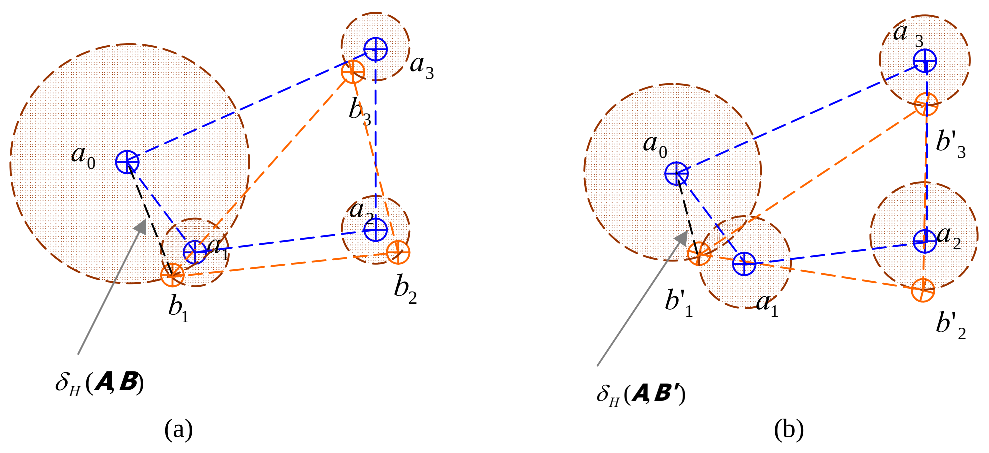
$$\delta_H(\boldsymbol{A},\boldsymbol{B}) = \|a_0-b_1\|.$$

Then assume there is another transformation that transforms the points in ***B*** to the other positions (see Figure 4.6b). Obviously, this time the transformation is not as good as the previous one because most of $b_i (i=1,2,3)$ don't lie in the *ε*-neighborhood of any points in A. However the Hausdorff distance shows that the transformation is better than the previous one. Although the minimum distance from each $a_i (i = 1,2,3)$ to its corresponding points in B becomes larger, but $\|a_0 - b'_1\|$ is still larger than any of them. Hence,

$$\delta_H(\boldsymbol{A}, \boldsymbol{B'}) = \|a_0 - b'_1\|,$$

and $\delta_H(\boldsymbol{A}, \boldsymbol{B}) > \delta_H(\boldsymbol{A}, \boldsymbol{B'})$.

The new transformation sacrifices most of the points that are used to be matched very well just to indulge one point that actually has no corresponding point. It cannot be considered as a better matching under such circumstance. The example shows that a smaller Hausdorff distance does not mean a better matching while there are some points without corresponding points to be matched in one set.



a_0
a_1
a_2
a_3
b_1
b_2
b_3
$\delta_H(\boldsymbol{A}, \boldsymbol{B})$
(a)
a_0
a_1
a_2
a_3
b'_1
b'_2
b'_3
$\delta_H(\boldsymbol{A}, \boldsymbol{B'})$
(b)

Figure 4.6
A counterexample that traditional Hausdorff distance fails to measure the matching quality.

Due to this shortcoming of the Hausdorff distance, some modifications to the Hausdorff distance have been introduced. One of them discussed in [32] selected the minimum distance among all the minimum distances,

$$h_k(\boldsymbol{B}, \boldsymbol{A}) = \min_{b \in \boldsymbol{B}}{}_k \min_{a \in \boldsymbol{A}} \|a - b\|.$$

Only one-way Hausdorff distance was calculated for the partial matching problems, in which $\boldsymbol{A}$ is a large set and $\boldsymbol{B}$ is a sub-set of $\boldsymbol{A}$. The previous example is such a case coincidently. This approach cannot be used directly for the matching problem discussed in this thesis, which is not a partial matching problem. Not only points in $\boldsymbol{A}$ may have no corresponding points in $\boldsymbol{B}$, but the points in $\boldsymbol{B}$ may have no corresponding points in $\boldsymbol{A}$ as well. In addition, the matching method to be discussed later assures at least one pair of points to be matched very well. No matter how bad the matching is, $h_k(\boldsymbol{B},\boldsymbol{A})$ or $h_k(\boldsymbol{A},\boldsymbol{B})$ will always be zero in this case. Hence it cannot be a measurement of how good the matching is.

A new measurement approach based on the Hausdorff distance is used here, which is called the Average Hausdorff Distance (AHD). The maximum distance among the minimum distance of each point to its corresponding point is no longer calculated, but the average of all minimum distances is calculated instead. For each point in ***A***, the distance from it to its nearest point in ***B*** is computed:

$$\tilde{\delta}(a,\boldsymbol{B}) = \min_{b\in\boldsymbol{B}}\|a-b\|, a \in \boldsymbol{A}.$$

The average distance of both $\tilde{\delta}(a,\boldsymbol{B})$ and $\tilde{\delta}(b,\boldsymbol{A})$ for all point in ***A*** and ***B*** are calculated,

$$\tilde{\delta}_A(\boldsymbol{A},\boldsymbol{B}) = \frac{\sum_{i=1}^{n}\tilde{\delta}(a_i,\boldsymbol{B})}{n}$$

and $$\tilde{\delta}_A(\boldsymbol{B},\boldsymbol{A}) = \frac{\sum_{i=1}^{m}\tilde{\delta}(b_i,\boldsymbol{A})}{m}$$

where n and m are the number of points in ***A*** and ***B*** respectively. Finally, the maximum value between them is selected as the measurement of how good the matching is,

$$\delta_A(\boldsymbol{A},\boldsymbol{B}) = \max(\tilde{\delta}_A(\boldsymbol{A},\boldsymbol{B}),\tilde{\delta}_A(\boldsymbol{B},\boldsymbol{A}))\,.$$

The step getting maximum value of each point to its nearest point in the traditional Hausdorff distance is replaced by getting average of these distances. It can prevent the situation in which the distance of a point to its nearest point is much larger than any of the others. However this approach may fail when the number of points is few. Fortunately, the point sets we have to deal with here always have enough points.

4.2.2 Transformation Matrix

Theoretically, if there are four pairs of points got from two point sets respectively in 3D space, the transformation that matches the two sets ***A*** and ***B*** is fixed. Knowing a transformation matrix *M,* which is a 4×4 matrix, each point or vector $P' = \begin{pmatrix}x' & y' & z' & 1\end{pmatrix}$ in 3D space $\mathbf{R}'^3$ can be transformed to $\mathbf{R}^3$ by *M*,

$$P = \begin{pmatrix}x & y & z & 1\end{pmatrix} = P'M^T$$

The transformation matrix *M* includes not only Euclidean transformations, like translations and rotations, but also other transformations, like scaling, if the four pairs of points are not well defined. Nonetheless, for the problem discussed in this thesis, all the other transformations except translations and rotations are not allowed. If one pair of points is defined to be on the same position (see Section 4.2.3) and the other three pairs of points are carefully selected, the transformation matrix *M* must be a Euclidean transformation and only rotation around this reference point is allowed. “Well selected” means the relative distances between any two of those three points in one combination are similar to the distances between the corresponding points in the other combination. Represent the transformation matrix *M* as follows,

$$M = \begin{pmatrix} & & & m_{03} \\ & M' & & m_{13} \\ & & & m_{23} \\ m_{30} & m_{31} & m_{32} & m_{33} \end{pmatrix},$$

where *M’* is a 3 by 3 matrix. When $m_{30} = m_{31} = m_{32} = m_{03} = m_{13} = m_{23} = 0, m_{33} = 1$, the determinant of *M’* equals 1, and $M'^T M' = I$, *M* is an Euclidean transformation that allows rotation only. The transformation matrix *M* can be calculated by:

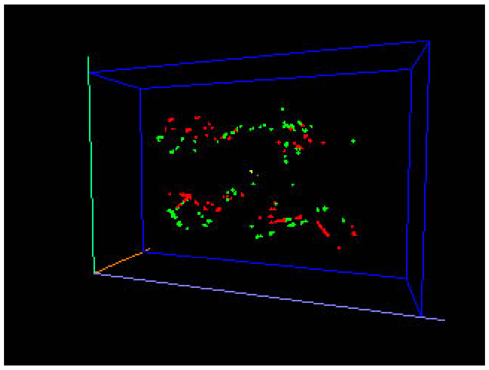
$$\begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ y_0 & y_1 & y_2 & y_3 \\ z_0 & z_1 & z_2 & z_3 \\ 1 & 1 & 1 & 1 \end{pmatrix} = M \begin{pmatrix} x'_0 & x'_1 & x'_2 & x'_3 \\ y'_0 & y'_1 & y'_2 & y'_3 \\ z'_0 & z'_1 & z'_2 & z'_3 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

where $x_i, y_i, z_i, x'_i, y'_i, z'_i$ are the coordinates corresponding to the points P_i and P'_i. P_3 and P'_3 are defined to be on the same position.

In practice, even with very well matched points found between the two sets, the positions of the corresponding points, the distances of each pair of corresponding points or the angles between each pair of corresponding points are not exactly equal. This is caused by not only the feature detection algorithm but also the rotation problems in discrete space. In this circumstance, the *M* would not be a strict rotation matrix. Tiny distortion occurs after the transformation. The distortion is harmful to the final alignment because the whole volume will also be rotated by this matrix in order to compare the samples before and after the bone cement injection. The distortion reduces the accuracy of matching.

