1-1-2012

A Sketch-Line Interaction Model for Image Slice-Based Examination and Region of Interest Delineation of 3D Image Data

Yueh-Shan Shih
Ryerson University

A SKETCH-LINE INTERACTION MODEL FOR IMAGE SLICE-BASED EXAMINATION AND REGION OF INTEREST DELINEATION OF 3D IMAGE DATA

by

Yueh-Shan Shih, BSc, Ryerson University, Toronto, Ontario, 2009

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Science

in the Program of

Computer Science

Toronto, Ontario, Canada, 2012

AUTHOR'S DECLARATION FOR ELECTRONIC SUBMISSION OF A THESIS

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my thesis may be made electronically available to the public.

A SKETCH-LINE INTERACTION MODEL FOR IMAGE SLICE-BASED EXAMINATION AND REGION OF INTEREST DELINEATION OF 3D IMAGE DATA

Yueh-Shan Shih

MSc, Computer Science, Ryerson University, 2012

ABSTRACT

This thesis explores the effectiveness of a novel interaction model for visualizing 3D image data. The interaction model is based on user-sketched line segments known as *sketch-lines*. This thesis shows that sketch-lines provide simple, fast and precise interactive image slice positioning and 3D region of interest (ROI) delineation in volume images. These two user interactions form the basis of many image visualization tasks, including image slice-based exploration and inspection, cutaway, surgical planning, and model-based segmentation. The sketch-line model is combined with the use of a subdivision surface as well as a new image slice rotation model to support the implementation of the user interactions. To evaluate the effectiveness of the sketch-line model, this thesis measures its performance and compares it to other interaction models with the aid of two user studies. Several experiments applying the model to various 3D medical images are also performed to demonstrate its functionality and consistency.

ACKNOWLEDGEMENTS

First of all, I would like to thank my supervisor Dr. Tim McInerney for helping me with this thesis. I want to thank him from the bottom of my heart for his patience and kindness. It would not have been possible to develop this thesis without his great help and excellent guidance. I would like to thank him not only for his proficient lead but also for his great attitude. It was a pleasure to work with him.

I would also like to thank the committee members, Dr. Jelena Misic, Dr. David Mason and Sophie Quigley for their ideas, suggestions and feedback. Their opinions have greatly improved the section on future work.

Last but not least, a very sincere thank to my parents and my family for their endless love and support. They are always there for me no matter what I have been through. Their encouragement and words are what make me strong. I could never thank them enough with words.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter I – Introduction

This thesis explores the effectiveness of an interaction model for visualizing and analyzing 3D image data. The model is based on the use of user-sketched line segments known as *sketch-lines*, which can be quickly and precisely drawn, both in 2D on image slices and in 3D on the surface of volume rendered objects, and then used to position image slices or delineate 3D regions of interest. The effective and efficient visualization and analysis of objects and object spatial interrelationships in 3D data, such as volume images, remains a fundamental goal in the scientific, industrial, and medical fields. In medicine, examining, exploring, navigating, delineating, cutting away regions of, highlighting, measuring, and segmenting 3D images such as MRI or CT scans are a few of the many tasks that radiologists, technicians, and surgeons must rapidly perform, given the huge number and size of data sets generated today.

Efficiently and intuitively generating useful views of objects in the 3D images, as well as delineating and segmenting 3D regions of interest for subsequent processing and analysis, requires the use of human-computer interaction models that are similar to familiar physical actions such as cutting, sketching, and sliding [27]. This direct human-computer interaction model is now well known and is important today due to the huge increase in touch and pen enabled screens and tablets. Furthermore, many view generation and region delineation tasks (for example, for surgical planning or for segmenting objects in noisy images) not only require efficiency and simplicity, but also precise control and accuracy. For this reason, image slice plane views of the 3D data, where all of the data on the projection plane is visible, are important for planning and analysis tasks [26].

In medicine, volume images can be thought of as a stack of 2D image slices (Figure 1) with the anatomical structures “buried” inside. Traditionally medical visualization packages provide the radiologists and surgeons with one or more 2D windows displaying image slices, along with a slider or other controls to rapidly scan through the slices (Figure 2).

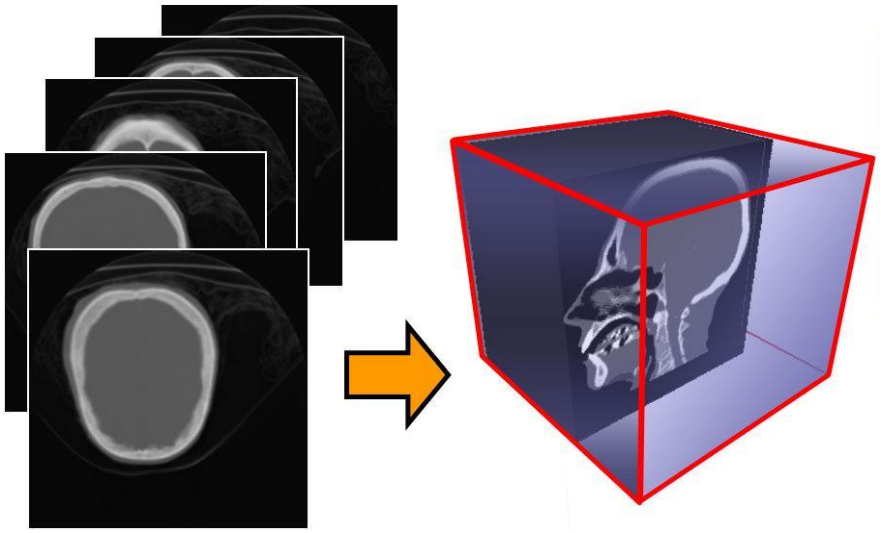

Figure 1 –Example of stack of 2D (CT) image slices [46].

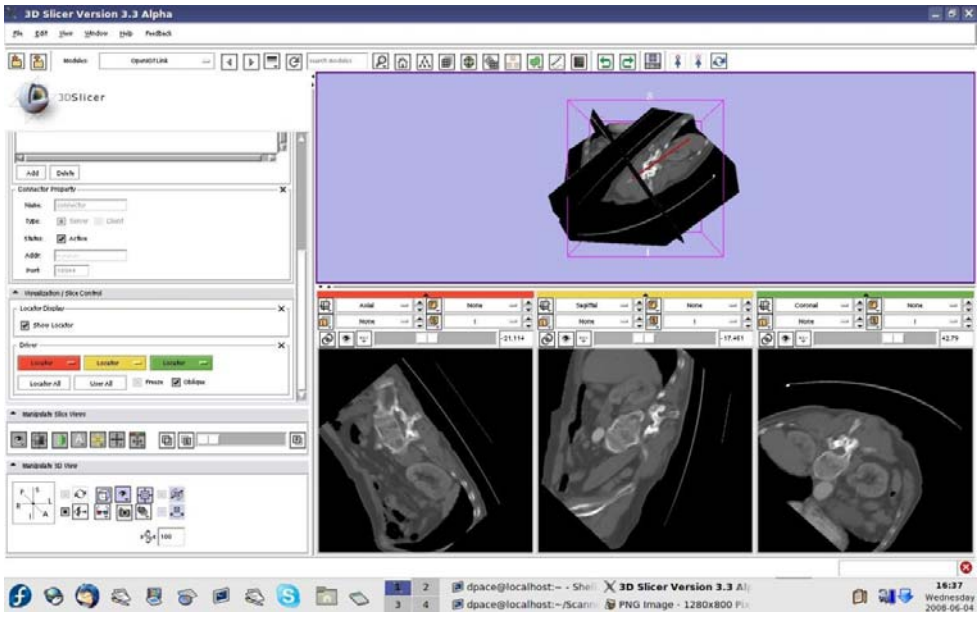

Figure 2 – Multiple 2D image slice views (bottom) and a 3D window (top) in 3DSlicer [13].

In recent years, a 3D display of the data, often in a separate window, is also provided using volume rendering and surface rendering techniques to draw anatomical structures. The 3D window also often shows rotated image slices (Figure 2 top).

That is, it is often useful to orient the image slices relative to the primary medial axis[1] of an anatomical structure in order to examine and measure cross sections of the structure [41]. Positioning of the image slice widget using standard *volume-relative* input actions (i.e. relative to the 3D scene usually defined by a bounding box that surrounds the volume image) can be tedious and time-consuming [19]. *Object-relative* positioning (i.e. relative to the surface of anatomical structures), on the other hand, is often more efficient and intuitive [1].

1.1 Contributions of this Thesis

This thesis explores the effectiveness of an interaction model that is based on the use of user-defined 2D and 3D line segments (and curve segments in 3D), known as *sketch-lines*, to perform object -relative image slice positioning and orientation as well as region of interest (ROI) delineation [29] (in both 2D and 3D). Specifically, the contributions are:

1) The extension and adaptation of the 2D sketch-line technique, first proposed in [1] [40], to 3D to enable fast and precise object-relative positioning [23] of image slice planes. Slices can be quickly positioned and oriented so that they are approximately orthogonal to the target object surface in any desired direction and can also be smoothly dragged along the curving surface of a data object.

[1] The primary medial axis of a closed three dimensional object such as the liver or kidney is, simply put, a space curve that defines the curving central axis of the object.

A series of quick 3D sketch-lines can be used to create a smooth spline curve that is on, or very close to, the object surface. This spline curve can be used to automatically constrain the slice position and orientation, allowing it to slide along the surface and avoid too much user input.

2) The development of a new image slice widget rotation technique that attempts to provide precise control of rotation while also minimizing input device movement and rotation interaction time.

3) The extension and development of the sketch-line technique for use in 3D ROI delineation. The sketch lines are combined with a subdivision surface to quickly construct an editable geometric model of the region, which can then be used for cutaway operations to aid in visualization of hidden objects, or as input to geometric model-based segmentation algorithms [1].

4) The conducting of two user studies to quantitatively measure the efficiency of the sketch-line interaction technique compared to several other well known interaction models, as well as to qualitatively assess the effectiveness of the interaction model based on user feedback.

1.2 Thesis Outline

Chapter 2 begins with a brief explanation of volume rendering and surface rendering as an aid in understanding the thesis contributions. It then presents a review of several data visualization and object modelling applications that provide 3D object positioning and rotation techniques.

Specifically, many data visualization packages provide 3D image exploration using 3D image slice widgets. The second major section reviews 3D region delineation, region clipping and cutaway techniques. A brief summary of interactive image segmentation techniques is also presented because segmentation is directly related to region delineation and region marking.

Chapter 3 describes the algorithms used to implement the sketch-line based exploration tool. This includes descriptions of sketch-line construction, subsequent image slice construction and orientation control, and the use of sketch-lines combined with a subdivision surface to define regions of interest known as *envelopes*.

Chapter 4 presents the results of several experiments demonstrating the use of sketch-lines for basic image slice positioning, as well as for data object surface-constrained image slice positioning. Experiments demonstrating the use of sketch-lines to create 3D regions of interest for cutaway and segmentation are also presented. Finally, the results of two user studies are presented that evaluate the effectiveness and efficiency of sketch-lines. The studies include both quantitative and qualitative information.

Chapter 5 concludes and summarizes the thesis and presents directions for future work as well as suggestions to improve the sketch-line model.

Chapter II – Literature Survey

This chapter is divided into three sections. The first section provides a brief explanation of volume rendering and surface rendering. The next section presents an overview of existing 3D data manipulation, exploration and navigation techniques, many of which can be found in commercial or academic medical visualization packages. Since many volume image exploration techniques often have 2D and 3D image slice widgets, control of image slice positioning [60] and slice rotation is an important part of volume image exploration (i.e. viewing cross-sections of the volume image from different positions and orientations) and navigation (controlling the view as the user "travels" through the volume image) . For this reason, a review of 3D object positioning and rotation techniques is also included. The final section presents a review of 3D region delineation, region clipping and cutaway techniques. A brief review of interactive image segmentation techniques is also presented because segmentation is directly related to region delineation.

2.1 Volume Rendering and Surface Rendering

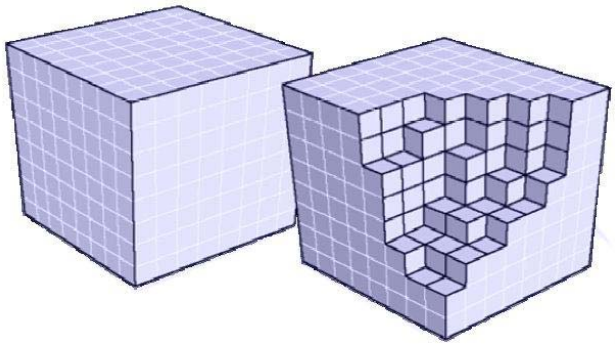

Figure 3 – Example of a voxel grid [46].

A brief introduction to volume rendering (VR) [11] [66] [39] and surface rendering (SR) [37] will help explain and show how a 3D object inside a volume image is rendered by computer [53]. A volumetric dataset is an array of scalar values that have been obtained by means of a scanning process (e.g. Magnetic Resonance Imaging (MRI), Computer Tomography (CT), Positron Emission Tomography (PET), etc.). The scalar values represent the intensity of a signal (e.g. x-rays for CT images) in different parts of the body region being scanned. The scanning is performed in a slice-by-slice fashion, with the final volume being assembled by stacking up the 2D slices into a regular grid of volumetric elements or *voxels* (Figure 3).

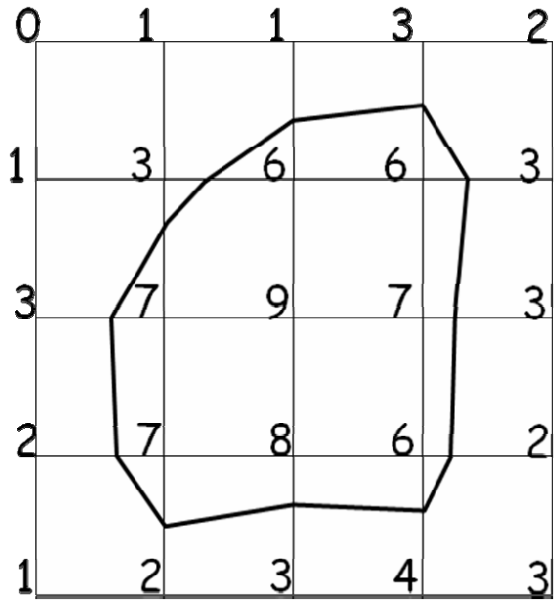

Figure 4 – A 2D illustrative example of finding an iso-contour of value = 5 in a 2D "voxel" grid. The numbers represent voxel intensity values. The algorithm "marches" through each square (cube in 3D) finding squares that contain voxel intensity values at the 4 corners that are higher and lower than the given value of 5. Interpolation is used along with a lookup table to construct line segments (triangles in 3D) within each of these squares [46].

An object rendered by surface rendering is divided into a mesh of triangles using algorithms such as the well-known marching cubes algorithm [14]. This algorithm "marches" through each cube in the voxel grid looking for voxel intensity values at the 8 corner points of the cube that are higher and lower than the threshold intensity value given by the user. If a cube contains both higher and lower values, the object surface must pass through this cube (Figure 4).

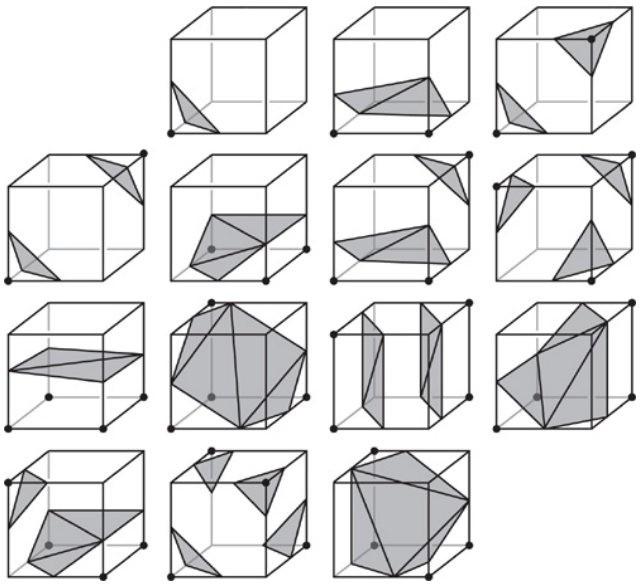

Figure 5 – A lookup table used to construct triangles within each voxel of a voxel grid that contain intensity values higher and lower than a given value. The combined triangles form an isosurface [46].

In this case, the exact position of the given intensity value along the edges of the cube are determined using linear interpolation. Triangles are constructed using a pre-defined lookup table (Figure 5) such that the intensity values in the cube higher than the given threshold value are separated from lower intensity values by the constructed triangles. The collection of all triangles represents the iso-surface. The graphics cards on modern computers have evolved to optimize the rendering of triangle meshes and today several million triangles can be rendered per second.

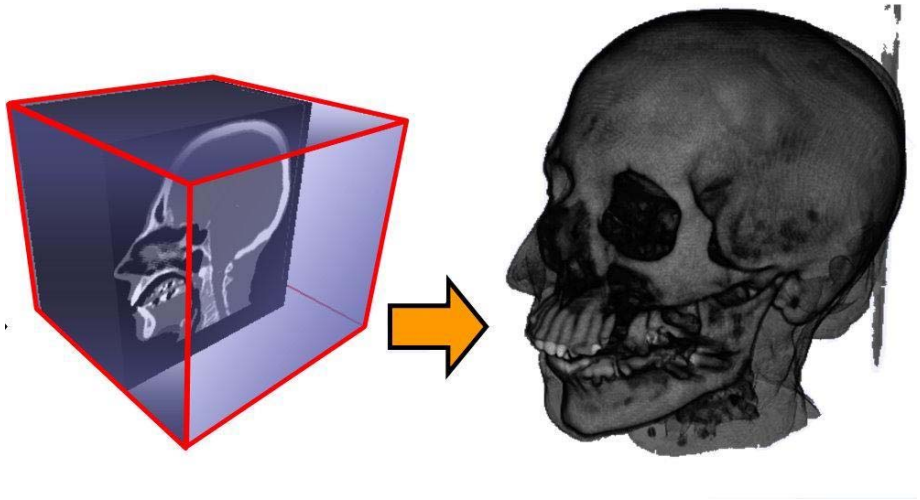

Figure 6 –Example of a volume rendered head CT scan where the transfer function has been set to render opaque bones and translucent skin [46].

Volume rendering (Figure 6) is the now the most common way to render objects in 3D from volume images. There are several volume rendering algorithms but with today's powerful graphics cards, ray casting is the most common algorithm as it is very easy to parallelize. Rays are *cast* into the volume from the current viewing point through each pixel on the screen window (Figure 7) and as voxels of the dataset are encountered by each ray, they are sampled. The sampled values are converted into RGBA (i.e. Red, Green, Blue, Alpha) values using a transfer function which is set by the user. The transfer function specifies the mapping of voxel intensity values to colors and opacities (i.e. 1 – transparency (Alpha) value). The sampled values are accumulated along each ray and blended to form the final color and opacity value at each pixel on the screen.

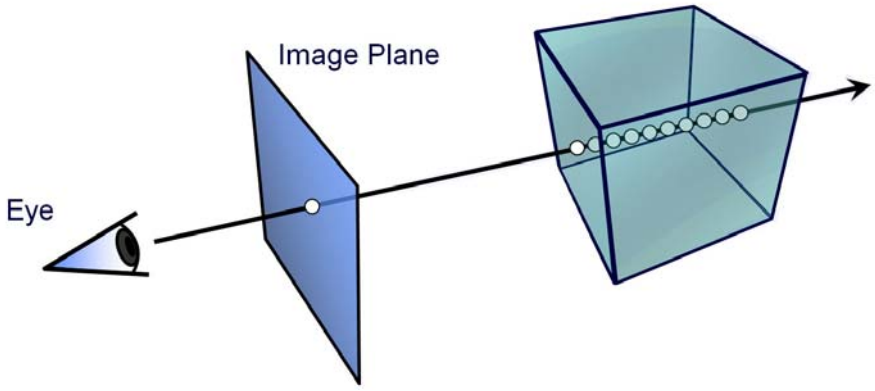

Figure 7 – Illustration of the ray casting algorithm [46].

