A Comparison of 2D and 3D Interfaces for Editing Surfaces Reconstructed from Contours

by

Julie Frances Waterhouse

A thesis

presented to the University of Waterloo

in fulfilment of the

thesis requirement for the degree of

Master of Mathematics

in

Computer Science

Waterloo, Ontario, Canada, 1996

Abstract

The last decade of computer technology has seen the proliferation of computer graphics applications. As technology advances, there is a growing fascination with three-dimensional (3D) object representations that likely comes from their greater ability to match "real life" than their two-dimensional (2D) counterparts. Unfortunately, the benefits of 3D editing are not without a price. Most techniques for manipulating objects in a 3D environment are developed for conventional hardware configurations that use 2D input devices and CRT displays. The difficulties lie in mapping 3D spatial relationships to 2D displays, and in mapping 2D user input to 3D object manipulation. This mapping problem is somewhat mitigated by adding constraints to the degrees of freedom in the manipulation task. 3D surfaces that have been reconstructed from contours are interesting to consider as targets of 3D interaction because they provide an inherent constraint on manipulation: point motion is restricted to a plane.

As part of my research, I implemented an interactive contour editor to edit 3D surfaces that were reconstructed from planar contours. More precisely, the editor is a tool for visualising a surface derived from a set of serial sections, and for removing deformations from this surface. It was designed specifically to remove artefacts from medical images of arteries.

I used the interface from my editor in an experiment that tested whether users were faster and more accurate at manipulating surfaces in a 2D environment or a 3D environment. At the outset of this study, I predicted that 2D would be better for editing deformations of a 2D nature. That prediction was borne out by my experimental results. I had also hoped that 3D would be superior as an editing environment for correcting deformations of a 3D nature. However, the 2D character of the data had a stronger effect on performance than did the 3D character of the deformation. Despite the inherent constraints in the surfaces, participants were faster at editing in 2D for all types of deformations, while maintaining a consistent accuracy between 2D and 3D. Participants did perceive a 3D environment to be better than a 2D environment for manipulating a group of points that spanned multiple contours, although this was not reflected in the quantitative results. The intuitive preference for 3D in this situation leads me to believe that it is worth continuing the search for a natural and effective interface for editing surfaces in a 3D environment.

Acknowledgements

I would like to thank the IBM Toronto Lab for providing financial support for this research.

I am most grateful to Rick Kazman and Stephen Mann for their patience and guidance as my supervisors. The efforts of Richard Bartels, William Cowan and George Labahn, who acted as readers for my thesis, were also invaluable.

I am indebted to Ross Ethier for involving me in the artery project, and appreciate the support I received from all of the associated people at the University of Toronto and the University of Western Ontario, in particular Jennifer Moore and David Steinman.

I am beholden to Bernhard Geiger for allowing me to incorporate his *nuages* software into my code; without it, I would have been unable to construct my surface editor.

I could not have completed this thesis without the encouragement of my family and friends. Special thanks to Andrew for his unconditional love and support even when I was at my maximum stress level. Thanks also to Sandra Loop for her help with my experiment design, and Rob Kroeger for providing insight into some flaws in my pretests. Thanks to all my friends whose patience I tried as participants in my experiments - especially Andrew, who became my "expert user."

Special mention must go to my Force 100 Tiger Spirit, Doc, who was unfailing in her devotion, as well as to SCOOPS for their supply of Funky Chunky Chocolate ice cream that kept me going through that last, long summer when Steve was not making one of his glorious batches of chocolate ice cream.

And of course, I must not forget the goats, without whose arteries I would have had no data.

Trademarks

Table of Contents

List of Tables

List of Figures

1. Introduction

The last decade of computer technology has seen the proliferation of computer graphics applications. They appear in such diverse fields as surgical training, flight simulation, computer-aided design (CAD), financial analysis and medical visualisation. Furthermore, as technology advances, there is a growing fascination with three-dimensional (3D) object representations. The appeal of 3D representations likely comes from their greater ability to match "real life" than their two-dimensional (2D) counterparts. Using a computer to reproduce the world around us has long been a "Holy Grail" of computer graphics. This sentiment is echoed in the popular enthusiasm for virtual reality.

The first of the 3D computer graphics applications[1] were primarily concerned with static visualisation. Later, hardware technology improved, making machines faster and making real-time object interaction possible. Such interaction is used not only for visualisation, but also for *editing*. Whereas visualisation describes viewing changes such as rotating and translating an object, editing refers to changes that deform the actual shape of the object. 3D editing techniques will be the focus of this thesis.

For visualisation used to envision a real-world object it in its entirety, a 3D view has an advantage over its 2D analogue [Wic89]. A 3D environment allows an object to be perceived as a whole, while a 2D view is restricted to only a cross-section or profile of an object.