Although there are other methods that can determine the rotation matrix and prevent the distortion at the same time, the distortion sometimes is useful to measure how good the matching is. When a bad matching is found, the three points lie a bit far from their corresponding points. The triangle comprised by these three points is different from the one comprised by these corresponding points, like length of the board or angles. Hence M would not be a strict rotation matrix around $P_3(P'_3)$. Translation, scaling, or even perspective are included. When M is applied to other points, distortion changes their positions too. The points that should overlap their corresponding points after transformation would not do so. The points far from $P_3(P'_3)$ are seriously affected by scaling and other non-rigid transformations. The AHD (Average Hausdoff Distance) will be large when one set matches to the other after such transformation. (see Figure 4.7) It is good to test how good the matching is by calculating AHD.

Since the matching points are carefully selected whose relative position is similar enough, the distortion will be very small. When m_{3i}, m_{i3}, and the determination of *M'* converge to their expected value, the transformation can be regarded as a rotation approximately. Actually, the $m_{3i}, m_{i3}, (i = 0,1,2)$ are of the order 10^{-6} when the acceptable matching was found in all the test cases. m_{33} always exactly equaled 1 and the determination of *M'* is very close to 1 of the same time. The distortion of the point set after rotating was hard to detect visually. Using such transformation on the whole volume, the distortion will be very tiny too, which would not bring about any negative effect while comparing the two samples.


(a) AHD=18.195

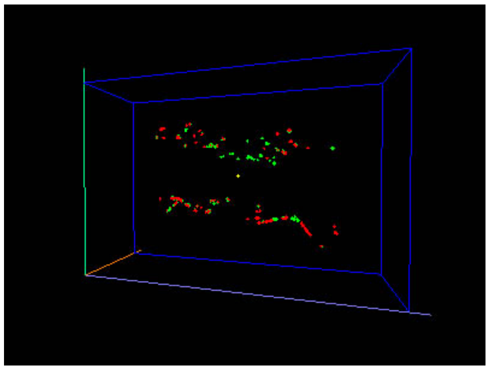

(b) AHD=3.959

Figure 4.7

The green points represent a group of points in one set and the red ones represent points in the other set.

(a) A very bad matching. Transformation matrix M enlarges the distortions heavily. The AHD value of this matching is extremely large. (b) A much better matching of the same pair of point sets.

4.2.3 Geometric Center of Volume

It has been mentioned that there is a point fixed when calculating transformation matrix. It is named P_3 or P'_3. It has also been mentioned that translation is allowed in this point matching problem discussed in this thesis. However the transformation *M* matrix only allows rotation around the point P_3 or P'_3. This is not a contradiction, for a transigent method is used to deal with the translation between the two point sets.

A bone can obviously be treated as a rigid object. Although the position or the angle is different in different scanning, the shape of the bone would not be changed at all. Consider the procedure of bone cement injection. Surgeons first dig a very small hole in the bone in order to insert the needle into the bone. The bone cement is then injected into the bone through the needle. Bone cement infiltrates into the gaps in the bone. Little bone cement overflows out the hole where it is injected. Surgeons always clean up the bone cement overflowing out of the surface of the bone for the bone cement exposing outside the bone is harmful to other organs. Comparing the sample bone before bone cement injection and the same one after injection, we can find:

- The surface or the shape of the bone has not been changed except for the tiny hole through which the bone cement has been injected.
- All the bone cement has been injected inside the bone, which has changed the internal structure of the bone, but has done nothing to the external structure of the bone.
- The allocation of the density of the bone has been totally changed after injection, so the center of gravity of the bone has been changed too.

The only thing that hasn’t been changed after injection is the shape of the bone. That means the geometric center of the bone remains unchanged relative to the bone. Hence it is still possible to find the similitude between the two samples in order to determine the translations.

The approach calculating the Center Of Geometry (COG) is very similar to the approach introduced in Chapter 3, which is used to detect the shape corner in a fixed cube. This time the fixed cube is the whole volume, and it is unnecessary to compare the result with the center-voxel.

First a threshold *t* is set to ignore noises in the volume. Some of the voxels have very low values and are located outside the bones. There are voxels with low values inside the bone too. Hence the value of *t* should be set very carefully so that it can ignore the noise but preserve the voxels inside the bone. In our experience, *t* is set to 5 in an 8 bit long data set for very few voxels in the bone have values lower than 5. Define the binary function as follow,

$$T_t(x,y,z) = \begin{cases} 1 & v(x,y,z) > t \\ 0 & otherwise \end{cases}$$

where $v(x,y,z)$ presents the value of the voxel corresponding the coordinates of (x,y,z). The volume is set to the binary volume now. Since all the voxels are presented by either 1 or 0, the changes in density after the injection can be ignored. No matter how much bone cement has been injected in, the two samples of the same bone scanned before and after the injection now are the same. Based on the method for calculating the CG, COG can be obtained from this binary volume. The formula that calculate the CG in discrete space is

$$\vec{CG} = \frac{\sum_x \sum_y \sum_z \vec{v}\rho(x,y,z)}{\sum_x \sum_y \sum_z \rho(x,y,z)}$$

where $\rho(x,y,z)$ of all voxels belonging to the bone equals 1. Hence,

$$\vec{COG} = \frac{\sum_{i=0}^{D_x} \sum_{j=0}^{D_y} \sum_{k=0}^{D_z} \vec{L}(i,j,k,T_t(i,j,k))}{\sum_{i=0}^{D_x} \sum_{j=0}^{D_y} \sum_{k=0}^{D_z} T_t(i,j,k)}$$

where

$$\vec{L}(x,y,z,w) = \begin{cases} (x,y,z) & w = 1 \\ (0,0,0) & otherwise \end{cases},$$

which returns out the coordinates of the voxel if its value is greater than *t*.

COGs of both sets are calculated. All the points in ***B*** are translated by vector $COG_B - COG_A$. It is presented by,

$$b'_i = b_i - (COG_B - COG_A), i = 1,2,...,m$$

The coordinates in ***A*** and ***B*** now are unified. By overlapping the centers of geometry, translation between the two can be ignored. To simplify the notation, b'_i that has been translated are still represented by b_i. In the rest of the thesis, b_i denotes points in ***B*** whose coordinates have been translated by $COG_B - COG_A$.

4.2.4 Seeking Corresponding Points

To match two point sets in 3D space, four pairs of points need to be found as the key points. The assumptions are as follows:

- One pair is defined, using the centers of geometry of the two sets, and they have been translated so that they overlapped each other.
- There are plenty of points in both sets.
- There must be points in the sets that can be found to match to each other.

The method for finding the remaining three pairs of key points now is introduced below.

There is an old method that finds out the position of the source of the radio wave, called radio fixing. The method was widely used to detect spies in World War II. The theory is very simple. Assume a radio station continuously broadcasts the radio wave. A radio wave detector, called radiophare, can receive the wave and figure out which direction the signal is the strongest. However it is unable to detect the distance to the radio station. So two units of radiophares are needed. Both of them can detect the direction of the source. They can fix the position of the source by combining the relative position of the units. (see Figure 4.8)

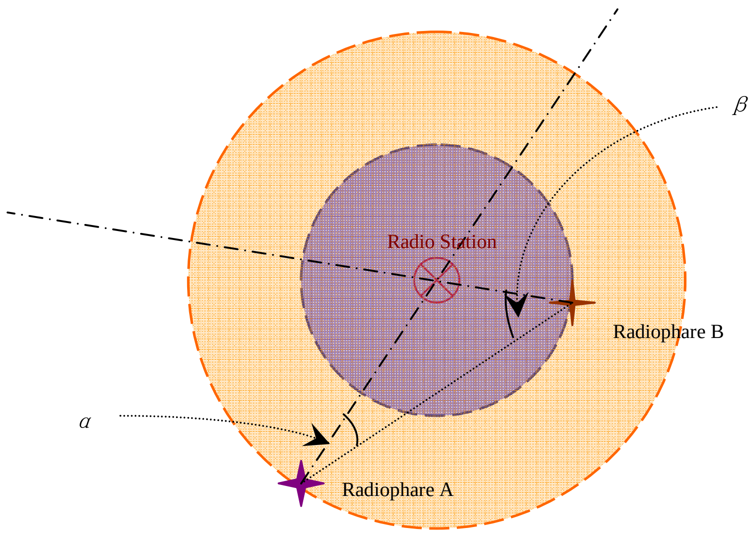


Figure 4.8

Radio fixing theory. Knowing the position of radiophere A, B and the angle *α*,*β*, the coordinates of radio station can be calculated by fundamental trigonometric functions.

The approach used here is very similar to the radio fixing. Assuming three points have been selected in ***A***, the corresponding points in ***B*** are going to be found. The difference between these two methods is, the radio fixing method can detect the direction of the source but not the distance; on the contrary in our method, the distance from the point to COG is known, but the direction is going to be found.

Select three key points in *A*

Three points in ***A*** are selected first as key points. Theoretically, these three points can be selected randomly except that all four points must be noncoplanar, including the COG. If the corresponding point of one of the key points cannot be found in ***B***, some other points in ***A*** will replace it. Since we assume that the number of feature points in each set is large enough, there must be three pair of points that can be found. Hence the replacement of the key points can be found somewhere. However random selection has many problems. First, the position of the points selected may be very close to each other, even the distance of them is as small as ε. While searching for the corresponding points in ***B***, it is very hard to determine which point the candidate corresponds to. The additional step that compares the distances from the candidate to both of the other two key points must be added. This is a waste of time. Second, it is hard to find the corresponding points of the key points in set ***B***. Because the key points have been selected randomly, it may be in anywhere in the list, which forces one to search the whole list of ***B***.