2.2 3D Data Manipulation, Exploration, and Navigation

There are many existing data visualization and object modelling software applications and systems, both commercial and non-commercial. Some example systems are VTK [33], ParaView [32], Amira [65], Maya [3], 3D Studio Max [4] and Blender [8].

VTK (Visualization Toolkit) is an open source C++ class library designed for 3D graphics and visualization application development, mostly using the C++ programming language. VTK contains many classes which can be used to visualize and explore 3D image data, including volume and surface rendering as well as image slice widgets. Figure 8 is an example VTK application showing volume rendering of the skull from a CT scan along with several image slices. VTK was used as the base software platform in this thesis.

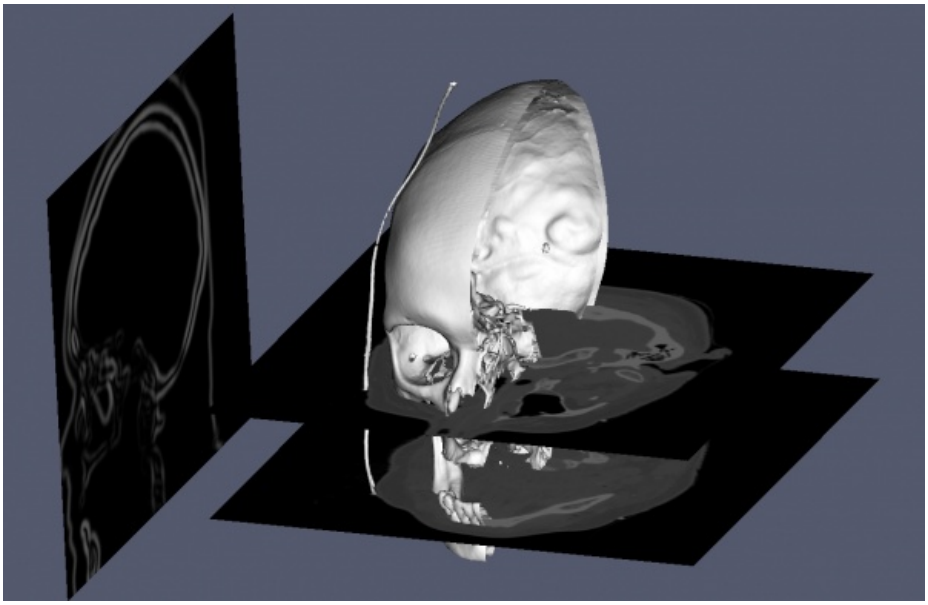

Figure 8 – VTK application demonstrating volume rendering combined with image slice display [33].

ParaView is an open source data visualization and analysis application build on top of VTK. It is primarily designed for scientific visualization and provides 3D interaction using a well-defined GUI (Graphical User Interface) along with several widgets for viewing and processing cross-sections of the data. Figure 9 shows an example of the ParaView interface.

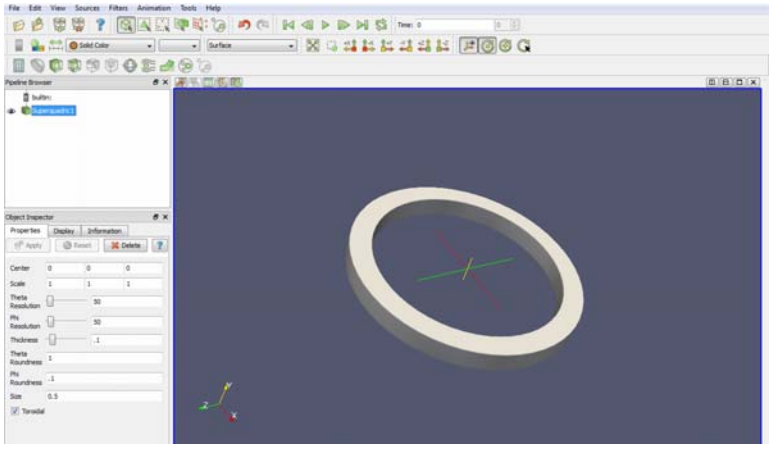

Figure 9 – ParaView interface with a torus object rendered in 3D [32].

Amira is designed for visualizing, manipulating and exploring scientific and medical datasets. It allows users to create, simulate, explore and analyze objects and images. It supports data slicing, volume rendering, and surface rendering. It also supports 3D image exploration and navigation using oriented image slices controlled using a widget (Figure 10).

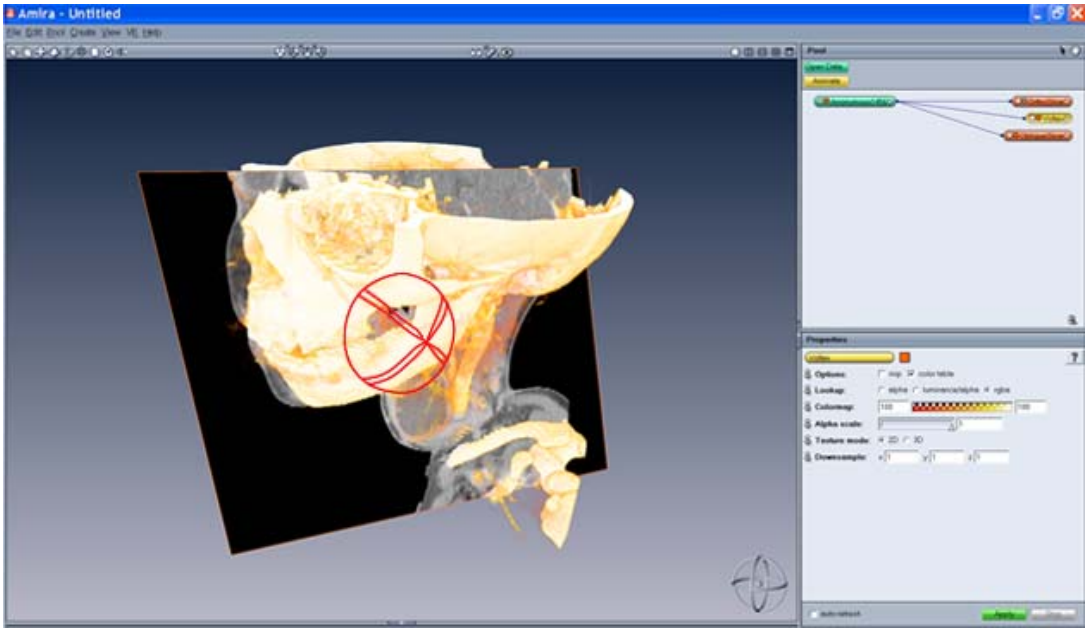

Figure 10 – Top: screenshot of Amira showing a 3D rendering of a CT scan of a head along with an image slice. The image slice can be oriented with the widget rotation handles shown in red [65].

Maya [3], 3D Studio Max, XSI, and Blender are all examples of 3D computer graphics packages designed for 3D object animation, modeling, simulation, and rendering. They are mostly used in the movie special effects and video games industries. An example interface of polygon mesh modeling in Maya is shown in Figure 11.

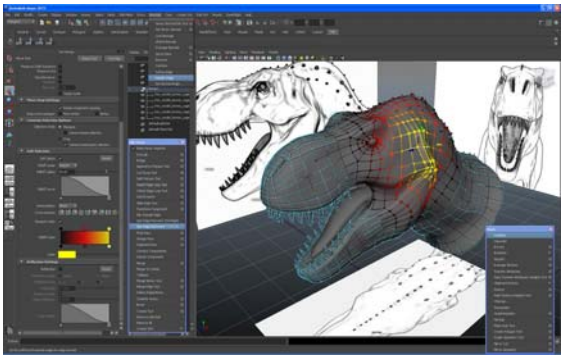

Figure 11 – Screenshot of Maya's interface showing an example of polygon mesh modeling [3].

All of these programs provide rendering and object manipulation techniques [47] as part of their core functionality [38] and all support object positioning technique [60] using 3D widgets. In Figure 12, a depiction of the 3D widget handles commonly used by the modelling packages is shown. Typically the widgets show visual representations of individual axes and circles that can be "grabbed" by the user to perform scene navigation and object manipulation operations such as translation, scale, and rotation, thus allowing the user to create a desired view.

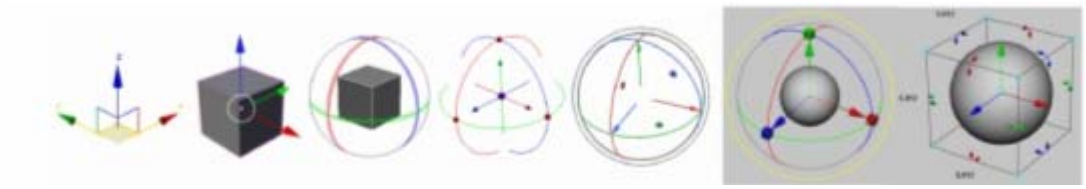

Figure 12 – Various visual representations of 3D widgets provided by various modelling packages that are used to rotate and translate objects. From left to right: 3DS Max, Blender, XSI, Houdini, Modo and Maya (gray-shaded region), respectively [54].

here are also many visualization application development systems (Figure 13) that are focussed on medical data, for example MeVisLab [43], ImageVis3D [61], Voreen [64] and 3DSlicer [13]. These systems support volume rendering (using GPU accelerated ray casting) and surface rendering, image processing, and interaction widgets.

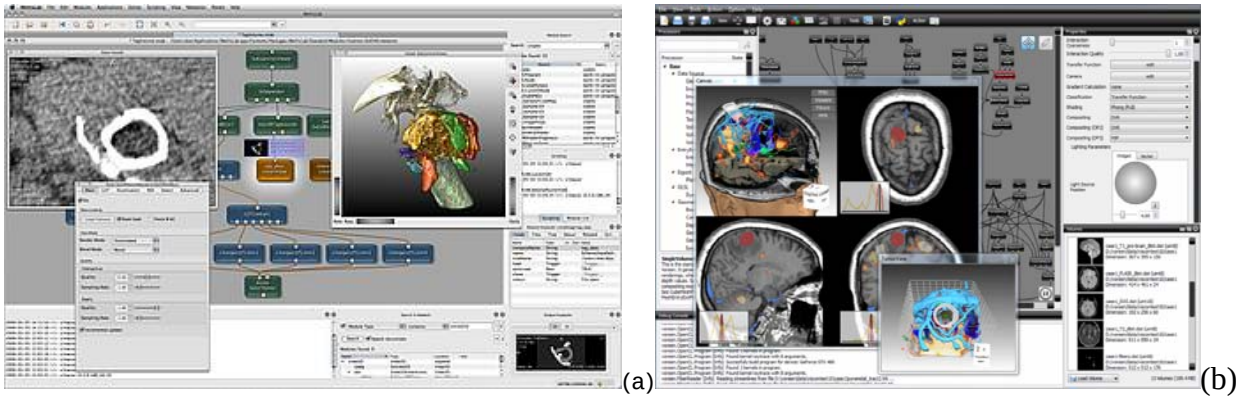

Figure 13 – Screenshots of (a) MeVisLab development environment [43], (b) Voreen's user interface [64].

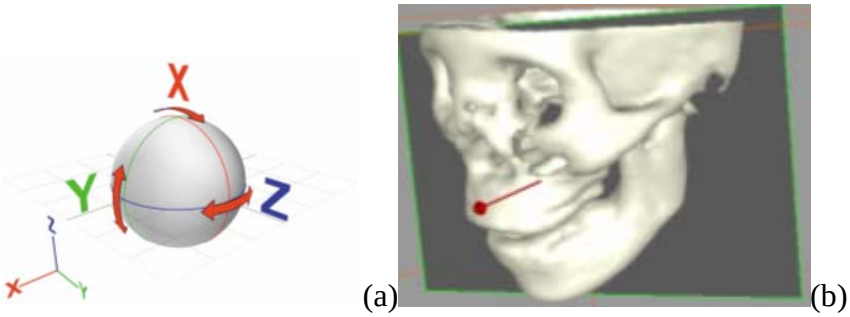

Figure 14 – Example rotation techniques: (a) virtual trackball/arc ball rotation technique with axes labelled [71]. (b) VTK virtual trackball style rotation (controlled via the red arrow handle) used to control image slice orientation.

All of the visualization platforms provide the ability to rotate the entire scene and most provide widgets to translate and rotate image slice planes into arbitrary 3D orientations. The standard and familiar arc ball interaction model [18] (Figure 14(a)) is commonly used both for scene and image slice rotation, where the scene/object is surrounded by a virtual sphere and mouse movements over the surface of this sphere are mapped to rotations.

This technique is fairly intuitive and easy to use but can have side effects [18]. One disadvantage is that the user can orient the 3D object in such way that is difficult to return to a previous state.

Using this technique to rotate an image slice (Figure 14(b)) may be too unconstrained in nature, especially when the image slice is combined with a volume rendering of a 3D anatomical structure - a useful technique for exploring certain slice views of the 3D volume image [59] - because it can be difficult to control the image slice position with respect to the 3D volume rendered object. In addition, the rotation handles may be hidden by the volume rendering. For this reason, many rotation models in medical visualization programs only allow the 3D image slice to rotate around single rotation axis at a time. By separating and limiting the range of the rotational *degrees of freedom*, users gain more precise control of rotation. This precise control is also desirable in the modelling applications [54] mentioned before. On the other hand, more user interaction can be needed to rotate an object into some orientations. In chapter 4, the results of a user study are presented comparing the arc ball style rotation against two constrained rotation techniques.

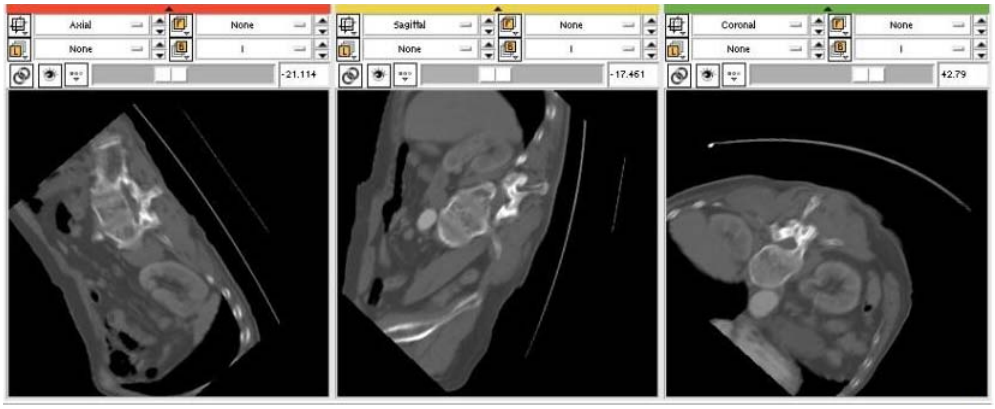

Figure 15 – Multiple 2D image slice views in 3DSlicer [13].

Most of the visualization applications display standard orthogonal view image slices (i.e. XY, YZ, XZ planes) in multiple windows (Figure 15) and allow the user to use a slider to quickly scan through (and explore) the image slices. Image slices are used a lot for detailed analysis and measurement of objects. Rotated image slices are also often displayed along with a volume rendered data object in the same window, along with a cross-sectional view in a separate window (Figure 16), to allow the user to explore the data using slices positioned with respect to the 3D rendered object [6]. Multiple 3D image slices can be used as well but as each image plane widget is often independent from the others, this can cause excessive interactions and visual clutter. Translating and rotating the image slice in the 3D world space with respect to the surface of the volume rendered object can also be a lot of work for the user.

Figure 16 – (Left) image slice positioned orthogonally with respect to the jaw surface and (right) corresponding 2D cross section view of the volume.

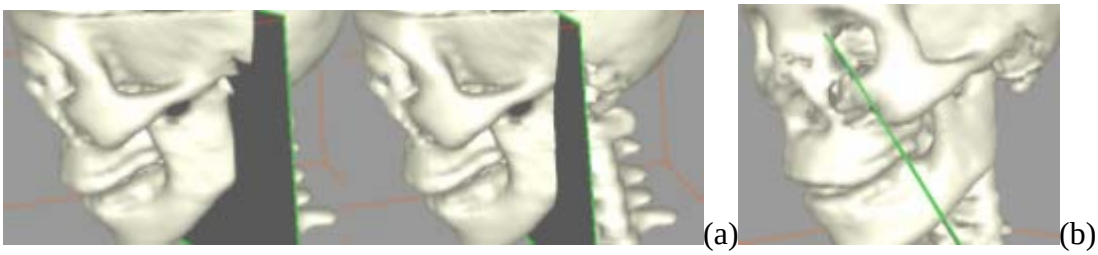

Figure 17 – (a) Object surface image slice positioning using a series of independent input actions. The user is forced to translate the image slice in world-space coordinates close to the new desired position on the jaw and then (b) rotate the slice, often in two directions, such that it is orthogonal to the jaw surface.

Typically, in most of the packages and systems, image slice widgets provide the ability to move the slice plane in a direction along its normal vector as in Figure 17(a). However, typically links between the slice plane manipulation and the 3D object do not exist. Orienting and positioning are performed only in "absolute" 3D world coordinates. If the user wants to translate and orient the slice plane so that it is orthogonal to the object surface (i.e. *relative* to the surface), such as the lower jaw region in Figure 17(b), they must perform a series of widget manipulations. They cannot specify the position and orientation in a direct way.

Furthermore, if the user desires to *slide* the image slice plane along the surface of the jaw such that it stays orthogonal to the surface (so they can examine relevant jaw cross sections in the associated 2D window), then a series of world-space translations followed by rotations to align the image slice will have to be performed. That is, they will have to perform multiple interactions independently, an obvious inefficiency. In short, the common problem is that slice-plane widget controllers do not allow object surface alignment, as the hinge-line based idea [39] attempted to provide.

2.3 Region Delineation, Cutaway and Segmentation

Selecting or surrounding arbitrarily shaped regions of interest in volumetric datasets in a simple and intuitive way and then representing the region with a geometric model, is a complex task. The geometric model representation can be a triangle mesh, a set of voxels, or an implicitly defined function such as the well-known Metaballs [9] technique.

Often the goal is to surround a 2D or 3D region of interest to form a closed contour (2D) or closed surface (3D) called an *envelope*[2]. An envelope can be used in cutaway operations to remove unwanted data in order to visualize hidden objects (Figure 18). In addition, an envelope can be used as input to a model-based segmentation algorithm which attempts to correctly label all the data voxels inside the envelope as belonging to a particular organ, such as the liver, or other anatomical structure.

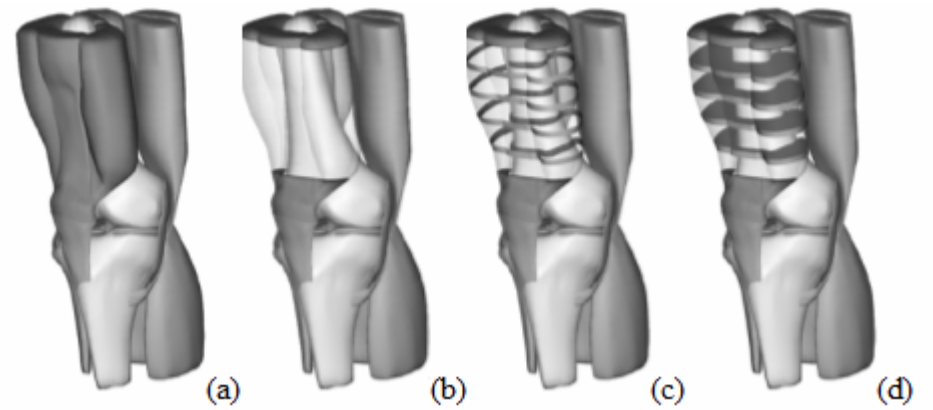

Figure 18 – (a) basic rendering without cutaway in knee dataset. (b) Muscle region selected and rendered as transparent. (c) Muscle region cutaway excepts for a series of "ribbons" or (d) solid "slices" [42].

Most visualization packages support the selection of simple rectangular box-shaped regions [21] for ROI selection. Standard tools such as cropping boxes, controlled using widgets, are often GPU accelerated and can be used to select the box-shaped regions in real time [20]. Transfer functions, mentioned in the first section of this chapter, can be set by the user to select voxels with a similar intensity value. However, the results are often unpredictable, especially in noisy volume images, and are not spatially localized.