For object editing, 3D interaction has the benefit of providing context for the user. It combines the three spatial dimensions in a natural way. Conversely, modelling packages that use three separate 2D orthogonal views for interaction (with a perspective view reserved for observation) are notoriously difficult to use as the user must integrate the different orthogonal views conceptually [Coo84]. The interaction style of a 3D environment more closely matches the user's mental model of handling objects in his daily activities [Hut85], and capitalises on the user's lifetime of experience in negotiating a 3D environment and manipulating 3D objects.

Unfortunately, the benefits of 3D editing are not without a price. Most techniques for manipulating objects in a 3D environment are developed for conventional hardware configurations that use 2D input devices and CRT displays. The difficulties lie in mapping 3D spatial relationships to 2D displays, and in mapping 2D user input to 3D object manipulation [Her92]. This *mapping problem* hampers the user's ability to indicate or perceive depth, making it awkward to interact with an object and, in particular, with specific points on an object's surface. It is particularly difficult to translate along an axis parallel to the line of sight, because the axis projects onto a point on the screen rather than a direction [Phi92].

I would like to determine whether it is possible to exploit the advantages of 3D interaction without falling prey to the mapping problem. Manipulating objects in three-dimensional space is a daunting task, having six degrees of freedom corresponding to the three axes of rotation and the three axes of translation [War90]. The simultaneous manipulation of three degrees of

[1] In the context of this thesis, the term "3D" will be used to refer to a 2D projective view of a 3D object, rather than a stereoscopic view.

freedom, let alone all six, is difficult. Studies have shown that users perform better and report a feeling of greater control in the environment when the degrees of freedom are restricted by constraints [Hou92].

On a computer, a user can manipulate objects either *directly*, or *indirectly*. In both interaction styles, attempts have been made to introduce constraints that mitigate the effects of the mapping problem.

Direct manipulation gives the user the impression of actually handling an object within the 3D environment. To accomplish this, however, the user must indicate a location in all three dimensions. Depth is most difficult, because it does not map well to existing 2D input devices. Various research groups have designed special software tools (herein referred to as *widgets*) to assist in this task by constraining 3D interaction [e.g., Sni92, Hou92, Her92, and Con92]. Widgets are encapsulations of geometry (their physical properties) and/or behaviour (the actions they perform) that are used to control or display information about application objects [Con92].

Indirect manipulation can also avoid some of the mapping difficulties by providing separate tools to control changes in each dimension. These are usually standard widgets such as sliders and dials [Che88] for tasks like translating or rotating an object. Unlike direct manipulation widgets, however, these tools are located outside of the 3D environment.

In this thesis, I examine the particular problem of surface editing using direct manipulation techniques. I strive to determine through experimentation whether a 2D environment or a 3D environment is more suitable for editing three-dimensional surfaces reconstructed from medical images. Such surfaces are created by joining together slices containing contours that were formed by cutting through an object with parallel planes. These slices are often referred to in the literature as *serial sections*. The lines used to connect points on contours in adjacent slices are often constructed in such a way as to form triangles. The surface generated for the object is thus a *triangular mesh*.

3D surfaces that have been reconstructed from contours[1] are interesting to consider as targets of 3D interaction because they provide an inherent constraint on manipulation. Due to the method used to create surfaces from contours, all the points on the resultant mesh lie in parallel planes. Thus the movement of points on a contour can occur only within the plane in which the contour lies. All manipulation of points on the surface is now constrained to two dimensions, so there is no longer a need to map 2D user input to 3D object manipulation for this type of interaction.

The tasks of selecting a point and of moving a point are fundamental to all surface editing operations. I therefore analysed these two tasks in an attempt to weigh the advantages and disadvantages of each environment for editing. The central research question of this thesis involves determining whether 3D editing benefits by constraining the data points to lie on a set of 2D planes. 2D editing of sequential slices may be more natural because it reflects the

[1] The assumption is that the surface being edited is the triangular mesh that results directly from the reconstruction, and not a smoother surface that has been fit to that mesh.

underlying structure of the data. On the other hand, 3D manipulation provides a context for the task; each contour is seen in relation to its neighbours.

Many applications employ 3D surface reconstruction from planar contour data. Mining engineers use contours to map mine workings. In CAD applications, a technique called *lofting* uses a set of contours to specify the geometry of an object. Biologists use serial sections through an organism to better understand its overall shape. Of interest in this research are the several imaging techniques of clinical medicine that provide data as a series of slices through an object. These include computed tomography (CT), ultrasound, as well as nuclear magnetic resonance (NMR) and magnetic resonance imaging (MRI).

