INTERACTIVE VOLUME VISUALIZATION IN A VIRTUAL ENVIRONMENT

by

Yu-Hang Siu

Submitted to the faculty of the Graduate School

in partial fulfillment of the requirements

for the degree

Master of Philosophy

in the Department of Computer Science and Engineering

The Chinese University of Hong Kong

August 1998

摘要

這篇論文的主要目的是探討如何在虛擬環境中發展出有效的體數據可視化(volume visualization)方法。傳統上體數據可以在二維的螢幕上顯現出來。但隨著電腦能力的提升和虛擬真實技術的進步，二維的顯示和操作方式已再不能滿足大部份的用家了。我們現在需要的是一個能容許三維顯示和操作的真正虛擬環境。

這個研究將會集中於體數據在虛擬環境中的操縱。這裡會提出一個新的體數據切割技術。這個技術主要建基於一個叫‘智能剪刀’(Intelligent Scissors)的二維圖像分割算法。我們採用一個相似的動態程序設計(dynamic programming) 技術去求出一個體數據表面上的關閉等值線。利用這條等值線我們可以求出一個切面。最後就可以用一個區域生長算法(region growing algorithm)去將那個體數據一分爲二。

我們亦會介紹一個新的等值面抽取(iso-surface extraction)方法。在醫學資料的曲面重組(surface reconstruction)過程中，爲了要有準確的結果，大多數圖像切面都需要考察。這個程序是很複雜的，因爲它需要抽取每一塊圖像切面的等值線和很多的人手修正。所以，我們把‘智能剪刀’的概念擴展和引用到體數據上。這個算法的主要目的是提供一個簡單易用的曲面重組界面和將涉及的圖像切面和人手修正減至最少。

最後，因爲三維的‘智能剪刀’可以抽取依付在曲面上的線條，這令它有找尋路徑的能力。如果一條管道的兩端是已知的，那麼這條路徑就可以憑三維‘智能剪刀’求出來。這個特性可以用來找出某些醫學體數據上的血管。我們亦做了一個實驗去證明它可以找出肺部體數據上的氣管位置。

Abstract

This thesis concerns how to achieve efficient volume visualization in virtual environment. Traditionally, volumetric data is visualized and displayed on a two-dimensional screen. With the increasing computational power and improving techniques for Virtual Reality(VR), people are no longer satisfied with two-dimensional input and display. Instead, a real virtual working environment with three-dimensional input and output is found to be more appropriate.

In this research, I concentrated on the manipulation of volumetric data in a virtual environment. A new technique for volume cutting is proposed. It is based on a technique called Intelligent Scissors, which is a two-dimensional image segmentation tool. I adapted a similar technique which used dynamic programming(DP) to find out closed contour along surfaces of volumetric data. A cutting surface is produced using the resulting contour. The volume can then be cut into two parts by a simple region growing algorithm.

Besides volume modeling, a new iso-surface extraction algorithm is also described. For surface reconstruction in medical imaging, every slice has to be examined in order to give accurate result. The process is troublesome since it requires every contour from every slice and it always involves a lot of manual corrections. Therefore, I proposed an algorithm that extended the idea of Intelligent Scissors and applied it on volumetric data. The aim of the proposed algorithm is to give a comprehensive interface for surface extraction while minimizing the number of slices involved and the number of manual corrections.

Since the extension of Intelligent Scissors can extract curves lying on surfaces, it gives the possibility of "path finding". Provided that the end points of a path are

known, the path can be found out by three-dimensional Intelligent Scissors. This property can be used to find out blood vessels in medical data and some experiments are carried out to extract paths in lung data.

Acknowledgements

Thanks my supervisors, Prof. P.A.Heng and Prof. H.Sun, for their invaluable advice during these two years. I would also like to thank Kevin Wong for his great contribution to the Volume Cutting algorithm, which is a fairly important part of this thesis. Finally, I would like to thank Mrs. Chan for proofreading the drafts.

Contents

List of Figures

Chapter 1

Introduction

Scientific visualization is an important field of computer graphics. Since many objects and natural phenomena surrounding us are 3D volumes of data, many scientists need various kinds of scientific visualization to examine their data sets, for example, computational fluid dynamics(CFD), medical CT/MRI imaging, volumetric data in molecular systems, earth science and seismic data about underground layers, etc. All of these fields play important roles in our life and careful examinations of the related data sets are often necessary. For instance, seismic data can be examined with a volumetric scheme, which provides geophysicists an alternative way to reveal the structure of geological layers and such information is proved to be useful for identifying potential oil reservoirs[40, 51].

Volume visualization as a subfield of scientific visualization concentrates on the examination of complex volumetric data. The main objective of volume visualization is to provide methods that can visualize internal structures of data, analyze and manipulate volumetric data efficiently and effectively. Among various applications, medical imaging is the main driving force for the development of volume visualization because of its direct practicality and importance. Medical data can be scanned from patient by computed tomography(CT), magnetic resonance imaging(MRI), or ultrasonography in a non-destructive and safe manner. Therefore, it is useful for many medical applications, for example, clinical diagnosis, orthopedic diagnosis, radiation therapy planning, medical education and surgical planning.

The other important issue is how to choose a suitable working environment for volume visualization. A two-dimensional display is suitable for two-dimensional computer graphics. For three-dimensional graphics, a normal display can still be able to give user a sense of depth by using shading, shadows, or even mirrors. However, it is not the most natural way to view and interact with 3D objects using 2D display and input. Therefore, in this research, I focus on volume visualization in a virtual environment and attempt to make efficient tools for volume manipulation in such an environment.

We discuss volume visualization in more detail in Section 1.1. A brief summary of the virtual environment we used is described in Section 1.2. Our research approach is described in Section 1.3. Finally, we give the thesis overview in Section 1.4.

1.1 Volume Visualization

The technique used to visualize three-dimensional volumetric data is called volume rendering. An element of the three-dimensional array is called a voxel. The most basic steps of volume rendering algorithms are assigning a color and opacity to each voxel in the array, making samplings in the volume, projecting the samples onto an image plane and compositing the samples together to get the final image. There are numerous volume rendering algorithms published in the past. Some of them are now only of historic interest because of the low quality of resulting images. Several important techniques are summarized as follows:

- **Rendering volume as shaded cubes.** In 1979, Herman and Liu[18] proposed an algorithm which rendered CT data as many shaded cubes. An appropriate threshold is applied to the volume to choose boundary voxels. Those voxels are rendered as opaque cubes represented by six equal-sized squares.

Each face of every cube can be shaded by standard shading algorithms and hidden surface removal is done by the Z-buffer algorithm. The main disadvantage of the algorithm is that it will form a solid with blocky effect, which can be improved by a low-pass filtering.

- **Surface construction by joining contours.** Rendering volume as shaded cubes causes aliasing effect and thus results in low quality images. Therefore, improved surface rendering techniques often use triangles to model the surface of volumetric data. One kind of such algorithms extracts contours from all image slices and then joins adjacent contours together by triangular strips. There are two main steps in such algorithms: first, closed contours from each slice are extracted by some edge detection algorithms[8, 26]; and a contour joining strategy is used to find optimal triangular strips to fit adjacent contours[19, 31].

  For contour detection, a novel technique called active contour(or snakes)[26] is used. Firstly an user has to make a rough approximation of the shape of contour. Each point in the boundary is associated with an energy function which is a combination of internal forces, such as boundary curvature, and external forces, like image gradient magnitude. The contour grows and shrinks in order to minimize its energy. The main drawback of the algorithm is, the user would never know what the final shape would be at the very beginning. If the final contour is not satisfactory, the user has to manually edit the resulting contour or restart the whole process with a new initial contour.

  Joining of contours is a more complex problem. One image slice may contain two or more contours of the desired object. A branch would occur if one of the adjacent planes contains more than one contours. Moreover, the problem becomes more complex if the distance between two planes are relatively large with respect to the size of cell. In[10], Meyers et al. broke the main problem

into four subproblems, namely, correspondence, tiling, branching, and surface fitting.

- **Marching cubes and Dividing cubes.** Instead of extracting contours, some algorithms extract triangles that define the surface of a constant intensity. *Lorensen and Cline* proposed *Marching Cubes*[28] to extract a list of triangles by marching through all the cubes. A cube is defined by eight surrounding voxels and each voxel is identified as either inside or outside the object according to a threshold intensity. Based on the configuration of voxels, each cube can be fitted in triangles that define part of the surface. Marching cubes is widely used to generate iso-surface because it is fast and can make high-resolution models. There are many variants of Marching cubes [35, 39, 38, 34] that either fasten the extraction process or decrease the number of triangles extracted.

  The Marching Cubes algorithm constructs iso-surface as polygonal elements. Another algorithm, called Dividing Cubes[7], renders the surface as 3D points with normal vector. Each cube in the volume is identified as either inside, outside, or intersecting the surface. A cube which lies on the surface is subdivided into small cubes such that each sub-cube is the same size of a display pixel. The normal vectors of the sub-cube surface are calculated by interpolating the gradient vectors of the original cube. Finally, the surface can be rendered by projecting the point surface elements on the view plane.

Previous algorithms are so-called surface rendering algorithms. The main disadvantage of surface rendering is that only surface information is shown and other information will be lost. For medical images, it may be important to see through the volume rather than only examine the iso-surface. Another disadvantage of surface rendering is that a binary decision must be made on the position of surface. This kind of iso-surface extraction can lead to the introduction of false positive (consider

unnecessary features) or false negative (discard necessary features).

To overcome these problems, another kind of volume rendering techniques called *direct volume rendering* can be used. Unlike surface rendering, volume rendering goes through the whole volume and create a 2D projection of the volumetric data. It can also give user an inside view of the volume by using transparency since in some cases, making binary decisions on fuzzy surface is inappropriate. Several major volume rendering techniques are summarized as follows:

- **Ray casting.** One of the most popular volume rendering techniques is *ray casting* . It assumes that volume is made up of many small particles which can emit, attenuate or scatter light. The amount of light emitted, attenuated or scattered by a particle depends on the particle's density. Parallel rays are casted into the volume and samples are taken along the rays. The color of a particular voxel can be determined by its gradient vector, viewer's position and light positions. Gradient vector(G) of a voxel (x,y,z) in a volume data set V can be approximated by central differences:

$$G_x = \frac{1}{2}\Delta x(V(x_{i+1}, y_j, z_k) - V(x_{i-1}, y_j, z_k)) \quad (1)$$

$$G_y = \frac{1}{2}\Delta y(V(x_i, y_{j+1}, z_k) - V(x_i, y_{j-1}, z_k)) \quad (2)$$

$$G_z = \frac{1}{2}\Delta z(V(x_i, y_j, z_{k+1}) - V(x_i, y_j, z_{k-1})) \quad (3)$$

  The samples along each ray are composited together to form the final color of the corresponding image pixel (see Fig. 1). The compositions of samples are done by the **over** operation. Assume every voxel X is assigned a color $C(X)$ and an opacity $\alpha(X)$, the **over** operation can be defined as:

$$C_{out} = C_{in}(1 - \alpha(X)) + C(X)\alpha(X) \quad (4)$$

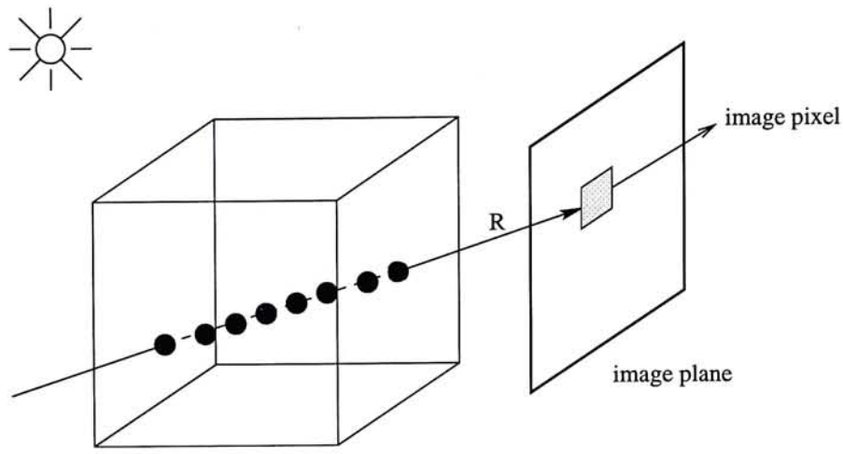


Figure 1: Composition of samples along the ray R

where C_{in} and C_{out} are the incoming and outgoing color for voxel X respectively. The overall intensity along the ray is given by:

$$C_{final} = \sum_{k=0}^{K} C(k)\alpha(k) \prod_{i=k+1}^{K} (1 - \alpha(i)) \tag{5}$$

where C_{final} is the resulting pixel intensity and K is the number of samples along the ray.

Besides volume rendering, ray casting can also be used to render iso-surface by assigning appropriate opacities to voxels and the quality can be comparative to some surface rendering algorithms like *marching cubes* . However, ray casting is very computational intensive. Although many acceleration techniques for it have been introduced, rendering by ray casting is still far from interactive.

- **Splatting.** Ray casting is an image space approach because the algorithm iterates over the image pixels. There is another kind of algorithms using an object space approach which iterates over the voxels. Splatting[48, 49] is one of such algorithms. For each voxel in the volume, it computes the contribution of

the voxel to the image pixels by using a filter footprint. The advantage of splatting is that it iterates over the voxels in storage order, unlike ray casting, which must compute the locations of sample points along different rays. Therefore, splatting uses much less addressing arithmetics than ray casting. However, the main disadvantage of splatting is that the computing of filter kernel is expensive and viewpoint dependent.

- **Frequency domain rendering.** The computational complexity of normal spatial domain projection algorithms are $O(n^3)$. By using the Fourier projection slice theorem, X-ray liked images can be made from volume with a complexity of $O(n^2)$. The Fourier projection slice theorem states that the inverse Fourier transform of a slice extracted from the frequency domain of a volume yields a projection of the volume in a direction perpendicular to the slice. A volume computed by a 3D discrete Fourier transform is firstly formed as a pre-processing step. X-ray images from arbitrary views can then be obtained by applying an inverse transform to a slice from the transformed volume which is passing through the origin.

- **Rendering by shear-warp factorization.**

  Both object space and image space approaches can produce high quality images but the rendering rate is far from interactive. In 1994, Lacroute [27] had proposed a new approach by using shear-warp factorization of the viewing transformation matrix. It is neither object space approach nor image space approach, but the performance is much better than the two approaches and can visualize a 128x128x84 data set in 1/5 second.

  Since the mapping from object space to image space uses a lot of computations to calculate the positions of samples, image space algorithms usually have slow performance. Lacroute pointed out that the addressing arithmetic can be

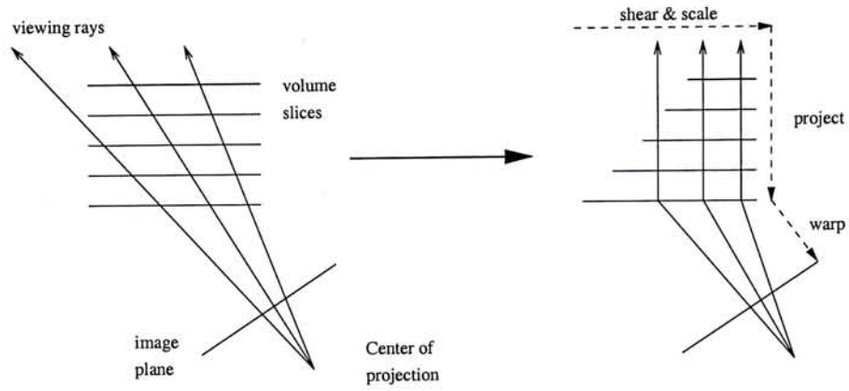


Figure 2: Shear and scale of slices

greatly reduced by transforming the volume into an intermediate coordinate system. It was found that rays can be made parallel to each other by shearing and scaling the volume slices. The viewing transformation can be decomposed into two matrix called shear matrix and warp matrix. The shearing and scaling of volume slices is done by the shear matrix, and the warping of the intermediate image to the final image is done by the warp matrix. From Figure 2, we can see that the projection to a 2D intermediate image would be more efficient by using this transformation. The shear-warp algorithm uses a run-length encoded volume for rendering and the intermediate image is also a run-length encoded data structure, which encodes runs of opaque and non-opaque pixels.

- **Rendering by 3D texture mapping.** With the aid of hardware-assisted 3D texture mapping, performance can be further improved. One can sample any slice inside a block of three-dimensional texture memory(see Fig. 4) by trilinear interpolation. Trilinear interpolation is the process for point sampling within a 3D box/cell given values at the vertices of the box. For the cell shown in Figure 3, if $V_{000}, V_{100}, V_{010}, ..., V_{111}$ are the values of the vertices, then the value of a point V_{xyz} inside the box is given by

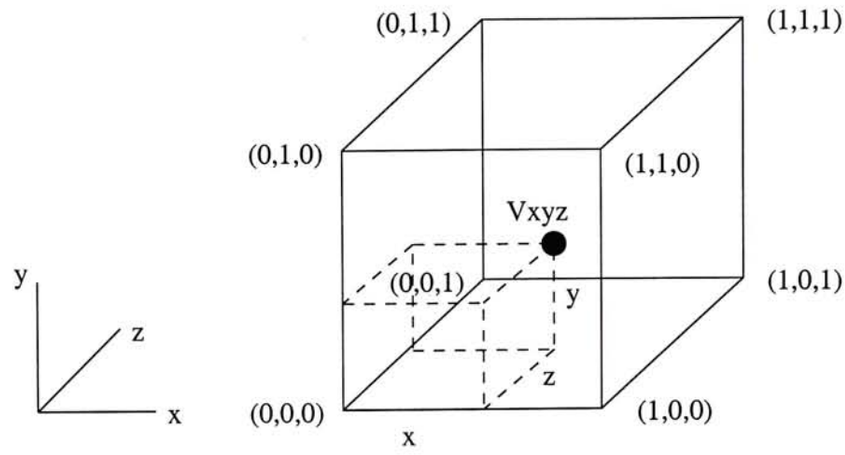


Figure 3: Trilinear interpolation

$$
\begin{aligned}
V_{xyz} \;=\; & V_{000}(1-x)(1-y)(1-z) \\
& +V_{100}x(1-y)(1-z) \\
& +V_{010}(1-x)y(1-z) \\
& +V_{001}(1-x)(1-y)z \\
& +V_{101}x(1-y)z \\
& +V_{011}(1-x)yz \\
& +V_{110}xy(1-z) \\
& +V_{111}xyz
\end{aligned} \tag{6}
$$

Cullip and Neumann[42] described how to perform volume rendering by using 3D texture mapping. The paper describes both the object space and image space approaches to do the task. As shown in Figure 5, object space approach uses slices perpendicular to one of the axis for sampling, and image space approach uses slices perpendicular to the viewing direction.