[2] Other region delineation tasks require only a surface "patch" be selected.

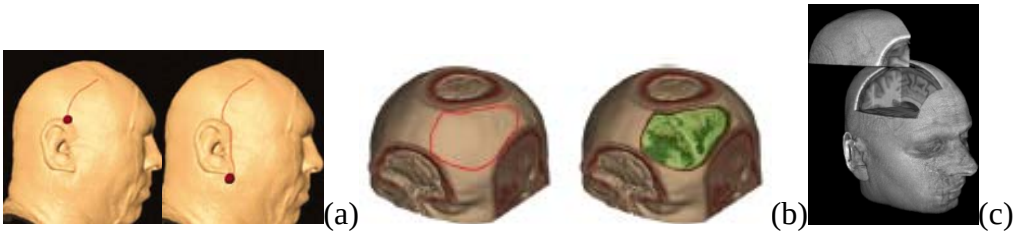

Figure 19 – Examples of (a) tracing[12] (b) painting[15] and (c) sculpting [67] for 3D ROI delination.

Most selection techniques [48] use some sort of interaction metaphor. Sketching [2] [51], tracing[12] [69], painting [15] [42], and sculpting [67] (Figure 19) are among the most common of these metaphors. Interactive segmentation techniques can be considered semi-automatic region selection and are described in Section 2.3.1. In the sketching metaphor, the user draws (using a mouse or pen or finger) lines or contours on the screen or directly on the object surface. These contours are then connected (extruded) or "inflated" to form a 3D envelope [2] [51]. The sculpting metaphor simulates cutting tools, often with a tool "tip" in the shape of a convex object such as a sphere or cube. The tool tip is positioned and/or moved along the data object surface and any voxels inside the tool are selected and sculpted away. Painting defines a surface/volumetric region, such as a circle or sphere, as the "brush" and as the "brush" is moved along the data object surface, the circles/spheres are combined to form the "paint" [15] [42]. Again, voxels inside the "painted" region are selected. Tracing [69] is similar to painting in that the user moves the cursor along the data object surface but draws a contour that outlines the region of interest. Tracing can be performed directly in 3D or in 2D on a series of parallel image slices and the resulting contours connected together to form a 3D ROI. Finally, some 3D region selection techniques make use of 3D widgets that define one or more planes [39]. The planes are often connected along a "hinge" line, defining, for example, a "spreader" tool. The widgets can be interactively "grabbed" and positioned to select an ROI.

All of these region selection techniques must somehow define the depth of penetration into the data object. This is a difficult problem common to all 3D region selection techniques, especially when using a 2D input device such as a mouse or touch screen. The depth of the 3D region selection is often controlled using a separate button or key [42] or gesture in the case of touch screens. Input devices that provide more degrees of freedom (Figure 20) can be used to move around the 3D world space, for example force feedback devices [55][56], but these devices are not precisely controllable, especially in noisy images, and are tedious and tiring to use due to shoulder muscle strain. They are more commonly used in surgical simulations [30] [44] [31] [10].

Figure 20 – Sensable Haptics device Phantom Omni [55].

The main uses of 3D ROI selection is to cut or clip away unwanted or occluding parts of the data object to reveal hidden structures, and for surgical planning or surgical simulation [67][28][70][34][35][58][39][68][25][17] (Figure 18). Figure 21 is an example of a virtual resection (selection followed by cut away) in 2D and 3D of a portion of the liver [34].

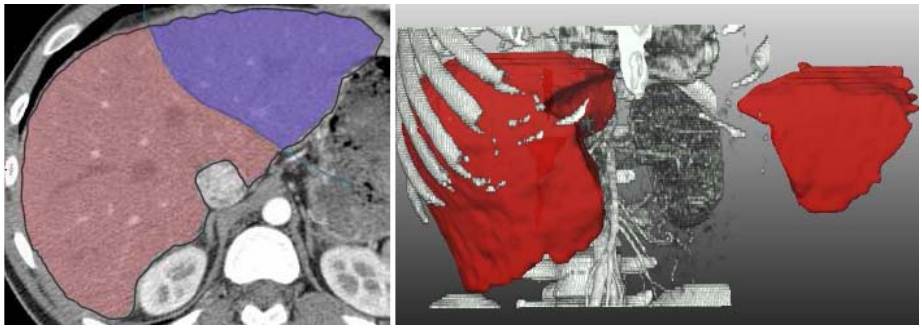

Figure 21 – Virtual liver selection and resection in 2D and 3D [34].

As mentioned above, many of the 3D selection techniques [63] are designed for removing occluding structures to view the interior structures. Ideally, the interaction models should be close to real actions, such as cutting into, cutting open, spreading apart and peeling actions. The use of these familiar actions allows the expert users to focus on the specific visualization task rather than on the cutting tool interface. However, combining all the features necessary to quickly, intuitively and precisely define a 3D complex-shaped ROI, such as the ability to edit the selected region, undo selection actions, control the depth of penetration of the selection, is still a challenge.

2.3.1 Interactive Segmentation

Segmentation can be defined as classifying or labeling data voxels as to the particular anatomical structure that they belong to. It can be thought of as a more automatic method of region delineation that attempts to delineate an entire object. That is, the user interactively selects voxels inside the object and the algorithm attempts to label all the voxels belonging to the object.

As most volume images are very noisy, segmentation is a challenging task, especially if the goal is completely automatic segmentation. In this case, the use of artificial intelligence techniques is often required.

There are many interactive semi-automatic segmentation techniques. Some of the more well-known algorithms are region growing [15] [16] [57], Random Walker [24], GraphCut [36], Level-sets [49], and deformable surface models [1]. In most of these algorithms, the user selects one or more *seed* voxels, using an input device, inside the anatomical structure of interest. Some of the algorithms require the user to draw several “brush strokes” on the target structure.

This type of information is used to transfer knowledge of the voxel intensity values (for example, the mean intensity, standard deviation etc.) of the target structure voxels to the algorithms, which then look at voxels neighboring the strokes or seeds and attempts to classify them as belonging to the target structure or not. This neighbor-voxel classification scheme continues until there are no more voxels that are part of the target structure. A more detailed explanation of these complex algorithms is beyond the scope of this thesis.

The main disadvantage of many of these algorithms is that they can “leak” or spread into neighboring objects (especially in noisy images) or also not grow or spread into the entire target object. Some of these algorithms, on the other hand, such as the Level-Set technique [49] and the deformable model technique [1], can also make use of a user-defined envelope that surrounds the target object and use this envelope to constrain the segmentation. This constraint helps to prevent leaks into the neighboring structures and helps to make sure that all the voxels of the target object are correctly labeled. This user defined envelope constraining mechanism is one of the contributions of this thesis.

Chapter III – Methodology and Implementation

Interacting with, visualizing and analyzing medical images are extremely important radiological and surgical planning tasks and many software packages [43] [61] [64] [13], research tools and algorithms have been developed over the past decade to support them. Radiologists and Medical technicians are very familiar with image slice planes. Viewing standard slice planes – coronal (XZ), sagittal (YZ), and axial (XY), is still one of the most common methods to navigate, explore and inspect volume images. Volume rendering is also common and is a powerful visualization tool, especially for high contrast CT volumes and segmented volumes. One of the main problems with many visualization packages is the amount of functionality, complex interfaces and different interaction models that are provided. This thesis explores the use of a *single* sketch-line based interaction model, both for fast and simple image slice positioning and for region of interest delineation, that enables visualization, inspection, cutaway and segmentation. This chapter will describe the use and implementation of the sketch-line technique to achieve these two operations. The sketch-line interaction model presented in this thesis is an extension of the 2D sketch-line model first proposed in [40] for use in image segmentation. The sketch-line model attempts to save time and effort by avoiding unnecessary manipulations of 3D objects. Only simple strokes directly on the volume rendered object surface or on an image slice are needed. Sketching lines is a simple and familiar action for users and the lines can be quickly and precisely drawn.

Enough information can be extracted from the sketch-line so that an image slice can be constructed relative to an object surface rendered in the 3D window, allowing cross sections of the object to be examined in a separate window.

In addition, following [1], by combining the sketch-lines with a subdivision surface and a sketch-line connection (extrusion) process, envelopes can be constructed that delineate complex-shaped regions of interest in an accurate way, both in 2D and 3D. These envelopes can then be used as a deformable model [1] and can be "snapped" [36] onto the ROI surface to segment all or part of a target data object.

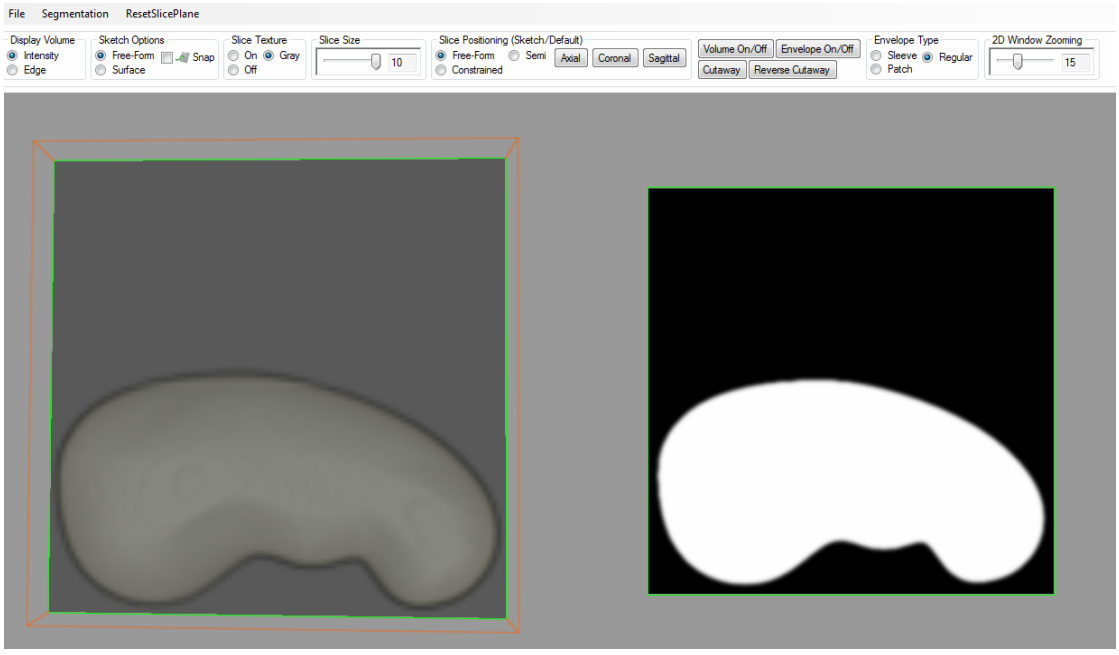

Figure 22 – Sketch-line system interface showing the 3D view (left) and the 2D cross-sectional view (right) of a segmented kidney dataset. The 3D view window contains a volume rendering of the data as well as an image slice plane (dark gray with green outline) that can be translated and rotated in 3D. The 2D view is automatically updated when the image slice plane is changed.

A combination of a 3D volume rendering view window and a 2D image slice window (Figure 22) is used to allow for navigation and exploration of a volume image [62]. Sketch-lines can be drawn in the same way both on the 3D object surface in the 3D view window and on the image slice in the 2D view window, resulting in a consistent interaction model. Simple 2D input devices, such as a mouse or pen and tablet, are all you need to draw the sketch-lines – there is no need for expensive and special 3D input devices. Furthermore, the result of the sketch-line input actions, either the construction and subsequent positioning of an image slice or the creation of an envelope, can be edited using 2D and 3D control “handles”.

3.1 The Sketch-line System Interface

The sketch-line system is built with VTK 5.6.0, Visual Studio 2010 and the C++ programming language under the Windows 7 environment. The sketch-line system GUI contains buttons, check boxes, radio buttons, sliders, and menus to activate various options (Figure 22). The sketch-line system interface is designed with two window views: the left window shows a 3D view of the volume image and the right window shows a 2D image slice view. The 3D view, called the 3D window, contains a volume rendering of the dataset, an image slice plane (known as the 3D slice plane) that can be positioned, scaled and rotated using a widget, and the outline of the volume image bounding box. The 2D window’s camera and slice plane are linked to the 3D slice plane’s orientation. Rotation transformations that are applied to the 3D slice to orient it are also applied to the 2D slice plane and 2D window camera so that a consistent “up” direction is maintained.

3.2 Sketch-Lines in 2D

The goal of a sketch-line is to maximize the amount of information (position, orientation, width, surface curvature, surface normal etc.) from the user to the algorithm with the least amount of effort. Sketch-lines are formed by positioning the mouse (on the surface of the object or on the 2D image slice) and pressing the left mouse button to establish the first endpoint of the line segment. The user then drags the mouse (while holding the left mouse button) to another location. In the 2D window (i.e. for a 2D sketch-line), as the user drags the mouse a line will be drawn from the first endpoint to the second endpoint defined by the current mouse position. The sketch-line is terminated by releasing the left mouse button. The process of sketching a 2D sketch-line is shown in Figure 23 (a) and (b).

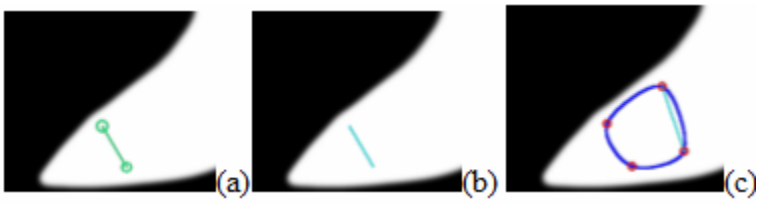

Figure 23 –2D Sketch-line: (a) Click and drag in 2D window. (b) A sketch-line is formed by click, drag and release in the 2D window. (c) A contour and control points are formed when two or more sketch lines are drawn.

A 2D sketch-line provides position, orientation and object width information. Sketch-lines can be connected together to form a closed contour (Figure 23 (c)). To maximize the amount of user information needed to create a contour envelope that accurately delineates an object cross-section, sketch-lines should ideally be drawn across the object cross-section so that they are approximately orthogonal to the primary medial axis of the object cross section.

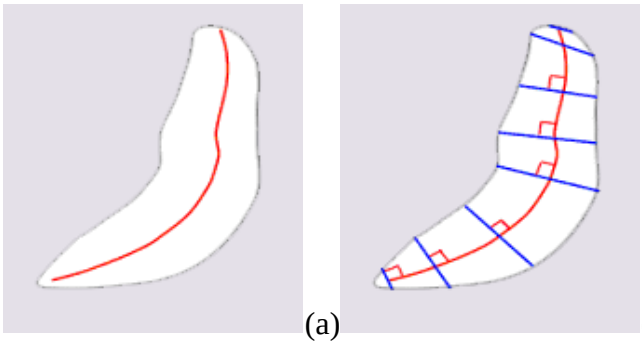

Figure 24 – Delineating an object cross section using 2D sketch lines: (a) the red curve is the approximate medial axis of the object cross section. (b) For optimal results, the user draws sketch-lines across the object approximately orthogonal to the medial axis.

Figure 24 and 25 illustrate this idea. In Figure 24(a), the red curve represents the approximate primary medial axis of this liver cross-section. The medial axis is the "center line" of the object; lines drawn across the object that intersect the medial axis at right angles will intersect two points on the object surface – one on either side of the axis. The distance from each of these two points to the medial axis is equal. In Figure 24(b), the user has sketched 8 line segments (blue) that roughly intersect the medial axis at right angles. The endpoints of the sketch-lines can then be connected and interpolated using a spline curve. As is clear in Figure 25, the resulting spline curve forms a closed contour that accurately delineates the object cross-section.

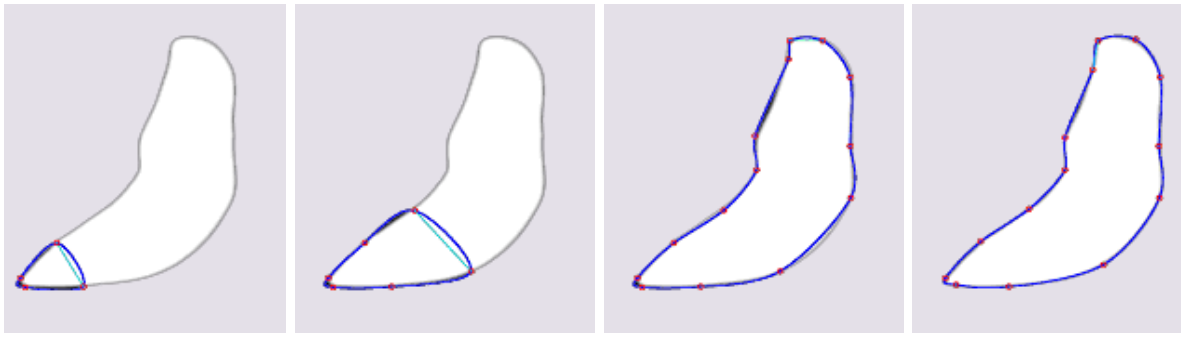

Figure 25 – Result of sketching 7 lines on an object cross section to form a closed contour. The spline control points (red) can be used to edit the spline shape.

This sketching process has a short learning curve – after some practice, it is often clear where to place the lines and how many to use. As the user adds another sketch-line, the spline curve is updated so the user can see the partial result. Based on experiments [2] a user can sketch each line in less than a second. Typically for a complex curving shape as in Figure 25, 5 to 8 sketch-lines are enough, depending on how much accuracy the user wants and what the 2D spline envelope will be used for.

To avoid local self-intersection of the resulting spline curve, the intersection of 2D sketch-lines is prevented. The light blue line in Figure 25 is called the *active edge*. When a new line is sketched, its endpoints are connected to the active edge endpoints and a new spline contour is created. The user can click on the spline contour in another location to change the active edge. This allows the user to create more complex shapes. Finally, the red dots in Figure 25 indicate the spline contour control points. These points can be selected and dragged to a new position. The smooth spline contour will be continuously updated with the new shape.

3.3 Sketch-lines in 3D

A 3D sketch-line is drawn on the surface of the volume rendered object using a similar left mouse click and drag input as in the 2D case. Because the surface of objects is curving, a 3D sketch-line is visually represented as a spline curve so that it appears to take on the approximate curvature of the object surface (Figure 26(a)). One of the main uses of a 3D sketch-line is to position and orient an image slice plane so that it is approximately orthogonal to the object surface (Figure 26(b) and (c)).

Object surface points and object surface normal vectors are sampled along the 3D sketch-line and this information is used to construct the necessary 3D coordinate system for the image slice. Furthermore, as in the 2D case, multiple sketch-lines can be drawn quickly on the object surface along the length of the region of interest. These 3D spline curves can be connected to form a mesh of triangles (Section 3.7). This "control" mesh is fed into a subdivision surface algorithm [1] which finely subdivides the triangles to form a smooth surface (Section 3.7).

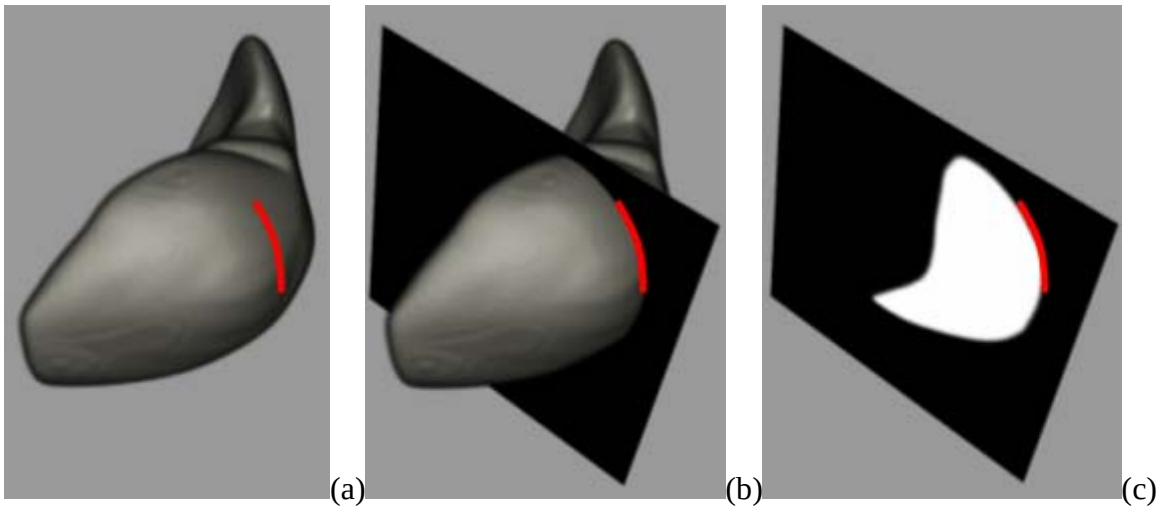

Figure 26 – 3D Sketch-line: (a) Sketching 3D "line" on volume rendering of liver from segmented volume image. (b) Positioning slice plane using 3D sketch-line. (c) Image slice showing how the sketch-line conforms to the liver surface.