To avoid these shortcomings, all the points in both sets are sorted by the distance to the COG, from far to near. The sorted sequence is represented as follows,

$$\Theta_A = (a_{i_1}, a_{i_2}, \ldots, a_{i_n})$$

where $a_p \in \boldsymbol{A}$, and $\left\| a_{i_k} - COG \right\| \geq \left\| a_{i_{k+1}} - COG \right\|$.

The first key points in ***A*** is chosen to be the farthest point to the COG, called $P_A^1 = a_{i_1}$. The second key points is chosen to be the nearest point to the COG, called $P_A^2 = a_{i_n}$. The last one is selected in the middle part of the point list. Strictly speaking, the first candidate of the third key point is the one on the one third position of the total list, called $P_A^3 = a_{i_{\lfloor \frac{n}{3} \rfloor}}$. By using the fastest sorting algorithm, the running time of sorting both sets is $O(n\log n + m\log m)$.

The above selection method avoids selecting two points which are too close to each other, because the three points are distributed throughout the sequence, and hence their distances would be far enough to the others. However it cannot ensure that these three points are not collinear. A remedial approach will later be introduced in the method on how to select the third key point P_A^3. Nevertheless for the time being, these key points are assumed not to be collinear and to be completely suitable for the later steps.

Search the first candidate in *B*

The first candidate corresponding to P_A^1 is presented by $P_B^1 = b_{j_r}$. Since the point set ***B*** has been sorted, $\Theta_B = (b_{j_0}, b_{j_1}, ..., b_{j_m})$, it is very easy and fast to search P_B^1. Obviously, P_B^1 cannot be the fastest point to COG in B, it should be the point whose distance to COG is the most similar to the distance from P_A^1 to COG. For $\forall b_q \in B$, $q \neq j_r$,

$$\left\| \|b_q - COG\| - \|P_A^1 - COG\| \right\| \geq \left\| \|b_{j_r} - COG\| - \|P_A^1 - COG\| \right\|.$$

There is no need to search all the points in ***B***, but only search from the beginning of the sequence Θ_B. The point b_{j_r},

where $\left\| \|b_{j_r} - COG\| - \|P_A^1 - COG\| \right\| \leq \left\| \|b_{j_{r+1}} - COG\| - \|P_A^1 - COG\| \right\|$, is selected.

Search the second candidate in *B*

The second candidate selected in ***B*** to match P_A^2 is represented by $P_B^2 = b_{j_s}$. There is no need to search the whole set ***B***, either. P_B^2 is selected by searching the sequence Θ_B from the tail to the head. The point P_B^2 cannot be selected in the same way as P_B^1 where only the distance to the COG is being considered. $\|P_B^2 - P_B^1\|$ must also be similar to $\|P_A^2 - P_A^1\|$. The points in the sequence that $\left\| \|b_j - P_B^1\| - \|P_A^2 - P_A^1\| \right\| > \varepsilon$ should not be considered. In the rest of the points in the sequence, P_B^2 is selected in the same way as P_B^1. The point b_{j_s}

where $\left\| \|b_{j_s} - COG\| - \|P_A^2 - COG\| \right\| \leq \left\| \|b_{j_{s-1}} - COG\| - \|P_A^2 - COG\| \right\|$, is selected to be P_B^2.

Search the third candidate in *B*

The third candidate selected in ***B*** to match P_A^3 is represented by $P_B^3 = b_{j_t}$. Since P_A^3 is in the middle part of the sequence, it cannot be searched like P_B^1 or P_B^2. There is no hint on where P_B^3 exactly is. This point may appear in any position of Θ_B. Hence, it has to be searched from the very beginning of the sequence Θ_B. P_B^3 also needs to satisfy the same condition as that in searching P_B^2,

$$\left\| \left\| b_{j_t} - P_B^1 \right\| - \left\| P_A^3 - P_A^1 \right\| \right\| \le \varepsilon$$

and

$$\left\| \left\| b_{j_t} - P_B^2 \right\| - \left\| P_A^3 - P_A^2 \right\| \right\| \le \varepsilon .$$

Points which do not satisfy these conditions in the sequence will be ignored and the rest of them are searched. The point b_{j_t}

where $\left\| \left\| b_{j_t} - COG \right\| - \left\| P_A^3 - COG \right\| \right\| \le \left\| \left\| b_{j_{t+1}} - COG \right\| - \left\| P_A^3 - COG \right\| \right\|$, is selected to be P_B^3.

To understand how the method works, please see Figure 4.9. Assume 3 key points have been selected in ***A***, and the COG has already be fixed, the candidates of P_B^1 are on the sphere that is centered by point COG and whose radius equals $\left\| P_A^1 - COG \right\|$. Here the sphere can be imaged as a shell whose thickness is 2ε. All the points in the shell should accord to the condition that is able to match P_A^1. The points with the closest distance from the COG comparing to $\left\| P_A^1 - COG \right\|$ is selected (see Figure 4.9a). In selecting the second point P_B^2, two such shells are imaged. One is centered at COG and its radius is $\left\| P_A^2 - COG \right\| \pm \varepsilon$. The other is centered at P_B^1 and its radius is $\left\| P_A^2 - P_A^1 \right\| \pm \varepsilon$. If two spheres intersect each other, the intersection will be a circle or just a point. When the two shells described above intersect each other, the intersection will be something like a ring. The intersection should be a very tiny block too if and only if COG, P_A^1, and P_A^2 are on the same line. However, these two shells must intersect each other for $\left\|P_A^2 - P_A^1\right\| + \left\|P_A^2 - COG\right\| \geq \left\|P_A^1 - COG\right\|$ (see Figure 4.9b). There are very few points located within this ring. By searching the sorted sequence of B, it can be determined quickly whether there are some points satisfying to this condition, and select the best one among them. If there are no points lying in this ring, it is also very quick to find out if $\left|\left\|b_j - COG\right\| - \left\|P_A^2 - COG\right\|\right| > \varepsilon$ (see Figure 4.13).

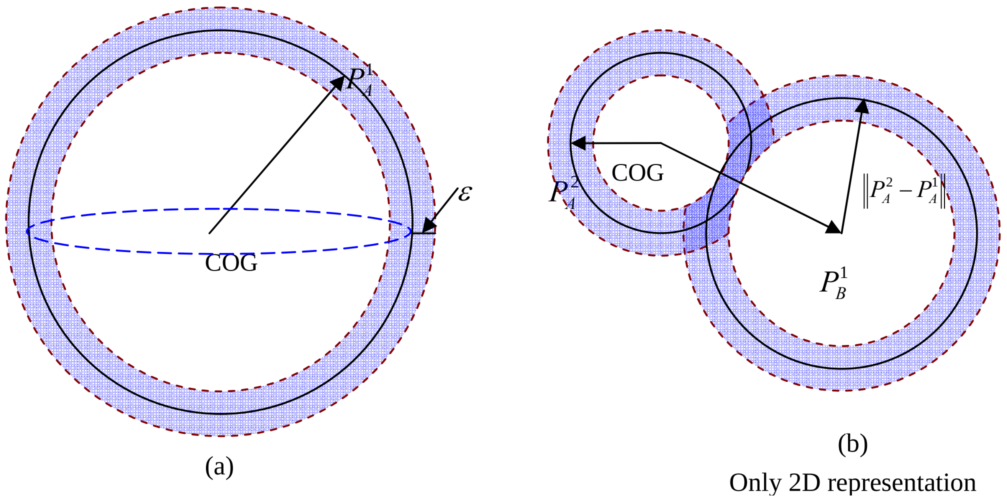


Figure 4.9

In searching P_B^3, there are more restrictions. First,

$$\left\|P_A^3 - COG\right\| - \varepsilon \leq \left\|P_B^3 - COG\right\| \leq \left\|P_A^3 - COG\right\| + \varepsilon,$$

which forms the first shell whose center is at COG and with radius $\left\|P_A^3 - COG\right\| \pm \varepsilon$.

Let S_1 denote the set of all the points in this shell. Second,

$$\left\|P_A^3 - P_A^1\right\| - \varepsilon \leq \left\|P_B^3 - P_B^1\right\| \leq \left\|P_A^3 - P_A^1\right\| + \varepsilon,$$

which forms the second shell whose center is at P_B^1, and with radius $\left\|P_A^3 - P_A^1\right\| \pm \varepsilon$.

Let S_2 denotes the set of all the points in this shell. Third,

$$\left\|P_A^3 - P_A^2\right\| - \varepsilon \leq \left\|P_B^3 - P_B^2\right\| \leq \left\|P_A^3 - P_A^2\right\| + \varepsilon,$$

which forms the last shell whose center is at P_B^2 and with radius $\left\|P_A^3 - P_A^2\right\| \pm \varepsilon$. Let S_3 denotes the set of all the points in this shell. (see Figure 4.10)

P_B^3 is in the set $S = S_1 \cap S_2 \cap S_3$. Theoretically, S has at most two points, but it cannot be an empty set.