A more detailed description of the image space approach had been presented by

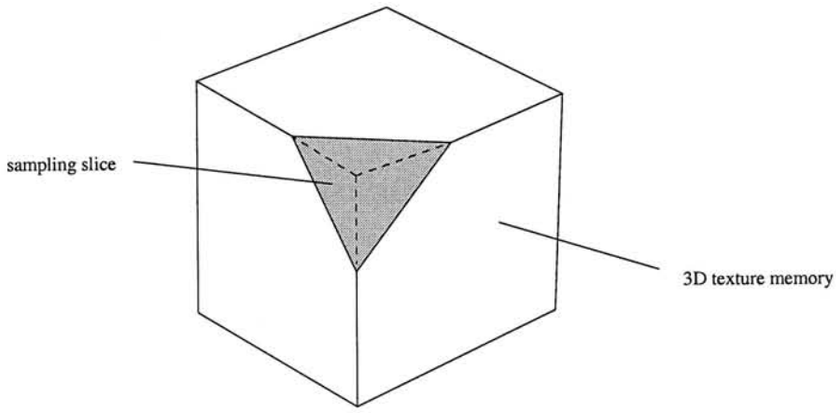


Figure 4: 3D texture mapping

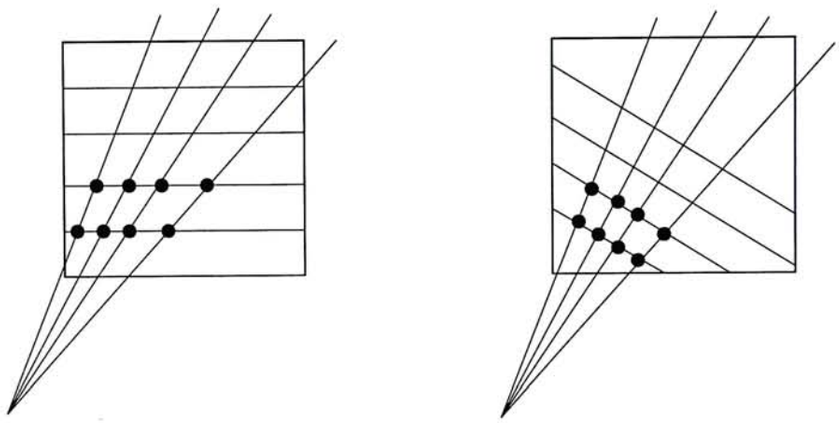


Figure 5: Sampling in Object and Image space

Wilson, Gelder and Wilhelms[50]. An octree encoding scheme has been applied along with 3D texture in [15]. That paper also proposes a template based Z-plane/block intersection method to accelerate the block projection computation.

Volume rendering by 3D texture mapping is fairly fast. It can achieve about 20 frames per second for the rendering of a 128^3 volume data. In this research, this technique would be used since it has remarkably faster speed than previous algorithms.

1.2 Virtual Environment

A suitable virtual working environment is important for volume visualization. It is easy to find that some volumetric operations are difficult or even impossible to perform using a 2D screen and a mouse. A virtual environment with three-dimensional display and three-dimensional interaction can help user to understand and interact with the volume data much easier by providing a rich set of spatial and depth cues. Existence of various VR applications for scientific visualization [2, 3, 5, 9, 20, 21, 43] in the past few years proves that it is more natural to manipulate high-dimensional data in a virtual environment.

In this research, the Virtual Workbench proposed in [43] is chosen to be the working platform. Virtual Workbench is a general-purposed working environment for dextrous work in 3D and some applications such as virtual windtunnel[4], blood vessel finding[44] and virtual stereotaxis[45] have also been implemented on it. Two main advantages of Virtual Workbench are:

- **High resolution three-dimensional display.** Virtual Workbench uses a mirror to reflect images from a normal monitor such that it constructs a "virtual work space" behind the mirror(see Fig. 6). Stereoscopic display can be achieved by using a pair of stereo glasses. The image quality is much better than those from Head-Mounted Display(HMD) since images are displayed on high resolution monitor.

- **Three-dimensional interaction.** In Virtual Workbench, object interaction is performed using a 6DOF pen-liked input device. With the mirror, user perceives a stereo virtual image within the work space. Therefore, the 3D pen can move freely within the work space without blocking the views of virtual images.

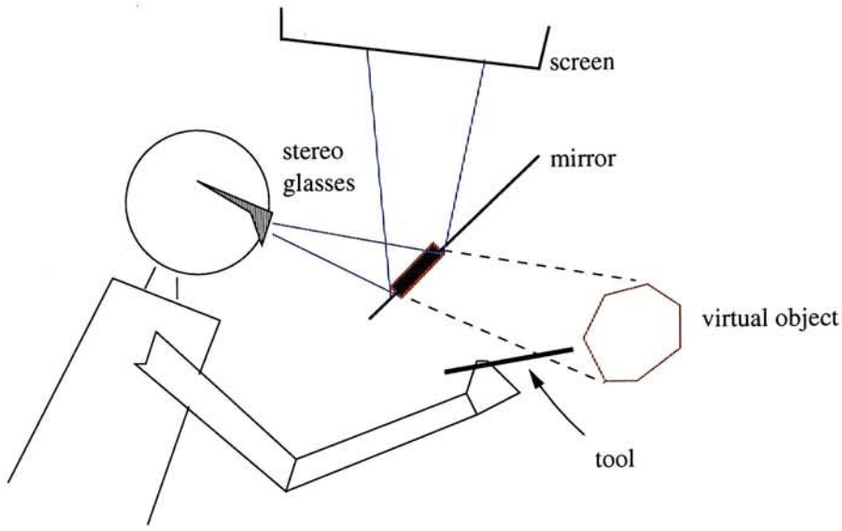


Figure 6: The Virtual Workbench

1.3 Approach

The main objective of this research is to find out effective volume manipulation methodologies that can help exploring volumetric data in a virtual environment interactively. Direct volume rendering is too computational intensive even using nowadays computers. Therefore, most applications considering direct volume rendering are still far from real-time. To improve the performance, unnecessary data from volume can be cut away and intention can be focused on volume of interest. Volume segmentation plays an important role to achieve this.

In this research, I borrow the idea from *Intelligent Scissors*[30, 41], which is a 2D image segmentation technique, and exploit its usefulness in volumetric data manipulation. It is essentially a dynamic programming technique for graph searching. Two approaches concerning Intelligent Scissors are taken in this report:

- **Contour extraction on surface mesh** Originally, Intelligent Scissors is applied to 2D image. We modify the idea a little bit by applying Intelligent Scissors to surface mesh generated from volume data. One can interactively select feature lines along the surface by this technique. After several successive

selections of segments, a closed contour which lies on the surface mesh can be formed. Methods are developed to cut volume data into two parts by using the closed contour.

- **Three-dimensional Intelligent Scissors** We extend the idea of 2D Intelligent Scissors to 3D. In the case of 2D, contours are lying along edges. By using 3D Intelligent Scissors, extracted lines now lie on surfaces. Iso-surface extraction becomes possible by using those lines. One problem of the extension is the huge amount of graph nodes needed to search. Therefore, methods have to be developed to cut away unnecessary voxels. One approach is to discard voxels with gradient less than a pre-defined amount. More about voxel pruning will be discussed later.

1.4 Thesis Overview

Chapter 2 gives a brief summary of *Intelligent Scissors* , which forms the basis of our new algorithms. We describe the original algorithm which detects contours in a 2D image. Its advantages against other similar algorithms will also be discussed.

Next, in Chapter 3 we describe the technique for volume cutting. We modify the algorithm of Intelligent Scissors to find out closed contours from complex volume surface.

Chapter 4 extends the concept of Intelligent Scissors from 2D to 3D. The extension introduces a huge amount of graph nodes and we show that how to accelerate the process by discarding unnecessary nodes.

In Chapter 5, we show the implementation of the algorithms on the Virtual Workbench environment. An intuitive interface is built in order to give user a more convenient 3D working environment. Some results of our algorithms are presented and analyzed here.

Finally, Chapter 6 summarizes the conclusions of this report. Implications for future research are also discussed.

Chapter 2

Contour Extraction

Contour extraction is an important step in volume segmentation, which in turn is very critical in some volume rendering applications because direct volume rendering is too computational intensive and segmentation can make visualization of the volume of interest possible in order to save time. Three-dimensional segmentation, however, is another complex problem and there is still no completely automatic algorithms which can generate satisfactory segmentation. Therefore, people tend to reduce the problem to two-dimensional. 2D closed contours can be extracted from parallel voxel slices first and then joined together to form a geometrical model(see Fig. 7) by some surface fitting algorithms[19, 31].

One main difficulty is how to find the best approximated contour. A popular technique called active contours or snakes[26, 8] has been widely used. However, as stated in Section 1.1, the main drawback is its inability to control the final shape of contour. Another kind of techniques use a graph searching formulation of dynamic programming(DP) to find globally optimal boundaries. In[30], *Mortensen and Barrett* proposed a method called *Intelligent Scissors* that can extract contours from 2D images interactively.

Throughout this chapter, we concentrate on describing the details of Intelligent Scissors, which forms the basis of our later algorithms. Section 2.1 introduces the concept of Intelligent Scissors. The main algorithm is described in Section 2.2. In Section 2.3, we discuss the cost function used in the technique.

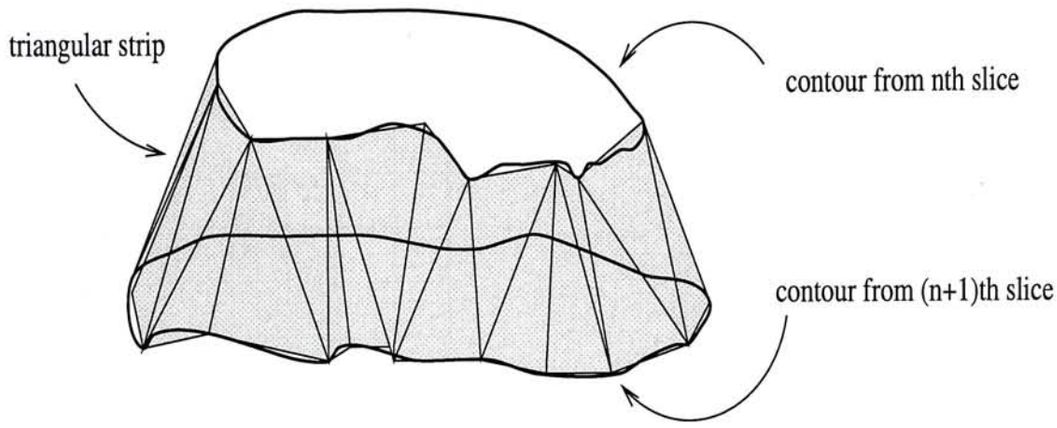
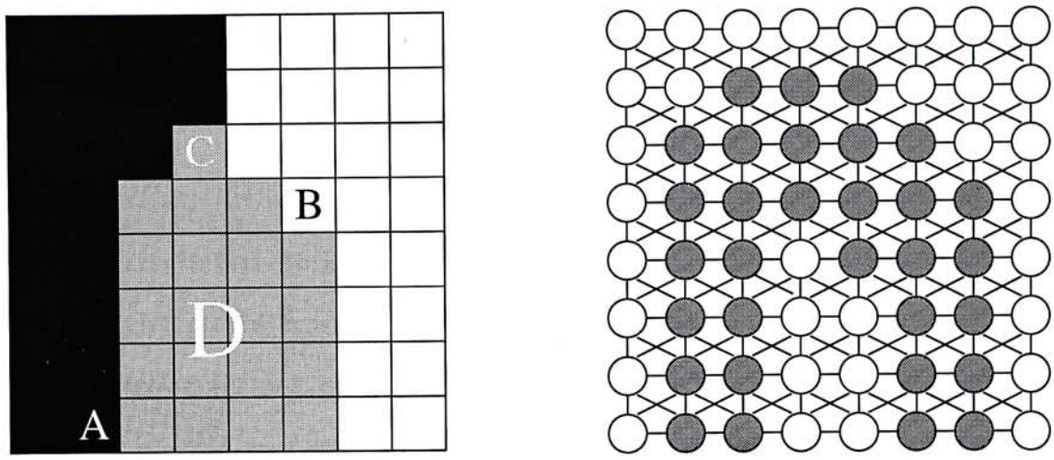


Figure 7: Surface fitting between parallel contours

2.1 Concept of Intelligent Scissors

Although Intelligent Scissors has an interesting name, it is basically a dynamic programming technique. There were some algorithms that also used a graph searching formulation of DP to find globally optimal boundaries[1, 12, 46, 52]. However, like snakes, those algorithms typically need an initial boundary template which is used to approximate the final shape of the desired contour. Since the template grows and shrinks in order to find an optimal contour, it makes each point on the contour moving with one degree of freedom within the 2D image. Therefore, those algorithms cannot be interactive and user cannot modify the result in the midway of the process.

Intelligent Scissors, on the other hand, allows user interactively select the most suitable boundary from a set of optimal boundaries given a seed point. The idea is simple. A 2D image is modeled as a graph with nodes at the pixel locations and edges are defined for neighbor pixels. Each edge is assigned a cost according to some gradient and frequency functions. Therefore, the problem of finding a contour between two pixels can now be solved using a shortest path algorithm, which finds a minimum cost path. *Stalling and Hege*[41] modified the algorithm in order to apply it to medical image segmentation. In their application the cost function is related to the image gradient only and the result is acceptable.



(a) (b)

Figure 8: (a) The original image (b) The graph form of (a). Grey nodes have greater gradient than white nodes.

Edge cost is assigned in a way that pixels lying on boundary features have lower cost. Figure 8(a) shows a 8x8 grey-scale image and Figure 8(b) is the corresponding graph formulation. Grey nodes in the graph represent the boundary points which have greater gradient than other nodes. So in this case, the edges between grey nodes have the lowest cost on average. The edges between grey nodes and white nodes have higher cost and those between white nodes have the highest cost. For example, if we want to find a path from pixel A to pixel B, the algorithm will automatically find a path going along the boundary and passing around pixel C. The reason is that the total cost of the path is the sum of the edges it passes. Therefore, it is unlikely that the resulting path would pass the region D, because the edges in region D tend to have greater cost on average. Intelligent Scissors finds the optimal path by Dijkstra's algorithm, which is discussed in the next section.

2.2 Dijkstra's Algorithm

To find a minimum cost path between two pixels, the shortest path algorithm called Dijkstra's algorithm[11] is used. In this algorithm, the total path cost of every pixel from a source are found by dynamic programming. This algorithm effectively computes the shortest paths from all nodes of the graph to a source node. The search need not be finished by one pass. Instead, it can be stopped when the search reaches the target position. It can save much time since unnecessary paths need not be calculated. If the seed point is changed, the searching must be started again. Time can be saved here since not all of the points need to be re-calculated. For example, u is the seed point and v is the new seed point. All points go to u passing v need not be re-calculated because if $p \rightarrow v \rightarrow u$ is the shortest path from a point p to u, then $p \rightarrow v$ must be the shortest path from p to v.

The idea of Dijkstra's algorithm is to utilize dynamic programming to update the cost of each point step by step. For each node u, there is a pointer $B(u)$ which points to one of its neighbor nodes such that a path from the clicked point to the seed point can be established quickly. Note that the cost function $c(u, v)$ can be preprocessed and the only changed function is $B(u)$, which indicates the path from u to the seed point s. The pseudo-code of Dijkstra's algorithm is as follows:

Dijktra'sAlgorithm

Definitions:

s	seed point
L	list of active nodes
$B(u)$	back pointers indicating the path
$P(u)$	TRUE if node u is made permanent
$T(u)$	total cost from u to s
$c(u, v)$	local cost of edge $u \rightarrow v$

$min(L)$ pop the node with minimum cost from L

Algorithm:

$P(u) \leftarrow FALSE$ for all u

$T(s) \leftarrow 0, T(u) \leftarrow \infty$ for $u \neq s$

$L \leftarrow$ {all nodes}

while $L \neq \emptyset$ do

$q \leftarrow min(L)$

$P(q) \leftarrow TRUE$

for each edge $q \rightarrow v$ such that $P(v) = FALSE$ do

if $T(v) > T(q) + c(q, v)$ then

$T(v) \leftarrow T(q) + c(q, v)$

$B(v) \leftarrow q$

end if

end for

end while

Figure 9 demonstrates how to find the cost map of a 4x4 image step by step. In order to simplify the demonstration, each node has only four neighbors instead of eight. The extension to eight neighbors can be done trivially by adding four more edges to each node. Figure 9(a) is the initial cost map and S is the seed point. There are three types of nodes. The first type is the nodes that are not yet visited. The second type is the reachable but not permanent nodes. The final one is the permanent nodes which have fixed total path cost. At the very beginning of the algorithm, the node S is marked as reached(but not permanent) and has total cost zero. Non-permanent nodes are put in a priority queue and in each stage the node with minimum total cost will be selected and marked as permanent. Figure 9(b)

shows that the only element in the priority queue, S, is marked as permanent and the cost of its neighbors is updated. Figure 9(b) and (c) shows the cost map with two and three permanent nodes respectively. The final cost map with all nodes marked as permanent is shown in Figure 9(e). Note that after each update of a node, the node's back pointer is modified and points to the node that makes the change. Therefore, eventually every node will have a back pointer that indicates the optimal path from that node to the seed point.

The cost function is directly related to the performance of the algorithm. A bad cost assignment can either make the resulting contour far away from the boundaries or make it too sensitive to noise. Therefore, we describe the cost function in the next section.

2.3 Cost Function

Since a minimum cost path has to be found and we want to extract contours near strong edge features, pixels near boundaries should have low costs and vice-versa. In [30], the cost function is composed of three functions: Laplacian zero-crossing(f_Z), gradient magnitude(f_G) and gradient direction(f_D). The cost of an edge between node u and node v is as follows:

$$cost(u,v) = w_Z \cdot f_Z(v) + w_D \cdot f_D(u,v) + w_G \cdot f_G(v) \tag{7}$$

where w_Z, w_D and w_G are the weights for the functions.