This smooth surface delineates and/or surrounds the region (i.e. envelopes it) and can be used to either cut away the region or be converted to a deformable surface model [1] and "snapped" down onto the region to take on its shape (i.e. segment it). In short, 3D sketch-lines are used to analyze, measure, and examine data.

3.3.1 3D Sketch-Line Construction

As mentioned, a spline curve is used to visually represent the 3D sketch-line. Constructing the curve proceeds as follows. The first point picked on the object surface is labelled *point1* (Figure 27). As the user drags the mouse and moves the cursor along a path on the object surface, surface points and surface normals are gathered using a 3D object "picker" class in VTK and are stored into an array (blue points in Figure 27). The current picked point under the cursor is labelled as *point2*. All of the picked surface points are averaged together to form the average pick point (Figure 27 (b)). In addition, all of the surface normals at the picked points are averaged together to form an average surface normal. Then, each picked point is subtracted, in turn, from the average pick point to form a direction vector (Figure 27 (c)).

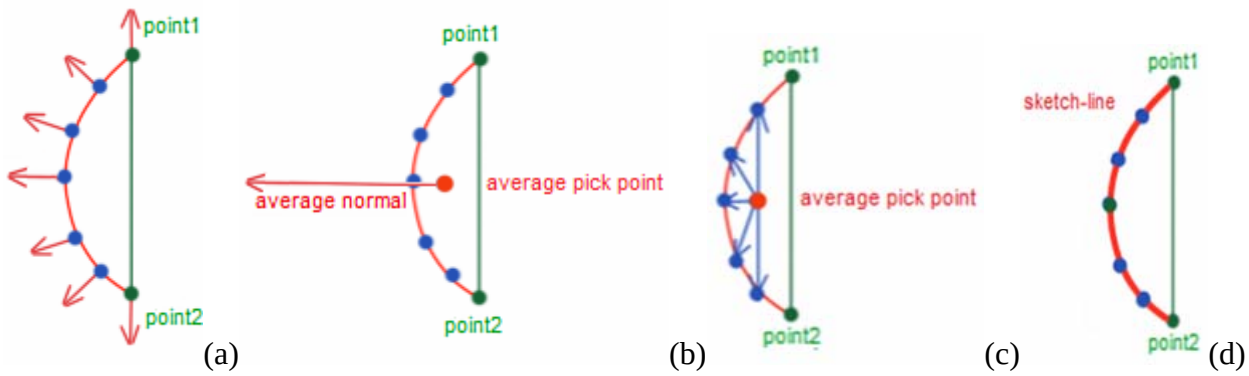

Figure 27 –2D illustration of 3D sketch-line construction. (a) the blue points are the sampled object surface points (and surface normals) "picked" as the cursor slides over them. (b) the red point is the average picked and the average surface normal (scaled for emphasis) is also shown. (c) vectors from the average picked point to each sampled surface point are formed. The vector most parallel to the average surface normal is chosen as the "middle" point of the 3D sketch line. (d): the chosen middle point (green) along with point1 and point2 are used to form the spline curve).

The picked point with a direction vector that is closest in direction to the positive average normal direction is chosen as the midpoint of the spline. That is, the 3 points (point1, point2, and the calculated midpoint) are used as spline control points and a smooth spline curve is constructed and displayed as the 3D sketch-line.

Basically, this 3D sketch-line construction attempts to create a spline curve that is always visible on the surface and conforms approximately to the surface shape. If the 3D sketch-line is drawn over part of the object surface that has an indentation (Figure 28 (e)), the sketch-line will still remain convex and go over the indentation.

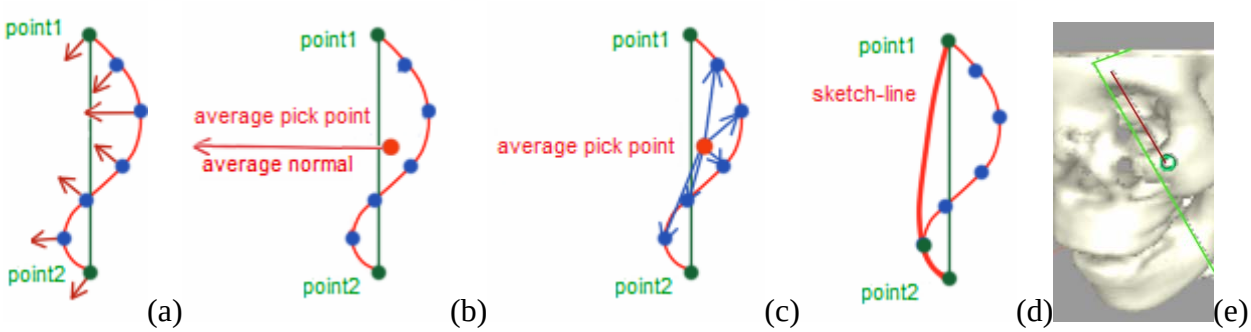

Figure 28 – Finding the middle point of the picked points when the sketched line goes over an indentation in the surface (as in the eye socket of the skull in (e)). (a) – (d) The sketch-line construction algorithm attempts to create a convex spline curve in order to remain visible to the user and also so that an envelope created using this sketch-line covers the surface indentation.

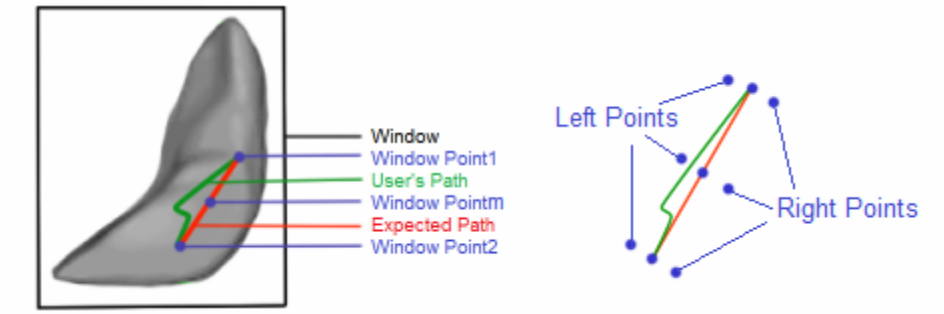

Figure 29 – 3D Sketch-line showing the path formed by the user when freely dragging point2 around (green) and the resulting sketch-line (red) constructed from the surface points sampled by projecting screen points between point1 and point2 onto the surface. The figure on the right shows how the accuracy of the constructed spline could be improved by picking more points to the left and right of the sampled points and averaging.

As the user drags the cursor over the surface a new point2 position is continuously updated, as is the calculated midpoint, and a new spline is rendered in real time. This allows the user to sketch more freely without too much concern about sketching exactly along a straight path over the object surface. However, to allow this sketching freedom, the path formed by user can be different than the expected curve path as shown in Figure 29.

The surface normal vectors of the picked points obtained may be clustered around one small area of the surface and therefore may not be a good sampling of surface normals along the expected path. This may result in an inaccurately oriented image slice plane. To avoid this problem, 2D screen window points corresponding to point1 and point2 are used to form a line in screen coordinate space. Points are sampled along this screen space line and are projected along a ray from the user's viewpoint (the camera position) onto the surface using VTK 3D object picker class. This algorithm generates a better sampling of surface points and normals along the path sketched by the user. From experiments, 5-10 points sampled surface points and surface normals are enough to form an accurate average surface normal. To increase accuracy, the number of sample points can be increased and/or surface points to the "left" and "right" of the sampled points can also be included (Figure 29 right).

3.4 Slice Plane Construction from 3D Sketch-lines

As stated previously, the 3D sketch-line contains enough information to position and orient an image slice plane that is roughly orthogonal to the object surface. First, the image slice is centered at the sketched spline curve midpoint.

Then, the image slice coordinate system is constructed using the average surface normal and the line segment formed from point1 and point2. Specifically, the normal vector of the slice plane is calculated from the cross product of the point1-point2 line and the average surface normal vector (Figure 30).

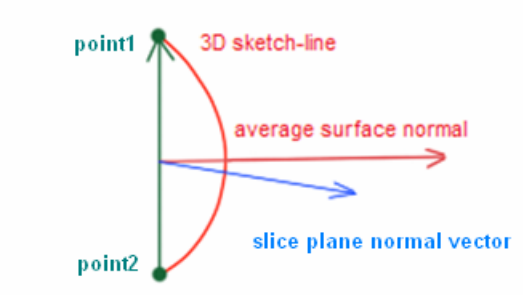

Figure 30 – Image slice normal vector is constructed by taking the cross product of the average surface normal and the point1-point2 line.

The “up” vector of the image slice (i.e. the “spin angle” around the image slice normal vector) is calculated so that the boundary edges of the image slice are parallel to the sides of the volume image bounding box. This image slice “spin angle” constraint makes image slices that appear “upright” in the 3D window from the user’s perspective. As most volume images have an idea of an “up” and “down” direction, for example the top of the head in a CT scan of the head which corresponds to the sense of “up” and “down” in the human body, this image slice spin adjustment helps the user make sense of the image slice view in the 3D window. It also helps to set an up and down direction in the 2D window view of the image slice. Finally, the image slice is translated with respect to the ‘x’ and ‘y’ axis of the slice plane so that it is centered within the volume image bounding box.

3.5 Slice Plane Position and Orientation Editing

Once the user sketches a line on the object surface and the image slice is instantaneously constructed and drawn, the user may want to edit or "fine-tune" the image slice position or orientation. As mentioned in Chapter 2, there are several possible rotation models that can be used to adjust the image slice orientation [5]. Since the "unconstrained" arc-ball style rotation model allows changes to two rotation directions at the same time and also often results in unpredictable image slice "spin angle" orientations, the user can spend extra time adjusting the spin angle of the image slice to an upright orientation. In this thesis, a new constrained rotation model was developed that uses a torus as a rotation "handle" (Figure 31 top).

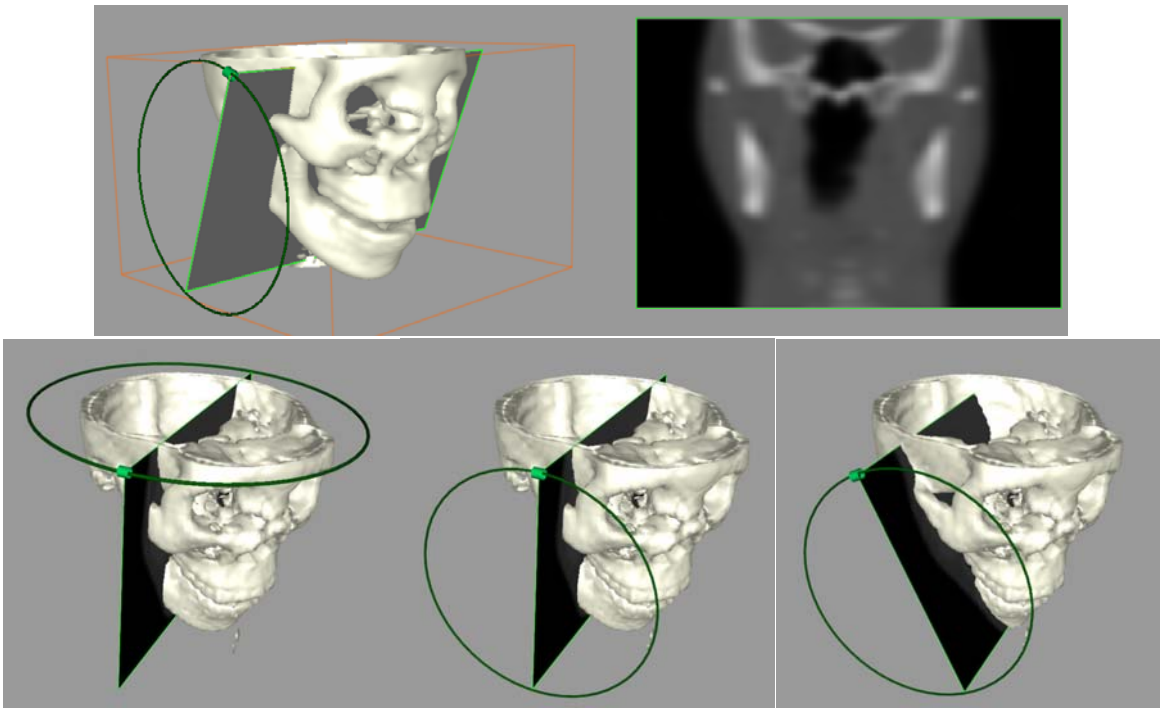

Figure 31 – Top: example of the torus slice rotation handle and cylinder marker that slides along the torus. Bottom: the user can select the marker and rotate the image slice in one direction, then press a key and the torus orientation changes (bottom middle) and the user can then rotate in the other direction. A third direction change (not shown) allows the user to spin the image slice around its normal vector.

A small cylindrical marker is also rendered which can be selected by the user and slides around the torus as the user moves the cursor along the torus. The torus model is a constrained rotation model that separates the rotational degrees of freedom to provide precise image slice orientation control as well as to prevent unwanted “spin angle” side effects.

To change the torus orientation and hence change the image slice rotation direction, the user presses the “Shift” key (Figure 31). That is, the user can instantly toggle between the two rotation directions and the mouse is always very close to the cylinder marker so the user can fluidly continue a rotation in a different direction. The torus acts as a guide and gently forces the user to move the cursor along the torus during the rotation adjustment, keeping the cursor close to the cylinder. If the cursor goes off the torus guide too much, the rotation will be deselected. Some flexibility is built-in to prevent the user from having to be too precise in their mouse positioning but still gaining the benefit of keeping the cursor close to the cylinder marker. Other rotation models that separate the rotational degrees of freedom do not provide this capability and the user can move the cursor far from the rotational control handle, forcing them to return the cursor to another handle or area on the slice plane when a change in rotation direction is desired. In chapter 4, the results of a user study comparing the torus model with an arc-ball style rotation model and an image slice “margin” constrained rotation model is presented. Another advantage of the torus rotation model is that the torus and cylinder can always be selected regardless of the current orientation of the image slice.

In summary, the torus rotation model main advantages are: 1) the torus/cylinder is always visible and selectable for rotation no matter the current slice plane orientation, 2) attempts to minimize mouse movements and mouse clutching [52] during switches in the direction of rotation therefore prevent unnecessary scene rotation. Other adjustments to the image slice can also be made. The user can select a point anywhere on the slice (other than the corner points) and "push" or translate the slice along its normal vector direction (Figure 32(a)). The user can also scale the slice using a slider as well as translate the slice using the mouse the two directions parallel to the edges of the slice by first pressing the "F1" function key (Figure 32(b)).

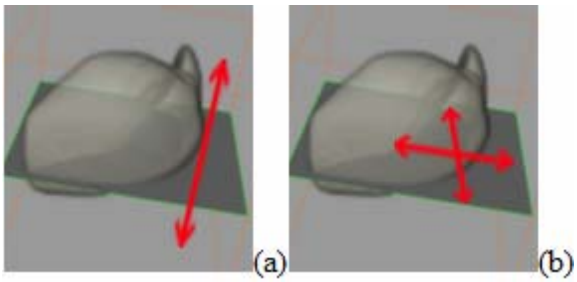

Figure 32 – Slice Positioning: (a) "pushing" the slice along its normal vector. (b) Translating the slice plane with 2 directions parallel.

3.6 Slice Plane Sliding

If the user wishes to look at the image cross-sections along a curving part of the object surface, many rotation techniques would force them to repeatedly and tediously "push" and rotate the image slice in 3D world coordinates so the slice is orthogonal to the surface. Using sketch-lines, this examination process can be quickly done. The user can quickly sketch a series of sketch-lines along the curving section (Figure 33) and enter into an image slice "snap" mode using a button press.

In this case, the midpoint of each sketch-line is connected and these points are used to form an interpolating spline (Figure 33). That is, the midpoints of the sketch-lines are connected to form a smooth curve and the curve goes through the midpoints. In addition, the average surface normal of each sketch-line is also smoothly interpolated using the same spline curve. Now when the user "pushes" the image slice the slice center point automatically follows the spline path and the slice normal is automatically interpolated. The result is the slice can slide back and forth along the curving section of the object surface while keeping itself approximately orthogonal to the surface (Figure 34).

Figure 33 –Smooth spline path formed by connecting the midpoint of multiple 3D sketch-lines.

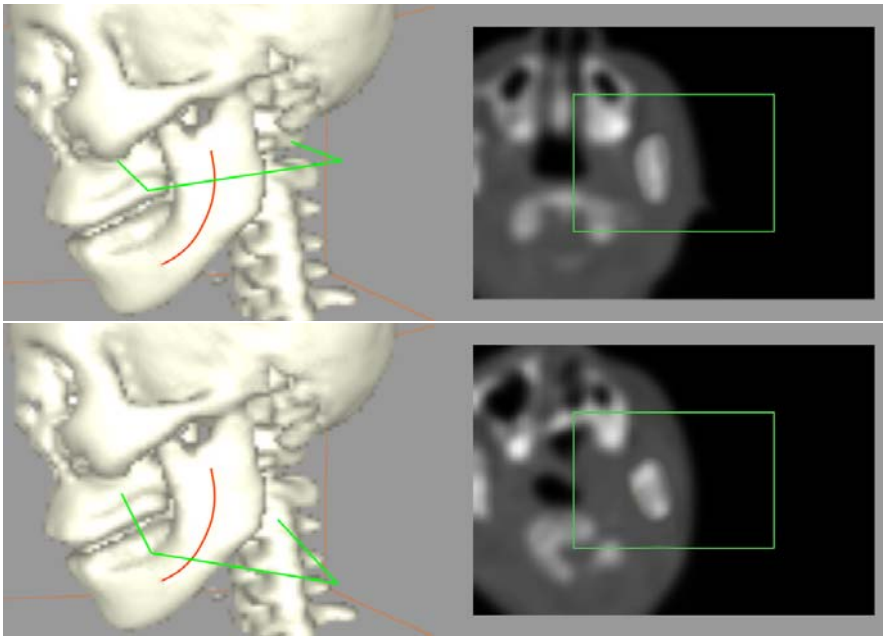

Figure 34 – Automatically sliding the image slice along the spline curve so that it is always roughly orthogonal to the object surface. The right images show the corresponding object cross sections.

3.7 Region Delineation Meshes and Region Envelope Meshes

As mentioned, both 2D and 3D sketch-lines can be connected to form a "control" mesh of triangles. A control mesh can then be input to an interpolatory subdivision surface algorithm [1] to form a smooth mesh of subdivided triangles. This *subdivision surface* mesh can be used to delineate, cutaway, or segment regions of interest.

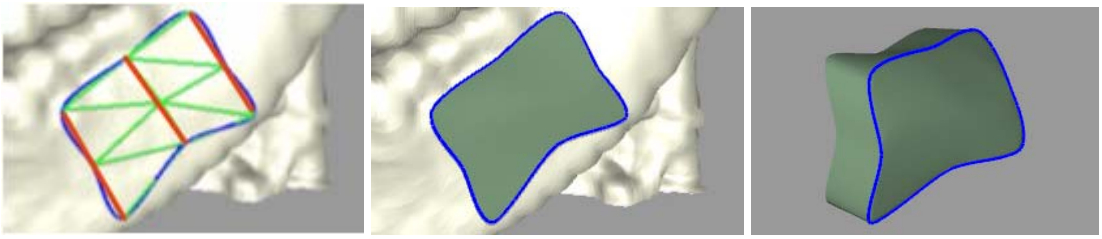

Figure 35 – Patch type of envelope. Left: 3 sketch lines (red) are connected and triangles are formed to create a control mesh. Middle: the control mesh can be subdivided into smaller triangles making a smooth surface patch. Right: the surface patch can be copied and moved inward and connected to the top patch to form an envelope.