I implemented a surface editor called SLICE (SimpLe Interactive Contour Editor) for the experimental component of the research, as well as for practical application in removing artefacts from MRI and CT scans of arteries. A fluid dynamics problem provides the context for the research in this thesis [Eth92]. Data in the form of serial sections through arteries are converted into a triangular surface mesh, which is edited to remove unwanted surface features such as bumps and dents. The resultant surface is transformed into a *tetrahedral volume mesh* (a partitioning of the space contained within the surface mesh into tetrahedral elements) for use in a fluid-flow analysis of the artery. The ultimate goal of that analysis is to determine whether a connection exists between the fluid flow in the artery and a disease known as intimal hyperplasia (thickening of the artery wall) [Eth92].

Artery blockage that interrupts the supply of blood to vital organs (chiefly the brain and heart) is responsible for roughly half of the deaths in most developed countries [Car78]. To renew blood flow in the artery, surgeons frequently implant living tissue, or grafts, to bypass the obstructions. Although the grafts are successful in the short-term, a significant number of them fail one or more years post-operatively [Eth92]. A primary cause of this failure is *intimal hyperplasia,* a progressive thickening of the parent vessel wall that eventually leads to occlusion of the graft *lumen* (the cavity inside the "tube" formed by the artery wall) [Dob94]. This usually occurs at the *anastomosis*, or junction between the original artery and the graft, and is then known as *anastotomic intimal hyperplasia*. A research project being conducted jointly by the University of Western Ontario and the University of Toronto (herein referred to as the UWT project) is attempting to model the flow fields in normal and diseased graft-artery anastomoses to better understand the role of hemodynamic effects (i.e., effects of blood flow) in graft failure. In the long-term, it is hoped that the research will lead to recommendations for a graft geometry designed to reduce the incidence of graft failure, as well as the development of hemodynamic criteria for early detection of intimal hyperplasia in patients [Eth92].

The data used to develop the artery models come from MRI and CT scans. In the future, histological serial sections (data from samples of sliced tissue) may be used as well. From these data, a tetrahedral volume mesh is produced for the finite element analysis of fluid flow in the vessel[1]. Inaccuracies in the data-gathering process, however, introduce artefacts (most often appearing as bumps and dents on the surface) into the contour data. To correct these errors, members of the UWT project create an intermediate triangular surface mesh that is edited to

[1] See Shames' book for a treatment of finite elements [Sha89].

eliminate the artefacts before proceeding with the generation of the volume mesh. The surface mesh must be edited by an expert user who can make informed decisions about how the surface needs to be modified.

In Chapter 2, I describe the issues in surface reconstruction that were of importance in selecting a reconstruction algorithm for the application. I discuss data formats and their consequences on the surface reconstruction algorithm used. I also present issues in 2D and 3D interaction techniques, and justify the use of direct manipulation.

In Chapter 3, I describe the implementation of SLICE, beginning with the requirements for the program. I explain the interface design that emerged from the task analysis and was chosen for the experiments.

In Chapter 4, I focus on the experimental portion of the research. I describe the purpose, subjects, design, experimental conditions, results and analysis of the experiments.

In Chapter 5, I present the conclusions of the research. I discuss future enhancements to the interface and their implications.

2. Background and Theory

In this thesis, I focus on the surface editing aspect of the UWT artery project. I have implemented a surface editor that first reconstructs a surface (triangular mesh) from contour data using existing software. The construction of this mesh is a multi-stage process that I discuss in some detail in this chapter.

With the surface editor, the user can move or delete points on the intermediate mesh, thus creating the final surface that will be used to produce the finite element volume mesh. In this chapter, I present issues in interaction that arise when users manipulate the mesh to change its shape.

The generation of that volume mesh, and the ensuing flow analysis are beyond the scope of this thesis, but are dealt with in Cuvelier's book [Cuv86].

2.1 Choosing a Tiler

The input to my editor is a set of planar contours that represent cross-sections through a 3D object. Figure 2.1 illustrates how the contours are formed.

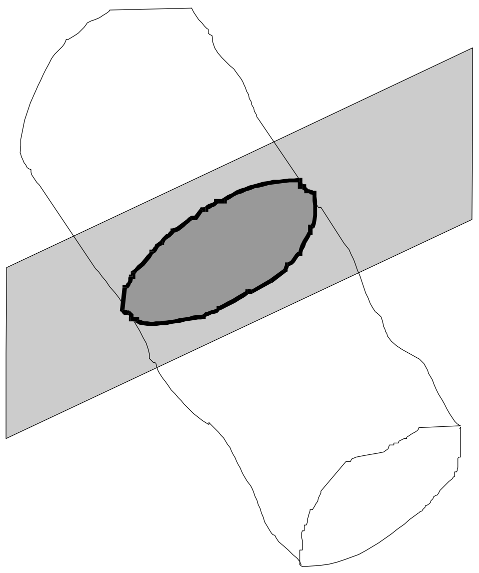


Figure 2-1: Contours are formed by intersecting a cutting plane with an object.