In the equation, f_Z is the Laplacian zero-crossing function which has output of 0 or 1, where 0 means a strong edge. The gradient of a node can be found by central difference. If I_x and I_y are the partials of a pixel in x and y directions respectively, the gradient of the pixel can be represented by a vector (I_x, I_y) and the gradient

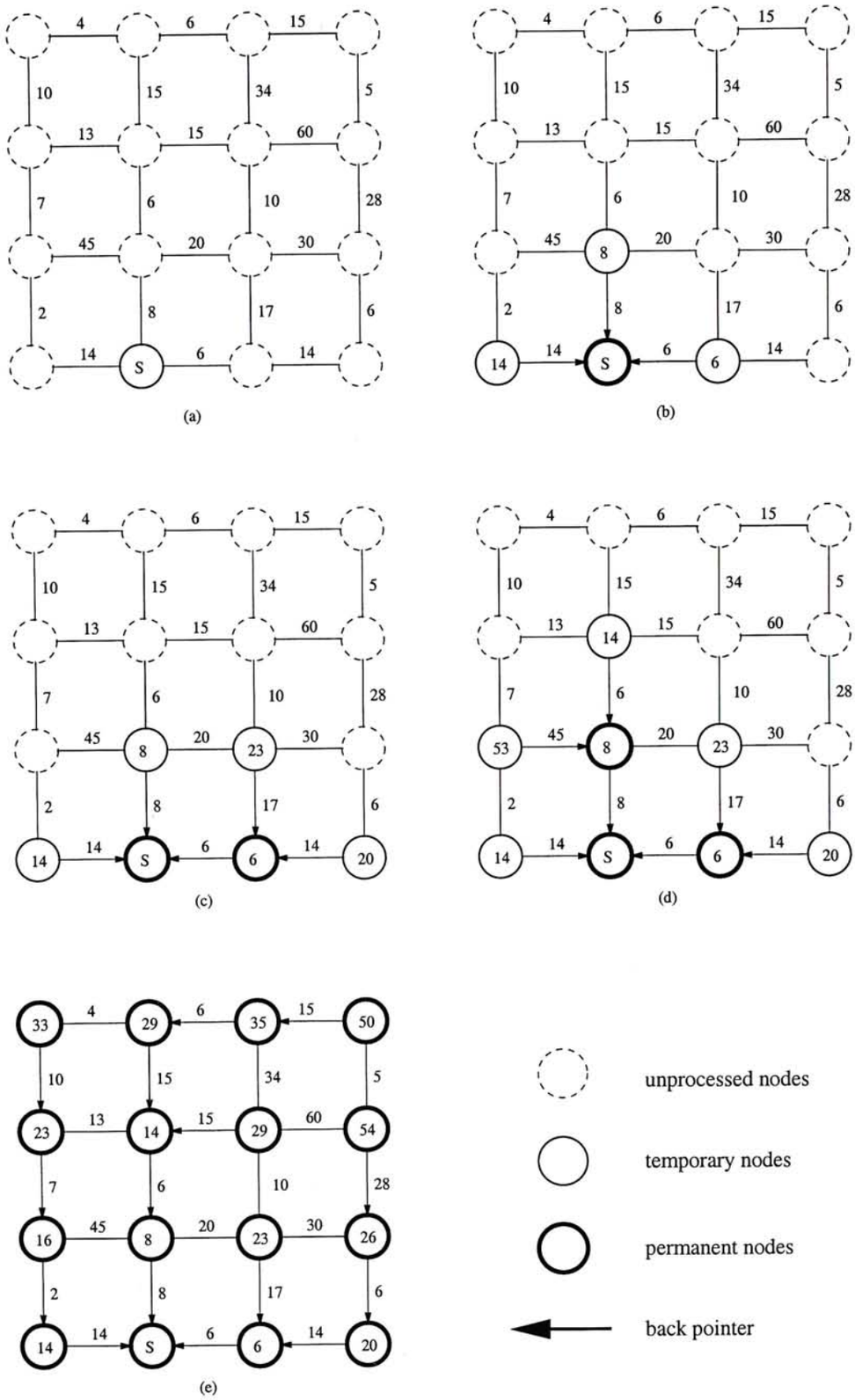


Figure 9: Cost map of a 4x4 image

magnitude G is given by

$$G = \sqrt{I_x^2 + I_y^2} \tag{8}$$

Since pixels with high gradients should have edges with low costs, the gradient magnitude function f_G can be approximated with

$$f_G = \frac{max(G) - G}{max(G)} = 1 - \frac{G}{max(G)} \tag{9}$$

The last function f_D controls the smoothness of the path. Therefore, the edge $p \rightarrow q$ in Figure 10(a) should have low edge cost and the one in Figure 10(b) should have higher cost because the line in (a) is smoother than the curve in (b). Thus f_D can be decided by the change of angles of the gradient vectors and the vector $(q-p)$. Let $D(p)$ be the vector perpendicular to the gradient vector of p (i.e. $D(p) = (I_y(p), -I_x(p))$. The smaller the angle between $D(p)$ and $(q-p)$ the smoother the curve is and the same for the case of $D(q)$ and $(q-p)$. The equation of f_D is given by:

$$f_D(p,q) = \frac{cos[d_p(p,q)]^{-1} + cos[d_q(p,q)]^{-1}}{\pi} \tag{10}$$

where

$$d_p(p,q) = D(p) \cdot L(p,q)$$
$$d_q(p,q) = D(q) \cdot L(p,q)$$

and

$$L(p,q) = \begin{cases} q-p & \text{if } D(p) \cdot (q-p) \geq 0 \\ p-q & \text{if } D(p) \cdot (q-p) < 0 \end{cases} \tag{11}$$

The value of f_D ranges from 0 to 1. It can smooth the shape of the contour by assigning high costs to sharp changed edges. However, in [41], *Stalling* suggested a

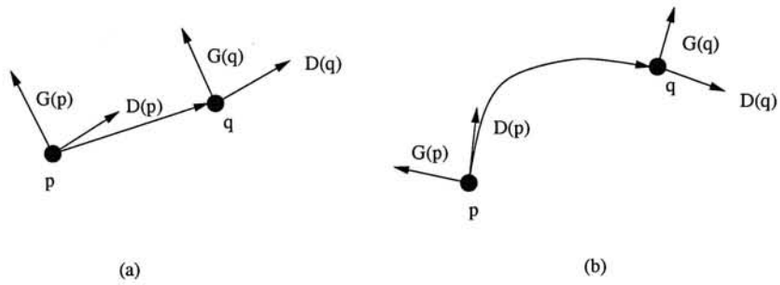


Figure 10: Illustration of the gradient direction function f_D

much simpler equation which uses only the image gradient magnitude. Good results are generated when it is applied to medical images. The simplified equation is

$$cost(u, v) = 255 - \frac{1}{2}(G(u) + G(v)) \tag{12}$$

2.4 Summary

In this chapter we have introduced a robust algorithm called Intelligent Scissors which can extract contours interactively from a 2D image. The algorithm provides a better segmentation tool than previous contour extraction techniques such as active contour by using dynamic programming.

The main advantage of Intelligent Scissors is that it allows interactive manual corrections of the contour. The process takes a short time for experienced users and the interface is easy to control. However, the drawback of this algorithm is that it is hard to predict the number of control points which define the closed contour and human interventions are inevitable.

The main algorithm used is the Dijkstra's shortest path algorithm. With appropriate definitions of edge costs, it is expected that extracted contours can go along with the real image boundaries. We borrow the idea of this algorithm and begin developing our new algorithms described in the next few chapters.

Chapter 3

Volume Cutting

The main goal of volume visualization is to give the user a clear understanding of a volumetric data set. Various approaches have been proposed in the past. For example, by setting voxels other than bone a opacity of zero, one can visualize the skull in a head data set by direct volume rendering. If we do not want to use direct volume rendering to examine the skull, we can use marching cubes to extract the iso-surface. Moreover, we can use some segmentation techniques to extract just the brain of human in a volume data. One can also use some simple tools like cutting planes to examine the interior structure of a volume. All these algorithms have their own pros and cons. One may prefer a particular algorithm in a particular situation. For example, experienced doctor may find that examining a X-ray slice is sufficient for clinical diagnosis. Although these algorithms have different features, the ultimate goal is the same: to examine the data in the greatest extent. Therefore, we can say that it is important to develop methods that can help the user to examine a volume data.

In this chapter we propose a new volume cutting algorithm which utilizes the technique of Intelligent Scissors. Instead of applying Intelligent Scissors to a 2D image, we now apply it to the surface mesh formed from a volume data. The main idea of this algorithm is described in Section 3.1. In Section 3.2, the application of Intelligent Scissors to surface mesh is presented. Finally, the internal cutting surface used to cut away voxels is described in Section 3.3.

3.1 Basic idea of the algorithm

Given a volume composed of two or more objects, how can we effectively cut it into two parts such that we can examine the interior structure more clearly? Segmentation is probably the most suitable approach for the task. However, current segmentation techniques still have many drawbacks, both in the aspects of quality and interactivity. Noisy data or fuzzy object boundaries make the segmentation more difficult. Moreover, some segmentation techniques need prior knowledge of the objects to be segmented, which makes the algorithms less adaptive to various data sets.

Based on the above observations, we conclude that a tool which can provide intuitive interface for volume separation is necessary. In this section, we concentrate on describing a new methodology which can help cutting the volume into meaningful parts. We are not going to give an accurate segmentation tool. What we concern here is a new method that can be used for interior structure examination of a volume data.

The basic idea of this algorithm can be illustrated by Figure 11. First a surface mesh is generated from the volume data by some mesh generation algorithms such as Geometrically Deformed Models(GDM)[29]. Second, user selects a closed contour which lies on the surface mesh. This contour extraction process is done by Intelligent Scissors described in Chapter 2. We modified the algorithm such that it works on surface mesh. Detail of the extension is described in the next section. After we get the contour, we build a cutting surface based on this contour using an algorithm described in Section 3.3. Now we get a surface mesh and a internal cutting surface. Voxelize these two surfaces and finally we can separate the volume by a simple region growing algorithm.

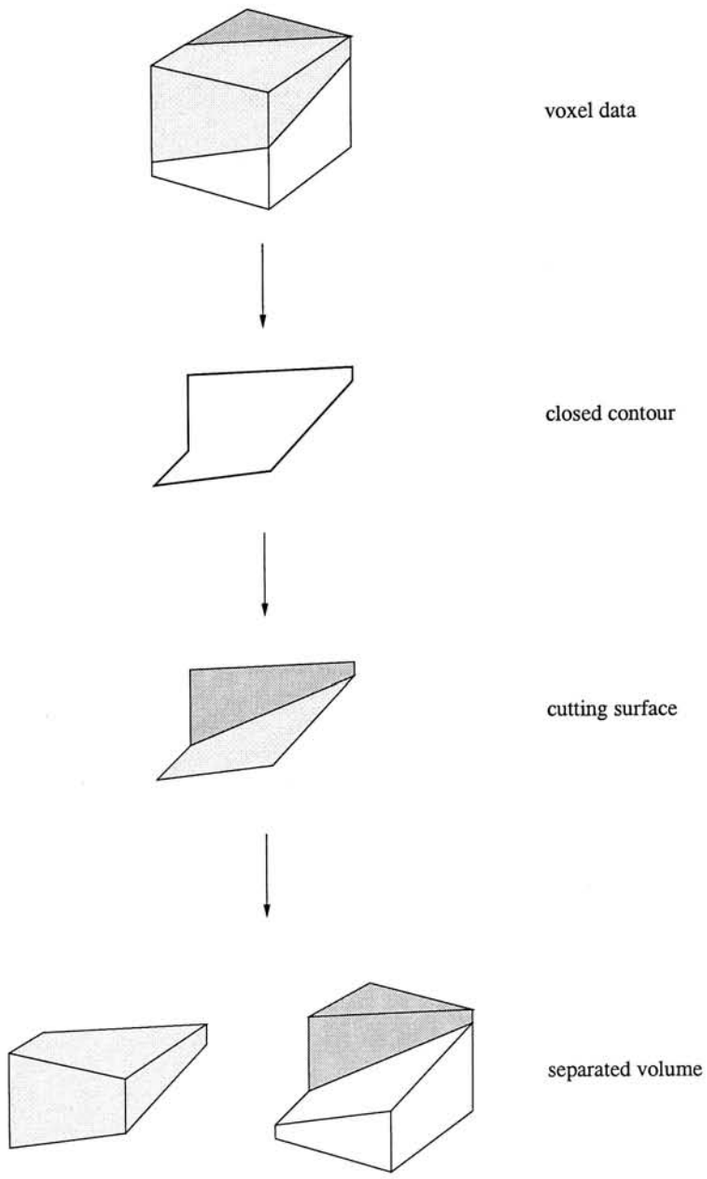


Figure 11: The basic idea of volume cutting

3.2 Intelligent Scissors on Surface Mesh

Intelligent Scissors is originally a tool for image segmentation. For a 2D image, the set of pixels is viewed as a graph with each pixel connected to its eight neighbors. From [30, 41], Intelligent Scissors has been proved to be an efficient method on 2D segmentation. Now we would like to modify the algorithm such that it can be applied to volume data.

To apply the algorithm to volumetric data, we must have a surface mesh of the data first. It is like that we should have a pixel graph first for a 2D image. There are some meshing algorithms that can build up the surface mesh from the volumetric data. For example, Miller et al.[29] proposed the *Geometrically Deformed Models*(GDM), which can get closed geometric models from volume data.

After we get the mesh, we should design a cost equation that assigns a cost to each mesh vertex. In the original algorithm, the cost equation is related to three functions, namely Laplacian zero-crossing, gradient magnitude and gradient direction. To find a contour on the surface mesh, we would like to modify the original cost equation such that the local cost between vertices u and v depends on their gradient magnitudes and gradient vectors. It is because geometrical information is more important than image information in a mesh. We also discard the Laplacian zero-crossing function since it is meaningless to apply a boundary detection function to a surface. The gradient vector of a mesh node can be obtained by trilinear interpolating the gradient vectors of the vertices of the cell containing the node(see Fig. 12). The resulting cost function is

$$cost(u,v) = \|p(u) - p(v)\| \left(w_g f_g(u,v) + w_n f_n(u,v)\right) \tag{13}$$

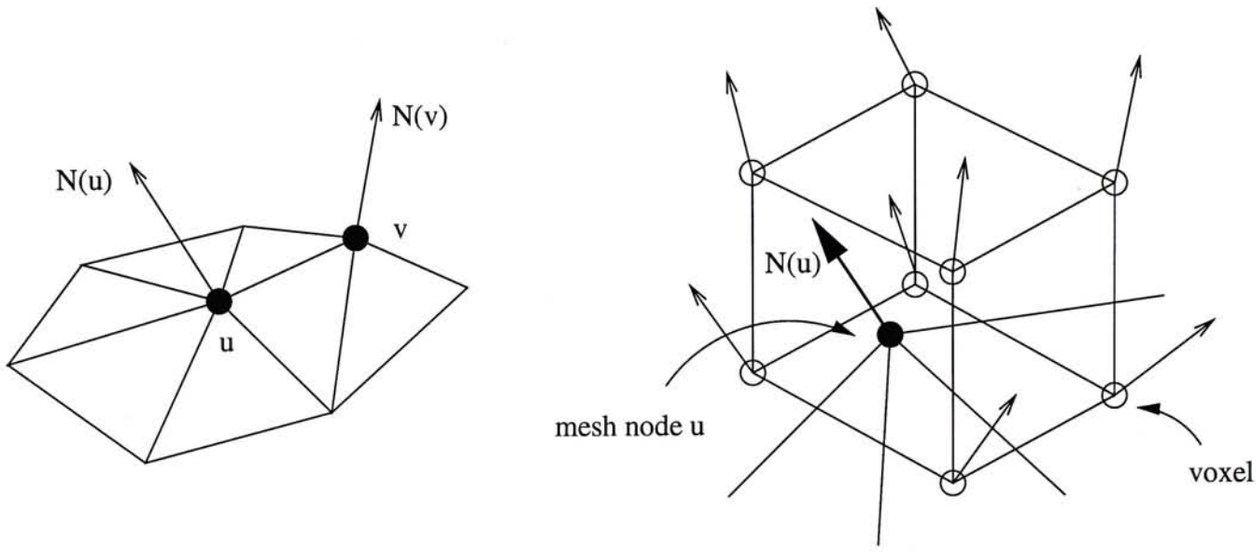


Figure 12: The gradient vector of a mesh node

where f_g and f_n are the gradient magnitude and vector functions

$$f_g(u,v) = 1 - \frac{G(u) + G(v)}{2max(G)}$$

$$f_n(u,v) = \frac{1 - N(u) \cdot N(v)}{2}$$

$p(u), p(v)$ are the position vectors at u and v respectively, and w_g, w_n are the weighting factors controlling the influence of f_g and f_n. The gradient magnitude of the gradient vector $N(u)$ is represented by $G(u)$. By using this formula, the Dijkstra algorithm tends to find a path which has large gradient changes and small normal changes. Note that the multiplication of $\|p(u) - p(v)\|$ is important because it guarantees that the shorter one of paths with the same cost would be selected.

A contour on the surface can be found by applying the technique described in Chapter 2. The main differences are the cost function and the number of edges attached to a node. In this algorithm, the degree of each node is not constant, unlike the case for 2D image. The rest is the same as the original algorithm.

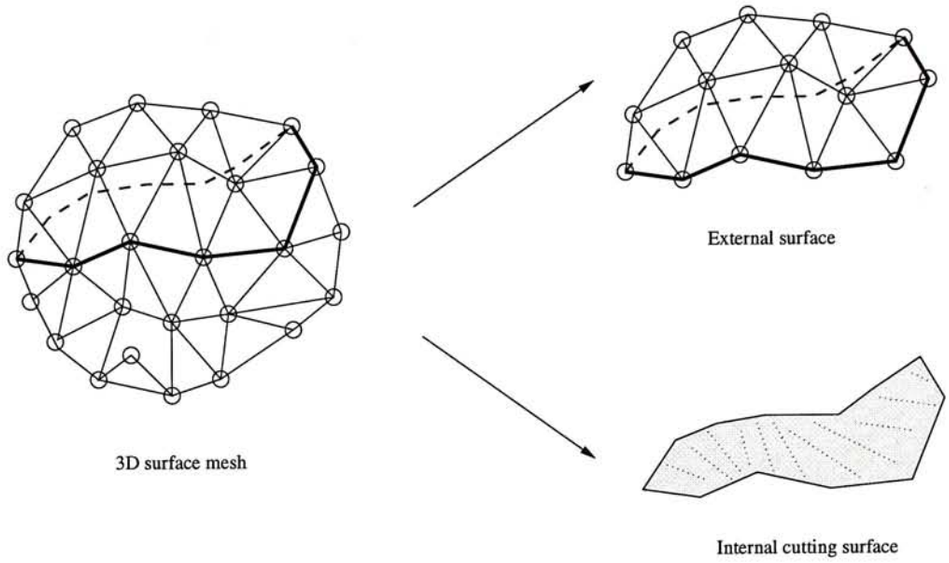


Figure 13: Generation of external and internal surfaces

3.3 Internal Cutting Surface

A closed contour is not enough for volume extraction. To separate the volume into two parts, we must have two surfaces. As shown in Figure 13, the first one is the external surface mesh. By cutting the mesh by the extracted contour, we can get the external surface by voxelizing the half part of the mesh. The second one is the internal cutting surface. The external surface can be obtained much easier than the internal one since there exists the surface mesh. Therefore, we must develop method that can extract the internal surface efficiently. Moreover, the internal surface should separate the volume in an intelligent way such that the surface is close to the object boundaries.

To build the surface, we first project the contour onto a plane, as shown in Figure 14. The plane is oriented as parallel to the contour as possible such that the projected area of the contour is the largest. The construction is done by shrinking the contour iteratively with minimization of the total surface cost.

The project plane is associated with three arrays, namely $c(x_p, y_p)$, $L(x_p, y_p)$ and

$V(x_p, y_p)$, where x_p and y_p are the projected coordinates. $c(x_p, y_p)$ is the cost of voxel projected on (x_p, y_p). $L(x_p, y_p)$ is the distance from the contour. $V(x_p, y_p)$ will store the voxel coordinates (x,y,z) that defines the shape of final cutting surface.

$L(x_p, y_p)$ is assigned at once after the projection of the contour. As shown in Figure 15, the distance map can be made by iteratively shrinking the projected contour. The number shown in each pixel is the distance to the contour in hops and the pixels outside the contour are assigned to infinity. We define $\mathcal{B}_k$ be the set of voxels which have a distance of k from the projected positions to the contour.

$$\mathcal{B}_k = \{(x, y, z) | L(x_p, y_p) = k\} \tag{14}$$

The cost of each voxel depends on two factors. The first one is the gradient and the second one is its continuity. The first factor makes it as near as the object boundaries. The second factor controls the shape of the surface such that it can be smoother. The cost function $C(x, y, z)$ of a voxel (x, y, z) is given by:

$$C(x, y, z) = -w_g G(x, y, z) + w_c S(x, y, z) \tag{15}$$

where $G(x, y, z)$ is the gradient magnitude and $S(x, y, z)$ is the surface continuity function. w_g and w_c are the positive weighting factors controlling the gradient function and the continuity function respectively. Therefore, the total cost of the cutting surface is the sum of voxels contained in the surface.