The sketch-line system currently supports 3 types of subdivision surface meshes: a closed surface mesh, a cylindrical mesh known as a "sleeve", and an open surface mesh known as a "patch" (Figure 35). A patch is a smooth open surface mesh that can be used to delineate a curving region of the data object surface. The patch can be copied and moved inward to form a closed "thin shell" mesh envelope (Figure 35 middle and right). This type of envelope is useful for cutting away thin shell structures such as the skin or the top of the skull. A sleeve is an open cylinder mesh that can be used to "delineate" a section of a curving cylindrical structure such as an artery. The sleeve can be snapped down onto the artery section, using the deformable surface model segmentation algorithm [1], to segment the artery section. The open cylinder sleeve mesh can also be capped at the ends of the cylinder to form a closed cylindrical mesh which can then be used as an envelope to cutaway the data object inside.

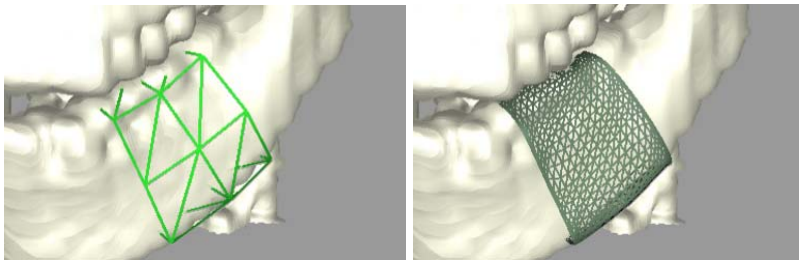

Figure 36 – Sleeve type of envelope. Left: 3 sketch lines are connected to form a cylindrical control mesh. Right: the control mesh can be subdivided into smaller triangles to form a smooth open-ended cylindrical mesh.

Finally, the closed surface mesh envelope can be constructed in two ways. A closed contour called a profile curve can be sketched in 2D on a series of image slices that show cross sections of a data object. The profile curves can be connected together to form a closed surface which can then be used to segment the data object [1] or used as an envelope to cut away the data object from the volume image.

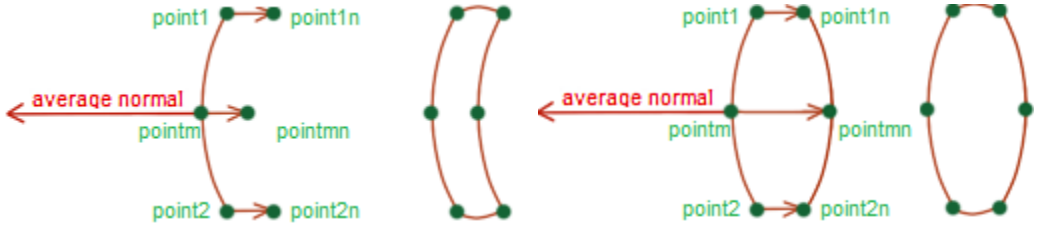

Figure 37 – Two types of profile curves construct from a single 3D sketch-line. Left: a sketch line can be copied and moved inwards (a user-defined distance) along the negative average surface normal vector direction. The two curves can be connected to form a closed profile curve. Right: in this type of profile curve, the middle point (*pointmn*) has been reflected in the copied curve so that its distance from the line joining *point1n* and *point2n*is the same as the distance of the original middle point from the line joining *point1* and *point2*.

A profile curve can also be constructed from a 3D sketch-line (Figure 37). The 3 control points of the 3D sketched spline curve are copied and moved inwards (i.e. in the direction of the negative average surface normal vector) a user-definable distance (using a slider) into the object (Figure 37 left). A surface mesh constructed from a series of these profile curves is useful for cutting away thin shell type objects such as the top of the skull.

The copied spline curve can also be reflected across an axis formed by the line joining point1 and point2 (Figure 37 right). The copied spline curve is connected to the sketched spline curve to form the profile curve. This style of profile curve is used for all other object shapes.

An example of a closed surface mesh created from a series of 3D sketch-lines is shown in Figure 38. Three sketch lines have been converted to profile curves (Figure 38 left and right) and are then stitched together to form a closed control mesh. The mesh is once again input to the subdivision surface algorithm to create the smooth closed surface mesh (Figure 38 middle).

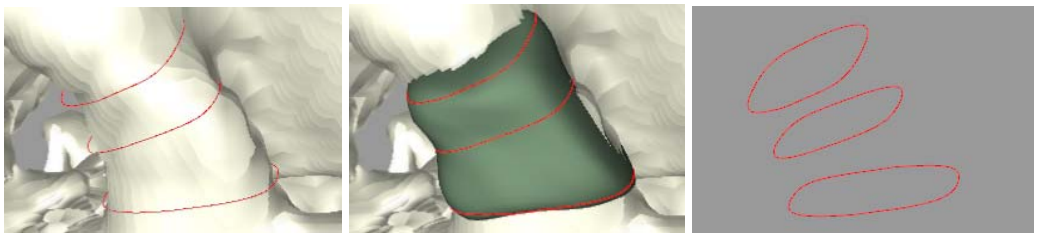

Figure 38 – Envelope mesh formed by connecting profile curves constructed from 3 sketch lines.

3.7.1 Envelope and Patch Editing

The envelope or patch meshes can be edited in several ways. The 3D sketch-lines are highlighted in red after the sketching process and drawn on top of the envelope mesh (Figure 39). The user can click on these red profile curves in the 3D window and the image slice will immediately be repositioned and re-oriented and the 2D window cross-section view updated. In addition, the profile curve will be displayed in the 2D window along with control points (Figure 39 and Figure 40) which can be selected and repositioned. These edits update the envelope/patch mesh in real time.

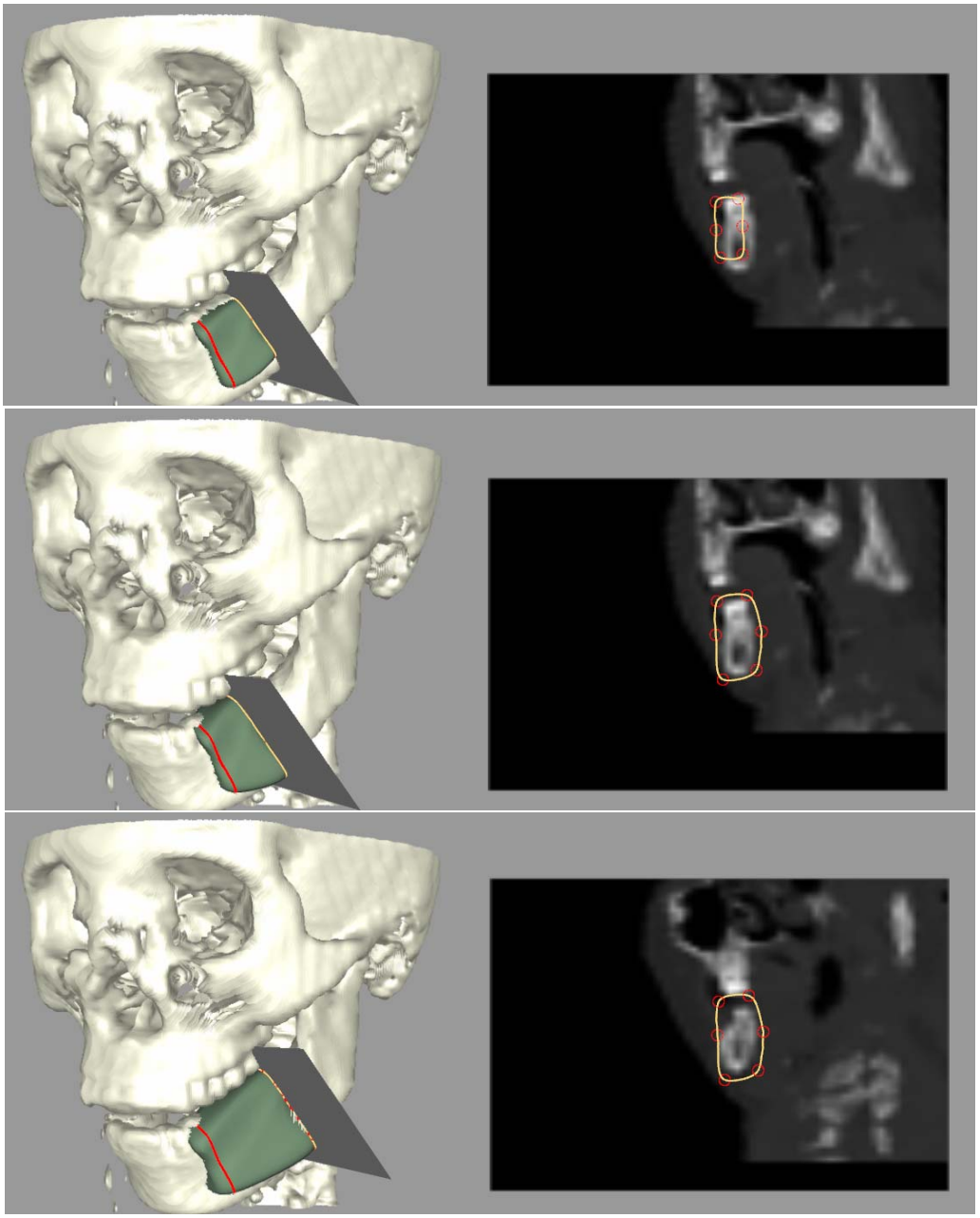

Figure 39 –Example of envelope editing. Top left: 2 sketch lines are used to create a closed surface envelope. The profile curves are shown in red in the 3D window and the user has selected one of them. The corresponding profile curve (yellow) is shown in the 2D window (top right) and the user can select the control points (red). Middle Image: the user has moved the control points and changed the shape of the profile curve to ensure the envelope surrounds the jaw region. Bottom Image: the user can also move the slice plane in its normal vector direction (or rotate it) and the envelope is attached to it and will be stretched.

In this way, if the initially sketched envelope does not quite cover the region of interest after the sketching process or if the depth of the mesh is not enough to cover the back side of the ROI, the user can edit the profile curves in 2D and see the results in the 3D window.

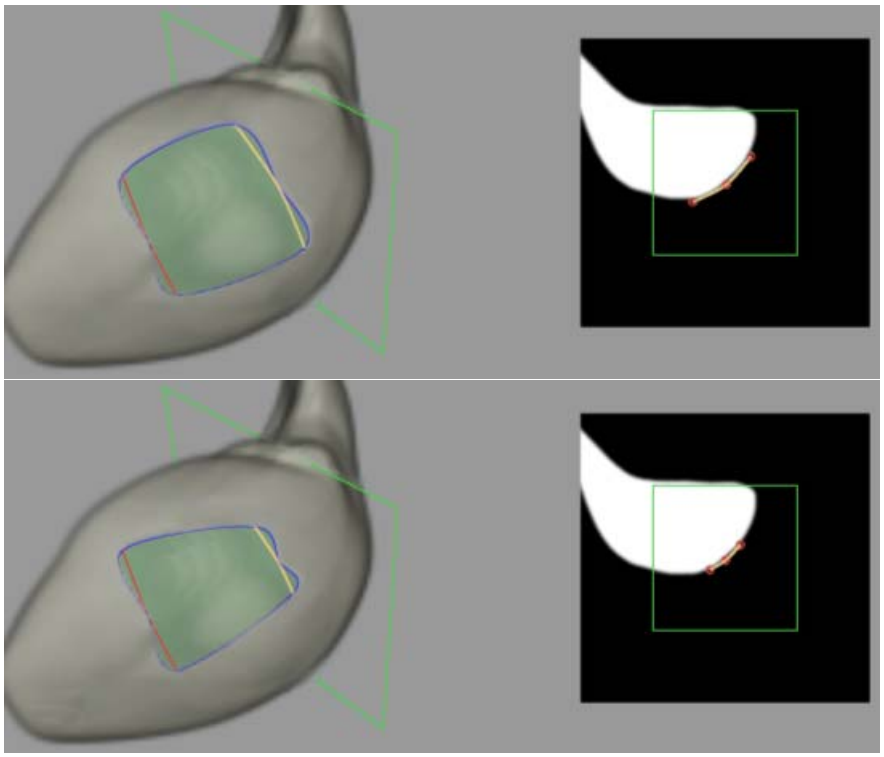

Figure 40 – Example of patch envelope editing in a segmented liver dataset. The profile curve constructed from a sketch line can be manipulated in the 2D window.

An undo operation is a very important part of any interaction model - the ability to return to a previous state [45]. When sketching lines in the 2D window and creating a contour, the “u” key can be used to undo the last sketch-line and return the contour to the previous state. Similarly in the 3D window the “u” key is use to undo sketch-lines in the 3D window. Users can click on “U/u” key to undo any mistakes. When “U” is pressed with the mouse in the 3D window, a 3D sketch-line will be undone. If there is a surface envelope, the undo will remove the last profile curve and return the envelope to the state it was in before the new line was sketched.

Chapter IV –Evaluation and Results

In this chapter several experiments demonstrating the uses of the sketch-line interaction model are presented. Then the results of two user studies, comparing sketch-lines and the torus image slice rotation technique to other interaction models, is presented.

4.1 3D Sketch-Lines for Image Slice Positioning

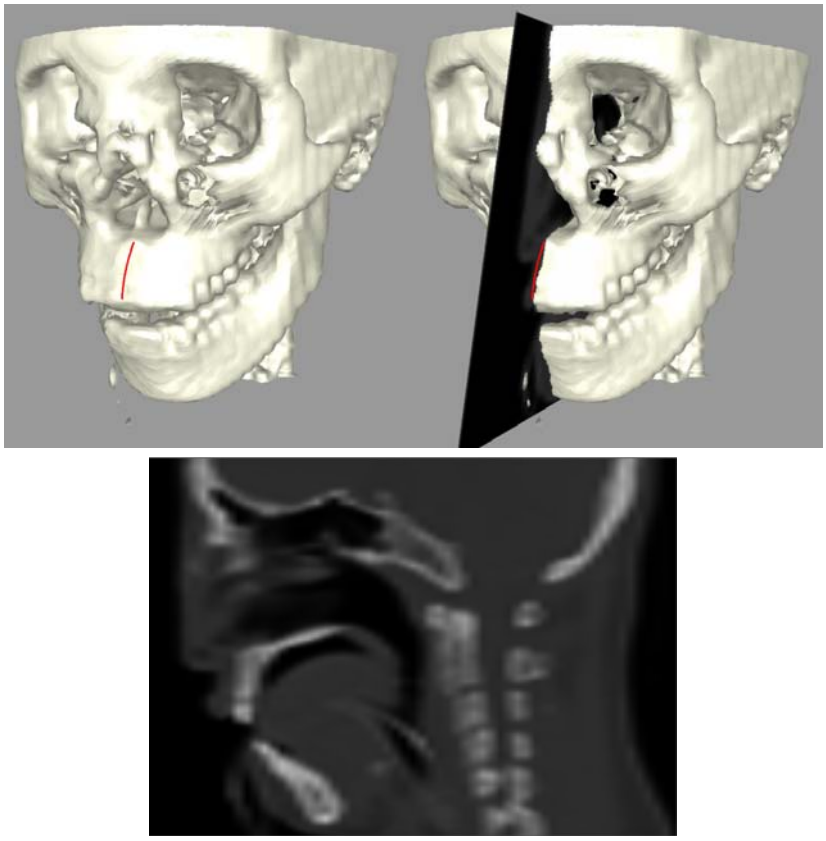

Figure 41 – A simple example of the image slice positioning after sketching on the skull surface in a CT scan of the head (top left). The image slice is instantly positioned (top right) and the skull cross-section is also displayed in a 2D window (bottom).

Two experiments were performed to demonstrate the basic idea of using sketch-lines to automatically position and orient an image slice position so that it is roughly orthogonal to the surface of a volume rendered object. Figure 41 demonstrates basic image slice positioning after a single sketch-line is drawn. In Figure 42, several sketch-lines were drawn along the curving surface of a segmented image of a liver. The image slice was initially positioned at the first sketch-line and can be “pushed” or “pulled” to slide along the liver surface, interpolating orientations between the sketch-lines and forcing it to remain approximately orthogonal to the liver surface.

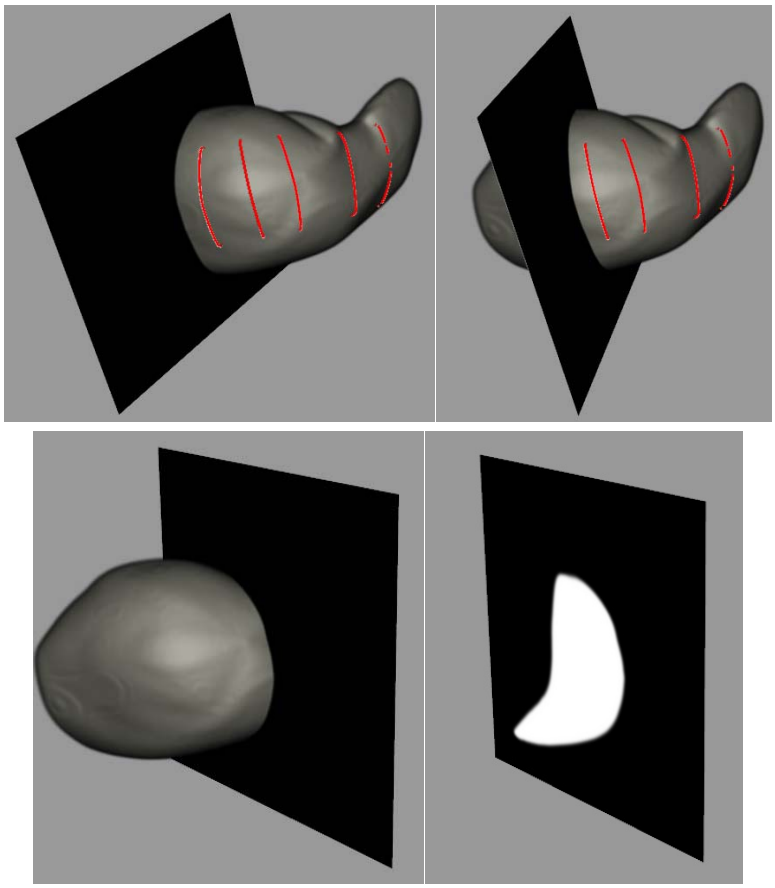

Figure 42 – Top left: a series of sketch-lines are drawn along the curving surface of a segmented liver dataset. An image slice is automatically positioned at the first sketch-line. Top right and bottom left: the slice can be “pushed” with the mouse and the slice will slide along the liver surface interpolating orientations between the sketch-lines and remaining approximately orthogonal to the liver surface. The lower right image shows the image slice with volume rendering turned off to reveal the cross section of the liver.

4.2 Cutaway

In the next series of experiments, the sketch-line interaction model was combined with a subdivision surface [1] and a sketch line connection process to create envelopes that surround 3D regions of interest of a volume rendered object. In Figure 43 a CT scan of a head was used and the skull was volume rendered (Figure 43 top left). Three sketch-lines were drawn on the forehead region and connected together and subdivided to form a smooth patch that conforms to the shape of the forehead (Figure 43 top middle). The sketch-lines can be edited in the 2D window (Figure 43 lower left) to stretch or shrink the sketch-line (and hence the patch). The patch was then extruded inward a user-defined distance (using the 2D image slice window as a guide (Figure 43 lower right) and the resulting envelope is shown in Figure 43 lower middle. The envelope was then used as a mask to cut a hole in the skull revealing the structures inside (Figure 43 top right).