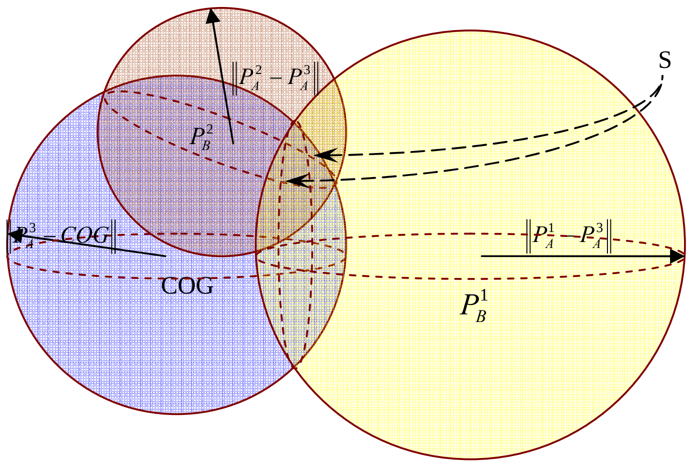
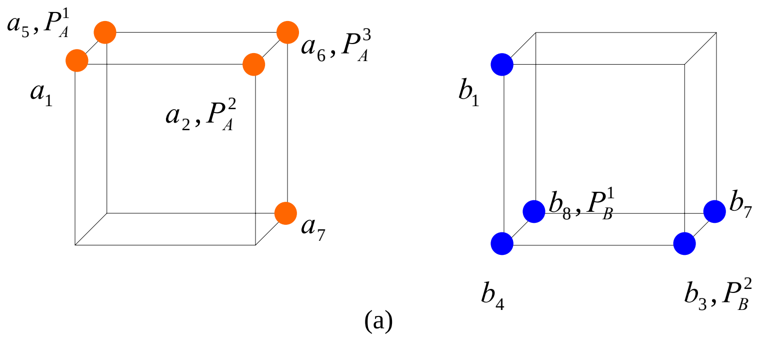


Figure 4.10 ε representation is omitted.
In continuous data space, S has two points, or one point if and only if P_A^1, P_A^2, P_A^3 is linear. S cannot be an empty set for $\|P_A^3 - P_A^1\| + \|P_A^3 - P_A^2\| \geq \|P_A^2 - P_A^1\| = \|P_B^2 - P_B^1\|$.

Since the ε-neighborhood is being considered, the intersections of these three shells are not two points but two tiny blocks. There may be several points lying in both blocks. Only points in one block are the right candidates. Here is a counterexample of what will happen when a wrong block is determined. (see Figure 4.11)
Assume the points in both sets are lying at the corner of a cube, but three of the eight points are missing. They are a_3, a_4, a_8 in ***A***, and b_2, b_5, b_6 in ***B***. The width, length and the height of the cube are all 1. In this special case, the distance from the COG to each point is specified to be equal, that means the COG is in the middle of the cube. Hence any point can be selected as P_A^i or P_B^i. Here assume $P_A^1 = a_5, P_A^2 = a_2, P_A^3 = a_6$. On the other hand, every point in ***B*** can be matched to any point in ***A***. Assume $P_B^1 = b_8, P_B^2 = b_3$, both b_4 and b_7 are candidates for P_B^3 because both of them lie in S that is specified by the three key points in ***A***.

When b_7 is selected as P_B^3, the transformation that matches ***B*** to ***A*** can hardly be solved by rotating around any axes.(see Figure 4.11b) The triangle $\Delta b_3 b_7 b_8$ is the mirror image of $\Delta a_5 a_2 a_6$. There is no possible Euclidean transformation that can transform ***B*** to ***A*** if these three pairs are selected. Nonetheless there should be a better match if b_4 has been selected as P_B^3. Although such an incorrect matching can be prevented if the AHD is carefully checked (the AHD of the matching is 0 if b_4 is selected as P_B^3, which is much smaller than if b_7 is selected as P_B^3), we still hope to prevent it before testing the AHD for computing the AHD is slow.



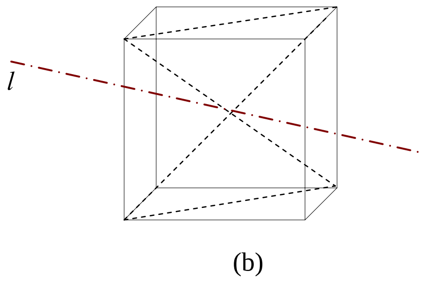


Figure 4.11
The original point sets. Big dots denote the points in the set. (b) ***B*** is rotated around *l* if right point is selected.

An additional step is added to avoid such situation. Define the Positive Normal Vector (PNV) of plane $COG\,P_\Psi^1 P_\Psi^2$ (where $\Psi = A, B$) as $(P_\Psi^1 - COG) \times (P_\Psi^2 - COG)$. The points on the same side with PNV as be considered. (see Figure 4.12)

This additional step is faster than calculating the AHD. It is very easy and fast to calculate the PNV, since

$$V = ((y_1 z_2 - z_1 y_2), -(x_1 z_2 - z_1 x_2), (x_1 y_2 - y_1 y_2)),$$

where $P_\Psi^1 - COG = (x_1, y_1, z_1)$ and $P_\Psi^2 - COG = (x_2, y_2, z_2)$. Determining the side of the plane $COG\, P_\Psi^1 P_\Psi^2$ on which the point lies is also easy. The cosine of the angle between PNV and $\psi_i - COG$ is given by,

$$\cos\theta = \frac{(P - COG)\cdot V}{|(P - COG\|V|},$$

where $P = \psi_i - COG$. θ lies in $(0, \frac{\pi}{2})$, if $\cos\theta > 0$. Since the denominator is always larger than zero, only $(P - COG)\cdot V$ needs to be calculated. This computation uses only addition and multiplication, and hence is very fast. This step is used for selecting both P_A^3 and P_B^3, which can save the time searching for points on the negative side. It also prevents selecting key points that are coplanar with the COG.

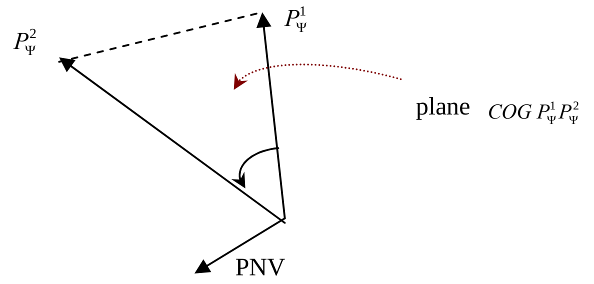


Figure 4.12
Only points on the positive side of the plane will be selected.

4.2.5 Partially Optimal Result

Theoretically, the optimal matching can be found if speed is not a consideration of the algorithm. Here is the slowest method to select key points and their corresponding points. Specify the three key points and search for their corresponding points one by one in the whole set of B. There is no doubt that this method can find out the optimal matching eventually, but the speed is extremely slow and the running time is $O(C_n^3 C_m^3 mn)$. There are C_n^3 possible combinations in ***A*** that can be selected as the key points and C_m^3 possible combinations that can be selected in ***B*** to match the key points in ***A***. For each available matching, the AHD should be tested, whose running time is $O(mn)$. Hence the total running time is $O(C_n^3 C_m^3 mn) \approx O(n^4 m^4)$. It is much slower than [40], whose running time is $O((m+n)^6 \log(mn))$.

Nonetheless the optimal solution is actually not necessary here. Speed is, however, an important consideration. Since all the corresponding points are selected in the ε-neighborhood of the key points, these three points has already been matched to the key points. Therefore, only three pairs of points can determine a matching. Finding out such points is a very easy and fast task if the method introduced in this thesis is used. In fact, it will be introduced as a speed-up method in a later section. However it may encounter a serious problem, which is called isomorphic combination. The three key points in ***A*** form a triangle. At the same time, their three corresponding points also form a triangle. If they lie in the ε-neighborhood of the key points, these two triangles have almost the same shape. If there are more than one such triangles that are composed of three points in ***B***, it has to be decided which combinations of three points should be selected. The AHD testing is still necessary to prevent such a situation. When key points match to their isomorphic combination, other points cannot be matched to their corresponding points. The AHD will then be a larger number and such a match can be eliminated easily.

In this section, we focus on how to select both key points and their corresponding points in the two sets skillfully in order to reach a lower AHD value and keep an acceptable speed at the same time.

It is unnecessary to compare every point in ***B*** with the key points for a candidate should lie in the ε-neighborhood of a key point. A candidate lies outside the ε-neighborhood of a key point when the difference between its distance to the COG and the distance from the key point to the COG is larger than ε. (see Figure 4.13)

Since the points in the set have been sorted by their distance to the COG, it is very easy and fast to determine which point in ***B*** are possible candidates. In addition, it can also select the best one to match the key point. The pseudocode describing the process of selecting candidate in the sorted sequence is as follows:

```
/* oa is the distance from the key point to the COG. Assume a is at the beginning of
sequence A*/

dis=MAX;
/*Initial the dis to maximum, which could be the maximum size of the volume. This
can assure the first comparing is valid. */
```

```
for (i=0;i<m;i++)
{
  ob=GetDistance(B(i));
  /*Get one point in B from the farthest to the closet and calculate its distance to
  COG*/

  if (dis<|ob-oa|)
  {
    if   (|ob-oa|>ε)
    {
      return -1;
      /*if |ob-oa|>ε, there is no point available. Return -1.*/
    }else
    {
      return i-1;
      /* if |ob-oa| is larger than the previous one, the previous b is the best candidate
      corresponding to a.*/
    }
  }else
  {
    dis=|ob-oa|;
    /* if this time the difference between ob and oa is smaller, then, the previous one
    is now the most similar one to oa. The next b will be test in the next loop.*/
  }
}
```

For the points like P_A^1 and P_A^2 at the two ends of the sequence, searching their candidate in ***B*** will be very fast. For the point P_A^3, searching its candidate is a bit slow. Although the first possible P_A^3 lies at the one third part of Θ_A, it doesn't mean the point corresponding to it also lies at the one third part of Θ_B. Hence P_B^3 needs to be searched from the beginning of Θ_B. P_B^3 can be found after searching about half of the sequence. It costs $O(m)$ at worst.