The surface continuity function measures the average distance of a voxel from other neighbor voxels. Surely not all voxels from the volume data have to be included for the calculation of the cost of a voxel. We define $\mathcal{A}_{(x,y,z)}$ be the set of involved voxels for the continuity calculation of a particular voxel and it is given by:

$$\mathcal{A}_{(x,y,z)} = \mathcal{N}_{(x,y,z)} \cap \mathcal{B}_{(L(x_p,y_p)-1)} \cap \mathcal{G} \tag{16}$$

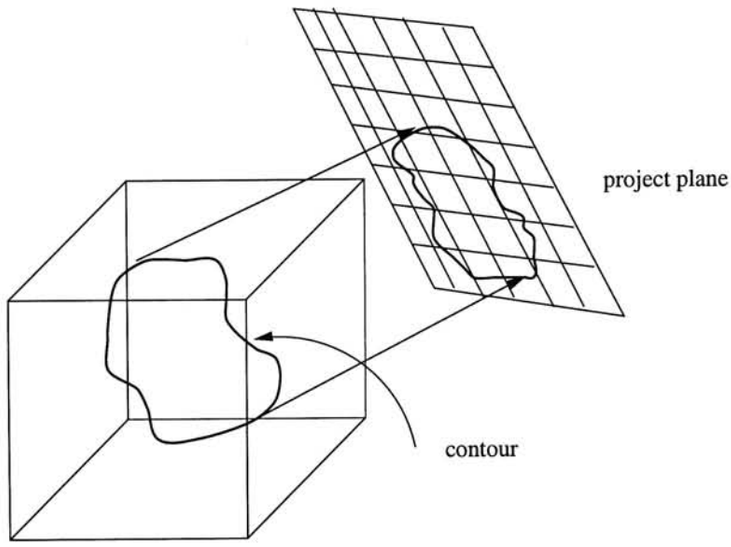


Figure 14: Projection of the contour on a plane

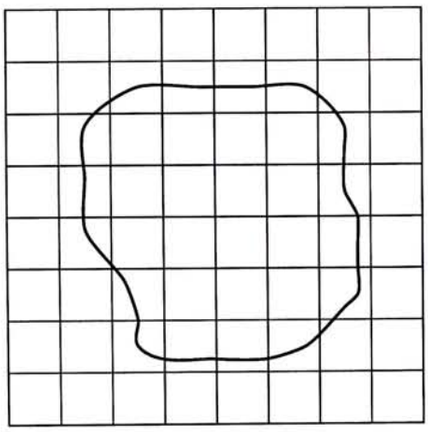
where

$$\mathcal{N}_{(x,y,z)} = \{(x',y',z') | \left\| (x'_p, y'_p) - (x_p, y_p) \right\| \leq \varepsilon\} \tag{17}$$

(x_p, y_p) and (x'_p, y'_p) are the projected point of (x, y, z) and (x', y', z') respectively. $\mathcal{G}$ is the set of voxels which have been guaranteed to be involved in the cutting surface. ε is a constant which is used to control how close two neighbor voxels should be on the project plane. Therefore, the surface continuity function is defined as follows:

$$S(x,y,z) = \frac{\sqrt{\sum_{\vec{v} \in \mathcal{A}_{(x,y,z)}} \|\vec{v} - (x,y,z)\|^2}}{|\mathcal{A}_{(x,y,z)}|} \tag{18}$$

As stated before, the surface is found by minimizing the total surface cost. It can be done by an iterative process which shrinks the projected contour gradually. In fact, the projected area is composed of many contours where the pixels of each contour have constant $L(x_p, y_p)$. Therefore, we can process the contours one by one to find out the surface. For example, Figure 15(b) is the projection of the contour in Figure 15(a). We then move on the smaller contour marked '1' in Figure 15(c). After all voxels have been found out by minimizing the contour cost, we move to the



(a)

		0	0	0	0		
	0					0	
	0					0	
	0					0	
		0				0	
		0	0	0	0		

(b)

		0	0	0	0		
	0	1	1	1	1	0	
	0	1			1	0	
	0	1			1	0	
		0	1	1	1	0	
		0	0	0	0		

(c)

		0	0	0	0		
	0	1	1	1	1	0	
	0	1	2	2	1	0	
	0	1	2	2	1	0	
		0	1	1	1	0	
		0	0	0	0		

(d)

Figure 15: Distance map $L(x_p, y_p)$ of the project plane

contour marked '2', and so on, until all contours have been processed. The cutting surface can be found using the following algorithm:

Initialization:

$\mathcal{G} \leftarrow \mathcal{C} \cap \mathcal{B}_0$

$k \leftarrow 1$

$\mathcal{D} \leftarrow \mathcal{B}_1$

$\forall i, j, V(i,j) \leftarrow null$

$\forall i, j, c(i,j) \leftarrow \infty$

Algorithm:

for each $(x, y, z) \in \mathcal{C}$ do

$c(x_p, y_p) \leftarrow 0$

$V(x_p, y_p) \leftarrow (x, y, z)$

// where (x_p, y_p) is the projected coordinates of (x, y, z)

end for

while $\mathcal{D} \neq \emptyset$ do

for each $(x, y, z) \in \mathcal{D}$ do

$tmp \leftarrow C(x,y,z) + \frac{\sum_{(x',y',z')} C(x',y',z')}{|\mathcal{A}_{(x,y,z)}|}$

if $tmp < c(x_p, y_p)$ then

$c(x_p, y_p) \leftarrow tmp$

$V(x_p, y_p) \leftarrow (x, y, z)$

end if

end for

$\mathcal{G} \leftarrow \bigcup_{i,j} V(i,j) - \{null\}$

$k \leftarrow k + 1$

$\mathcal{D} \leftarrow \mathcal{B}_k$

end while

After extracting the cutting surface, the final step is simple. To separate the volume, we can have two alternatives. The first one is region growing. After the voxelization of the external and internal surfaces, we can select a voxel in the volume and perform a simple region growth to get the region of interest. The second method uses the algorithm introduced in [22], which can produce volume data from triangular mesh. We can add up the external and internal meshes together to form a closed mesh. Triangulating the mesh and the voxelization algorithm can be used straightly.

3.4 Summary

In this chapter, a new volume cutting method is proposed. The aim of this algorithm is to give user a convenient interface that can separate a volume into two parts by considering the object boundaries. A closed contour is extracted using an extension of Intelligent Scissors, which is an efficient 2D image segmentation tool. We modified it such that it works on surface mesh. A cutting surface extraction algorithm is also proposed. By using these two surfaces, volume can be separated into two parts easily. The detail of this work can be found in [6].

Chapter 4

Three-dimensional Intelligent Scissors

The Intelligent Scissors stated in Chapter 2 works quite well in 2D applications. It motivates us to extend the algorithm to three-dimensional, which is applied to volumetric data instead of 2D images. Unlike the volume cutting algorithm proposed in Chapter 3, it works not only on surface mesh from volume data, but works on the *whole* volume data with all voxels as graph nodes.

The characteristic of 2D Intelligent Scissors is to find out a curve lying on edge features given the starting and ending points. In volumetric data, however, our new algorithm now extracts lines lying on *surfaces*, instead of *boundaries* in the case of 2D.

The extension from 2D to 3D is straight forward. Like the pixel graph in a 2D image, each voxel of a volume is now a graph node and every node has 6, 18 or 26 edges attached to it, depending on whether the graph is decided to be 6-connected, 18-connected or 26-connected. Figure 16 shows the voxels involved for 6, 18 and 26-connectivity respectively. Note that the speed of finding a 3D path is directly related to the number of nodes and edges in the graph. For example, the graph constructed from a 128x128x128 volume data set will have 2 million nodes and about 6 million, 18 million or 26 million edges for 6, 18 or 26-connectivity respectively. Therefore, using a 6-connected graph can find a contour with low quality quickly and a better quality contour can be found by a 26-connected graph with lower speed. The algorithm is about the same as the previous one except the increasing number of nodes and edges.

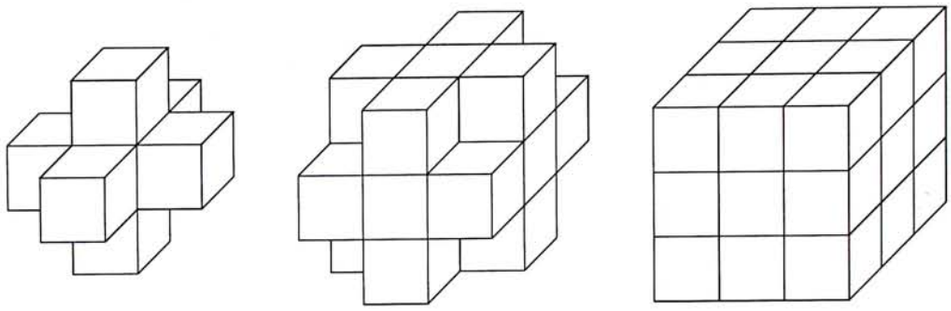


Figure 16: The 6, 18 and 26 connectivity

4.1 3D Graph Construction

The graph construction for 3D Intelligent Scissors can be straight forward. Except the outermost voxels of the volume, each voxel can be assigned 6, 18 or 28 edges which connect to its neighbor voxels. However, if we construct the graph using this straight forward approach, we will be in trouble later. The reason is that it will produce a huge amount of nodes and edges in the graph. Several tens of seconds can be taken to extract just one contour from a graph with 1 million nodes and 26 million edges even on powerful computer. It is because the Dijkstra's algorithm has running time $O(|E| + |V|log|V|)$ if Fibonacci heap[16] is used to implement the priority queue (see Appendix A), where V is the set of nodes and E is the set of edges.

Therefore, we must develop some methods which can reduce the number of nodes and edges. Since voxels with zero gradient are usually not preferred in the graph searching, the elimination of those voxels can reduce a large number of voxels. However, simply cutting away those voxels can cause problems. Figure 17(a) shows an image composed of two line segments. Now consider finding a path from pixel A to pixel B. By Intelligent Scissors, a segment similar to the red line should be found easily. However, if we cut away voxels of zero gradient, the image will be broken into two groups, as shown in Figure 17(b) (non-zero gradients are represented by shaded squares). Since a crack separates the groups, the search originates from node A would

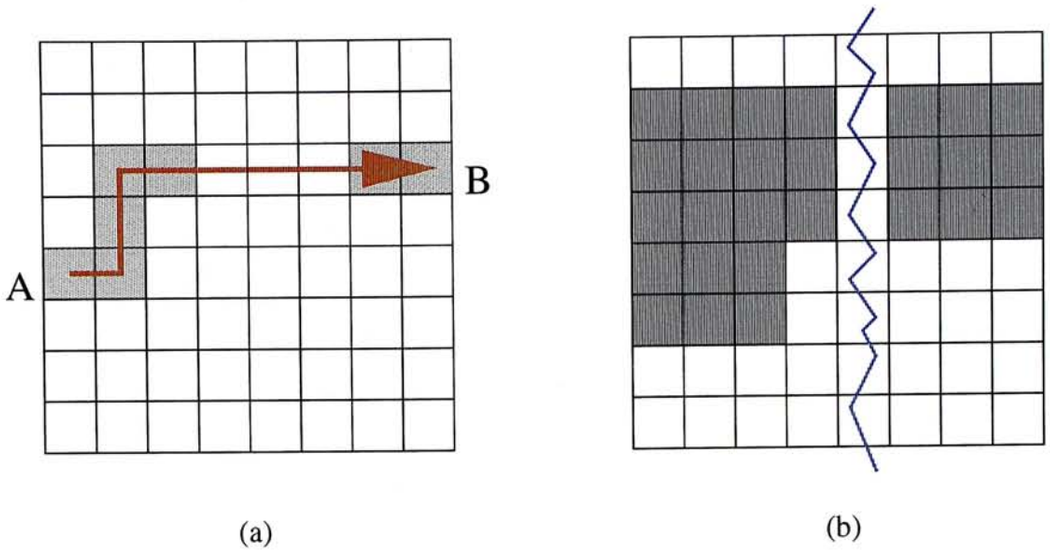
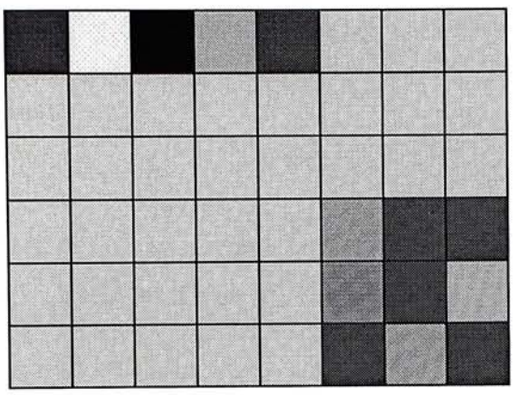


Figure 17: (a) Original image (b) Image gradient

never reach the other group containing B. Thus it is impossible to find a path from A to B. In some medical data, the boundaries between objects can be not clear and may contain zero gradient voxels. Therefore, a simple node pruning is not reasonable.

Now the problem is how to reduce the number of voxels but preserve the connectivities of objects inside the volume. Obviously, we cannot eliminate *all* voxels which have zero gradient. Instead, we should keep some of them and use those voxels to connect between different groups. To solve the problem, we use a data structure called IsoRegion[32], which is used to accelerate ray casting originally. Usually, a volumetric data set contains many empty or homogeneous regions. Each of these regions contains voxels of the same value. Particularly, if the shape of the region is a cube and the size of it is in the form of $(2d+1)^3$ for $d \geq 0$, we say that the region is an IsoRegion and d is called the dimension of that IsoRegion.

IsoRegion can be represented as a 3D array where every element indicates the dimension of the corresponding IsoRegion at that voxel. Therefore, a voxel u has a IsoRegion number n means that the $(2n+1)^3$ voxels of the cube centered at u have the same voxel intensity. Figure 18 shows a 2D image and its corresponding IsoRegion



(a)

0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0
0	1	1	1	0	0	0	0
0	1	2	1	0	0	0	0
0	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0

(b)

Figure 18: IsoRegion of a 2D image

map. The algorithm for the IsoRegion construction can be found in Appendix B.

Since the edge length of an IsoRegion must be odd, it guarantees that there must be *one* voxel placed at the center. Now we modify the previous node pruning algorithm a little bit: all voxels with zero gradient are cut away, except the center voxels of IsoRegions. Connectivity can be established by linking some of the boundary voxels of an IsoRegion to the center voxel(Fig. 19). Linking *all* boundary voxels to the center can produce too many edges if the dimension of the IsoRegion is large. Therefore, only 26 boundary voxels corresponding to the directions of 26-connectivity are considered. Figure 20(a) is the IsoRegion map of an image. According to our algorithm, all voxels inside an IsoRegion are pruned away, except the boundary voxels and the center voxel. Higher priority is given to larger IsoRegions. For example, voxels in Figure 18 marked '1' corresponds to IsoRegions of dimension one. However, they are still cut away, as shown in Figure 20(b), since they are contained by larger IsoRegions of dimension two. Finally, the graph can be completed by linking the boundary voxels to the center voxels, as shown in Figure 20(c). Usually not all boundary voxels of an IsoRegion are present after pruning because they are contained in other IsoRegions. Links are added to those present boundary voxels only. The

pseudo code of the graph construction algorithm is as follows:

Initializations:

$V \leftarrow \{\}$ /* The set of graph node*/

$E \leftarrow \{\}$ /* The set of graph edge */

$I(u) = 0$ for all voxel u /* The IsoRegion map */

$A \leftarrow \{\}$

Algorithm:

$I = makeIsoRegion()$ /* Make IsoRegion map, see Appendix B */

/* Add nodes */

for each voxel u in the volume

if $I(u) = 0$ or $I(u) \geq I(v)$ for all neighbor v then

$V = V \cup \{u\}$

end if

end for

/* Add edges */

for each node $u \in V$

if $I(u) = 0$ then

$A \leftarrow$ { set of neighbor nodes }

else

$A \leftarrow$ { set of the 26 boundary nodes }

end if

for each node $v \in A$

if $(u, v) \notin E$ then

$E = E \cup \{(u, v)\}$

end if

end for

end for

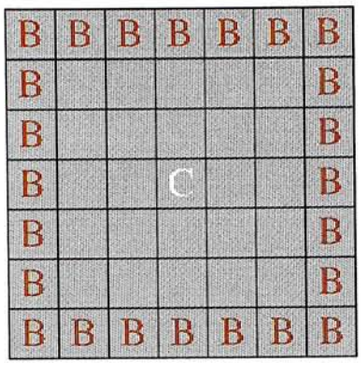


Figure 19: Definition of boundary and center voxels in IsoRegion

4.2 Cost Function

The cost function is about the same as the one described in Section 2.3. The main difference is that it is now applied to volumetric data. Here we use the 3D image gradient and Laplacian zero-crossing functions to evaluate the cost of each edge. Let f_g and f_z be the gradient and Laplacian zero-crossing functions respectively. If $p(u)$ is the position vector of u, then the cost equation is

$$cost(u,v) = \|p(u) - p(v)\| \left(w_g f_g(u,v) + w_z f_z(u,v)\right) \tag{19}$$

where w_g and w_z are the weighting factors controlling the gradient and Laplacian zero-crossing functions respectively. Experiments show that $w_g = 0.7$ and $w_z = 0.3$ give acceptable results.

If (I_x, I_y, I_z) is the gradient vector of a voxel u, the gradient function of a voxel can be represented by

$$f_g(u,v) = 1 - \frac{G(u) + G(v)}{max(G)} \tag{20}$$

where

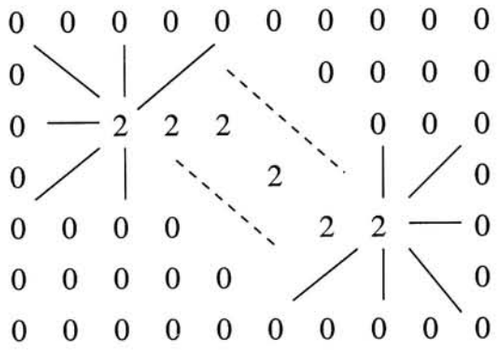
$$G(u) = \sqrt{I_x^2(u) + I_y^2(u) + I_z^2(u)} \tag{21}$$

0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	0	0	0	0
0	1	2	2	2	1	1	0	0	0
0	1	1	1	1	2	1	1	1	0
0	0	0	0	1	1	2	2	1	0
0	0	0	0	0	1	1	1	1	0
0	0	0	0	0	0	0	0	0	0

(a)

0	0	0	0	0	0	0	0	0	0
0						0	0	0	0
0		2	2	2			0	0	0
0					2				0
0	0	0	0			2	2		0
0	0	0	0	0					0
0	0	0	0	0	0	0	0	0	0

(b)



(c)

Figure 20: (a) The IsoRegion map (b) After node pruning (c) Linking

Finally, the equation of Laplacian zero-crossing function is given by

$$f_z(u,v) = \begin{cases} 0 & \text{if } \nabla^2 V(u) * \nabla^2 V(v) \leq 0 \\ 1 & \text{if } \nabla^2 V(u) * \nabla^2 V(v) > 0 \end{cases} \tag{22}$$

where

$$\nabla^2 V(u) = \left\{ \frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} + \frac{\partial^2 V}{\partial z^2} \right\} \tag{23}$$

and it can be approximated by the following discrete form

$$\nabla^2 V(x,y,z) \approx (27V(x,y,z) - \sum_{i=-1}^{1} \sum_{j=-1}^{1} \sum_{k=-1}^{1} V(x+i, y+j, z+k))/26 \tag{24}$$

4.3 Applications

So far we have described what three-dimensional Intelligent Scissors is and how to construct the corresponding graph. In this section we would like to discuss its usefulness by proposing two applications. The first one is a surface extraction algorithm and the second one is a vessel tracking algorithm.