From these contours, I must reconstruct the surface of the object. A mesh generation program, or *tiler*, creates a triangular mesh from the contour data. It builds the mesh by joining points on one contour to corresponding points on a contour in a neighbouring slice as shown in Figure 2.2. I considered several tilers for use in the project including the "Surfaces from Contours" package written by Meyers, Painter and Sloan of the Universities of Washington and

Alabama, Jules Bloomenthal's "Implicit Surface Polygonizer" [Hec92] and a surface meshing package from Mark Jones of Swansea [Jon94]. I chose a tiler called *nuages* that was written at INRIA, France by Bernhard Geiger [Gei93] for incorporation into the surface editor. Of the limited selection of available tilers, it best fulfilled the basic requirements of the project.

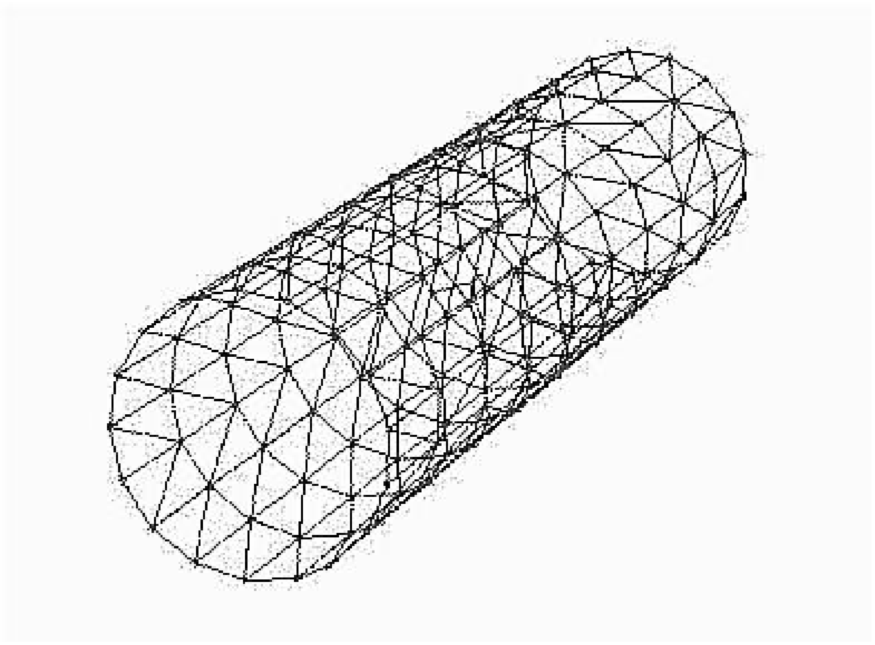


Figure 2-2: Example of a simple surface mesh.

There were several considerations in choosing the tiler that generates the initial surface mesh. The first was availability at no cost. Of the tilers that met this criterion, I sought one that could generate a mesh at interactive speed. This is desirable so that a new mesh can be generated as the user edits the original mesh. The tiler's source code also needed to be readily modifiable so that I could build the editor interface on top of it. In addition, to aid in maintaining interactive speed, it is important that the entire mesh need not be recreated when only a small portion of it is changed. Thus, the tiler code needed to allow for local mesh reconstruction.

The nature of the artery data determines which approach to surface reconstruction is most appropriate for the tiler. There are two approaches to the problem: volume-based and surface-based. Volume-based methods are used when the data are available as a 3D lattice of points. Surface-based approaches require that the data define the intersection of a surface and a plane of sectioning [Mey92]. For this project, data are provided as a set of closed contours from parallel slices through an artery. A tiler that uses a surface-based approach is thus preferable.

The chosen tiler must also solve the basic meshing problems of *correspondence*, *tiling* and *branching*. These are illustrated in Figure 2.3.