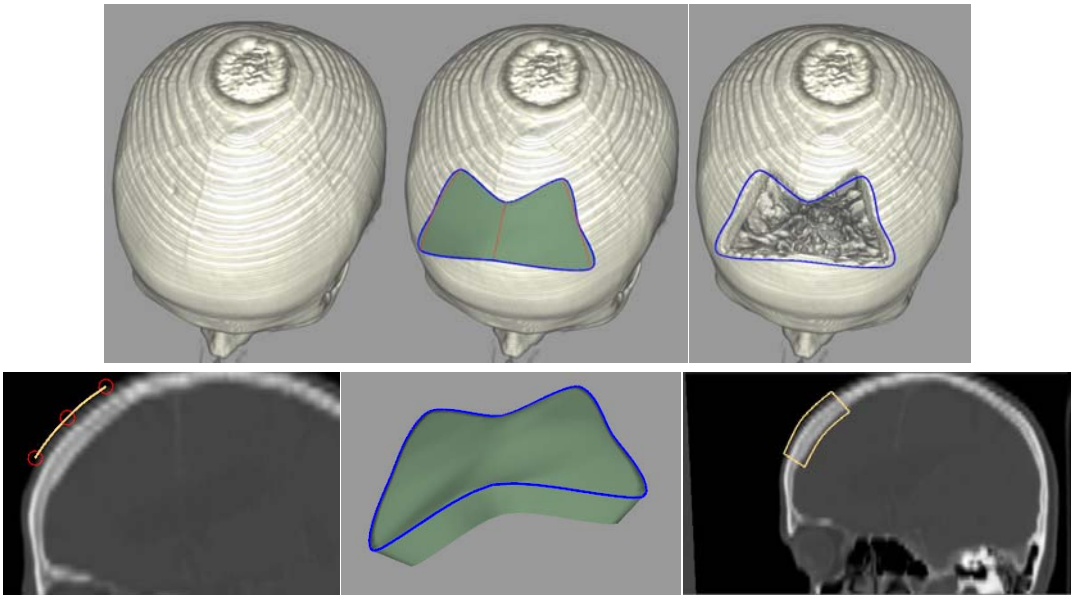

Figure 43 –Three sketch-lines are drawn on the forehead of a CT head scan [50] (top middle) forming a surface patch. The sketch-lines can then be edited in the 2D window (lower left). The patch can be extruded inward to form an envelope (lower middle and lower right) and a hole can then be cut in the skull (upper right).

In a second cut away experiment, 3 lines were sketched along the jaw of a low resolution CT scan of the head (Figure 44). Each sketch-line was converted to a profile curve (Section 3.7) with a user-defined depth. The profile curves were connected to form a cylindrical mesh (a "sleeve") and the mesh was subdivided (Figure 44 middle). The 2D window can be used to make sure that sleeve is surrounding the target jaw region. The open ends of the sleeve were then "capped" resulting in a closed cylindrical mesh and the envelope was used to cut the jaw region away (Figure 44 right).

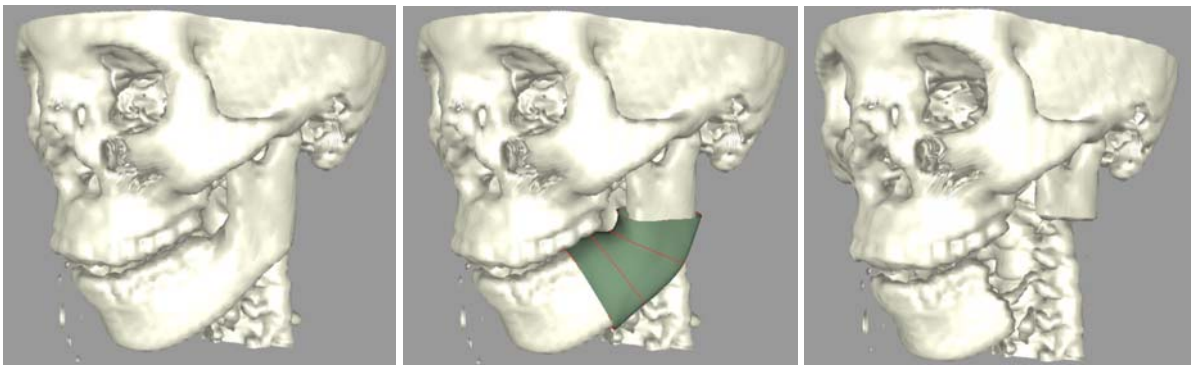

Figure 44 – 3 sketch-lines are drawn on a target jaw region in a CT scan of the head. A profile curve is constructed from each sketch-line, with a user-defined depth, and the profile curves are connected and subdivided to form a smooth "sleeve" mesh (middle). The sleeve, an open cylindrical mesh, is then "capped" and the envelope is used to cut the jaw region away (right).

For the final cutaway experiment, an envelope editing featured is demonstrated. In Figure 45, 2 sketch-lines were drawn on the surface of the liver. Profile curves were constructed and connected then subdivided to form a closed surface envelope (Figure 45 left). The user can select a profile curve and an image slice will be positioned there. The user can edit the profile curve in the 2D window or alternatively, translate and/or rotate the image slice automatically pulling on the profile curve and hence stretching/shrinking the envelope (Figure 45 middle). The envelope was used to cut away a portion of the liver (Figure 45 right).

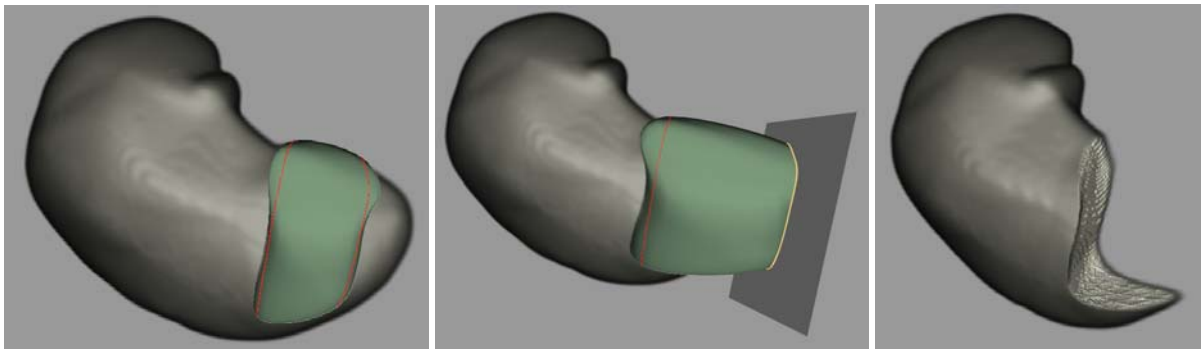

Figure 45 – Two sketch-lines are drawn on the surface of the liver. A profile curve is constructed from each sketch-line, with a user defined depth, and the curves are connected and subdivided to form a closed envelope (left). A profile curve can be selected and an image slice is positioned. The curve can be edited in the 2D window or the image slice can be translated and rotated (middle) to stretch/shrink the envelope. The envelope can be used to cut away the voxels inside it (right).

4.3 Segmentation

In this section several experiments were performed to demonstrate the use of sketch-line constructed envelopes for volume image segmentation. The subdivision surface envelopes are input to a deformable model segmentation algorithm developed in [1]. In the first experiment, similar to the jaw cutaway in the previous section, a sleeve model was constructed with 3 sketch-lines along a portion of the jaw (Figure 46 top left). In Figure 46 top right, the sketched sleeve model is fairly accurately surrounding the jaw. With many segmentation algorithms, including deformable surface model algorithms, the better the initial model, the better chance that the segmentation will perform well and “snap” or fit onto the correct object boundaries. This is especially true in noisy medical images where the chance of the deformable model incorrectly snapping onto the boundaries of neighbouring objects is much higher. Figure 46 lower left and right shows the result of the model fitting onto the jaw region.

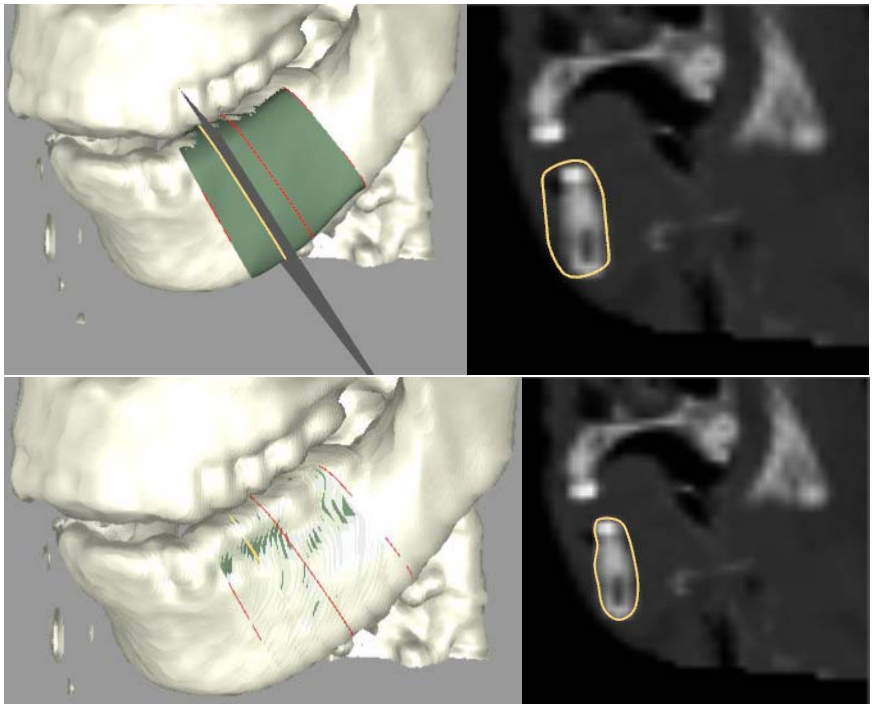

Figure 46 – 3 sketch-lines are drawn on a target jaw region in a CT scan of the head. A profile curve is constructed from each sketch-line, with a user-defined depth, and the profile curves are connected and subdivided to form a smooth "sleeve" mesh (upper left). The accuracy of the initial sleeve mesh is apparent in the 2D window showing the sleeve cross section (upper right). The sleeve mesh is input to a deformable model fitting algorithm and is fitted to the jaw boundary (lower left and right).

In the next experiment, another sleeve model was constructed using 5 sketch-lines along a portion of the right femoral artery in a contrast-enhanced CT scan of the lower body (Figure 47). The initial model is again fairly accurate (Figure 47 top) and the deformable model fitting algorithm was able to accurately segment the artery portion (Figure 47 bottom). The segmentation was validated through visual inspection only.

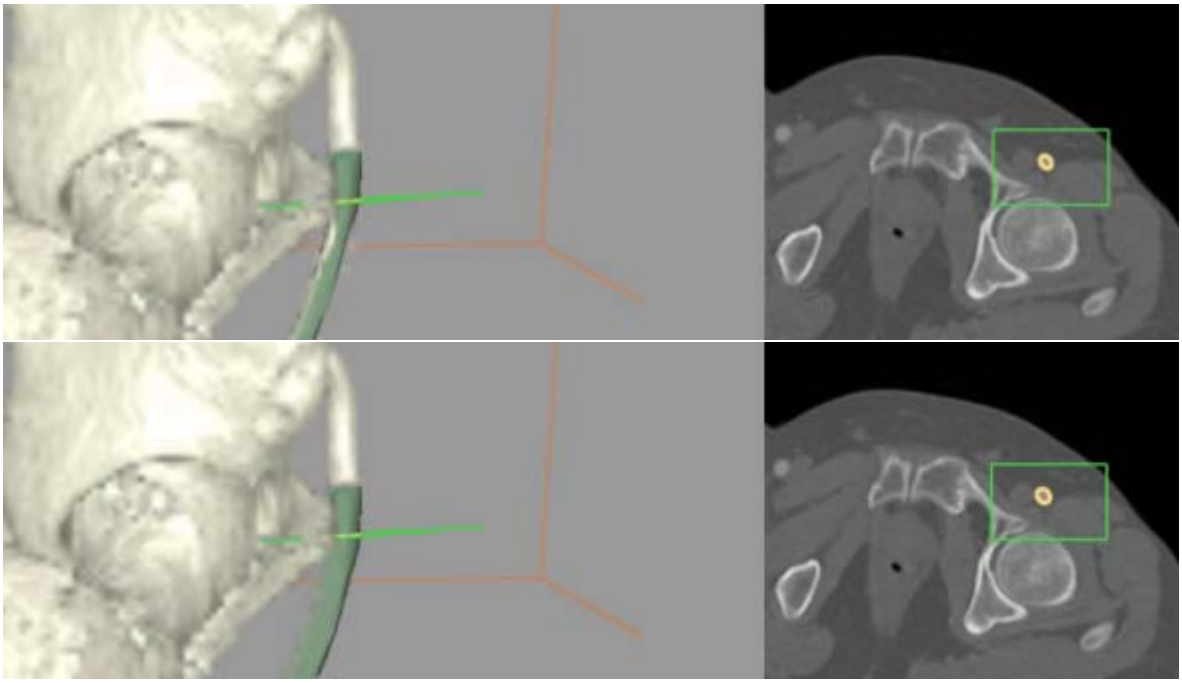

Figure 47 – 5 sketch-lines are drawn on a portion of the right femoral artery in a contrast-enhanced CT scan of the lower body [50]. An accurate sleeve model is constructed (top) which is then fitted to the artery surface (bottom) resulting in an accurate (via visual inspection) segmentation.

4.4 User Study 1: Sketch-Lines and Torus Rotation for Image Slice Positioning

The goal of the first user study was to determine the effectiveness and efficiency of the sketch-line technique for image slice positioning and also to determine the effectiveness and efficiency of the torus image slice rotation algorithm. The study was divided into two parts. The first part compared the torus rotation technique against two other common image slice rotation techniques. One rotation technique, labelled the “margin” technique in this study, is similar to torus rotation where rotation is performed in only one direction at a time.

In margin rotation (Figure 48(a)), the user selects either the top or bottom margin region of the slice plane (the red margin lines in Figure 48(a)) and “pushes” or “pulls” on this region with the mouse to rotate the slice about the vertical and horizontal axis of the slice.

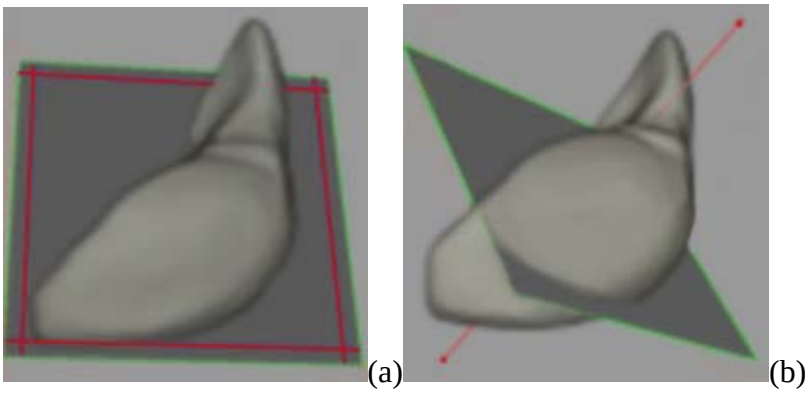
(a) (b)

Figure 48 – Depiction of slice plane rotation “handles” for: (a) “margin” rotation (b) “pivot” rotation.

The four corner regions defined by the red margin lines can be selected and are used to “spin” the slice about its normal vector. The amount of rotation is determined by how far the mouse is dragged. In this technique, the mouse can be dragged far from the initially selected margin region. When the user wishes to change rotation direction, the mouse has to be dragged back to the opposite margin. Furthermore, if the user is viewing the whole scene such that the image slice is viewed “on edge” then the user must rotate the entire scene (know as *scene rotation*) so that the margins are visible and can be selected. As mentioned before, the torus rotation technique was designed to overcome these limitations.

A second rotation technique, labelled “pivot” rotation, is an unconstrained technique that allows the user to change the orientation in two directions.

The user selects and pulls/pushes on the pivot axis handle (the red line in Figure 48(b)) and the handle is designed to be always perpendicular to the slice plane. To spin the slice plane, the shift key is pressed and held as the mouse is moved.

The unconstrained nature of the pivot technique means that the users can often overshoot the desired rotation or correctly align one rotation direction but not both. Also, this model often results in a "spin" side effect and the user must spend time correcting the spin angle.

A total of 11 people, 10 male and 1 female, were recruited from the undergraduate and graduate computer science student body using an advertisement on the department bulletin board as well as in-class announcements. They were on average 20 years old with an average of 39 hours of mouse usage per week and 9 hours of video game playing per week. 6 of the participants reported being familiar with 3D graphics packages such as 3D Studio Max, Maya, and Blender. A mouse was used as the input device due to its familiarity (not all participants had experience with touch screen or pen input devices). The trials performed in the study required approximately 1 hour.

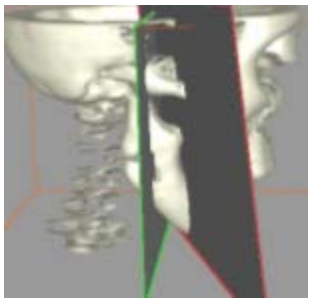

Figure 49 – In the first part of the study users were asked to rotate the green slice plane, initially in a standard orientation, so that its orientation matched (within a tolerance) a target slice plane (red).

In the first part of the within-subjects study, users were asked to orient a slice plane, using a randomly assigned rotation technique, from a standard position so that it matches the orientation of target slice plane (Figure 49) within a tolerance.

Once the target was matched, a new target appeared. The time taken to perform the matches as well as the number of mouse clicks etc. was measured. Each user was instructed on the required task and each of the 3 rotation techniques was demonstrated.

In addition, a user was allowed to practice the rotation techniques for a few minutes each and then performed a practice trial. Each trial consisted of 10 slice matching tasks. Each user was assigned the 3 rotation techniques in random order and the 10 target slice plane orientations were presented to each user in a random order. A total of 3 trials were performed for the 3 rotation techniques for a total of 90 matching tasks. Once the users finished the trials, they were asked to fill out a form asking some qualitative questions about the rotation techniques.

Since the torus rotation technique was designed to be useable regardless of the 3D scene orientation (i.e. would rarely require an initial scene rotation to select the torus and begin rotation) a hypothesis was formed that torus rotation would require less user scene interaction, measured by the number of mouse clicks and scene rotation time required, than the other two techniques. Also, since the torus rotation was designed such that the mouse was always close to the cylinder control handle when the user changed rotation direction, a second hypothesis was formed that torus rotation would require fewer mouse clicks and less rotation time than the other two techniques. Finally, a third hypothesis was formed that torus rotation would require less spin rotation time and fewer mouse clicks than pivot rotation due to the spin angle side effects of the pivot technique. Table 1 shows the results of the study.

As for the results, the first hypothesis was proven to be partly true by a statistical analysis (T-Test and P-Test). Torus rotation requires fewer scene rotation mouse clicks (Two-tailed $p < 0.04$) than margin rotation.

However, while torus rotation required less scene rotation time and mouse clicks than pivot rotation, the results were not statistically significant. Similar results were observed for the second and third hypotheses. That is, torus rotation required statistically significant less rotation time and spin mouse clicks and time than margin rotation but not statistically significant less clicks and time than pivot rotation.

Torus vs. Margin and Pivot	Scene Rotation Time	Scene Mouse Clicks	Slice Rotate Time	Rotate Mouse Clicks	Slice Spin Time	Spin Mouse Clicks	Total AVG Time	Total Mouse Clicks
AVG: Margin	1.22	1.79	2.48	2.48	0.65	0.55	4.33	5.1
AVG: Pivot	1.13	1.47	2.31	1.69	0.4	0.37	3.83	3.87
AVG: Torus	0.79	1.07	1.84	1.97	0.37	0.35	2.99	3.46
STD: Margin	0.77	1.17	0.85	0.79	0.36	0.25	1.55	2.12
STD: Pivot	1.07	1.4	1.58	0.57	0.37	0.35	2.35	2.15
STD: Torus	0.43	0.58	0.39	0.23	0.25	0.18	0.74	0.75
T-TEST: Torus vs. Margin	-2.14	-2.43	-2.50	-2.22	-4.54	-2.98	-3.17	-2.72
P-TEST: Torus vs. Margin	0.06	0.04	0.03	0.05	0	0.02	0.01	0.02
Probability: Torus vs. Margin	95.26	97.06	97.35	95.83	99.89	98.81	99.1	98.17
T-TEST: Torus vs. Pivot	-1.31	-1.22	-0.97	1.60	-0.22	-0.31	-1.20	-0.74
P-TEST: Torus vs. Pivot	0.22	0.25	0.36	0.14	0.83	0.76	0.26	0.48
Probability: Torus vs. Pivot	82.8	80.46	73.01	88.89	42.73	46.45	79.9	64.37
T-TEST: Pivot vs. Margin	-0.34	-0.99	-0.45	-4.55	-1.51	-1.51	-0.86	-1.93
P-TEST: Pivot vs. Margin	0.74	0.35	0.66	0	0.16	0.17	0.41	0.09
Probability: Pivot vs. Margin	47.65	73.63	52.58	99.89	87.29	87.22	68.8	93.33

Table 1 – User study 1 measurements: Torus rotation vs. Margin and Pivot rotation.