4.3.1 Surface Extraction

Finding contour in one slice by Intelligent Scissors is fast. The original technique can be used to extract contours from different slices and surface can be reconstructed by joining the contours together[19, 14, 54]. However, the process of extracting **every** slice is troublesome. Moreover, the process of joining adjacent contours is complex. Therefore, we hope that the process can be simplified by using as few slices as possible and the joining process is left for 3D Intelligent Scissors. Theoretically, lines extracted by 3D Intelligent Scissors tend to lie on iso-surface. It means that a surface inside a volume can be reconstructed if we use enough 3D lines. In this section, we show that

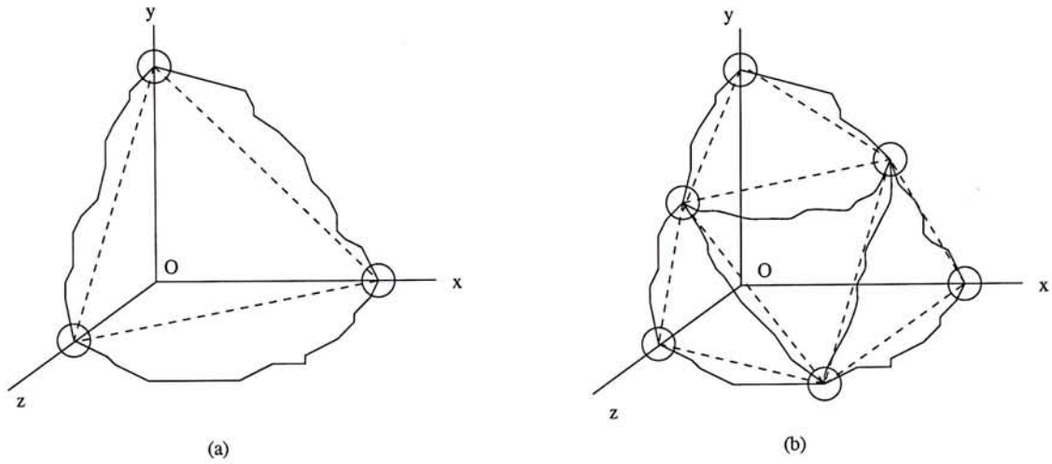


Figure 21: (a) The initial triangle (b) Four triangles after the first subdivision

how to construct iso-surface using three slices which are parallel to the XY, YZ and ZX-plane respectively.

Firstly, we define three contours in the XY-plane, YZ-plane and ZX-plane respectively. The ends of contours are joined together such that it forms a "deformed triangle"(see Fig. 21(a)). The planes can be selected arbitrarily inside the volume and the extractions of the contours in their corresponding planes can be simply done by 2D Intelligent Scissors.

Surface reconstruction is done by applying a recursive subdivision to the "deformed triangle" by subdividing the contours and joining the mid-points together. As a result, four smaller "triangles" will be formed (Fig. 21(b)). Further subdivisions are performed depending on the level of detail(LOD) and finally, a multi-resolution surface mesh can be formed. If a mesh has t triangles, then the mesh of the next level will have $4t$ triangles. Therefore, in general, a mesh of level n has 4^n triangles. Figure 22 shows six surface meshes of LOD 0, 1, 2, 3, 4 and 5 respectively. The volumetric data is a CT head of size 128x128x64 and the shown surfaces are the outermost skin from one of the eight equal sized octants.

The contours are in voxel level. It means that every contour is composed of voxels

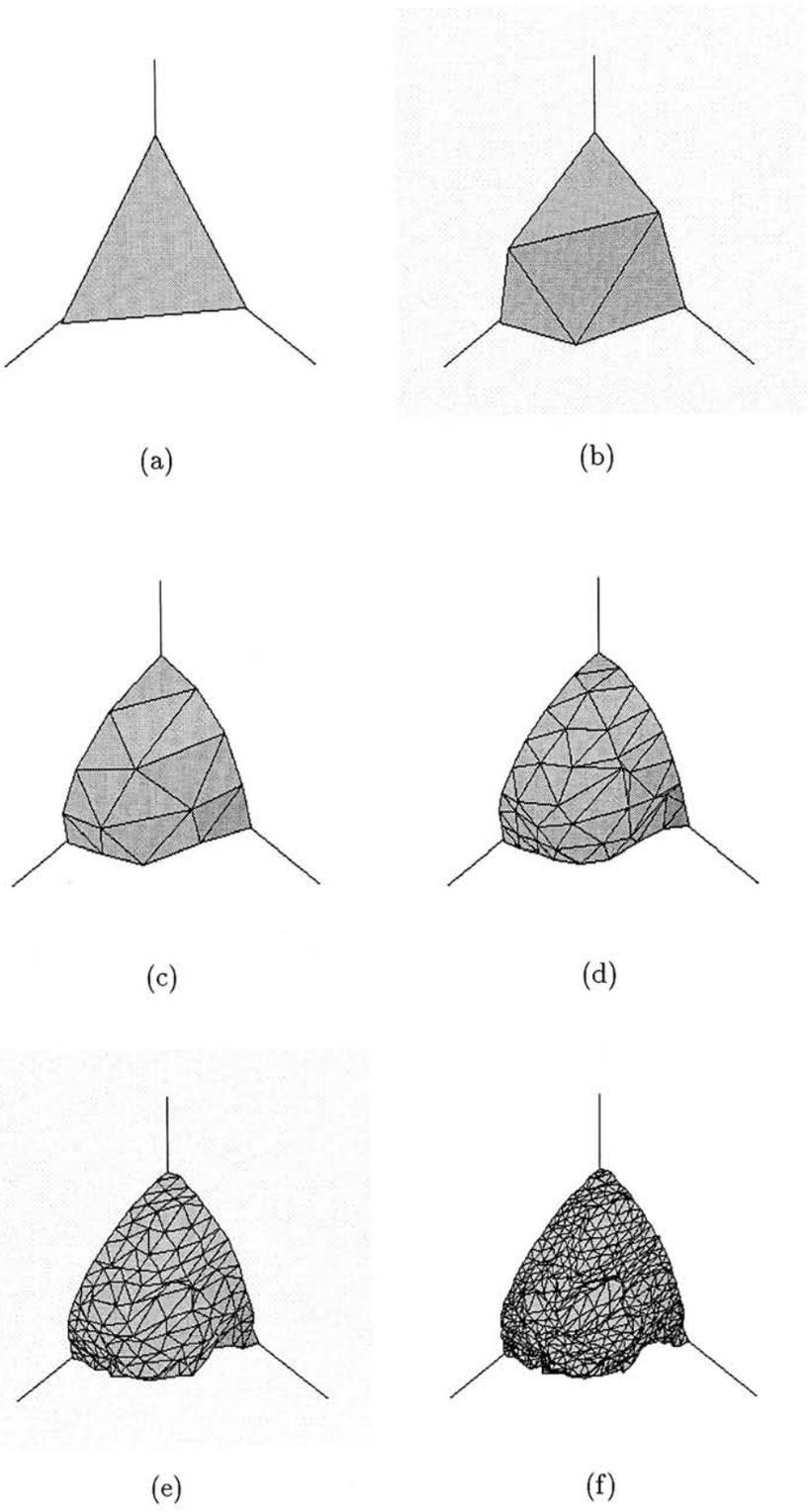


Figure 22: Surfaces for 0, 1, 2, 3, 4, 5 subdivisions respectively

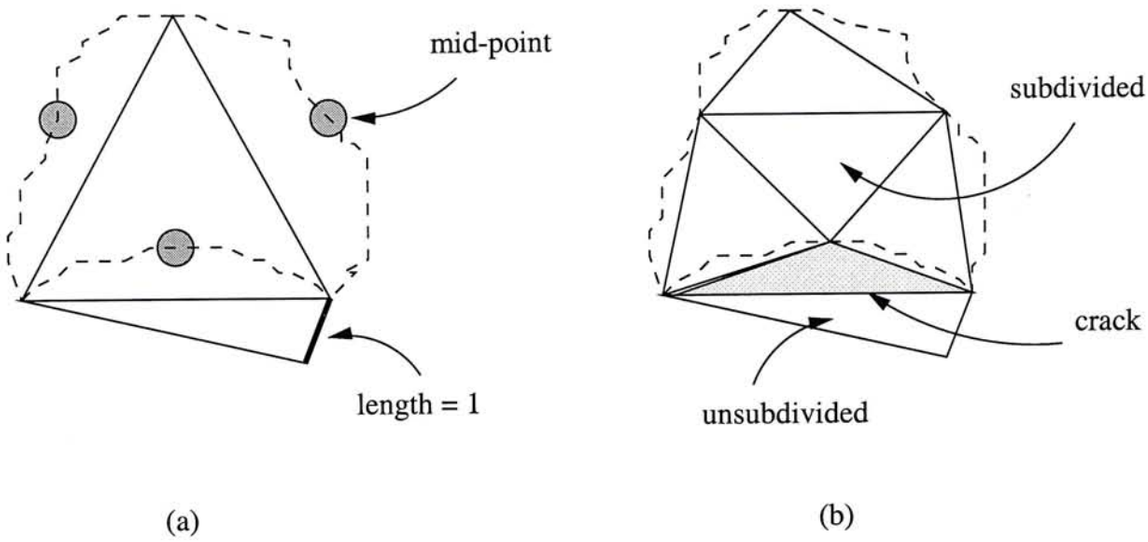


Figure 23: (a) Two triangles before subdivision (b) A crack appears if only one is subdivided

and the midpoint of each contour is also a voxel. Therefore, we say that a contour is of length n if it is composed of $n + 1$ voxels. One problem of the subdivision is that if the length of any one edge is one, i.e., composed of two voxels, the triangle cannot be subdivided into four triangles normally. Thus a crack will be formed if we left it unsubdivided (Fig. 23). The solution is to classify all triangles into four cases and subdivide a triangle according to its configuration. The first case is the normal case where the length of every edge of the triangle is greater than 2 voxels. The second case consists of one edge with length equals to one. The third and fourth cases consist of two and three edges with lengths equal to one respectively. The subdivision methods of these four cases are shown in Figure 24. Following these subdivision rules will make the output surface model a closed mesh, which can be subjected to geometric operations.

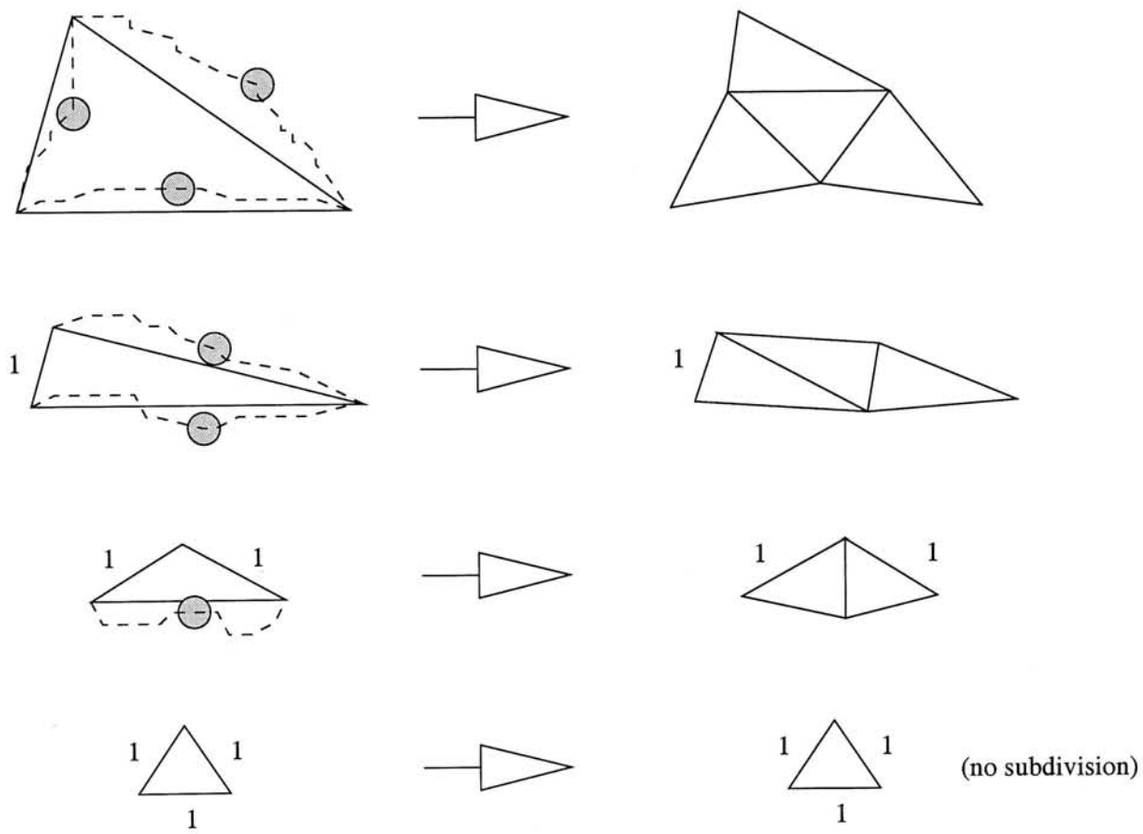


Figure 24: The subdivision rules for the four cases

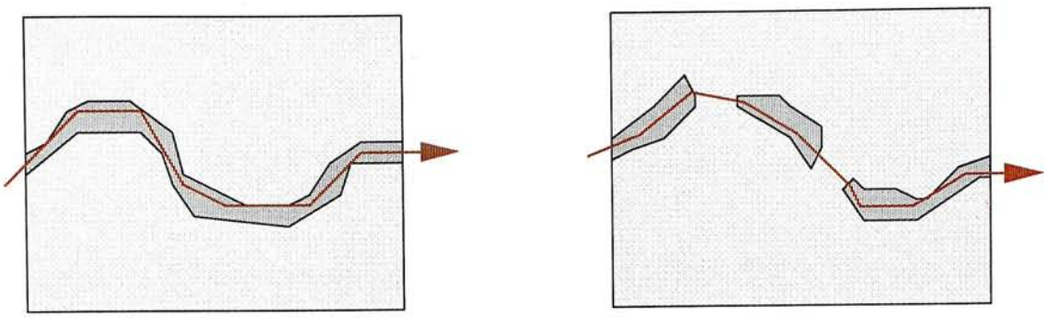
;

4.3.2 Vessel Tracking

In medical data, various kinds of vessel tracking are sometimes needed because clear descriptions of the vessels are important for clinical diagnosis and surgery planning. For example, the human liver can be divided into 8 different segments and these segments can be distinguished according to the branching pattern of the blood vessels. In neurosurgery, a clear understanding of cerebral vasculature can help a lot in the surgery planning. The third example is heart diseases. Congenital or acquired heart diseases may need surgical intervention and dilation of narrowed vessels. The understanding of the patient's actual vessel morphology is very important to the surgery.

Various approaches[53, 47, 13, 36] have been taken to find out blood vessels automatically or semi-automatically in the past. They build the vessel tree either based on multiple X-ray images(biplane angiograms) or directly from volume data using some region growing or image processing techniques. In this section, we propose another method that finds such paths using three-dimensional Intelligent Scissors. 3D Intelligent Scissors has this ability since the line extracted by it tends to "climb" along surfaces. Therefore, a rough path can be found out by the algorithm if the end points of a narrow tunnel are selected as the starting and ending voxels (see Fig. 25(a)). Another advantage of using 3D Intelligent Scissors for the task is it can successfully extract the path even the actual path is broken (see Fig. 25(b)). In some medical data, the diameter of a blood vessel may sometimes smaller than one voxel. Therefore, it makes some region growing algorithms[53] unsuitable in such cases.

A path between two voxels can be found out easily by Dijkstra's algorithm(Section 2.2). Normally Dijkstra's algorithm originates from the source node and stop the searching if the target node is reached. It means that the graph is only partially filled if a "point to point" shortest path is to be found. Therefore, if the source node is fixed, the partially filled graph can be re-used in order to save time, since some or most of



(a) (b)

Figure 25: Path finding in volumetric data

the nodes have already been traversed. This property can be applied to the construction of a vessel tree since all branches of the tree have a common source/root node. The building of a vessel tree can be done by a simple algorithm:

Initializations:

$r \leftarrow$ root node

$T \leftarrow$ { all target nodes }

$G \leftarrow$ the graph (V, E) /* $V =$ { all nodes}, $E =$ { all edges} */

$B(u) \leftarrow nil$ for all $u \in V$

/* $B(u) = v$ if there is an extracted path containing edge (u,v) */

Algorithm:

for each $u \in T$

if $B(u) = nil$ then

$Dijkstra(G, B, r, u)$

endif

end for

As a result, the path information is stored in the node array $B(u)$, which is the back pointers that indicate the paths (see Fig. 26(a)). Therefore, a path from root node r to a target node v can be found by traversing the back pointers and the path

can be represented as

$$path = \{v, B(v), B(B(v)), B(B(B(v))), ..., r\} \tag{25}$$

Other than the path extraction, the locations of branch points are also important in some applications, for example, the navigation of lung air ways. A branch point is the intersection of two different paths originated from a single source node (see Fig. 26(b)). The locations can be found out by examining all elements of array B. If a node v is pointed by other two or more nodes, we say that v is a branch point. The following algorithm finds out all branch points and stores the branch nodes in K:

Initializations:

$C(u) \leftarrow 0$ for all $u \in V$

$K \leftarrow \{\}$ /* the array stores branch nodes */

Algorithm:

for each node $u \in V$

$C(B(u)) \leftarrow C(B(u)) + 1$

end for

for each node $u \in V$

if $C(u) > 1$ then

$K = K \cup \{u\}$

end if

end for

4.4 Summary

In this chapter, we extend the idea of two-dimensional Intelligent Scissors to three-dimensional. The graph construction of volumetric data introduces a huge amount

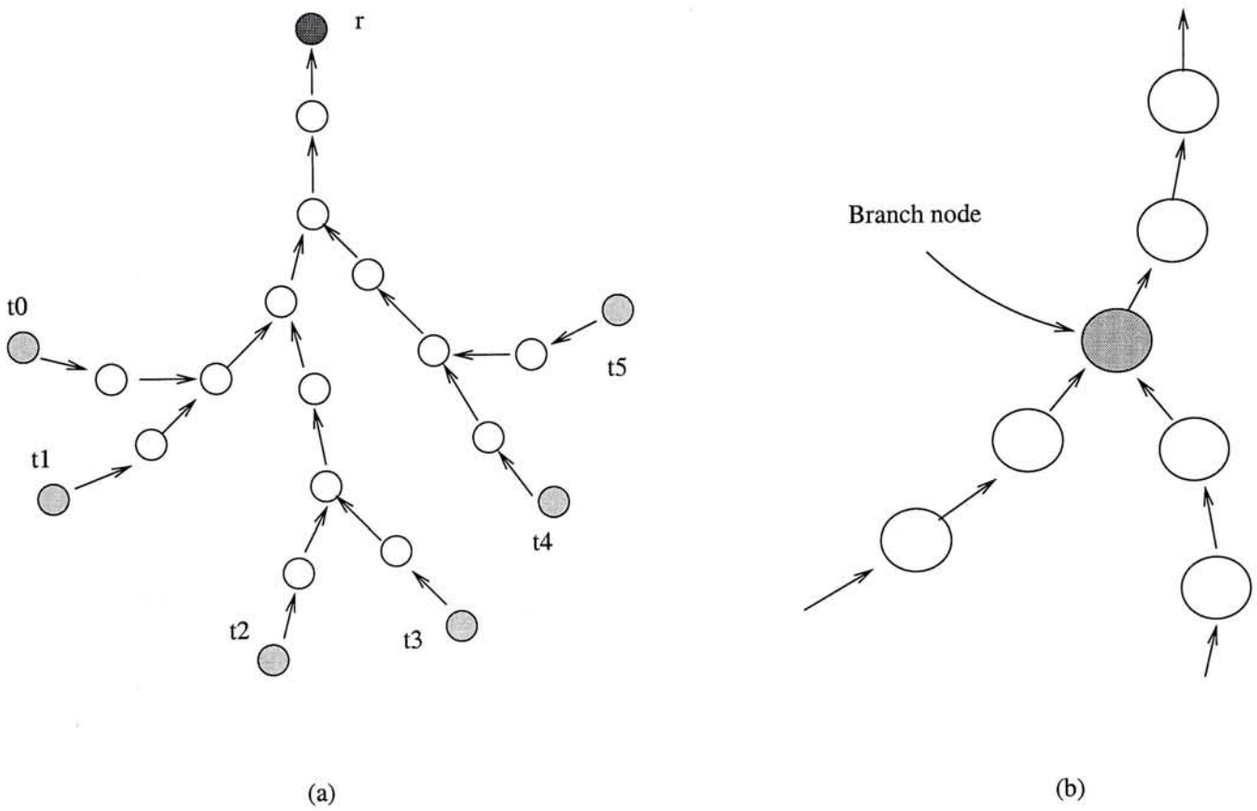


Figure 26: (a) The vessel tree built from back pointers $B(u)$ (b) Branch node is pointed by two or more other nodes

of nodes and edges, which severely slows down the contour extraction algorithm. In order to reduce both the number of nodes and edges, we cut away most of the nodes inside homogeneous regions and leave some of them for connections of different groups.