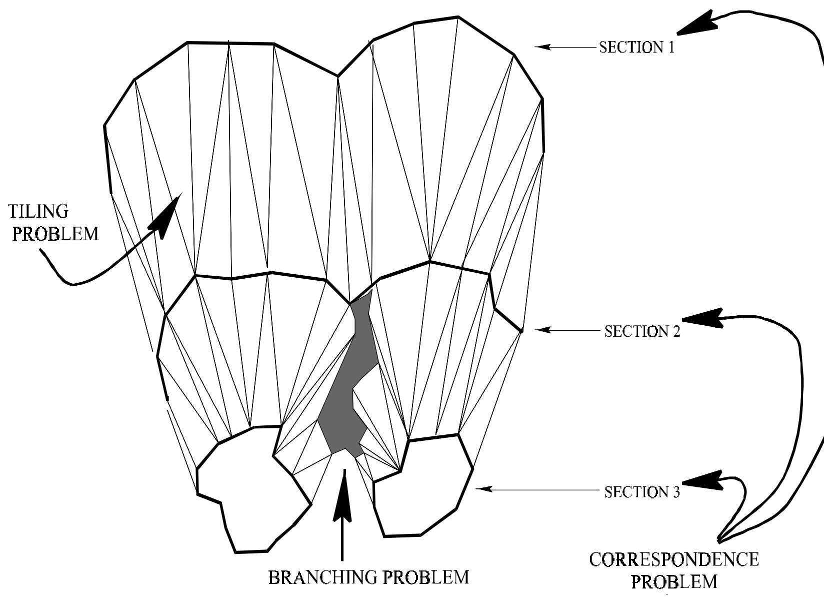


Figure 2-3: These contours illustrate some of the problems encountered in reconstructing a surface.

2.1.1 Correspondence

The *correspondence* problem refers to the issue of deciding which contours should be connected by the surface. Solutions include comparing the shape of contours in adjacent slices, and determining overlap in the plane (e.g., slices made perpendicular to the primary axis of a right circular cylinder have exactly the same shape - a circle, and line up precisely with one another when projected onto a plane). The coarse topology of the final surface is determined by the topological adjacency relationships between the contours of the data set. If there are multiple contours in a section, the contours must be organised into groups representing individual objects.

2.1.2 Tiling

The *tiling* problem deals with how the contours should be connected. Given points on pairs of contours from adjacent sections, the task is to generate the "best" topological adjacency relationship between these points. The difficulty lies in the fact that the tiling problem is severely under-constrained. In other words, there are many different tilings that could interpolate a given set of contours. Some metric must be chosen to determine what is meant by "best" topological adjacency relationship. Some commonly used metrics include area, volume, matching direction, span length, matching normalised arc length, and various non-local metrics. All of these metrics perform poorly with certain pathological examples. There are, however, possible improvements such as normalisation for position, size and small rotations [Mey92].

Ideally, the tiler used by the surface editor should create surfaces without twists. Figure 2.4 illustrates a surface with a twist that was constructed from actual artery data. Figure 2.5 provides a simpler example to illustrate the problem. Twists in the surface are caused by joining the wrong points on adjacent contours and may cause problems when attempting to generate a volume mesh. The tetrahedral volume elements are generated based on the triangles in the surface mesh. If the triangles are twisted, the algorithm for generating the volume elements will produce a poor discretisation of the volume; i.e., the tetrahedra will also be twisted, and will therefore not accurately reflect the flow of forces within the object.

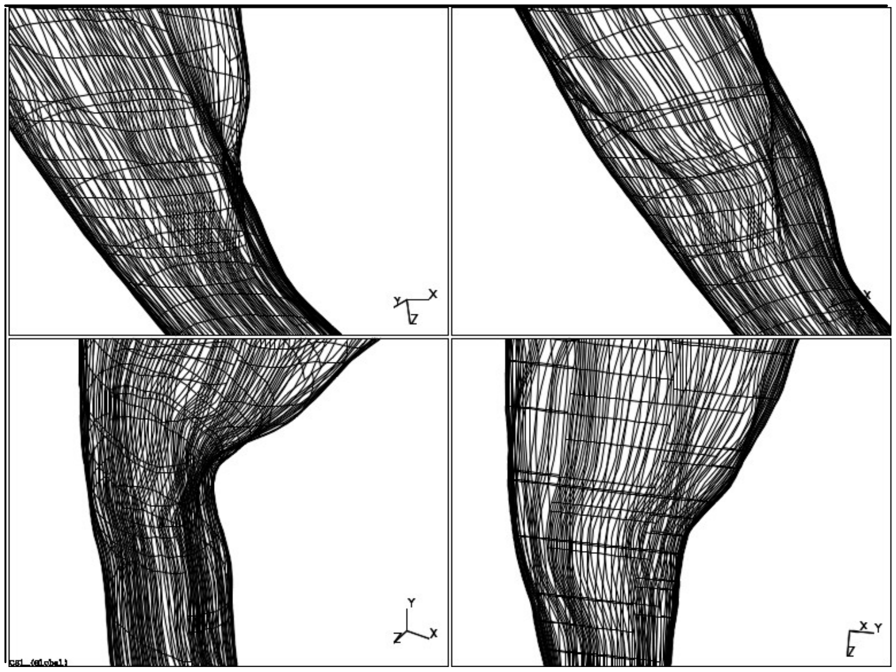


Figure 2-4: A twist in the surface is visible in the longitudinal lines.