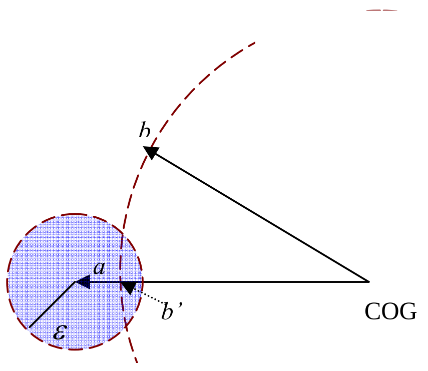


Figure 4.13

When $\left\| \|a - COG\| - \|b - COG\| \right\| > \varepsilon$, b lies outside the ε-neighborhood of a key point whatever the rotation around *COG* is.

Proof: Assume there is a rotation that brings *b* to *b'* around *COG*, where *b'*, *a* and *COG* are collinear. Since $\left\| \|a - COG\| - \|b - COG\| \right\| > \varepsilon$, and $\|a - b'\|$ is minimum now, so there is no rotation that can transform *b* into the ε-neighborhood of *a*.

By searching for points in the sequence for the best one lying in the ε-neighborhood of the key points and ignoring those lying outside the ε-neighborhood, all the three selected points must match to the key points very well. However it does not mean other points will match their corresponding points well too. To get a better matching, more key points and their candidates should be test. All the possible combinations of key points in ***A*** will be tested. Some of the combinations cannot find valid candidates in ***B*** and the AHD for such cases would not be tested. Otherwise, the AHD is tested and the combination and its corresponding candidates with minimum AHD is selected to be the final result. The pseudocode is as follows:

```
ahd=MAX;
/* Initial ahd to MAX. This can assure the first comparing is valid. */
while (Available())     //outer loop
/* If there are still points never been tested, continue the loop. */
{
  while (Available()) //middle loop
  /* If there are still points never tested, continue the loop. */
  {
    while (Available())     //inner loop
    /* If there are still points never tested, continue the loop. */
    {
      Pa1=Select1InA();
      Pa2=Select2InA();
      Pa3=Select3InA();
      Pb1=Search1InB(Pa1);
      Pb2=Search2InB(Pa1,Pa2,Pb1);
      Pb3=Search3InB(Pa1,Pa2,Pa3,Pb1,Pb2);

      if (Pb3>=0)
      /* If Pb3>=0, three candidates corresponding to their key points are found. */
      {
        newAhd= TestAHD(Pa1,Pa2,Pa3,Pb1,Pb2,Pb3);
        /* Test the AHD of this matching. */
        if (ahd>newAhd)
        {
          ahd=newAhd;
          KeepCombination(Pa1,Pa2,Pa3,Pb1,Pb2,Pb3);
          /* If the new AHD is smaller then the previous one, remember the current
          matching. If there is no other matching whose AHD is smaller than that of
          this, the matching is selected to be the final matching. */
        }
      }
      Ignore(Pa3,3);
      /* Ignore the current $P_A^3$. This one won't be selected in the next inner loop. */
    }
    Reset(3);
    Ignore(Pa2,2);
    /* Reset the point sequence A, for all the points ignored should be re-selected in
    the next middle-loop except the Pa2. */
  }
  Reset(2);
  Ignore(Pa1,1);
  /* Reset the point sequence A, for all the points ignored should be re-selected in the
  next middle-loop except the Pa1. */
}
```

Here are three loops: inner loop, middle loop, and outer loop. In the inner loop, P_A^1 and P_A^2 are fixed for each different P_A^3. Function *Select1InA()*, *Select2InA()*, and *Select3InA()* are used to select key points in ***A***. Any points in ***A*** that is marked by the function *Ignore()* will be ignored, for such points have been selected and tested before. Corresponding candidates for the selected points are searched in the sequence Θ_B by the approach introduced above by function *Search1InB()*, *Search2InB()*, and *Search3InB()*. If all the corresponding points are found, the AHD of this combination is tested and remembered if it is smaller than any previous ones. No matter whether the corresponding points have been found or not, the current P_A^3 is blocked by setting the flag in a corresponding array after the current search. In the next inner loop, this point will be ignored when selecting P_A^3 by checking the flag array. The function *Available()* return a positive value if there are still points that have not be flagged. The inner loop completes when all the points have been selected and tested. This time, P_A^2 is flagged. Function *Reset(i)* resets the elements marked by *i* in the flag array. Hence all the points will be selected as P_A^3 again except the one that has been selected as P_A^2, because its corresponding element in flag array has been set to *2*. The outer loop works as the same.

It is easy to observe the running time of this algorithm is $O(3n^4 + 3n^3m + n^4m)$. There are three loops, each of them need to do *n* times. The main body nesting in the inner loop does several jobs, selecting three candidates, which costs linear time at worst, searching P_B^1 , P_B^2, and P_B^3, which costs linear time at worst, and calculating the AHD. In the worst situation, the running time of the main body is $O(3n + 3m + nm)$. Including the three loops, the total running time at worst is $O(3n^4 + 3n^3m + n^4m)$. It is a bit faster than [40], but it can not reach the optimal result because not all the possible combinations in ***B*** have been tested. However, this method can get the optimal result among the combinations that candidate points lie in the ε-neighborhood of all the possible given key points, so called partial optimal result.

4.3 Speed-up Technique

In this section, some speed up techniques will be discussed. Carefully designed data structure and programming skills can be helpful to increase the speed.

4.3.1 Increasing Speed in Searching P_B^3

P_B^3 is a bit troublesome to be searched for it is in the middle of Θ_B. When P_A^3 is at the i^{th} position of Θ_A, it does not mean that P_B^3 is at the same position. All the

Θ_B should be searched to avoid missing any suitable P_B^3. This step can be finished in $O(m)$ time. Although it can be finished in linear time, it has to be done once in each iteration.

Here is a method for the speeding up the search for P_B^3. Assume that P_A^3 is at the i^{th} position of the sequence, and the search for P_B^3 begins at the i^{th} position of Θ_B. Compare the distance from the current element to the COG with $\left\|P_A^3 - COG\right\|$. If $\left\|b_{j_i} - COG\right\| < \left\|P_A^3 - COG\right\|$, search forwards, else search backwards. This method may have to search many points if the difference between $\left\|P_A^3 - COG\right\|$ and $\left\|b_{j_i} - COG\right\|$ is too large.

Another method is using a hash table. A hash table is a referential index in which each element points to the position of a group of elements in another list or sequence. A well designed hash function $h(k)$ maps a small number of keys k, called collisions, onto a unique integer i. If the number of collision is sufficiently small, the hash table gives $O(1)$ search time.

Let D_i^A denote the i^{th} element in Θ_A and D_j^B denote the j^{th} element in Θ_B. Design a hash function $h(k)$, put a group of elements $a_{i-q}, a_{i-q+1}, ..., a_i, a_{i+1}, ..., a_{i+p}$ such that, $h(k)$ gives out a unique integer if $D_{i-q}^A - D_{i+p}^A \leq c \cdot \varepsilon$. The hash function is defined as follows,

$$h(k) = \left\lfloor \frac{D_{MAX} - k}{c \cdot \varepsilon} \right\rfloor,$$

where D_{MAX} is the longest distance from the points in the two sets to the COG, and c is a constant. Operator $\lfloor x \rfloor$ gets the maximum integer smaller than x. This function maps a group of collisions onto a unique integer. An anti-hash function is also defined to map the unique integer to a group of elements in Θ_B,

$$h'(k) = \begin{cases} -1 & if\{b_j \mid 0 \le D_{MAX} - ck\cdot\varepsilon - D_j^B \le c\cdot\varepsilon\} = \phi \\ \min_j\{b_j \mid 0 \le D_{MAX} - ck\cdot\varepsilon - D_j^B \le c\cdot\varepsilon\} & otherwise \end{cases}$$

and $h''(k) = \{b_{h'(k)}, b_{h'(k)+1}, \ldots, b_{h'(k+1)}\}$.