Upon completion of all trials the participants were asked to fill out the questionnaire and indicate their level of agreement or disagreement with each statement using a 5-point Likert scale. The questionnaire asked the participants whether each rotation technique was easy to control and whether it was easy to learn. The results are presented in Table 2 and Figure 50. On average, torus rotation obtained the highest scores.

The participants were also asked to state their favourite rotation technique. The results were that 3/11 listed the pivot technique as their favourite, 2/11 listed margin as their favourite and 6/11 listed torus as their favourite.

Participants were also asked to comment on the techniques. 3 users commented that pivot rotation was easy to control, intuitive, natural, and fast. 5 users commented that the torus technique “makes sense”, was easy to use, and that the visual cue provided by the torus allowed them to understand the effect of the rotation. The 2 users who liked the margin technique liked it for its simplicity and liked that no key presses were required for spin.

Participants were also asked to list their least favourite technique. The results were that 6/11 disliked pivot, mostly due to the difficulty in controlling the rotation to achieve the expected result. Next, 4/11 disliked margin the most and commented that it was slow to achieve the rotation result and also that it was non-intuitive as well as requiring more scene rotation. Only 1 user disliked the torus technique the most and the comment was that it felt “unnatural”.

5-Point Likert Scale	Margin	Pivot	Torus
AVG: Easy to Learn	4.00	3.36	4.27
STD: Easy to Learn	0.77	1.21	1.19
AVG: Easy to Control	3.45	3.36	4.27
STD: Easy to Control	1.04	1.21	0.79

Table 2 – User Study 1: Torus rotation vs. Margin and Pivot questionnaire results.

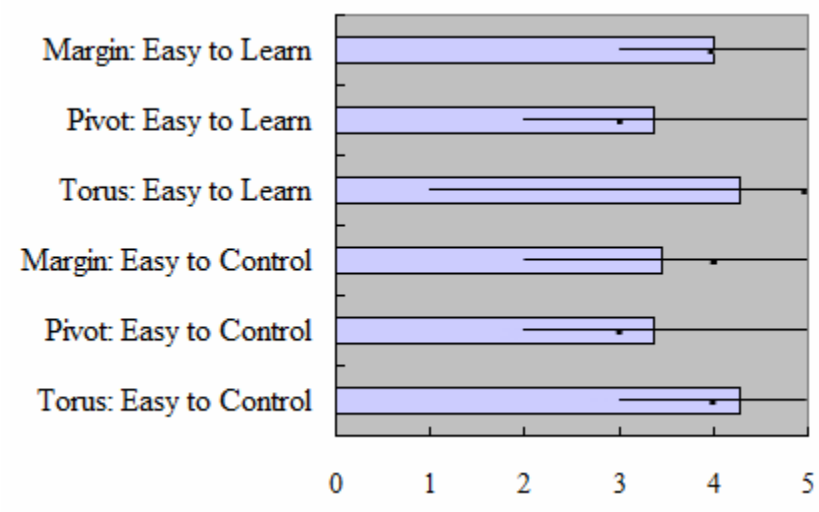

Figure 50 – User Study 1: Torus rotation vs. Margin and Pivot questionnaire results. The bar shows the median value, the thin black line shows the minimum and maximum.

The second part of the study compared the sketch-line slice plane positioning and orienting technique with a standard "push"/rotate technique. The low resolution CT dataset of the skull was used as the target object slice positioning as it is clearly rendered and very fast to rotate. In the "push"/rotate technique the user simply selects any point on the slice plane and moves the mouse to "push" and "pull" the slice plane in a direction parallel with its normal vector. Since the torus technique was considered precise, it was the only technique made available to the users to rotate the slice plane.

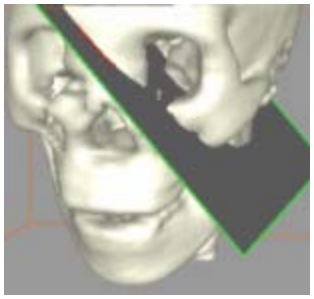

Figure 51 –In the second part of the study users were asked to sketch on the surface of the skull for slice plane positioning.

The second part of the study proceeded similarly to the first. A target slice plane was presented to the users, positioned and oriented relative to the skull surface. For the push/rotate technique, a slice plane in a standard position was provided and the users had to manipulate it to match the target plane. For the sketch-line technique (Figure 51), the user had to sketch a line near the target slice in order to match it. The users were permitted to fine-tune the sketch-line result using the standard push/rotate technique. Each trial consisted of 10 test cases. A total of 3 trials were performed for both techniques for a total of 30 test cases each. Once again, users were allowed to practice the two techniques. A questionnaire was filled out at the end.

The basic hypothesis in this part of the study was that the sketch-line technique would need fewer mouse clicks and less time to position and orient the slice plane with respect to the skull surface. The push/rotate technique requires significant user input to slide and orient the slice. Table 3 summarize results. The hypothesis was verified ($p < 0.05$). A breakdown of the amount of push, rotate, spin, and scene mouse clicks and time is also included for completeness.

Sketch vs. Push	Scene Rotation Time	Scene Mouse Clicks	Slice Push Time	Push Mouse Clicks	Slice Rotate Time	Rotate Mouse Clicks	Slice Spin Time	Spin Mouse Clicks	Total AVG Time	Total Mouse Clicks
AVG: Push	2.07	2.47	0.70	0.75	3.07	2.62	0.57	0.50	6.40	6.46
AVG: Sketch	2.52	2.41	0.39	0.49	0.61	0.66	0.11	0.11	5.30	4.89
STD: Push	0.69	1.18	0.46	0.52	0.65	0.44	0.42	0.34	1.60	2.06
STD: Sketch	0.79	0.86	0.41	0.54	0.41	0.32	0.12	0.08	1.47	1.27
T-TEST	1.52	-0.20	-2.86	-2.38	-11.97	-13.66	-3.79	-3.52	-2.61	-3.67
P-TEST	0.16	0.84	0.02	0.04	0.00	0.00	0.00	0.01	0.03	0.01
Probability	87.49	42.02	98.55	96.80	100.00	100.00	99.67	99.50	97.81	99.60

Table 3 – User Study1measurments: Sketch-line vs. Push/Rotate.

The questionnaire asked the participants whether the sketch-line and push/rotate techniques were easy to control and whether they were easy to learn.

The results are presented in Table 4 and Figure 52. On average, the results were roughly equal. The participants were also asked to state their favourite. The results were that 2/11 listed push/rotate as their favourite while 9/11 listed sketch-line as their favourite. The 2 users who preferred push/rotate commented that it was difficult to predict the position and orientation of the slice plane after sketching.

5-Point Likert Scale	Push/Rotate	Sketch-Line
AVG: Easy to Learn	4.64	4.18
STD: Easy to Learn	0.67	0.75
AVG: Easy to Control	4.18	3.91
STD: Easy to Control	0.87	0.54

Table 4 – User Study 1: Sketch-lines vs. Push/Rotate questionnaire results.

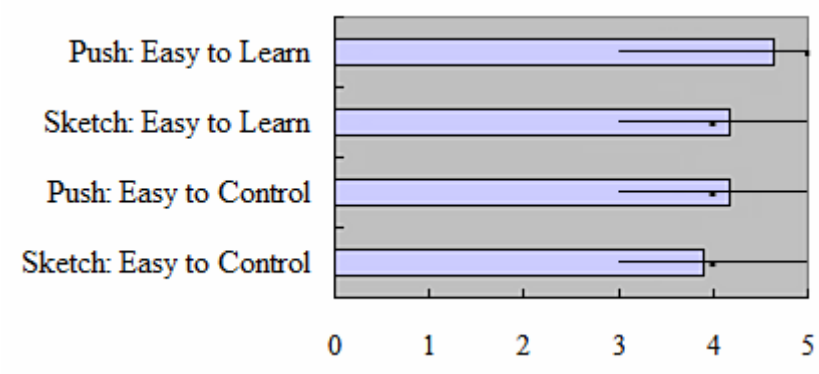

Figure 52 – User Study 1: Sketch-lines vs. Push/Rotate questionnaire results. The bar shows the median value, the thin black line shows the minimum and maximum.

4.5 User Study 2: Sketch-Line versus Tracing and Painting for 2D Region Delineation

The second user study is a comparison of sketch-line region delineation against two other common region delineation techniques: tracing and painting.

Users were asked to delineate 2D contours using one of the 3 techniques. The time taken to delineate, the editing time required, as well as some other model specific quantities were measured.

A 2D delineation task was chosen for several reasons. The most basic reason was that an implementation of 3D tracing and 3D painting was not available. However, if the implementations were available, there would be no guarantee that the software used the same type of input device or that all implementations would support real-time delineation. In addition, the implementations may also each require different input actions (e.g. a key press and/or mouse wheel) to control envelope depth. A 2D contour delineation task, on the other hand, can be performed in real time for all three models, has no depth control requirement, and uses essentially the same interactions as the 3D delineation task. Also, the 2D task is very simple to explain and to grasp by naive users. Finally, designing target contours and measuring delineation time as well as delineation precision was all simplified in 2D.

First, a simple explanation of tracing and painting is in order. Tracing is exactly as it sounds – the user uses a mouse or some other input device and moves the cursor along the target contour, tracing out a smooth spline curve in an attempt to match the given contour (Figure 53).

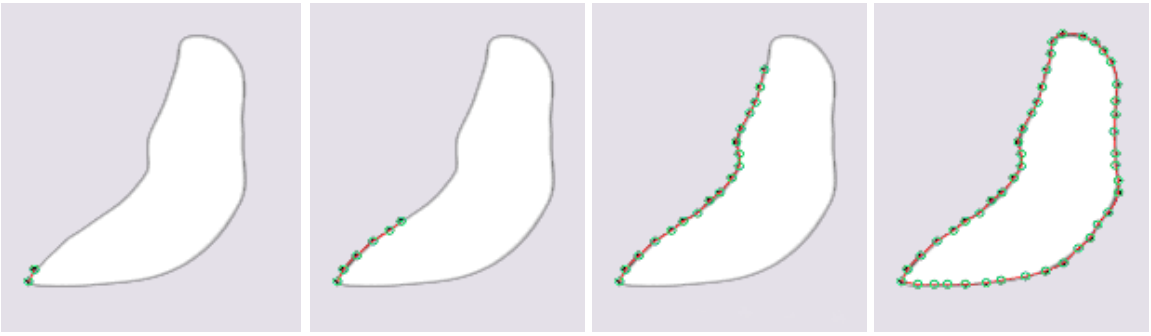

Figure 53 – Contour Outlining: Tracing a spline curve around a target contour in a 2D window. The control points are shown in green.

The user can, at any time, stop and select a spline control point and edit the shape of the spline by repositioning the control point. The user may also use a slider to change the distance between control points as the tracing proceeds.

In painting, the uses moves a circle (a “paint brush tip”) around the interior of the target contour and the circles are blended together (the blended circles are the “paint”) forming a closed contour (Figure 54 top). To edit the painted contour, the user holds down the right mouse button and the circle becomes an “eraser” (Figure 54 bottom). The user may also use a slider to change the size of the paint circle (i.e. brush tip). Tracing and Painting were chosen as the two alternative interaction models as they are both common and familiar to users, even from childhood, and both are simple to learn.

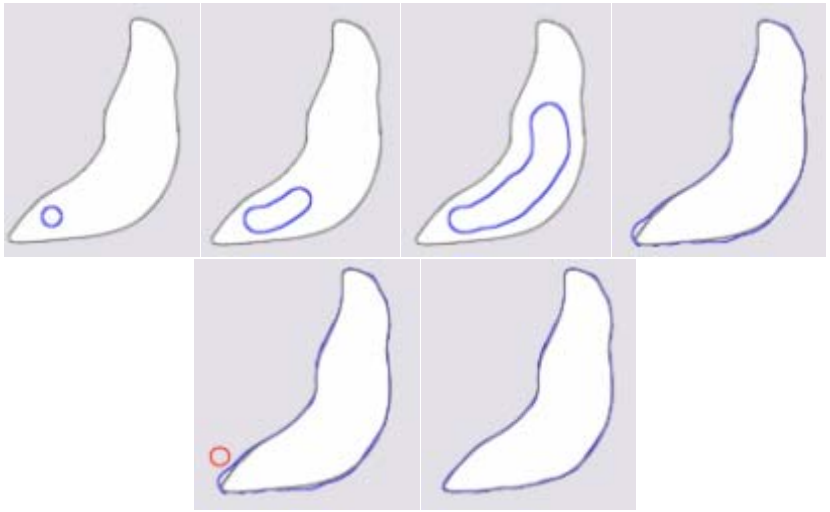

Figure 54 – Contour Outlining: Top - painting the interior of the target contour using a circular “brush tip”. Bottom – pressing and holding the right mouse button changes the circle to an “eraser” or “paint remover” allowing the painted contour to be edited.

Finally, it should be noted that although the task of delineating a contour in a "clean" (i.e. noise-free) image can be automatically performed using well known segmentation algorithms, for noisy medical images (which is the norm), automatically or semi-automatically delineating a region of interest is an open research question. Many existing segmentation algorithms use a user-drawn region as a starting point.

Similar to user study 1, 11 people (undergraduate and graduate computer science students) participated in the within-subjects study. The 11 students in this study were different than the students in the first user study. Out of 11 users, 8 were male and 3 were female with an average age of 22 years, 35 hours of mouse usage per week and 8 hours of video games per week. The study took approximately one hour to complete. Each trial consisted of 10 contour matching tasks, with contours of different shapes and lengths and shape complexity (Figure 55). The users were asked to delineate each target contour as quickly and as precisely as possible. In one trial, the users could edit their result. Each user performed 3 trials and the delineation technique (sketch, trace, paint) as well as the presentation order of the contours was randomized. Each user was allowed to practice each of the 3 techniques. Once the trials were completed, users were asked to fill out a questionnaire.

Figure 55 – User Study2: Contours used for test cases for users to trace/sketch/paint on.

Due to the nature of the sketch-line technique, the first hypothesis formed was that the sketch-line model would be more precise (without editing) than paint and trace. Precision was measured by calculating the 2-sided Hausdorff distance [7] between the user contour and the exact contour. A second hypothesis was that sketching would be faster (with editing) than paint and trace and that the editing time would be less. A final hypothesis was that sketching would require fewer control points to form the spline curve than tracing. The results summarized in Table 5 and Table 6.

In the first trial, users were asked to delineate the target contour as quickly and accurately as possible but were not allowed to edit their results. The delineation technique (sketch-line, tracing, and painting) as well as the order of presentation of the target contours was randomized.

Sketch vs. Trace and Paint	Control Points	Number of Parameter Changes	AVG Time	Precision Error (mm)
AVG: Trace	26.53	0.00	15.41	5.33
AVG: Paint	0.00	1.11	23.12	5.47
AVG: Sketch	13.84	0.00	14.10	3.89
STD: Trace	2.26	0.02	7.55	1.58
STD: Paint	0.00	0.72	9.96	1.36
STD: Sketch	1.62	0.00	4.19	1.38
T-TEST: Sketch vs. Trace	-16.88	N/A	-0.70	-3.62
P-TEST: Sketch vs. Trace	0.00	N/A	0.50	0.00
Probability: Sketch vs. Trace	100.00	N/A	62.92	99.63
T-TEST: Sketch vs. Paint	N/A	N/A	-3.51	-3.90
P-TEST: Sketch vs. Paint	N/A	N/A	0.01	0.00
Probability: Sketch vs. Paint	N/A	N/A	99.56	99.77
T-TEST: Trace vs. Paint	N/A	-5.09	-3.01	0.26
P-TEST: Trace vs. Paint	N/A	0.00	0.01	0.80
Probability: Trace vs. Paint	N/A	99.96	98.97	44.35

Table 5 – User Study2: Sketch-lines vs. Tracing and Painting without editing. Parameter changes refer to changes in the control point spacing for Trace and changes in the brush tip radius for Paint.

The results of the first trial are presented in Table 5. The first hypothesis was validated ($p < 0.005$ for sketch vs. trace and $p < 0.003$ for sketch vs. paint). The sketched contour was more accurate than the traced or painted contour. In Table 5, the total time taken to perform the contour delineations is presented when the users were not permitted to edit their results. In this case, sketch-lines were faster than either trace or paint but this result was only statistically significant for sketch vs. paint ($p < 0.02$).

Sketch vs Trace and Paint	Control Points	Number of Parameter Changes	Edit Time	AVG Time
AVG: Trace	27.87	0.01	16.37	33.66
AVG: Paint	0.00	0.94	14.70	41.29
AVG: Sketch	14.50	0.00	8.88	25.39
STD: Trace	4.37	0.02	7.18	9.33
STD: Paint	0.00	0.77	4.25	11.19
STD: Sketch	1.83	0.00	4.95	6.36
T-TEST: Sketch vs. Trace	-11.67	N/A	-4.12	-5.53
P-TEST: Sketch vs. Trace	0.00	N/A	0.00	0.00
Probability: Sketch vs. Trace	100.00	N/A	99.84	99.98
T-TEST: Sketch vs. Paint	N/A	N/A	-4.60	-7.21
P-TEST: Sketch vs. Paint	N/A	N/A	0.00	0.00
Probability: Sketch vs. Paint	N/A	N/A	99.92	100.00
T-TEST: Trace vs. Paint	N/A	-3.97	0.82	-3.18
P-TEST: Trace vs. Paint	N/A	0.00	0.43	0.01
Probability: Trace vs. Paint	N/A	99.80	67.49	99.23

Table 6 – User Study2: Sketch-lines vs. Tracing and Painting with editing.

The second hypothesis was also validated. The results are presented in Table 6. Both the average time required to delineate the contours using sketch-lines as well as the average editing time required were significantly less for sketch-lines than trace ($p < 0.003$) and paint ($p < 0.001$).

The final hypothesis was also validated and the results are presented in Table 5 and Table 6 (under the column heading “Control Points”). Sketch-lines required significantly fewer control points to delineate the contours than tracing ($p < 3.9E-07$).

Upon completion of all trials the participants were asked to fill out the questionnaire and indicate their level of agreement or disagreement with each statement using a 5-point Likert scale. The questionnaire asked the participants whether each rotation technique was easy to control and whether it was easy to learn. The results are presented in Table 7 and Figure 56. All participants scored Tracing and Sketching both “easy to control” and “easy to learn”.

Painting scored slightly lower on “easy to control”. The participants were also asked to state their favourite technique. The expectation here was that since tracing and painting techniques are similar to simple pen tracing and painting on paper and almost everyone has experience with this in childhood, then sketch-lines would not be chosen as the favourite technique despite its superior performance. However, 4/11 users preferred sketching while 2/11 liked painting and 5/11 preferred tracing.

Some comments were that painting is easy to use for a quick rough outline of a target contour. People who liked sketch commented that it was fast, precise, easy to edit, easy to learn, easy to visualize and convenient. Those who liked tracing commented that it was efficient and easy to focus resulting in fewer mistakes. Participants were also asked to name their least favourite technique. The results were that 7/11 disliked painting commenting that painting was hard to predict and control precisely without making mistakes. Only one user disliked sketching because it required the user to sketch precisely. For tracing, 3/11 disliked it and commented that it was hard to trace using the mouse and get an accurate result.

5-Point Likert Scale	Trace	Sketch	Paint
AVG: Easy to Learn	4.91	4.82	4.36
STD: Easy to Learn	0.30	0.40	0.92
AVG: Easy to Control	4.09	4.27	3.18
STD: Easy to Control	0.94	0.79	1.33

Table 7 – User Study2: Sketch-lines vs. Tracing and Painting questionnaire results.