In Figure 2.5, the grey, dotted lines show how the two contours should be connected. The black, solid lines show what happens when the surface is twisted. Imagine rotating the top circle by two points while the grey connections are in place. This would result in the black connections.

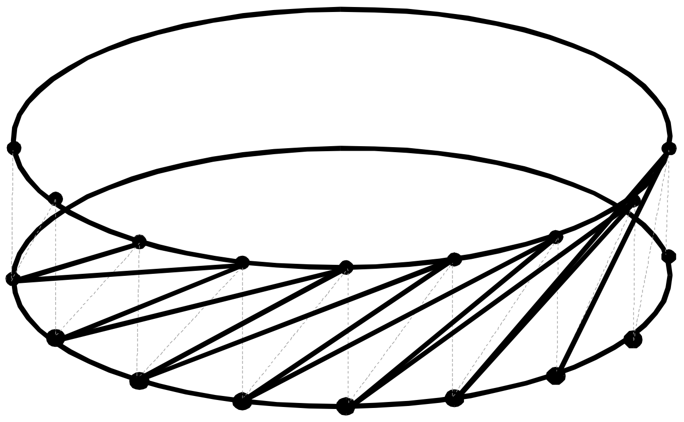


Figure 2-5: A simplified example of a surface twist .

2.1.3 Branching

Since anastotomic intimal hyperplasia occurs at graft-artery junctions, it is crucial that the meshing algorithm used by the tiler be able to handle branching structures in the surface. Branching is said to occur when the object is represented by a different number of contours in adjacent sections.

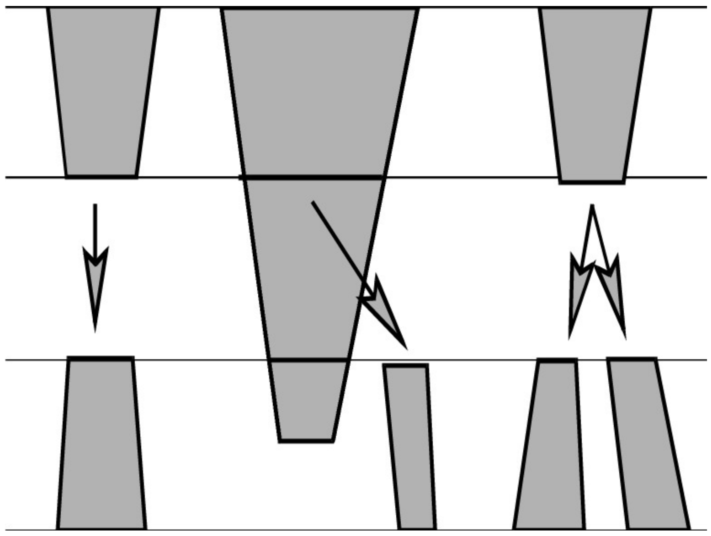


Figure 2-6: Left: Simply-connected cylinders; Centre: A two-cylinder branch; Right: A three-cylinder branch.

Branches may be in the form of a pair of simply-connected cylinders, a two-cylinder branch, or a three-cylinder branch [Mey92] (Figure 2.6). The difficulty in meshing branching surfaces is determining how to mesh the area of the surface at the forking point, or junction. The information provided by consecutive slices at this point is always incomplete, since the two slices contain different numbers of contours. The idea behind the solution to the branching problem implemented by the chosen tiler is to model the implied saddle surface at the junction by adding fabricated vertices between adjacent contours to form composite contours [Gei93]. The solution to the tiling and branching problems determines the surface topology and its coarse geometry.

2.1.4 Surface-fitting

Once the surface, including branches, has been tiled, its precise geometry can be determined. This involves finding a smooth surface that interpolates or approximates the vertices of the mesh and maintains the same topology. Interpolation is used if the data are precise specifications of an object, whereas approximation is appropriate if the data are noisy or otherwise imprecise. The most commonly used method employs a series of parametric surface patches [Far93]. The vertices of the mesh are the control points of the surface patches, and the topology of the mesh determines which vertices are used in each patch.

The simplest method of surface-fitting is to merely use the triangular faces of the mesh as the surface. This method is not usually satisfactory unless the contours sample the original surface very finely. In addition, since the surface produced by using just the triangular mesh may have discontinuities, it is not always ideal for creating a finite element volume mesh. For the research purposes at hand, however, a surface formed by the triangular faces of the mesh is desirable. The *nuages* tiler forms this mesh simply by joining the points on adjacent contours. This means that, unlike a fitted surface whose points lie anywhere in 3-space, all the points on the triangular mesh lie in the parallel planes that contain the contours. This property of the mesh will be exploited to provide a constraint on 3D interaction with the mesh.