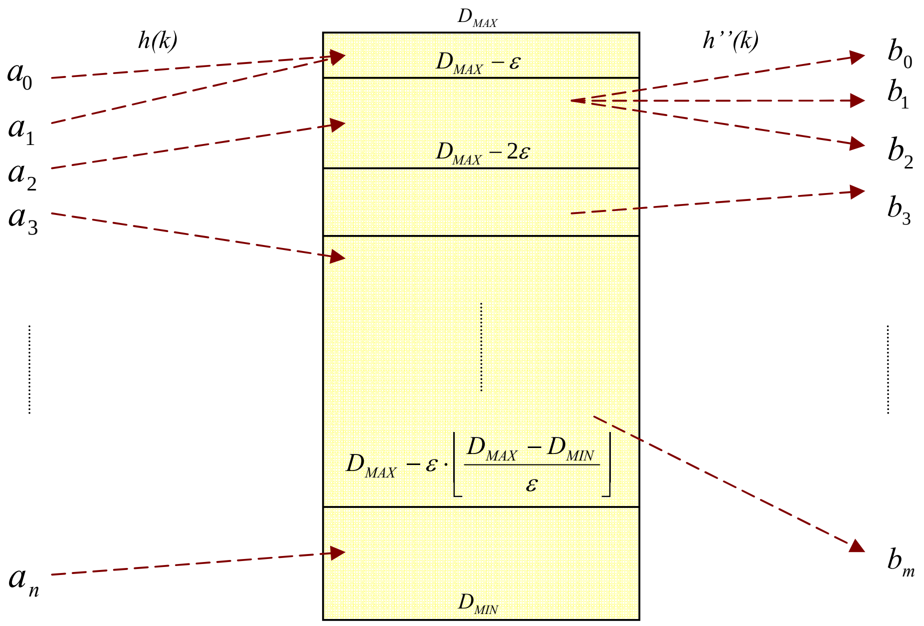


Figure 4.14
Elements are mapped by Hash function and anti-hash function.

An array, called hash array, stores the index of Θ_B indexed by $h(D_i^A)$. Any a_i is mapped to a unique integer by $h(D_i^A) = p$. This integer p is the index of the hash array. The value of p^{th} element in the hash array is given by $h'(p)$, which points to the first element of the available points is Θ_B. These points must be suitable to such condition: $\left|D_i^A - D_j^B\right| \le c\cdot\varepsilon$. It is easy to prove:

Let a_i is mapped to a group of points $b_j, b_{j+1}, \ldots, b_{j+s}$ in Θ_B listed from far to near with respect to COG, via $h(D_i^A) = p$ and $h'(p)$. Due to the definition of the hash function and the anti hash function, the following equations hold: $D_{MAX} - cp\cdot\varepsilon \ge D_i^A \ge D_{MAX} - c(p+1)\cdot\varepsilon \to (1)$.

And for any $t, j \le t \le j+s$, $D_{MAX} - cp\cdot\varepsilon \ge D_t^B \ge D_{MAX} - c(p+1)\cdot\varepsilon \to (2)$.

$D_i^A \ge D_{MAX} - c(p+1)\cdot\varepsilon \to (3)$

$D_{MAX} - cp\cdot\varepsilon \ge D_t^B \to (4)$,

Add (3) and (4),

$D_{MAX} - cp\cdot\varepsilon + D_i^A \ge D_t^B + D_{MAX} - c(p+1)\cdot\varepsilon \to (5)$

$\Rightarrow D_i^A - D_t^B \ge -c\cdot\varepsilon \to (6)$

$D_{MAX} - cp\cdot\varepsilon \ge D_i^A \to (7)$

$D_t^B \ge D_{MAX} - c(p+1)\cdot\varepsilon \to (8)$,

Add (7) and (8)

$D_{MAX} - cp\cdot\varepsilon + D_t^B \ge D_i^A + D_{MAX} - c(p+1)\cdot\varepsilon \to (9)$

$\Rightarrow c\cdot\varepsilon \ge D_i^A - D_t^B \to (10)$

Combine (6) and (10), there is

$c\cdot\varepsilon \ge D_i^A - D_t^B \ge -c\cdot\varepsilon \Rightarrow \left|D_i^A - D_t^B\right| \le c\cdot\varepsilon$.

When c is set to 1, all $b_j, b_{j+1}, ..., b_{j+s}$ must lie in the ε-neighborhood of a_i. That is how the hash table works. By calculating the hash function and anti hash function, very few points need to be searched. It can be used in searching for all candidates in ***B***. Hence the running time is decreased to $O(3n^4 + 3n^3 + n^4 m)$, and its initialization costs only linear time.

4.3.2 Simplified Loop

In practice, P_A^3 is found a bit difficult to be selected because there are too many restrictions on it. Hence there are few points in the list that can be suitable candidates for it. Even P_A^3 has been selected, P_B^3 is even more difficult to be found. The inner loop in the previous pseudocode fishes in the air most of time. Hence it is replaced in the implementation. P_A^3 is no longer selected one by one. Only P_A^1 and P_A^2 will be changed for a fixed P_A^3. In addition, the AHD calculation was moved to the outer loop.

Not all the suitable combination of key points will be tested, only the one first found will be tested:

```
Pb3=-1;
/* Initial Pb3 to enter the loop*/

while (Available())
{
  while (Pb3<0)
  {
    while(Pb3<0)
    {
      Pa1=Select1InA();
      Pa2=Select2InA();
      Pa3=Select3InA();

      Pb1=Search1InB(Pa1);
      Pb2=Search2InB(Pa1,Pa2,Pb1);
      Pb3=Search3InB(Pa1,Pa2,Pa3,Pb1,Pb2);
      Ignore((Pa1,1);
    }
    Ignore(Pa2,2);
    Reset(1);
  }
  if (Pb3>=0)
  /* If Pb3>=0, three candidates corresponding to their key points are found. */
  {
    newAhd= TestAHD(Pa1,Pa2,Pa3,Pb1,Pb2,Pb3);
    /* Test the AHD of this matching. */
    if (ahd>newAhd)
    {
      ahd=newAhd;
      KeepCombination(Pa1,Pa2,Pa3,Pb1,Pb2,Pb3);
      /* If the new AHD is smaller then the previous one, remember the current
      matching. If there is no other matching whose AHD is smaller than that of this,
      the matching is selected to be the final matching. */
    }
  }
  Ignore(Pa3,3);
  Reset(2);
  Reset(1);
}
```

The major modification to the simple loop is how to choose P_A^3. In the inner loop, P_A^3 could be treated as a fixed point for different combinations of other two points, provided that the current P_A^3 satisfies all the selection conditions. Actually, P_A^3 will be changed almost every time for different combination of P_A^1 and P_A^2. The accepted P_A^3 at end of the inner loop is the first one that all the candidates of three key points can be found in ***B***. The running time decreases to $O(n(3n^3 + 3n^2 m + mn))$, or $O(n(3n^3 + 3n^2 + mn))$ if a hash table is used to search for P_B^3.

4.3.3 Black List

Black List (BL) is a technique that prevents re-searching those specific combinations of key points which have no corresponding matching points. Notice in the simple loop, some combinations of key points will appear again. If such combinations have no corresponding points, it is a waste of time to search for their corresponding points in a later time. For example, three points are selected as key points in the first loop,

$$P_A^1 = a_{i_u}, P_A^2 = a_{i_v}, P_A^3 = a_{i_w}.$$

However no corresponding points are found for these points. a_{i_u} is first marked and replaced by another point $a_{i_{u'}}$. P_A^3 may also be changed to $a_{i_{w'}}$ for some reasons. It may be on the negative side of $COG\,P_A^1 P_A^2$, or it may be too close to the other key points. If the corresponding points of this combination have been found, $a_{i_{w'}}$ will be marked and $a_{i_{u'}}$ will be reset finally. In the next iteration, the combination $P_A^1 = a_{i_u}, P_A^2 = a_{i_v}, P_A^3 = a_{i_w}$ appears again. Combinations of points like that do not need to be searched again in order to save times. Hence BL is introduced.

BL is a 1-dimentinal array whose size is n^3. Each combination $P_A^1 = a_{i_u}, P_A^2 = a_{i_v}, P_A^3 = a_{i_w}$ is recorded by a unique element in this array whose index is $i_u \cdot i_v \cdot i_w$. Each element is a Boolean value which is initialized to false. If a combination of key points has no corresponding points, its corresponding element in the BL is set to true. When the combination is selected next time, it is ignored if its value in BL is true. It could save a lot of time when searching for P_B^x .

The only concern of BL is the cost of memory space. The size of BL is n^3. For a sequence with hundreds of points, the cost of memory space is not very large, especially for the PC nowadays.

The effect of BL is very remarkable. The table 4.1 shows the running time of the program using BL or not:

Samples	Number of Points		Running Time(s) All using Simplified Loop			Total Time saving
	A	***B***	Simplified Loop only	Black List	BL & Hash Table	
S1	113	119	0.540	0.250	0.221	59.1%
S2	97	98	0.340	0.230	0.191	43.8%
T1	175	147	1.503	0.731	0.430	71.4%
T2	292	336	12.368	4.757	1.973	84.0%
T3	231	221	4.156	1.853	1.221	70.6%
T4	194	235	3.796	1.542	1.292	66.0%
T5	141	138	1.062	0.431	0.181	83.0%
T6	375	291	17.726	5.447	3.896	78.0%
T7	175	175	2.263	1.832	1.111	50.9%

Table 4.1

CPU: Pentium II, 300MHz.

4.3.4 ε-neighborhood Determination

The setting of ε-neighborhood influences the speed too. The greater the ε is, the more combinations can find their corresponding points, and the more chances the AHD needs to be tested, and hence the slower the speed is. On the other hand, if the ε is set to too small, it may be unable to find the matching points, especially for the case when there are fewer points in the sets. No-matching is reported if the ε is too small in such cases.