We apply the three-dimensional Intelligent Scissors in two areas: surface extraction and vessel tracking. Since lines extracted by the algorithm tend to lie on surfaces, enough number of such lines can reconstruct the surface inside the volume. The surface is built by a recursive subdivision of an initial abstact triangle with three non-straight edges. The output is a topologically simple surface mesh bounded by the initial deformed triangle.

By the same characteristic, a vessel can be tracked out provided the two ends of the vessel are known. This can be used to find blood vessels, cerebral vasculature or lung air ways. It is also demonstrated how to find out branch points of a vessel tree in order to have a virtual fly-through of the volume data.

The main contribution of 3D Intelligent Scissors is that it can extract contours in volumetric data. With this property, we can extract meaningful curves lying on surfaces. The algorithm would be more useful if we can extract contours in an interactive rate. However, the huge amount of nodes and edges in a graph makes the process far from interactive and it becomes the major drawback of the algorithm. It is obvious that we need to improve our graph pruning algorithm and graph searching technique in order to improve the performance.

In the next section, we show the implementations of these new algorithms on a virtual environment called Virtual Workbench. User-friendly interfaces are built in order to give user an easy and efficient control of these algorithms on volumetric data.

Chapter 5

Implementations in a Virtual Environment

When dealing with high-dimensional data, a two-dimensional interface is usually not sufficient. Instead, a higher dimension interface is preferred. Particularly, Virtual Workbench[43] is one of such interfaces that is suitable for the manipulation of three-dimensional data and provides stereo display and three-dimensional interaction. Various applications have shown that it is an appropriate virtual interface for three-dimensional dextrous work[4, 44, 45].

In previous chapters, we have described a new algorithm for volume cutting which is based on the technique of dynamic programming. By using a closed contour lying on the surface of a volume, the volume can be cut into two parts (see Chapter 3). An extension of the 2D Intelligent Scissors to 3D is also presented in Chapter 4. Based on this extension, two new algorithms that extract surfaces and vessels from volumetric data are developed.

In this chapter, we show the implementations of our algorithms on Virtual Workbench. Some experiments have been carried out and all of the results in this chapter have been measured on a SGI Octane R10000 MXI workstation. A programming library called *BrixMed* [37] is used. It is mainly designed for the Virtual Workbench and many useful classes and their call-back functions are well defined, including plane, volume, toolrack, tools, lights,...etc. A snapshot of the working environment is shown in Figure 27. Typically the environment consists the target objects(the volumetric

Figure 27: A typical working environment in Virtual Workbench

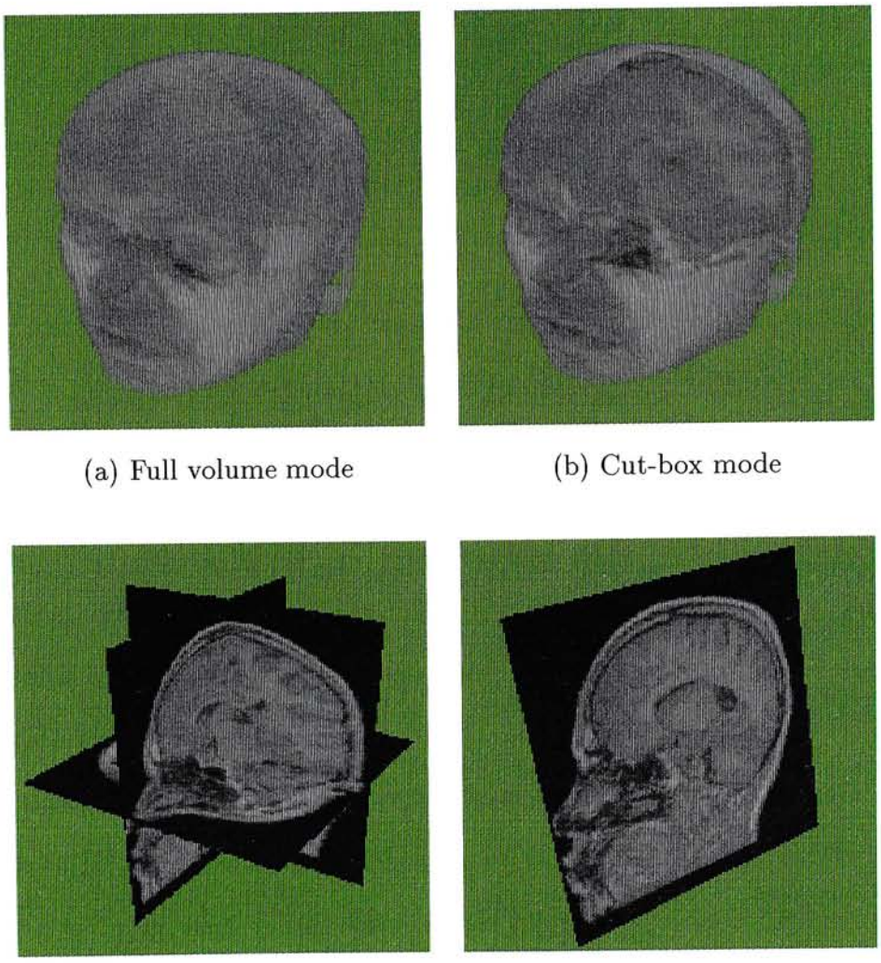
head), a tool(the pen) and a toolrack. Different tools for different operations can be selected by clicking the buttons of the toolrack.

In *BrixMed* , volumetric data is rendered using 3D texture mapping and there are four modes to display the volume, namely, full volume mode, cut-box mode, triplanar mode and monoplanar mode. Figure 28 shows the effects of applying these display modes to a volume.

5.1 Volume Cutting

In Chapter 3 we introduce a new algorithm for cutting volumetric data. The program is implemented on Virtual Workbench and a simple interface is built. The cutting process is separated into two parts. The first part is contour definition, which selects and edits a closed contour. This contour is used to cut the volume into half later. The second part is a cutting surface construction and the volume can be separated by region growing using this surface as boundary.

Before the contour definition, a surface mesh has to be built first. For simplicity,



(a) Full volume mode

(b) Cut-box mode

(c) Triplanar mode

(d) Monoplanar mode

Figure 28: The four volume display modes

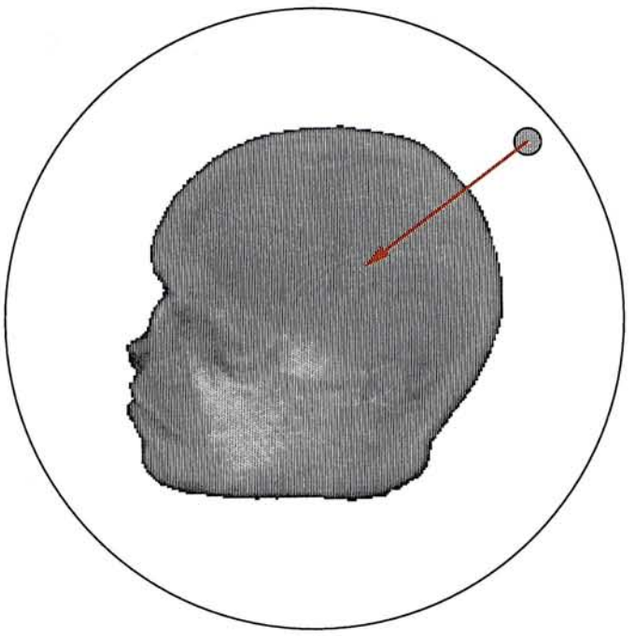


Figure 29: Environmental mapping of a volume

we use an environment map to represent the mesh. It is constructed by projecting rays from a sphere containing the volume to the center and sampling along the rays in order to decide whether the rays reach the surface or not(see Fig. 29). The major drawback of this method is that every point on the surface must be visible from one point inside the volume, otherwise the mesh would be incorrect. In fact, other more complicated and accurate mesh extraction algorithms can be used here and our algorithm works in the same way. However, since mesh extraction is not the main concern of our cutting algorithm, we choose to generate the surface mesh using a simple algorithm.

The first step of our cutting algorithm is to define a closed contour lying on the surface mesh. A point on the mesh can be chosen using a six degree-of-freedom(6DOF) stylus. When the stylus moves near the surface mesh, the nearest mesh point will appear as a sphere. Points with higher gradient magnitudes can be selected by the stylus with the help of a 3D snap. Every time we choose a mesh point, the mesh points covered by the snap are also considered such that a better edge point can be

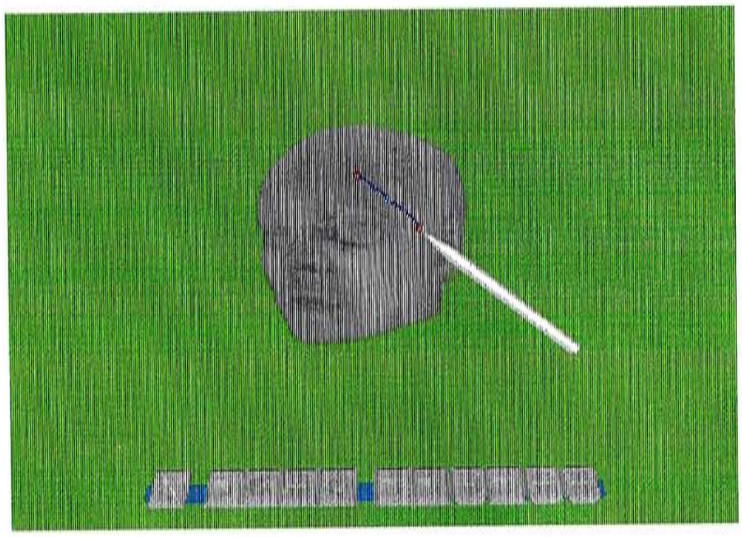
selected. Figure 30 shows the procedure for defining a closed contour by the interface.

After the contour definition, the rest of the process is automatic. Firstly, based on the closed contour, an internal cutting surface can be constructed using the algorithm described in Section 3.3. Since we now have a surface mesh and the internal cutting surface, volume separation can be done by a simple region growing algorithm. Figure 31(a)-(f) show some results of this application. Figure 31(a) is the original volume, which is a 128x128x64 CT head. Figure 31(b) is the state after defining a closed contour surrounding the nose and Figure 31(c) is the volume after cutting away the nose. Figure 31(d)-(f) is a similar case except that instead of the nose, a part of the face is cut off.

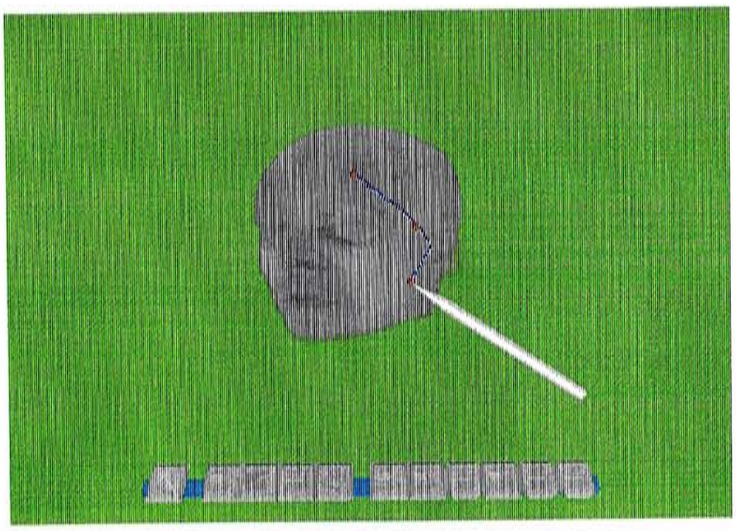
5.2 Surface Extraction

The algorithm described in Section 4.3.1 shows how to extract surface from volumetric data by using three joined contours. The three contours can be lying on any three planes. For simplicity, we use the XY-, YZ- and ZX-plane only. Note that it works in the same way even three arbitrary planes are used.

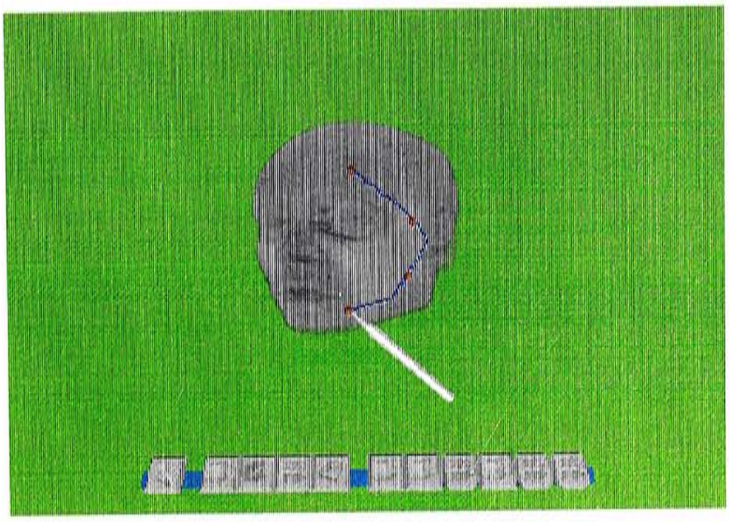
To extract the surface, we have to extract the three boundary contours first. The extraction is done by the interface shown in Figure 32. The interface consists of three planes from the volume, which can be rendered using the triplanar mode. At the very beginning, the end points of each contour are fixed and the three points have the greatest value along their corresponding axes by default (Fig. 32(a)). Three sliders are used to tune the three points to the best positions (Fig. 32(b)). After placing the three points correctly, the user can use a 3D stylus to edit the contours (Fig. 33(a)). New nodes are inserted into the contours in order to change their shapes. For example, if u and v are the end nodes of a contour and the path $u \rightarrow v$ is found by the algorithm, a node p can be inserted between u and v such that a new contour $u \rightarrow p \rightarrow v$ can be



(a)



(b)



(c)

Figure 30: Closed contour definition

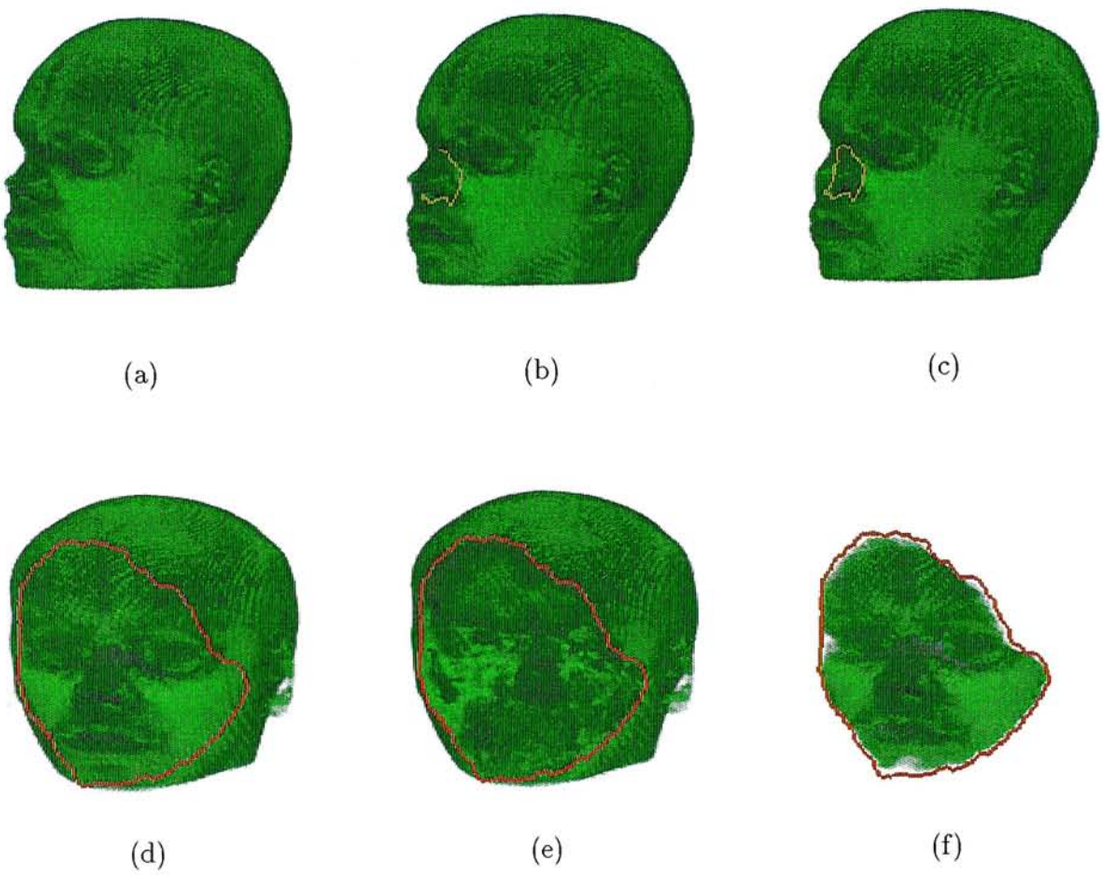


Figure 31: Volume cutting by a closed contour

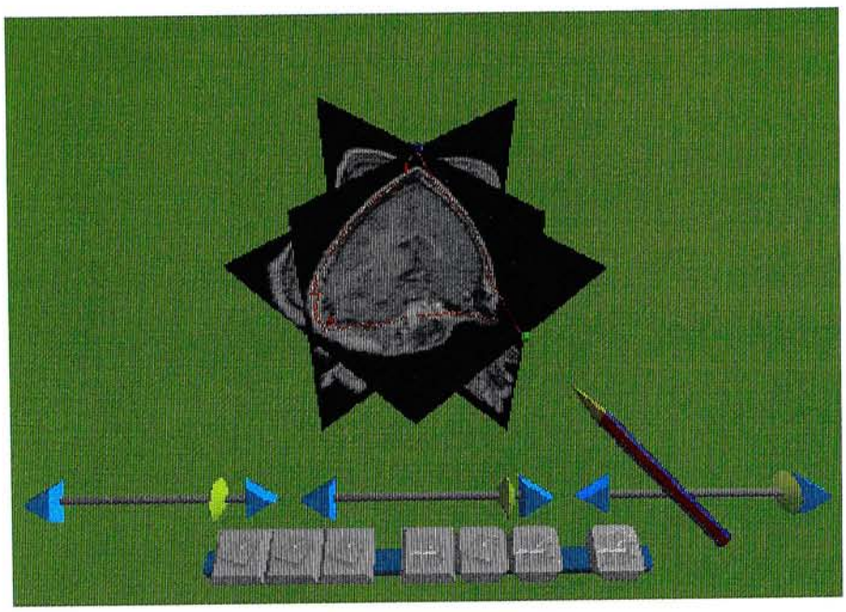
formed. The leftmost three buttons in the menu bar control which contour is going to be edited. When a plane is selected, a line perpendicular to the plane will be drawn from the tip of the stylus to the plane, in order to indicate the position accurately. Figure 33(b) shows the final view of the three edited contours. The speed of finding and editing the three contours is fast here since the graph searches are restricted to two-dimensional planes only. Later construction of the surface, however, consumes much more time because the searches are done throughout the volume.