2.2 Object Interaction

The process of creating an editing interface that improves on the current one (described later in Section 2.2.3) requires some analysis. First, both the benefits and the drawbacks of the existing interface must be fully understood. Second, potential problems in the replacement interface must be recognised, and solutions to them devised.

Before performing this analysis, the concepts of *direct manipulation* and *constraints* must be clarified. I will examine direct manipulation first.

2.2.1 Direct Manipulation

Direct manipulation interfaces (DMIs) were first identified by Ben Schneiderman [Jac89]. The term “direct manipulation” stems from the philosophy that users should manipulate objects in a computer program by a means that is analogous to the way they manipulate objects in space [Jac89, Hut85]. The essence of a DMI is that rather than carrying on a dialogue *about* an object, the user operates directly *on* it, making the interface almost transparent [Hut85].

In contrast, an indirect manipulation technique for object deformation would require many different sliders, thereby using excessive screen real-estate. While related controls such as sliders for rotation may be logically grouped, users can rarely correlate changes in the controls with corresponding changes in the deformed objects [Sni92a]. This large cognitive “distance” between the user and the tool results from that fact that, by definition, indirect manipulation tools are located outside the 3D environment.

Direct manipulation widgets such as trackballs [Str92, Con92], handles [Hou92, Con92, Str92], bounding boxes [Str92, Hou92] and shadows [Her92, Jau95] have the advantage that they exist in the scene along with the objects they manipulate [Str92] thereby reducing the cognitive load on the user.

Demands on both the user's short-term and long-term memory are also reduced in a DMI. For long-term, the user must commit to memory only a few generic manipulation commands from which most specific operations can be derived. Short-term memory load is reduced by having internal state data always displayed; most commands that change values are reflected immediately in changes in the objects. Additionally, DMIs often have less modes than an indirect manipulation interface of equivalent functionality [Jac89]. This is also less taxing on the user's memory.

Due to its nature, direct manipulation is easiest to apply to a problem domain that has a concrete graphical representation. Editing the surfaces of arteries is such an application, and the advantages of direct manipulation that I have presented make it a natural interface choice for this task. ROSS is an example of an existing product that provides a DMI for interactive visualisation of objects reconstructed from serial sections [NAS93]. Unfortunately, ROSS does not provide any surface editing capability.

Any system that employs a DMI, however, is susceptible to the mapping problem (see Chapter 1 for discussion of this problem). Often, such systems fail to provide sufficient feedback as to how motion of the input device produces transformations on the object [Phi88]. Widgets can supply additional feedback in the form of spatial cues, and even reveal their functionality through their geometry [Hou92, Con92]. Their real benefit, though, comes from the constraints they impose on interaction.

2.2.2 Constraints on 3D Interaction

Borning describes a constraint as “a relation that must be maintained” [Bor86]. For 3D interaction, constraints can be thought of as a means of restricting motion, most often by fixing it to a particular dimension, or set of dimensions. This effectively reduces the number of degrees of freedom in the interaction. Rigid body motion in 3D has six degrees of freedom for positional and angular placement [War90], and object deformations have many more [Phi88].

Simultaneous manipulation of many degrees of freedom may make interaction too difficult. Snibbe observes that “a tool can be made more effective by removing unnecessary degrees of freedom with constraints” [Sni92a]. Houde reports similar findings in her experiments with a handle-box interface [Hou92]. Hsu et al. also note that the number of degrees of freedom presented to the user in free-form deformation can be overwhelming [Hsu92]. Herndon et al. constrain transformations to a plane with their shadow widgets in an attempt to make 3D manipulation easier for the user [Her92].

Because of the demonstrated value of constraints in 3D interaction, 3D surfaces that have been reconstructed from contours seem to lend themselves well to deformation through 3D interaction. They have an inherent “relation that must be maintained”: contours must remain planar. Since all points on the surface are part of some contour (recall from Chapter 1 that this is a property of the surface mesh), movement of points is restricted to the plane. The common mapping problem is thus diminished by eliminating a dimension for translation. I therefore hope that a 3D DMI will be successful for editing the artery data.

2.2.3 Current Interaction Techniques

To provide an improvement over the existing artery editing interface, it is necessary to understand both the positive and negative aspects of that interface. Current editing techniques in the UWT project are somewhat haphazard. The user determines contours that need editing by examining an initial reconstruction of the artery data using the solid modeller package I-DEAS[1].

The user then edits contours requiring corrections in one of three ways [Moo95]. The first method is the most primitive, and difficult to use. The user manually plots the cross-section containing the problematic contour, and determines where the fault lies. He then changes the coordinates directly in the slice file using a text editor.