It is necessary to use a function to set the ε automatically according to the number of the points in the sets. Since the size of the cube that is used to detect feature points is 15, the error distance maximizes to half of its size. Hence 7 is the maximum error distance that can be accepted. The position of the feature points is much more accurate than what is expected. 7 is even large. An experiential function is used to determine the ε:

$$\varepsilon = 7 - \min(\frac{m+n}{200}, 6),$$

which specifies $\varepsilon \in [1,7]$.

Table 4.2 shows the running time of different ε settings:

Samples	Number of Points		Running Time with different ε(s)			AHD		
	A	B	7.0	2.5	Auto-determining	7.0	2.5	Auto-determining
S1	113	119	0.300	0.181	0.221	1.705	1.789	1.761
S2	97	98	0.180	0.120	0.191	5.014	9.913	5.014
T1	175	147	0.521	0.351	0.430	2.363	2.426	2.363
T2	292	336	6.780	2.424	1.973	3.283	4.144	3.943
T3	231	221	2.130	1.191	1.221	2.671	2.522	2.671
T4	194	235	2.504	0.981	1.292	3.642	4.715	3.959
T5	141	138	0.701	0.281	0.181	7.283	9.169	9.169
T6	375	291	9.214	4.987	3.896	2.598	2.834	3.139
T7	175	175	1.592	1.292	1.111	2.502	2.468	2.468

Table 4.2

CPU: Pentium II, 300MHz. Simplified loop, BL and Hash Table is used.

It is very clear that smaller ε can really increase the speed, because fewer candidates for each combination of key points are selected and tested by AHD algorithm. On the other hand, lager ε sometimes cannot achieve a better matching. A very important point should be noted that three key points matched by their corresponding points very well does not mean other points can be matched by their corresponding points. Although there are more candidates selected when ε is set larger, simplified loop only selects the first available pair of points as P_A^3 and P_B^3. This pair may not as good as the later ones. Anyway, the auto-determining function works very well. The speed is much faster than ε=7. The AHDs are smaller than the other two in most cases.

4.3.5 Other Speed-up Techniques

One of the speed-up techniques that sacrifices accuracy has been mentioned once in 4.2.5. With a carefully set ε, AHD does not need to be tested for each available combination of key points and their corresponding points because each corresponding point must lie in the ε-neighborhood of a key point. Theoretically speaking, such three pairs of points are already matched. Since AHD testing is no longer needed, running time is saved. Considering the worst situation, the corresponding points are found after all possible combinations have been tested. The running time is only $O((n+m)^3)$. If other speed-up techniques have been used, like hash table, the running time is decreased to $O(n^3)$. Table 4.3 shows how fast this method is.

This speed-up technique sacrifices the accuracy. It only makes an available matching but not an optimal matching, even a better matching. The result could be a disaster for some cases, such as T2 and T4, especially with a larger ε. Nonetheless under a well controlled ε, this method performs well and the speed is extremely fast. However, this method sometime cannot match two sets if isomorphic combinations exist in one of the sets.

Samples	Number of Points		Running Time with different ε(s)		AHD		
	A	B	No AHD Testing	Simplified Loop/BL/Hash Table/Auto-matically ε determination	No AHD Testing		Simplified Loop/BL/Hash Table/Automatically ε determination
					ε=7.0	Automatically ε determination	
S1	113	119	0.000	0.221	2.578	2.578	1.761
S2	97	98	0.000	0.191	6.813	6.813	5.014
T1	175	147	0.000	0.430	8.697	8.697	2.363
T2	292	336	0.100	1.973	12.610	3.945	3.943
T3	231	221	0.000	1.221	6.047	2.697	2.671
T4	194	235	0.000	1.292	18.195	3.674	3.959
T5	141	138	0.000	0.181	9.172	9.172	9.169
T6	375	291	0.100	3.896	2.928	3.154	3.139
T7	175	175	0.000	1.111	2.507	2.507	2.468

Table 4.3
CPU: Pentium II, 300MHz.
The running time of No AHD Testing algorithm is the same no mater how much ε is.

There is another method suitable for some special cases. It works well under the assumption that the center of the feature points is quite different with COG. The feature points are usually detected on one side of the bone in such cases. The definition of "the center of the feature points" (COF) is as follows,

$$COF_A = \frac{\sum_{i=0}^{n} a_i}{n}, COF_B = \frac{\sum_{i=0}^{m} b_i}{m}.$$

COF actually is the average of all coordinates in *x,y,z* axis respectively of all feature points in on set. The description of this method is as follows,

1. Calculate COF in each set.
2. Find a transformation to overlap $COF_B - COG$ to $COF_A - COG$. It is very easy to find a transformation to overlap two vectors in 3D space.
3. Transform all the points in B by this transformation matrix. Now there is only one degree of freedom left to match two sets, which is the rotation around the vector $COF_A - COG$.
4. Project all points in both sets onto a uniform sphere centered by COG and represent them in polar coordinates with latitude and longitude only. The axis of the uniform sphere is $COF_A - COG$. Now the problem finding a matching allowing rotation in 3D space is transformed to the problem finding a matching allowing translation in 2D space.
5. Find an optimal matching by using some algorithms working under translations, like the algorithm introduced in [35].

This method is very fast and the result is optimal too. If the algorithm introduced in [35] is used to find the matching in the last step, the running time is $O(nm(n+m)\log nm)$.

The shortcoming of this method is that it cannot handle the situation when the feature points are distributes in the space uniformly. In this case the COG and COF are very close, and this causes a great error when overlapping the two vectors.

Chapter 5

Alignment and Comparison

Since the transformation matrix *M* has been obtained, the total volume now is transformed by this matrix:

$$V' = \begin{pmatrix} x' & y' & z' & 1 \end{pmatrix} = VM$$

Since the matrix has floating-point elements, the new coordinates are not integer either, which is not suitable in discrete space. So,

$$V'' = \begin{pmatrix} \lfloor x'+0.5 \rfloor & \lfloor y'+0.5 \rfloor & \lfloor z'+0.5 \rfloor & 1 \end{pmatrix}$$

After alignment, comparing two volumes becomes very easy:

$$V_C(x,y,z) = \left| V_O(x,y,z) - V''(x,y,z) \right|$$

where $V_O(x,y,z)$ is the value of the voxel whose coordinates are *(x,y,z)* in the sample volume.

Chapter 6

Conclusion

In this thesis, series of methods are introduced to align two volumes of samples, which contain the same bone but one of them has been scanned after bone cement injection. The shape of the bone cement extracted from the data, and surgeons can then easily observe, evaluate the bone cement injected into the bone. It is no longer necessary to cut off the bone to measure the result of the injection, which can totally destroy the sample. Students of Orthopaedic Surgery who have less experience can be trained to distinguish the part belonging to bone cement from the other tissue by studying the extracted part of bone cement. Researchers in computer graphic also can test their new classification method by comparing with the direct results.

The research is based on the VISBONE system, which have solved the volume visualization problem. However the old version of VISBONE runs on a high-end SGI workstation and programming and maintenance is difficult. Hence at the very beginning of this research, a new version of VISBONE that can run on the PC platform with the help of the VolumePro chipset has been developed. The new version performs better than the old version and it can easily be expanded. The new version also has collected all the useful features of the original VISBONE, especially the Transfer Function Editing and Cut Plane. These features help us observe the internal structure of the bone, analyze the density of the bone cement, which is very helpful for the future researches.

However, some automatic methods for extracting the bone cement without comparison were planned to be developed at first. After several months' endeavor, the results were not as satisfactory as we expected. Two main problems could not be solved no matter which methods were used. One was that the boundary between the bone cement and the harder part of the bone can not be detected because they adhere together and the densities are similar. The other was that the extracted part could not be proved that it is exactly the part belonging to the bone cement, though it looks like the bone cement visually. These two problems were the motivation to develop an approach to compare two real samples automatically.

Detecting feature points in the mass of voxels was the first task. Traditional feature point detectors had been found not to be suitable for volume data sets, especially the volume containing human organs. Traditional feature point detectors usually use small filters that could not deal with large structures. On the other hand, they are unable to produce stable result when the volume is rotated. Although large size filters have been developed, plenty of decimal calculations are needed. Our feature point detection method is based on statistics instead of fixed filters. Feature points at sharp corner of the bone cortices are detected by analyzing the statistic data of the voxels' allocations in a cube with a specified size. This method could avoid the influence of changing connection relationship among the voxels when the volume is rotated in discrete space. It is also faster than the traditional filters because it has only integer addition and multiplication only. This method could be used widely for its flexibility. The size of the testing cube, the sharpness of the corner, the threshold of density all could be set to fit special tasks. In practice, this method works very well. Most of the expected points could be detected in both samples. The accuracy of feature point detection eases the difficulty in later works.