After the contour definitions, the surface bounded by these three contours can be constructed by recursive subdivisions. User is required to choose the level of detail which represents the subdivision levels that the process is going to perform (see Fig. 34(a)). A surface mesh can then be extracted and Figure 34(b) shows the resulting surface from a corner of a 128x128x64 CT head. It is the same surface shown in Figure 22 of Section 4.3.1. Table 1 shows the timing results that needed to generate the surface shown in Figure 34(b) with different levels of detail. The data set is the same for all four cases except the positions of the three planes. Each graph is 18-connected. For each case four surfaces are produced with levels 1 to 4.

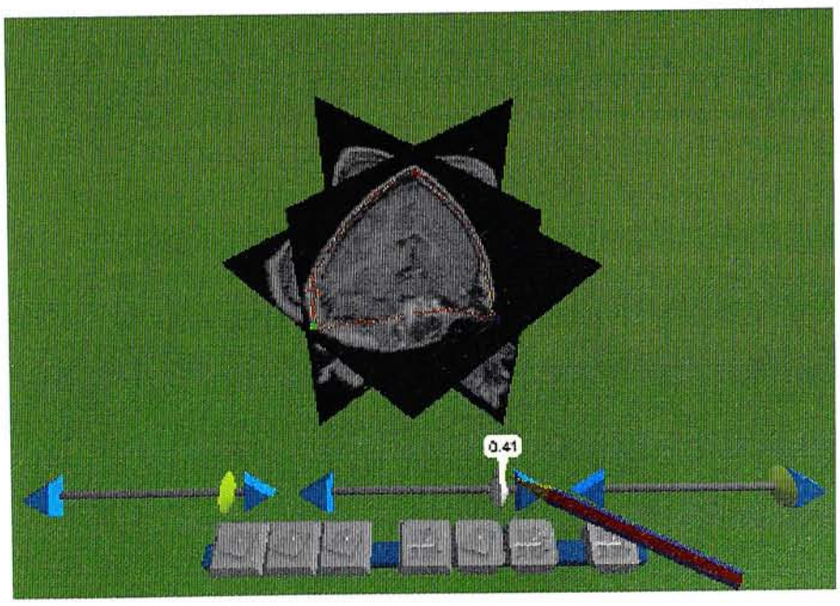
5.3 Vessel Tracking

In this section, the implementation of the vessel tracking algorithm on Virtual Workbench is shown and it is applied to the tracking of lung air ways. Commonly, vessels from medical data are found by region growing[53], 3D image processing techniques[47] or tracing manually[44]. In Section 4.3.2, we has presented an alternative method that uses dynamic programming to find a vessel between two voxels. Here we show that how the algorithm actually works in a virtual environment.

To search for vessels in a volume, we have to construct the corresponding 3D graph first. The data set we used is a 128x128x64 lung data set. A straight forward

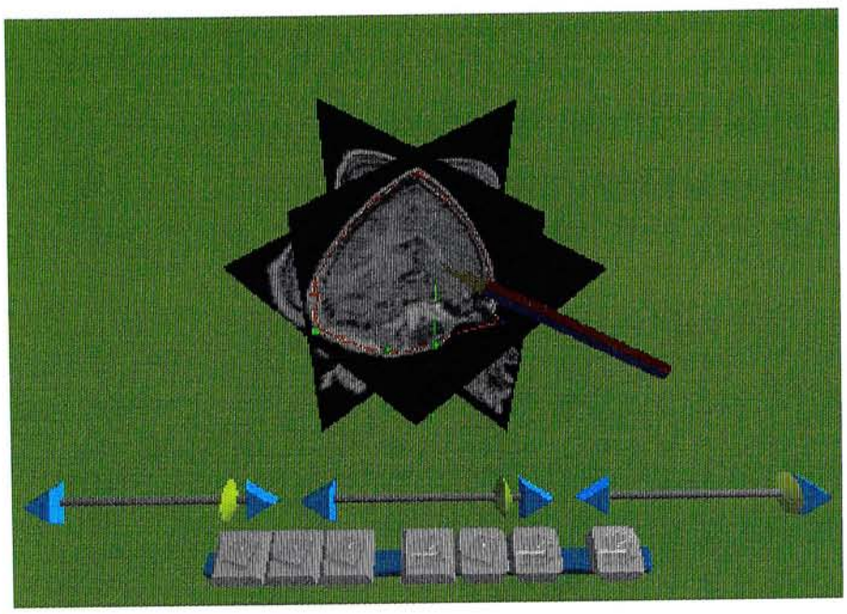


(a)

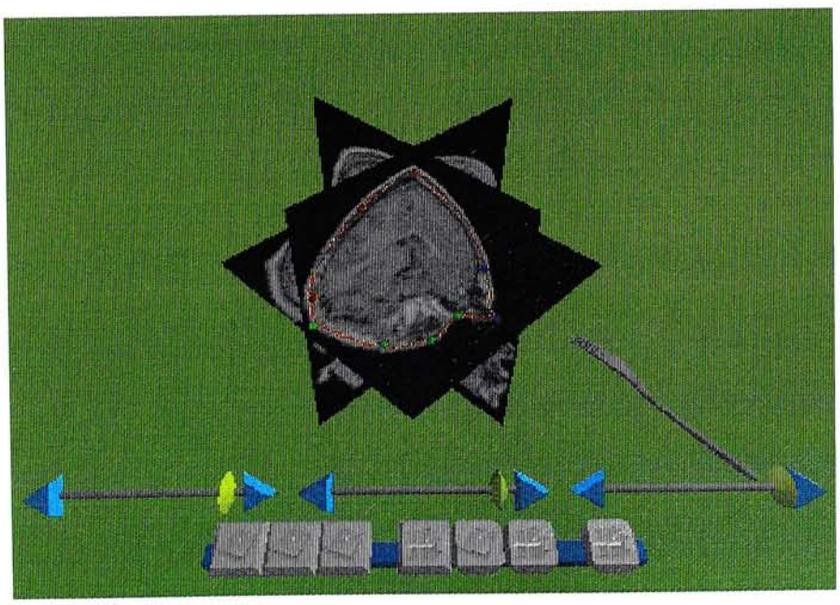


(b)

Figure 32: Joint points adjustment

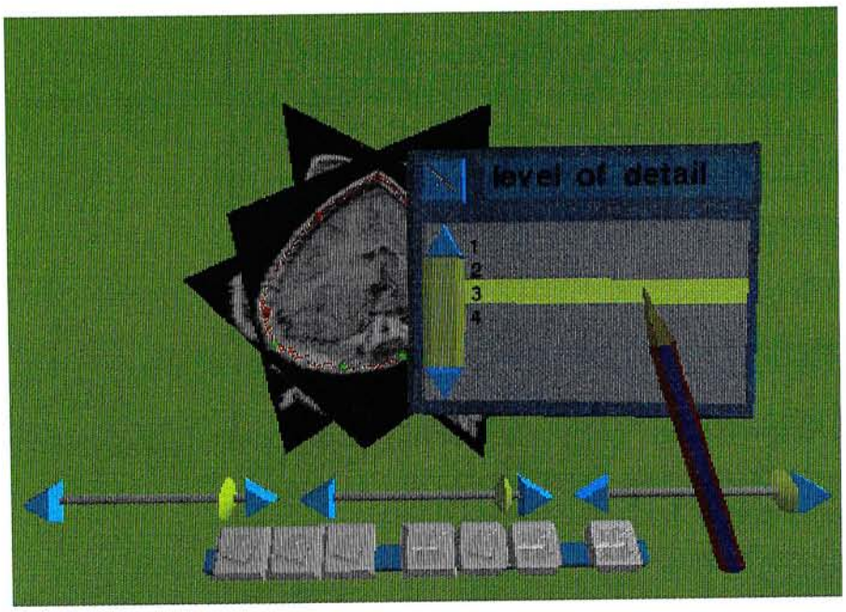


(a)

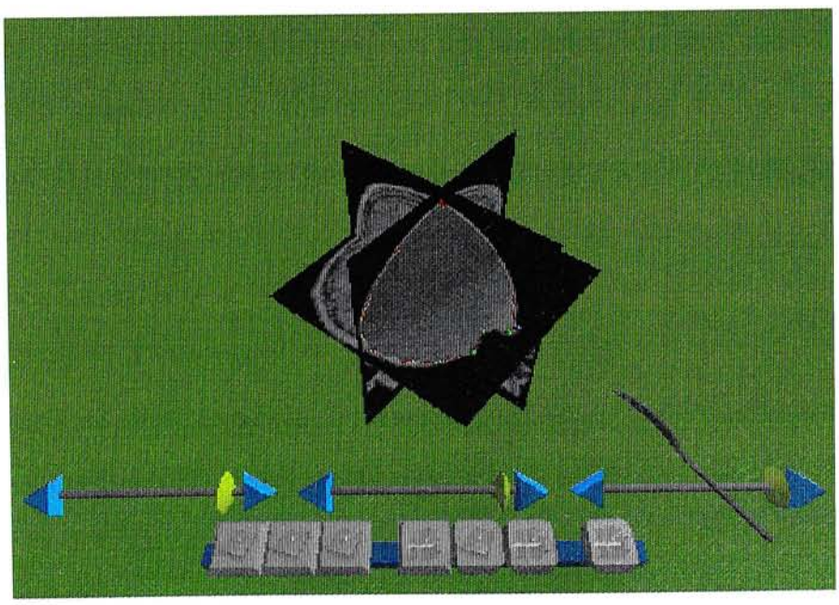


(b)

Figure 33: Contour editing by a 3D stylus



(a)



(b)

Figure 34: Construction of the surface

size	no. nodes	no. edges	level	no. triangles	CPU time(sec.)
32x32x32	10435	69572	1	4	0.07
			2	16	0.27
			3	58	0.85
			4	202	2.68
64x32x32	30289	236704	1	4	0.68
			2	16	2.08
			3	64	4.32
			4	250	9.31
64x64x32	78609	664090	1	4	3.03
			2	16	7.48
			3	64	12.51
			4	256	22.95
64x64x64	149560	1293068	1	4	7.12
			2	16	16.86
			3	64	28.16
			4	256	47.97

Table 1: Timing results for surface construction

construction of the volume would produce 1 million nodes and about 9 million edges for 18-connected graph. Since the numbers of nodes and edges are too large, the volume is thresholded such that voxels with intensity smaller than 30 are set to zero. Moreover, since the upper part of the volume contains unnecessary data, the search is limited to voxels with $y < 90$. After thresholding and node pruning, the number of nodes is 372820 and the number of edges is 3158320.

The interface of this program is simple(Fig. 35). User can choose the display mode of the volume by the middle four buttons in order to spot the vessel end points more easily. The two right buttons on the menu bar control the selections of vessels. After pressing the first button, the voxel selected by the 3D stylus will be set as the root of a vessel tree. Only one voxel can be set as root since only one tree is going to be extracted in this program. The second button enables the tool to select an arbitrary number of terminals. Figure 35 shows the steps of defining the vessel

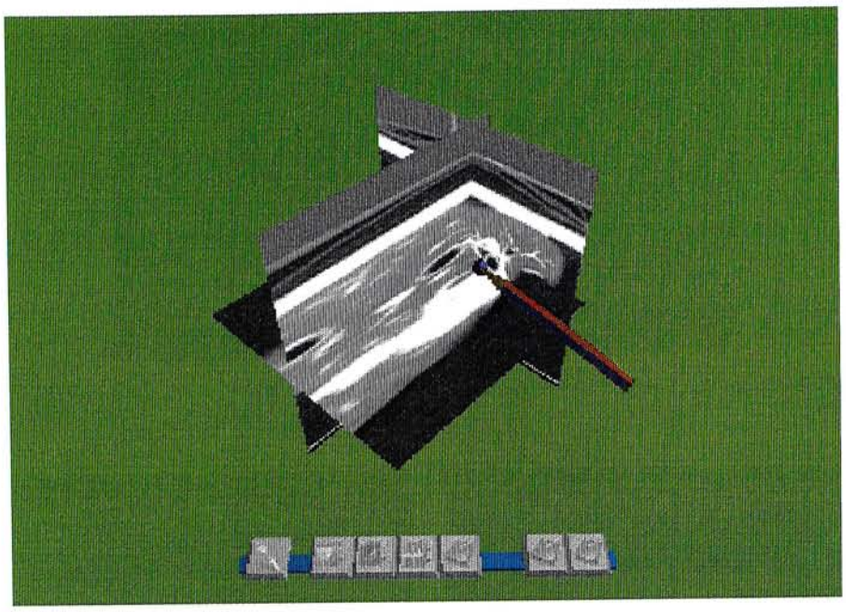
size	no. nodes	no. edges	CPU time(sec.)	$(\|V\|log\|V\| + \|E\|)/100000$
32x32x32	32500	268165	0.83	3.12
64x23x32	65088	547833	1.85	6.42
64x64x32	113374	982738	3.49	11.55
64x64x64	227503	2024591	8.24	23.91
128x64x64	390123	3470100	15.15	41.27

Table 2: Timing results for vessel founding

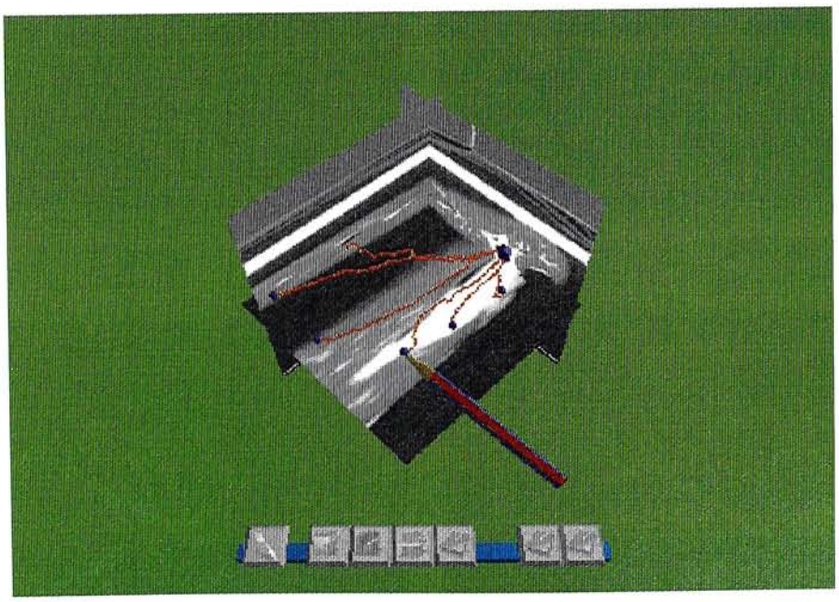
tree. In Figure 35(a), the voxel with a small sphere is the root selected by the stylus. Vessels can be found by specifying the ends of the vessels(Fig. 35(b)). The volume in Figure 36(a)-(e) is rendered in monoplanar mode and we can see that the extracted vessel(red line) is passing through one of the air ways. Provided that the graph is present and only one tree is needed to be found, the extraction of vessels from that root is fast. The most time consuming part of the algorithm is to find the paths from all nodes to the root node. It is basically a *single-source shortest path problem* . If the root node is fixed, this calculation is needed to be done once. Table 2 shows the CPU time needed to find all paths provided a root node is given. The data set is the same lung data used before and only part of it is used each time. As shown in appendix A, this algorithm has a complexity of $O(|V|log|V| + |E|)$. The last field of the table computes $|V|log|V| + |E|$ and it can be seen that the increasing rates of this value and the time is approximately linear.

5.4 Summary

In this chapter, we show the implementations of our algorithms described in previous chapters. For the manipulations of volumetric data, the platform is important. The working environment should be able to provide a high-resolution stereo display and true 3D interaction. Therefore, we choose the Virtual Workbench as our working environment, which can be used to do some dextrous work.

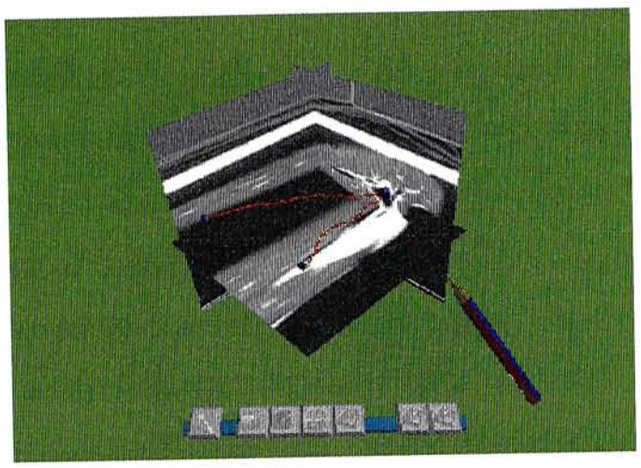


(a) root selection

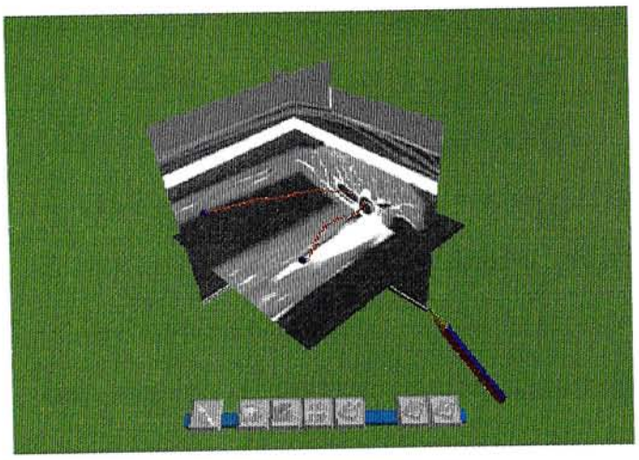


(b) terminal selection

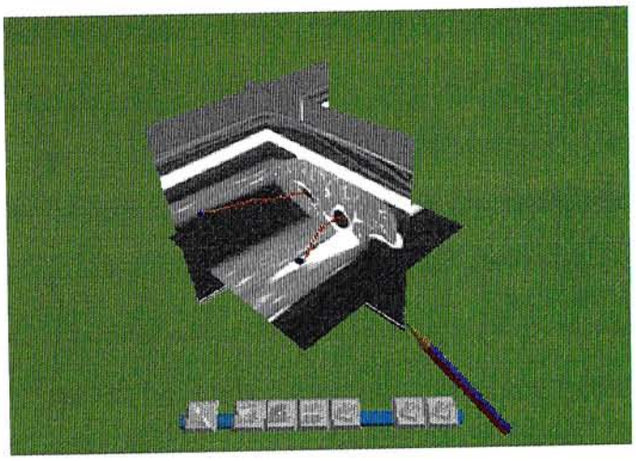
Figure 35: Construction of the vessel tree



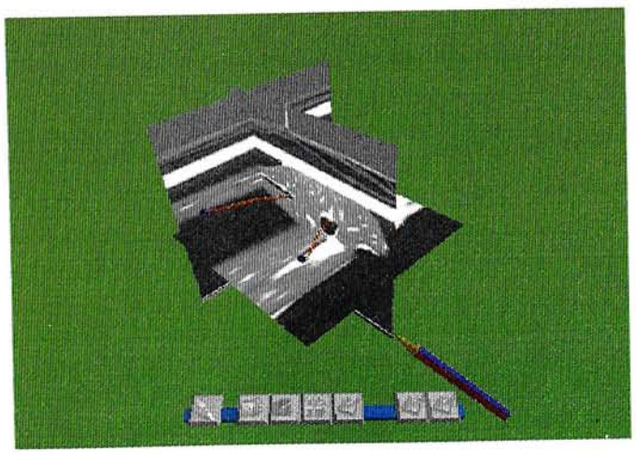
(a)



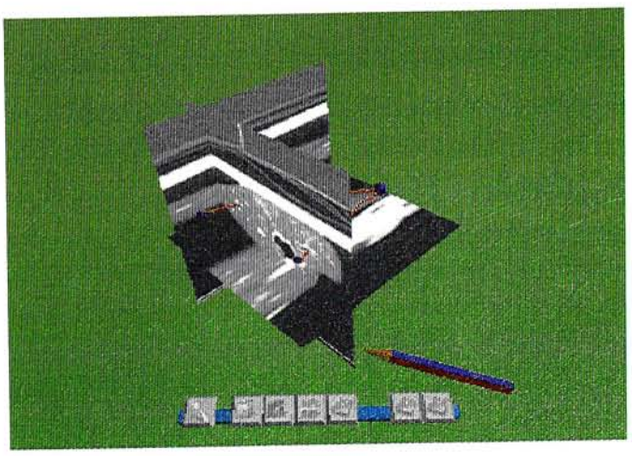
(b)



(c)



(d)



(e)

Figure 36: The extracted vessel passes through the actual path

In the previous two chapters, one volume cutting algorithm, one surface extraction algorithm and one vessel tracking algorithm are proposed. The first one is based on the 2D Intelligent Scissors and the last two are based on the 3D Intelligent Scissors. Three programs have been written for the algorithms using the *BrixMed* programming library. Each interface of the programs has four buttons in common. These buttons control the display modes of the volume. In *BrixMed* , four display modes are supported, namely, full volume mode, cut-box mode, triplanar mode and monoplanar mode. Using these display modes can help identifying features inside volumetric data in a more convenient and efficient way. The three interfaces are very simple and have only 2-6 buttons related to the corresponding algorithm. For example, the vessel tracking program only have two buttons related to the algorithm. One for root locating and one for terminal locating.