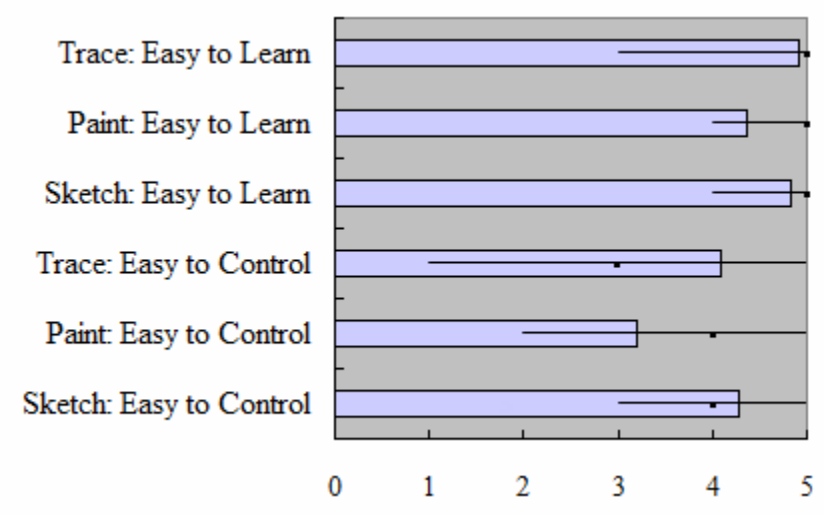

Figure 56 – User Study2: Sketch-lines vs. Tracing and Painting questionnaire results.

Chapter V – Conclusion and Future Work

This thesis presented a sketch line interaction model for visualizing and analyzing 3D images. The sketch-line model provides a single consistent interaction model that can be used for several common volume image visualization and analysis tasks. Specifically, it overcomes existing problems of image slice positioning and 3D region delineation found in many medical visualization packages due to overly complex GUI's and multiple and limited interaction models that are based on volume-relative input actions rather than object-relative input. It provides a more seamless bridge between a 3D volume rendering view of the data and 2D image slice views for cross section views. Furthermore, by combining the interaction model with a spline curve, a subdivision surface and an extrusion process, the user is able to create accurate and editable geometric meshes that delineate and surround 3D regions of interest. The sketch-line model is simple to learn and is precise. The torus rotation model works with the sketch-line image slice placement to provide precise editing of slice orientation.

Sketch-lines are designed to be drawn with a mouse or pen or touch screen input device – there is no need for expensive high-degree-of-freedom devices which can be uncomfortable to use and lead to shoulder strain and fatigue [22]. Its aim is to avoid tedious manual editing and manipulation of objects that are manipulated based on volume coordinates rather than relative to data object surfaces. The results of two user studies support the effectiveness and efficiency of the sketch-line interaction model when compared to other interaction techniques.

Furthermore, several experiments were performed in an effort to demonstrate the potential uses of the sketch-line model when it is combined with volume rendering and used in a multiple window system for image slice exploration of volume images, and when it is combined with a subdivision surface and an extrusion process for 3D region delineation.

5.1 Future Work

There are several areas for improvement and extension to the sketch line system. Firstly, it is currently somewhat difficult to sketch on narrow anatomical structures, such as small arteries. An accurate zoom feature would be beneficial in these situations. Also, in noisy medical images, the volume rendering of objects results in "fuzzy" surfaces are also difficult to sketch. More surface samples within a larger region around the cursor need to be taken as the user sketches. To further reduce the noise, a simple convolution operation with a Gaussian blurring could be used on the collected samples. Currently VTK does not support a transfer function GUI and volume rendering values are all hard-coded. In addition, the volume rendering quality in VTK is rather poor. A possible solution would be to port the sketch-line system to MeVisLab [43] which supports interactive, high-quality volume rendering as well as many VTK C++ classes. Furthermore, currently the envelopes can be created and manipulated in real time. However, both the cutaway operation and the deformable model segmentation algorithm are slow by today's standards and need to be GPU accelerated. Fortunately, both of these algorithms can be readily parallelized.

Finally, currently there is no prevention of local and global self intersection of the various envelopes. This is a common problem of parametric surface representations. Additional constraints need to be imposed to prevent local intersection of sketch-lines and profile curves.

Useful extensions to the system include the ability to add a new sketch line between two existing sketch lines as well as adding new sketch lines to an existing sketch line defined region in oredr to form protrusions or branches. Finally, further user studies are needed that directly compare GPU accelerated versions of 3D sketch lines, 3D tracing and 3D painting techniques using different kinds of input devices (pen, tablets and force feedback input devices). Ideally these studies would have medical professionals as participants.

References

[1] Aliroteh, M., & McInerney, T. (2007). *SketchSurfaces: Sketch-Line Initialized Deformable Surfaces for Efficient and Controllable Interactive 3D Medical Image Segmentation.* In ISVC '07: Proceedings of the 3rd international conference on Advances in visual computing, Lake Tahoe, NV, 542-553.

[2] Araújo, B. D., Jorge, J., Sousa, M. C., Samavati, F., & Wyvill, B. (2004). *MIBlob: a tool for medical visualization and modelling using sketches.* In SIGGRAPH '04*:* ACM SIGGRAPH 2004 Posters, Los Angeles, California, 107.

[3] Autodesk. (2005). *Autodesk Maya.* Retrieved December 8, 2011, form http://usa.autodesk.com/maya/

[4] Autodesk. (2009). *Autodesk 3ds Max Products.* Retrieved December 8, 2011, from http://www.autodesk.com/3dsmax

[5] Bade, R., Ritter, F., & Preim, B. (2005). *Usability Comparison of Mouse-based Interaction Techniques for Predictable 3d Rotation*. In SG '05: Proceedings of 5th International Symposium on Smart Graphics, Frauenwörth Cloister, Germany, 138–150.

[6] Barraclough, R. W., Binkley, D., Danicic, S., Harman, M., Hierons. R. M., Kiss, Á., Laurence, M., & Ouarbya, L. (2010). *A trajectory-based strict semantics for program slicing.* Theor. Comput. Sci., 411(11-13), 1372-1386.

[7] Beauchemin, M., Thomson, K. P. B., Edwards, G. (1998). *On the Hausdorff distance used for the evaluation of segmentation results.* In Canadian Journal of Remote Sensing, 24(1), 3-8.

[8] BlenderFoundation. (2011). *Blender 2.60.* Retrieved December 8, 2011, form http://www.blender.org

[9] Bloomenthal, J., & Wyvil, B. (Eds.). (1997). *Introduction to Implicit Surfaces.* San Francisco, CA: Morgan Kaufmann Publishers Inc.

[10] Bornik, A., Beichel, R., Kruijff, E., Reitinger, B., & Schmalstieg, D. (2006). *A Hybrid User Interface for Manipulation of Volumetric Medical Data.* In 3DUI '06: Proceedings of the 3D User interfaces, Alexandria, VI, 29-36.

[11] Bruckner, S., Grimm, S., Kanitsar, A., & Groller, M. E. (2006). *Illustrative Context Preserving Exploration of Volume Data*. IEEE Transactions on Visualization and Computer Graphics, 12(6), 1559-1569.

[12] Bruyns, C., & Senger, S. (2001). Interactive cutting of 3D surface meshes. Computer & Graphics, 25(4), 635-642.

[13] BWH and 3D Slicer. (2011). *3D Slicer 4.0.* Retrieved December, 8, 2011, from http://www.slicer.org

[14] Chan, S. L., & Purisima, E. O. (1998). *A new tetrahedral tesselation scheme for isosurface generation.* Computers and Graphics, 22(1), 83-90.

[15] Chen, H. J., Samavati, F. F., & Sousa, M. C. (2008). *GPU-based point radiation for interactive volume sculpting and segmentation.* Visual Computer, 24(7), 689-698.

[16] Chen, H. L. J., Samavati, F.F., Sousa, M. C., & Mitchell, J. R. (2006). *Sketch-Based Volumetric Seeded Region Growing*, In SBM '06: Proceedings of the 3rd Eurographics Workshop on Sketch-Based Interface and Modeling, Vienna, Austria, 123-129.

[17] Coffin, C., & Hollerer, T. (2006). *Interactive Perspective Cut-away Views for General 3D Scenes.* In VR '06: Proceedings of the IEEE conference on Virtual Reality, California, Santa Barbara, 25-28.

[18] Diepenbrock, S., Ropinski, T., & Hinrichs, K. (2011). *Context-aware volume navigation.* In PACIFICVIS '11: Proceedings of the 2011 IEEE Pacific Visualization Symposium, Munster, Germany, 11-18.

[19] Elmqvist, N., & Fekete, J. (2008). *Semantic pointing for object picking in complex 3D environments.* In GI '08: Proceedings of Graphics interface 2008, Windsor, Ontario, 243-250.

[20] Engel, K., Hadwiger, M., Kniss, J. M., Lefohn, A. E., Salama, C. R., & Weiskopf, D. (2004). *Real-time volume graphics.* In SIGGRAPH '04: Proceedings of ACM SIGGRAPH 2004 Course Notes, Los Angeles, CA, 29.

[21] Fuchs, R., Welker, V., & Hornegger, J. (2010). *Non-convex polyhedral volume of interest selection.* Computerized medical imaging and graphics the official journal of the Computerized Medical Imaging Society. 34(2), 105-113.

[22] Gallo, L., Pietro, G. D., Coronato, A., & Marra, I. (2008). *Toward a natural interface to virtual medical imaging environments.* In AVI '08: *Proceedings of the Working Conference on Advanced Visual interfaces*, Napoli, Italy, 429-432.

[23] Glencross, M., Chalmers, A. G., Lin, M. C., Otaduy, M. A., & Gutierrez, D. (2006). *Exploiting perception in high-fidelity virtual environments*. In SIGGRAPH '06: ACM SIGGRAPH 2006 Courses, Boston, Massachusetts, 1.

[24] Grady, L. (2006). *Random walks for image segmentation.* IEEE Trans. on Pattern Analysis and Machine Intelligence, 28 (11), 1768–1783.

[25] Harders, M., Steinemann, D., Gross, M., & Székely, G. (2005). *A hybrid cutting approach for hysteroscopy simulation.* In MICCAI'05: Proceedings of the 8th international conference on Medical image computing and computer-assisted intervention - Volume Part II , Palm Springs, CA, 567-574.

[26] He, C., Lewis, A., & Jo., J. (2007). *A Novel Human Computer Interaction Paradigm for Volume Visualization in Projection-Based Virtual Environments.* In SG '07: Proceedings of the 8th International Symposium on Smart Graphics, Kyoto, Japan, 49-60.

[27] Hinckley, K., Pausch, R., Goble, J. C., & Kassell, N. F. (1994). *A survey of design issues in spatial input.* In UIST '94: Proceedings of the 7th annual ACM symposium on User interface software and technology, Marina del Rey, California, 213–222.

[28] Huff, R., Dietrich, C. A., Nedel, L. P., Freitas, C. M. D. S., Comba, J. L. D., & Olabarriaga, S. D. (2006). *Erasing, digging and clipping in volumetric datasets with one or two hands.* In VRCIA '06: Proceedings of the 2006 ACM international Conference on Virtual Reality Continuum and Its Applications, Hong Kong, China, 271-278.

[29] Jang, Y. & Woo, W. (2011). *Stroke-based semi-automatic region of interest detection algorithm for in-situ painting recognition*. In Proceedings of the 2011 international conference on Virtual and mixed reality: systems and applications - Volume Part II, Orlando, FL, 167-176.

[30] Jung, Y., Recker, R., Olbrich, M., & Bockholt, U. (2008). *Using X3D for medical training simulations.* In Web3D '08: Proceedings of the 13th international Symposium on 3D Web Technology, Los Angeles, California, 43-51.

[31] Kim, K., & Park, J. (2009). *Virtual bone drilling for dental implant surgery training.* In VRST '09: Proceedings of the 16th ACM Symposium on Virtual Reality Software and Technology, Kyoto, Japan, 91-94.

[32] Kitware. (2006). *ParaView*. Retrieved December, 8, 2011, from http://www.paraview.org/

[33] Kitware. (2002). *Visualization Toolkit.* Retrieved December, 8, 2011, from http://www.vtk.org/

[34] Konrad-Verse, O., Littmann, A., & Preim, B. (2004). *Virtual Resection with a Deformable Cutting Plane.* In SimVis '04: Proceedings of Simulation und Visualisierung 2004, SCS, Ghent, 203-214.

[35] Li, W., Ritter, L., Agrawala, M., Curless, B., & Salesin, D. (2007). *Interactive cutaway illustrations of complex 3D models.* In SIGGRAPH '07: Proceedings of ACM SIGGRAPH 2007 papers, San Diego, California, 31.

[36] Li, Y., Sun, J., Tang, C., & Shun, H. (2004). *Lazy snapping.* ACM Transactions on Graphics, 23(3), 303–308.

[37] Loop, C., & Blinn, J. (2006). *Real-time GPU rendering of piecewise algebraic surfaces.* In SIGGRAPH '06: Proceeding of ACM SIGGRAPH 2006 Papers, Boston, Massachusetts, 664-670.

[38] McClymont, J., Shuralyov, D., & Stuerzlinger, W. (2011). *Comparison of 3D Navigation Interfaces.* In VECIMS: Proceedings of Virtual Environments Human-Computer Interfaces and Measurement Systems, Ontario, Canada, 1-6.

[39] McGuffin, M. J., Tancau, L., & Balakrishnan, R. (2003). *Using Deformations for Browsing Volumetric Data.* In VIS '03: Proceedings of the 14th IEEE Visualization 2003, Washington, Ontario, Canada, 401-408.

[40] McInerney, T. (2008). *SketchSnakes: Sketch-Line Initialized Snakes for Efficient Interactive Medical Image Segmentation.* Computerized Medical Imaging and Graphics, 32(5), 331-352.

[41] McInerney, T., & Broughton, S. (2006). *HingeSlicer: Interactive Exploration of Volume Images Using Extended 3D Slice Plane Widgets.* In GI '06: Proceedings of Graphics Interface 2006, Quebec, Canada, 171-178.

[42] McInerney, T., & Crawford, P. (2010). *RibbonView: Interactive Context-Preserving Cutaways of Anatomical Surface Meshes.* In ISVC '10: Proceedings of the 6th International Symposium on Visual Computing, Las Vegas, NV, 533-544.

[43] MeVis Medical Solutions. (2011). *MeVisLab 2.2.1.* Retrieved December, 8, 2011, from http://www.mevislab.de/

[44] Moritz, E., Hagen, H., Wischgoll, T., & Meyer, J. (2007). *Usability of multiple degree-of freedom input devices and virtual reality displays for interactive visual data analysis.* In VRST '07: Proceedings of the 2007 ACM Symposium on Virtual Reality Software and Technology, Newport Beach, California, 243-244.

[45] Nakamura, T., & Igarashi, T. (2008). *An application-independent system for visualizing user operation history.* In UIST '08: Proceedings of the 21st Annual ACM Symposium on User interface Software and Technology, Monterey, CA, 23-32.

[46] Nguyen, H. (2007). *GPU Gems 3.* Boston, MA: Pearson Education Inc.

[47] Oh, J., Stuerzlinger, W. (2005). *Moving objects with 2D input devices in CAD systems and Desktop Virtual Environments.* In GI '05: Proceedings of Graphics Interface 2005, Victoria, British Columbia, 195-202.

[48] Oh, J., Stuerzlinger, W., & Dadgari, D. (2006). *Group Selection Techniques for Efficient 3D Modeling.* In 3DUI '06: Proceedings of the 3D User Interfaces, Arizona, Tucson, 95-102.

[49] Paragios, N. (2003). *A levelset approach for shape-driven segmentation and tracking of the left ventricle.* IEEE Transactions on Medical Imaging, 22(6), 773–776.

[50] Pixmeo. (2011). Osirix 4.0. Retrieved January, 5, 2011, from http://www.osirix-viewer.com/index.html

[51] Pühringer, N. (2009). *Sketch-based Modelling for Volume Visualization.* Retrieved January, 5, 2012, from InstituteofComputerGraphicsandAlgorithms, Vienna University of Technology, http://www.cg.tuwien.ac.at/research/publications/2009/puehringer-2009-sbm/puehringer-2009-sbm-paper.pdf

[52] Reuter, P., Riviere, G., Couture, N., Mahut, S., & Espinasse, L. (2010). *ArcheoTUI—Driving virtual reassemblies with tangible 3D interaction.* Journal on Computing and Cultural Heritage, 3(2), 1-13.

[53] Ropinski, T., Steinicke, F., & Hinrichs, K. (2005). *Interactive Importance-Driven Visualization Techniques for Medical Volume Data.* In VMV'05: Proceedings of the 10th International Fall Workshop on Vision, Modeling and Visualization, Erlangen, Germany, 273-280.

[54] Schmidt, R., Singh, K., & Balakrishnan, R. (2008). *Sketching and Composing Widgets for 3D Manipulation.* Computer Graphics Forum, 27(2), 301-310.

[55] SensAble. (1993). *Sensable*. Retrieved December, 8, 2011, from http://www.sensable.com

[56] SenseGraphics AB. (2004). *H3D.org Open Source Haptics.* Retrieved December, 8, 2011, from http://www.h3dapi.org

[57] Sherbondy, A., Houston, M., & Napel, S. (2003). *Fast Volume Segmentation With Simultaneous Visualization Using Programmable Graphics Hardware.* In VIS '03: Proceedings of the 14th IEEE Visualization 2003, CA, USA, 171-176.

[58] Sifakis, E., Der, K. G., & Fedkiw, R. (2007). *Arbitrary cutting of deformable tetrahedralized objects.* In SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation, San Diego, California, 73-80.

[59] Teather, R. J., & Stuerzlinger, W. (2008). *Assessing the Effects of Orientation and Device on (Constrained) 3D Movement Techniques.* In 3DUI '08: Proceedings of the 2008 IEEE Symposium on 3D User Interfaces, Reno, NE, 43-50.

[60] Teather, R. J., & Stuerzlinger, W. (2007). *Guidelines for 3D positioning techniques.* In Future Play '07: Proceedings of the 2007 Conference on Future Play, Toronto, Canada, 61-68.

[61] The Center for Integrative Biomedical Computing at the University of Utah. (2011). *ImageVis3D 2.0.1.* Retrieved December, 8, 2011, from http://www.sci.utah.edu/cibc/software/41-imagevis3d.html

[62] Tory, M., Moller, T., Atkins, M. S., & Kirkpatrick, A. E. (2004). *Combining 2D and 3D views for orientation and relative position tasks.* In CHI '04: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Vienna, Austria, 73-80.

[63] Ulinski, A., Zanbaka, C., Wartell, Z., Goolkasian, P., & Hodges., L.F. (2007). *Two handed selection techniques for volumetric data.* In 3DUI '07: Proceedings of IEEE Symposium on 3D User Interfaces 2007, Charlotte, NC, 107–114.

[64] University of Münster and Linköping University. (2011). *Voreen volume rendering engine.* Retrieved December, 8, 2011, from http://www.voreen.org/

[65] Visage Imaging GmbH. (2011). *Amira 5.4*. Retrieved December, 8, 2011, from http://www.amira.com

[66] Viola, I., Kanitsar, A., & Gröller, M. E. (2004). *Importance-Driven Volume Rendering.* In VIS '04: Proceedings of the conference on Visualization '04, Vienna University of Technology, Austria, 139-146.

[67] Weiskopf, D., Engel, K.,& Ertl, T. (2003). *Interactive clipping techniques for texture-based volume visualization and volume shading.* IEEE Transactions on Visualization and Computer Graphics, 9(3), 298-312.

[68] Xiang, D., Tian, J., Yang, F., Yang, Q., Zhang, X., Li, Q., & Liu, X. (2011). *Skeleton Cuts – An Efficient Segmentation Method for Volume Rendering.* IEEE Transactions on Visualization and Computer Graphics, 17(9), 1295-1306.

[69] Zachow, S., Gladilin, E., Sader, R., & Zeilhofer, H. (2003). *Draw and cut: intuitive 3D osteotomy planning on polygonal bone models.* International Congress Series, 1256, 362–369.

[70] Zhang, Q., Eagleson, R., & Peters, T. M. (2011). *Rapid scalar value classification and volume clipping for interactive 3D medical image visualization.* The Visual Computer: International Journal of Computer Graphics. 27(1), 3-19.

[71] 3D MAX-TUTORIALS.COM. (n.d.). *Using Transform Gizmos.* Retrieved December, 8, 2011, from http://www.3dmax-tutorials.com/Transform_Gizmo.html