The step of matching two sets of feature points may be the most difficult part in the thesis. Literature review has been done before the work began. Not many papers were found that present methods suitable for the special case we met. A matching method that balances the speed and accuracy as well as possible was expected to be developed, but not a method which can achieve the optimal results for the best matching is considered to be useless sometimes in the discrete space. Based on the fundamental geometry theory, our method implemented the matching by seeking three key points and their corresponding points, which all of them lied in some fixed error region of the key points. This is done after the translation between the two sets has been solved by a transigent method, in which the Center of Geometry of the bone is introduced. An easy and direct way to calculate the transformation matrix is also used not only for transforming the points in the set but also measuring how good the matching was. The transformation matrix got by this way might lead to distortions, which is the exact effect we wanted. Worse matching with greater distortion, which could easily be tested, would soon be eliminated. Better matching with less distortion is acceptable for the distortion will amke no effects visually at least. Traditional measurement, called Hausdorff Distance, for measuring how good the matching was in point pattern matching has also been modified to suit our special case.

Other speed-up techniques as well as skillful selection for key points are implemented in the program. Skillful selection for key points makes it unnecessary to seek corresponding points for all the combinations of key points. Thus, the running time is reduced. Others, like the hash table, improve the speed efficiently.

Although most of the approaches introduced in this thesis have been implemented, source codes for each step have not been combined together. A complete system is expected to be done first in the future works. We hope to develop several methods to analyze the structure of the bone cement automatically after it has been extracted. The development of a method for extracting the bone cement automatically in the volume data is still under way.

Contents

Bibliography

[1] W. E. Lorensen and H. E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm.

[2] Dirk Bartz and Michael Meiβner. Voxels versus Polygons: A Comparative Approach for Volume Graphics. *Volume Graphics: Springer Book*, pp171-183

[3] Tuy H, Tuy L. Direct 2D display of 3D objects. *IEEE Computer Graphics and Applications*, 1984; 4(10): 29-33.

[4] Levoy M. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 1988; 8(3): 29-37.

[5] Frieder G, Gordon D, Reynolds R. Back-to-front display of voxel-based objects. *IEEE Computer Graphics and Applications*, 1985; 5(1): 52-59.

[6] Wilhelms J, van Geldern A. A coherent projection approach for direct volume rendering. In: Proc. *ACM SIGGRAPH Conference*, 1991; 275-284.

[7] Westover Lee. Footprint evaluation for volume rendering. Proceeding of SIGRAPH '90. *Computer Graphics*, 24(4): 367-376, August 1990.

[8] Robert A. Drebin, Loren Carpenter, and Pat Hanranhan. Volume rendering. *Computer Graphics*, 22(4): 65-74, July 1988. ACM SIGRAPH '88 Conference Proceedings.

[9] P Lacroute and M. Levoy. Fast volume rendering using a shear-warp factorization of the view transform. *Computer Graphics*, Proceedings of SIGRAPH 94, pages 451-457, July 1994.

[10] Hanspeter Pfister, Jan Hardenbergh, Jim Knittel, Hugh Lauer, Larry Seiler. The VolumePro Real-Time Ray-Casting System.

[11] P. Lacroute. Analysis of a parallel volume rendering system based on the shear-warp factorization. *IEEE Transactions on Visualization and Computer Graphics*, 2(3): 218-231, September 1996.

[12] Yi-King Choi, Leong J.C.Y., Lu, W.W., Wenping Wang. VISBONE: 3D visualization of bone mineral density. *Computer Graphics and Applications*, 1999. Proceedings. Seventh Pacific Conference on, 1999 Page(s): 138 -146, 321.

[13] H. Pfister and A Kaufman. Cube-4 – A scalable architecture for real-time volume rendering. In *1996 ACM/IEEE Symposium on Volume Visualization*, pages 47-54, San Francisco, CA, October 1996.

[14] R. Osborne, H. Pfister, H. Lauer, N. McKenzie, S. Gibson, W. Hiatt, and T. Ohkami. EM-Cube: An architecture for low-cost real-time volume rendering. In Proceeding of the SIGRAPH/Eurographics *Workshop on Graphics Hardware*, pages 131-138, Los Angeles, CA, August 1997.

[15] R. Yagel and A. Kaufman. Template-based volume viewing. *Computer Graphics Forum, Proceeding Eurographics*, 11(3): 153-167, September 1992.

[16] P. Schröder and G. Stoll. Data parallel volume rendering as line drawing. In 1992 *Workshop on Volume Visualization*, pages 25-31, Boston, MA, October 1992.

[17] T. Laine, D. Schlenzka, K. Mäkitalo, K. Tallroth, L. Nolte, and H. Visarius. Improved Accuracy of Pedicle Screw Insertion With Computer-Assisted Surgery. *SPINE*, Volume 22, Number 11, 1997, pp. 1254-1258.

[18] Olga Sourina, Alexei Sourin, and Howe Tet Sen. Virtual Orthopedic Surgery Training on Personal Computer. *International Journal of Information Technology*, Volume 6, No. 1, May 2000.

[19] *http://www.osteonics.com/simplex_us/pages/intro.html*

[20] *http://www.irc-biomed-materials.qmul.ac.uk/Pages/Research/Bonecement/Research-Bonecement.htm*

[21] W.W. Lu, JCY Leong, YW Li, KCM Cheung, KDK Luk, KY Chiu, A Holmes, SP Chow. Injectable Bioactive Bone Cement for Spinal Surgery: A Developmental and An in vitro Biomechanical and Morphological Study.

[22] G.D.Rubin, C.F.Beullieu, V.Anngiro, et al.: "Perspective Volume Rendering of CT and MR Images: Applications for Endoscopic Imaging", *Radiology*, Vol.199, pp.321-330 (1996).

[23]M. W. Vannier, R. L. Butterfield, D. Jordon, W. A. Murphy, R. G. Levitt, and M. Gado. Multispectral Analysis of Magnetic Resonance Images. *Radiology*, vol. 154, no. 1, pp. 221-224, 1985.

[24] M. C. Clark, L. O. Hall, D. B. Goldgof, L. P. Clark, R. P. Velthuizen, and M. S. Silbiger. MRI Segmentation Using Fuzzy Clustering Techniques. *IEEE Eng. In Medicine and Biology*, vol. 13, pp. 730-742, 1994.

[25] Yoshinobu Sato, Carl-Fredrik Westin, Abhir Bhalerao, Shin Nakajima, Nobuyuki Shiraga, Shinichi Tamura and Ron Kikinis. Tissue Classification Based on 3D Local Intensity Structure for Volume Rendering. *IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS*, Vol. 6, No. 2, April-June 2000.

[26] Y. Sato, S. Nakajima, H. Atsumi, T. Koller, G. Gerig, S. Yoshida, and R. Kikinis. 3D Multiscale Line Filter for Segmentation and Visualization of Curvilinear Structures in Medical Images. *Proc. Joint Conf. Computer Vision, Virtual Reality, and Robotics in Medicine and Medical Robotics and Computer-Assisted Surgery '97*, pp. 213-222, 1997.

[27] Daniel Cohen, and Arie Kaufman. Scan-Conversion Algorithms for Linear and Quadratic Objects. *Volume Visualization*, Arie Kaufman. IEEE Computer Society Press Tutorial.

[28] David G. Morgenthaler, and Azriel Rosenfeld. Surface in Three-Dimensional Digital Images. *Volume Visualization*, Arie Kaufman. IEEE Computer Society Press Tutorial.

[29] Hui Chen, Wenping Wang, and Ralph Martin. Building Panoramas from Photographs Taken with An Uncalibrated Hand-Held Camera.

[30] Steven W. Zucker, and Robert A. Hummel. A Three- Dimensional Edge Operator. *Volume Visualization*, Arie Kaufman. IEEE Computer Society Press Tutorial.

[31] C. John. A Computational Approach to Edge Detection. *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 8(6), pp. 679-698, Nov. 1986.

[32] Helmut Alt, and Leonidas J. Guibas. Discrete Geometric Shapes: Matching, Interpolation, and Approximation. *Handbook of Computational Geometry*, Edited by J.-R. Sack, and J. Urrutia.

[33] M.D. Atkinson. An Optimal Algorithm for Geometrical Congruence. *J. Algorithms 8* (1987), 159-172.

[34] H. Alt, K. Melhlhom, H. Wagener and E. Welzl, Congruence. Similarity and Symmetries of Geometric Objects. *Discrete Compu. Geom*. 3(1988), 237-256.

[35] E. M. Arkin, K. Kedem, J. S. B. Mitchell, J. Sprinzak and M. Werman. Matching points into pairwise-disjoint noise regions: Combinatorial bounds and algorithms. *ORSA J. Comput*. 4(4) (1992), 375-386.

[36] A. Efrat, and A. Itai. Improvements on Bottleneck Matching and Related Problems Using Geometry. *Proc. 12th Annu. ACM Sympos. Comput. Geom*. (1996),301-310.

[37] P. J. Heffernan and S. Schirra. Approximate Decision Algorithms for Points Set Congruence. *Comput. Geom*. 4(1994), 137-156.

[38] B. Behrends. Algorithms zur Erkennung der ε-Kongruenz von Punktmengen und Polygonen. *M.S. thesis, Freie Univ. Berlin, Institute for Computer Science (1990).*

[39] D. P. Huttenlocher, K. Kedem, and M. Sharir. The upper envelope of Voronoi surfaces and its applications. *Discrete Comput*. Geom. 9(1993), 267-291.

[40] D. P. Huttenlocher, K. Kedem, and J. M. Kleinberg. On Dynamic Voronoi Diagrams and the Minimum Hausdorff Distance for Point Sets Under Euclidean Motion in the Plane. *Proc. 8th Annu. ACM Sympos. Comput. Geom*. (1992), 110-120.