The applications are not real-time. It is because the graph needed to be searched is too large, even we have used a graph node pruning algorithm to cut away most of the unnecessary nodes. To have interactive performance, some kinds of volume compression or node clustering algorithms have to be used, in order to reduce both the number of nodes and edges in the volume graph.

Chapter 6

Conclusions

In this thesis, manipulations and feature extractions of volumetric data are the major subjects. The performance of pure volume visualization is not the main concern in this work. With the help of hardware assisted 3D texture mapping technique, volume visualization can already be interactive nowadays. With small data set(e.g., 128x128x128), it is not hard to achieve 20 frames per second in a small window like 128x128. One of the major drawbacks of 3D texture mapping is that the performance is directly related to the amount of available texture memory. Moreover, usually only high-end graphics workstations have this functionality. However, we believe that the rapid evolution of computer hardware would make 3D texture mapping a common feature of standard workstations eventually. Therefore, we only focus on how to manipulate and understand volumetric data in the greatest extent by the help of a virtual working environment.

In this chapter, we give a summary of our work and some future research directions and improvement are proposed.

6.1 Summary of Results

The work in this thesis mainly emphasizes on volume cutting and feature extraction algorithms. These algorithms are summarized as follows:

- **Application of Intelligent Scissors on volume cutting.** *Intelligent Scissors* is a successful 2D image segmentation tool. An image is transformed into a graph and boundary features are found by Dijkstra's shortest path algorithm. By successive extractions of contours, a closed contour can be formed for image segmentation. We applied it to volumetric data such that a closed contour on the volume surface can be extracted in a similar manner. The volume can then be separated into half based on the contour.

- **Extension of Intelligent Scissors from 2D to 3D.** Although Intelligent Scissors has been applied to volumetric data in the volume cutting algorithm, it is still a 2D contour extraction tool. We extended the tool to 3D by constructing a 3D graph of the volume. A contour between two voxels is extracted by Dijkstra's algorithm and feature lines lying on the surface inside the volume can be found.

- **An alternative surface extraction algorithm.** Surface extraction is an important process in volume segmentation because it gives user a more clear view of the interior part of a volume. Since contours extracted by 3D Intelligent Scissors tend to lie on boundary features, sufficient number of such contours should be able to reconstruct the surface. In this thesis, a surface construction algorithm based on three manually traced contours is proposed. Experiments show that the algorithm works well for the extraction of simple surface.

- **A vessel tracking algorithm.** 3D Intelligent Scissors has the ability of finding path between two voxels since the gradients of voxels along vessel should be high and this characteristic fits the 3D Intelligent Scissors algorithm quite well. Experiments have been done on lung data in order to find out the air ways.

6.2 Future Directions

We have demonstrated how to reconstruct surface and extract vessels from volumetric data by 3D Intelligent Scissors. The basic idea is borrowed from the corresponding 2D image segmentation tool, which is a fast and robust contour tracking algorithm. However, we can see that the performance degrades after extending the algorithm from 2D to 3D due to the huge number of nodes and edges. Although we have reduced the size of graph significantly ($40 \sim 60\%$) by using the IsoRegion data structure, the size is still unreasonable for interactive contour extraction. Time for extracting a vessel tree is still acceptable since it only consists of one root node and the search is only needed once. However, for the surface construction algorithm, we can see that it takes a long time to finish the extraction since the surface consists of a large number of contours with different starting and ending points.

We can see that if we want to improve the performance of our algorithms, we have to further reduce the size of the volume graph. Volume compression techniques like hierarchical compression[24], vector quantization[33] or wavelet[17] may work. Data coherence is important in volume compression and IsoRegion is just one of the many structures that deal with data coherence. A cooperation of various volume compression algorithms and the graph construction algorithm would surely further reduce the size greatly.

Another possible improvement is the automatic extraction of vessels in volumetric data. At this stage, each vessel is found by specifying its two end voxels manually. In an actual application, the vessels needed to be extracted may be too many such that manually locating all end points would be time consuming. In our algorithm, each node of the graph would have a back pointer and a distance variable, which indicates the total cost from that voxel to the root voxel. By examining this variable, we can probably determine whether a voxel is a terminal or not, since paths pass through

homogeneous region tend to have high total cost. Selecting a pre-defined number of voxels with low total cost and reasonable length may extract a large number of correct paths. These voxels may be further evaluated by applying a region growing algorithm, which determines that if the voxel is a tip of a line with certain length, say, 10 voxels.

Finally, a rich set of visualization tools can be integrated into Virtual Workbench. Many techniques are useful if it can be implemented in Virtual Workbench. Especially those need stereo display and 3D manipulation of data. For example, a virtual fly-through along the lung air ways immediately after vessel extractions may be interesting.

Appendix A

Performance of Dijkstra's Shortest Path Algorithm

Recall that the performance of either 2D or 3D Intelligent Scissors depends greatly on the complexity of Dijkstra's algorithm. The algorithm maintains a list of labeled nodes and the list supports the following functions:

- **insert(n,k)** Insert a node n with a key value k into the list.
- **delete_min()** Delete the node with the lowest key value from the list.
- **decrease(n,k)** Decrease the key value of node n to k.

Lists that support these functions are also called priority queues. Usually priority queues are implemented using *heaps* , which is a tree structure that keeps the node with minimum key value as root node. Various kinds of heaps were proposed[25, 23] but most of them require $O(log|V|)$ time for each of the operations. Assume that V and E are the sets of nodes and edges of the graph respectively. Dijkstra's algorithm requires at most $|V|-1$ delete_min() operations, at most $|V|$ insert operations and at most $|E|$ decrease operations. Therefore, Dijkstra's algorithm using normal heaps will have a complexity of $O(|E|log|V|)$.

In 1984, Fredman and Tarjan[16] proposed a data structure called *Fibonacci heaps*, which reduces the complexity of insert() and decrease() operations to $O(1)$. Therefore, the complexity of Dijkstra's algorithm using Fibonacci heap is $O(|E|+|V|log|V|)$.

Appendix B

IsoRegion Construction

The following algorithm is quoted from [32] and it constructs IsoRegion from volumetric data.

Initial step:

for all voxel position (x, y, z)

if $V(x+i, y+j, z+k) = V(x, y, z)$

$\forall i, j, k \in [-1, 1]$

then

IsoRegion$(x, y, z) \leftarrow 1$

else

IsoRegion$(x, y, z) \leftarrow 0$

Iteration step:

$d \leftarrow 1$

repeat

for all voxel position (x, y, z)

if IsoRegion$(w) \geq d$

$\forall w \in N_D(x, y, z)$

then

IsoRegion$(x, y, z) \leftarrow$ IsoRegion$(x, y, z) + 1$

$d \leftarrow d + 1$

until no further IsoRegion found

Bibliography

[1] A.MARTELLI. An application of heuristic search methods to edge and contour detection. In *Commmunications of the ACM* (1976), pp. 73–83.

[2] BAJURA, M., FUCHS, H., AND OHBUCHI, R. Merging virtual objects with the real world: Seeing ultrasound imagery within the patient. *Computer Graphics 26*, 2 (July 1992), 203–210.

[3] BRYSON, S. Virtual spacetime:an environment for the visualization of curved spacetimes via geodesic flows. Tech. Rep. RNR-92-009, NASA NAS, Mar. 1992.

[4] BRYSON, S. Projects in VR: The virtual windtunnel on the Virtual Workbench. *IEEE Computer Graphics and Applications 17*, 4 (July/Aug. 1997), 15–15.

[5] BRYSON, S., AND LEVITT, C. The virtual windtunnel: An environment for the exploration of three-dimensional unsteady flows. In *Visualization '91* (1991), pp. 17–24.

[6] C.H.WONG, Y.H.SIU, P.A.HENG, AND H.SUN. Interactive volume cutting. In *Graphics Interface '98 (to be published)* (1998).

[7] CLINE, H. E., LORENSEN, W. E., LUDKE, S., CRAWFORD, C. R., AND TEETER, B. C. Two algorithms for the reconstruction of surfaces from tomograms. *Medical Physics 15*, 3 (June 1988), 320–327.

[8] COHEN, L. D. On active contour models and balloons. *Computer Vision, Graphics, and Image Processing. Image Understanding 53*, 2 (Mar. 1991), 211–218.

[9] CRUZ-NEIRA, C., SANDIN, D. J., AND DEFANTI, T. A. Surround-screen projection-based virtual reality: The design and implementation of the CAVE. *Computer Graphics 27*, Annual Conference Series (1993), 135–142.

[10] D., M., S., S., AND K.R., S. Surface from contour: the corespondence and branching problem. In *ACM transactions on Graphics* (July 92).

[11] DIJKSTRA, E. A note on two problems in connexion with graphs. *Numerische Mathematik 1* (1959), 269–270.

[12] D.L.POPE, D.L.PARKER, D.E.GUSTAFSON, AND P.D.CLAYTON. Dynamic search algorithms in left ventricular border recognition and analysis of coronary arteries. In *IEEE Proceedings of Computers in Cardiology* (1984), pp. 71–75.

[13] EHRICKE, H.-H., DONNER, K., KOLLER, W., AND STRASSER, W. Visualization of vasculature from volume data. *Computers and Graphics 18*, 3 (May 1994), 395–406.

[14] EKOULE, A. B., PEYRIN, F. C., AND ODET, C. L. A triangulation algorithm from arbitrary shaped multiple planar contours. *ACM Transactions on Graphics 10*, 2 (Apr. 1991), 182–199.

[15] FANG, S., SRINIVASAN, R., HUANG, S., AND RAGHAVAN, R. Deformable volume rendering by 3d texture mapping and octree encoding. In *IEEE Visualization '96* (Oct. 1996), IEEE. ISBN 0-89791-864-9.

[16] FREDMAN, M., AND TARJAN, R. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM 34* (1987), 596–615.

[17] GROSS, M. H., LIPPERT, L., AND KURMANN, C. Compression domain volume rendering for distributed environments. *Computer Graphics Forum 14*, 3 (1997).

[18] HERMAN, G. T., AND LIU, H. K. Three-dimensional display of human organs from computed tomograms. *Computer Graphics and Image Processing 9*, 1 (Jan. 1979), 1–21.

[19] H.FUCHS, Z.M., K., AND S.P., U. Optimal surface reconstruction from planar contours. In *Comm. ACM* (1977), pp. 693–702.

[20] HONG, L., MURAKI, S., KAUFMAN, A., BARTZ, D., AND HE, T. Virtual voyage: Interactive navigation in the human colon. In *SIGGRAPH 97 Conference Proceedings* (Aug. 1997), T. Whitted, Ed., Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 27–34.

[21] JASWAL, V. S. CAVEvis: Distributed real-time visualization of time-varying scalar and vector fields using the CAVE virtual reality theater. In *IEEE Visualization 97* (Nov. 1997), R. Yagel and H. Hagen, Eds., IEEE, pp. 301–308.

[22] JONES, M. W. The production of volume data from triangular meshes using voxelisation. *Computer Graphics Forum 15*, 5 (Dec. 1996), 311–318.

[23] J.VUILLEMIN. A data structure for manipulating priority queues. In *Comm. ACM* (1978), pp. 309–314.

[24] J.WILHELMS, AND GELDER, A. Multi-dimensional trees for controlled volume rendering and compression. In *ACM Siggraph Symposium on Volume Visualization 1994* (1994), pp. 27–34.

[25] J.W.J.WILLIAMS. Algorithm 323: Heapsort. In *Comm. ACM* (1964), pp. 347–348.

[26] KASS, M., WITKIN, A., AND TERZOPOULOS, D. Snakes: Active contour models. In *First International Conference on Computer Vision, (London, England, June 8–11, 1987)* (Washington, DC., 1987), IEEE Computer Society Press, pp. 259–268.

[27] LACROUTE, P., AND LEVOY, M. Fast volume rendering using a shear-warp factorization of the viewing transformation. *Computer Graphics 28*, Annual Conference Series (1994), 451–458.

[28] LORENSEN, W. E., AND CLINE, H. E. Marching cubes: a high resolution 3D surface construction algorithm. In *SIGGRAPH '87 Conference Proceedings (Anaheim, CA, July 27–31, 1987)* (July 1987), M. C. Stone, Ed., Computer Graphics, Volume 21, Number 4, pp. 163–170.

[29] MILLER, J. V., BREEN, D. E., LORENSEN, W. E., O'BARA, R. M., AND WOZNY, M. J. Geometrically deformed models: A method for extracting closed geometric models from volume data. *Computer Graphics (SIGGRAPH '91 Proceedings) 25*, 4 (July 1991), 217–226.

[30] MORTENSEN, E. N., AND BARRETT, W. A. Intelligent scissors for image composition. In *SIGGRAPH 95 Conference Proceedings* (Aug. 1995), R. Cook, Ed., Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 191–198. held in Los Angeles, California, 06-11 August 1995.

[31] OLIVA, J. M., PERRIN, M., AND COQUILLART, S. 3D reconstruction of complex polyhedral shapes from contours using a simplified generalized Voronoi diagram. *Computer Graphics Forum 15*, 3 (Sept. 1996), C397–C408.

[32] P.F.FUNG, AND P.A.HENG. Efficient volume rendering by isoregion leaping acceleration. In *WSCG'98 Computer Graphics and Visualisation* (1998).

[33] P.NING, AND L.HESSELINK. Fast volume rendering of compressed data. In *IEEE Visualization Proceedings 93* (1993), pp. 11–18.

[34] POSTON, T., NGUYEN, H., HENG, P.-A., AND WONG, T.-T. Skeleton climbing: fast isosurfaces with fewer triangles. In *Proceedings of Pacific Graphics'97* (Oct. 1997), pp. 117–126.

[35] RÖLL, S., HAASE, A., AND VON KIENLIN, M. Fast generation of leakproof surfaces from well-defined objects by a modified marching cubes algorithm. *Computer Graphics Forum 14*, 2 (June 1995), 127–138.

[36] ROSS, T., HANDELS, H., BREUER, U., AND SZABO, K. 3D visualization of microvascular blood vessel networks. *Computers and Graphics 19*, 1 (Jan.–Feb. 1995), 89–??

[37] SERRA, L., AND HERN, N. *BrixMed 1.2 User's Guide and Reference Manual.* Institute of Systems Science at National University of Singapore, 1997.

[38] SHEKHAR, R., FAYAD, E., YAGEL, R., AND CORNHILL, F. Octree-based decimation of marching cubes surfaces. In *Proceedings of Visualization'96* (Sept. 1996), pp. 335–342.

[39] SHU, R., ZHOU, C., AND KANKANHALLI, M. S. Adaptive marching cubes. *The Visual Computer 11*, 4 (1995), 202–217. ISSN 0178-2789.

[40] SIMS, D. Applications: From the ground up: building a high-resolution seismic model. *IEEE Computer Graphics and Applications 15*, 4 (July 1995), 15–17.

[41] STALLING, D., AND HEGE, H.-C. Intelligent scissors for medical image segmentation. In *Proceedings of 4th Freiburger Workshop* Digitale Bildverarbeitung in der Medizin, *Freiburg* (Mar. 1996), B. Arnolds, H. Müller, D. Saupe, and T. Tolxdorff, Eds., pp. 32–36.

[42] T.J.CULLIP, AND U.NEWMAN. Accelerating volume reconstruction with 3d texture hardware. Tech. rep., University of North Carolina, 93.

[43] T.POSTON, AND L.SERRA. The virtual workbench: dextrous VR. In *Proc of the Virtual Reality Software and Technology '94 Conference* (94), pp. 111–122.

[44] T.POSTON, L.SERRA, H.NG, P.A.HENG, AND B.C.CHUA. Interactive tube finding on a virtual workbench. In *Second International Symposium on Medical Robotics and Computer Assisted Surgery* (1995).

[45] T.POSTON, W.L.NOWINSKI, L.SERRA, B.C.CHUA, H.NG, AND P.K.PILLAY. The brain bench: Virtual stereotaxis for rapid neurosurgery planning and training. In *Proc of Visualization in Biomedical Computing 1996* (1996), pp. 491–500.

[46] U.MONTANARI. On the optimal detection of curves in noisy pictures. In *Commmunications of the ACM* (1971), pp. 335–345.

[47] WAHLE, A., OSWALD, H., AND FLECK, E. 3D heart-vessel reconstruction from biplane angiograms. *IEEE Computer Graphics and Applications 16*, 1 (Jan. 1996), 65–73.

[48] WESTOVER, L. Footprint evaluation for volume rendering. *Computer Graphics* (August 90).

[49] WESTOVER, L. Splatting - a parallel, feed-forward volume rendering algorithm. Tech. rep., Dept. of Computer Science, UNC at chapel Hill, July 91.

[50] WILSON, O., GELDER, A. V., AND WILHELMS, J. Direct volume rendering via 3d textures. Tech. rep., University of California, Santa Cruz, June 94.

[51] WOLFE, JR., R. H., AND LIU, C. N. Interactive visualization of 3D seismic data: a volumetric method. *IEEE Computer Graphics and Applications 8*, 4 (July 1988), 24–30.

[52] Y.P.Chien, and K.S.Fu. A decision function method for boundary detection. In *Computer Graphics and Image Processing* (1974).

[53] Zahlten, C., H.Jürgens, and H.-O.Peitgen. Reconstruction of branching blood vessels from CT-data. In *Visualization in Scientific Computing* (1995), pp. 41–52.

[54] Zyda, M. J., Jones, A. R., and Hogan, P. G. Surface construction from planar contours. *Computers and Graphics 11*, 4 (1987), 393–408.