The second way in which the contours are processed is by a low-pass filter that smoothes out tiny indentations in the surface. The user can choose the degree to which the curves are smoothed, if any. This filter does nothing to remove larger bumps and indentations.

The third, and only interactive method, is to move points on the contours using a curve editing feature of I-DEAS. When the user picks and moves a point on the curve (termed *primary motion*), adjacent points on the curve, in a local region around the picked point, are also affected (termed *secondary motion*) (Figure 2.7). The user can specify the area of effect and the nature of the secondary motion through a dialogue-style interface [SDR93].

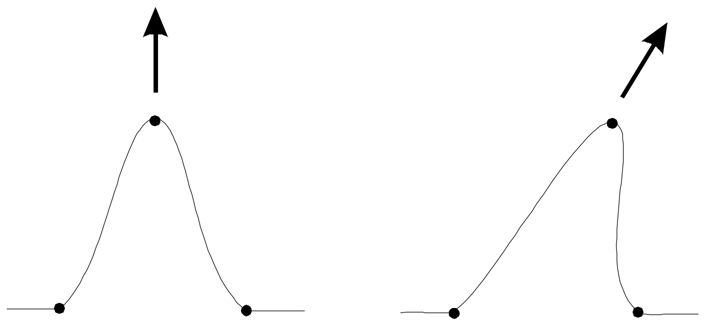


Figure 2-7: Example of how a curve segment is affected by the primary motion of a picked point.

The user moves points in the current *workplane*. For artery editing, this is chosen to be the plane containing the contour of interest. The user can limit the view to only this workplane, providing a 2D view of a cross-section. Alternatively, the user can display all of the contours in the data set at once in a 3D perspective view. Although the latter view gives the user a context in which to work, it is seldom used. Even with the ability to rotate, pan and zoom in this view, having all of the contours present makes it difficult to pick and control the desired points for movement [Moo95]. Attempts to interact with points on a particular contour are confused by the interfering presence of other contours. This may be because the surface itself is not present in the view to give full meaning to the contour lines. Unfortunately, the surface reconstruction algorithm used by I-DEAS seems fragile, i.e., the final surface cannot be constructed until the glitches in the contours are corrected.

Because editing in the perspective view is difficult when only the contours are visible, a 2D view of a single cross-section is chosen most often as the editing environment for the arteries. Even then, users of the system frequently resort to using the text editor out of frustration.

[1] I-DEAS is a CAD/CAM product designed to aid in the manipulation of freeform geometry [SDR93].

2.2.4 Anticipated Improvements

3D display representations better match the user's mental model of 3D spatial information (such as flight path or contour data) than do their 2D counterparts [Wic89, Wic92]. It thus seems reasonable that a 3D interface for artery editing might be an improvement over the current 2D editing technique, at least for some tasks. The aim of the experiments conducted as part of this research is to determine for which tasks, if any, this holds true. The experimental hypothesis is that for tasks of a 2D nature (i.e., tasks involving only a single contour and hence a single plane), a 2D interface will be superior to a 3D interface. This superiority would exists because of the unnecessary complicating factor of the extra dimension in 3D, and the resultant mapping problem.

On the other hand, for tasks of a more global, 3D nature, such as flattening a bump that spans several contours, I hypothesise that a 3D interface would prove more effective than a 2D interface of equivalent functionality. This is primarily because the user's mental model of the artery is of a 3D nature. In addition, the mapping problem is diminished due to the inherent constraints in the artery data. Finally, the *visual momentum* provided by presenting the artery as a unified object, rather than a series of slices, may aid the user in 3D interaction [Woo84, Wic92]. Users may become cognitively “lost” as they traverse through multiple displays pertaining to different aspects of the same system. Visual momentum is an engineering solution to this problem. See [Woo84] for a more complete treatment of this topic.

The 3D surface editor interface that I evaluate alongside the current 2D method of interacting with the artery data will use direct manipulation. As well as delivering all of the benefits outlined in Section 2.2.1, this will allow a fair comparison to be made with the existing interface, which also uses direct manipulation.

3. Implementation

Although I hope that SLICE's interface is generic enough to be used with any contour-editing application, SLICE was primarily designed to fulfil the requirements of the UWT project. In my research, the interaction requirements of that project receive particular attention, and are mapped into high-level viewing and editing tasks. Those tasks in turn are broken down into the low-level component tasks that drive the interface design process. Out of this process, both SLICE's DMI for object interaction and its GUI emerge. The GUI integrates the functionality of the DMI with SLICE's meshing capabilities. This chapter covers the evolution of the SLICE interface, beginning with the requirements of the UWT project, and concluding with the interface used in the experimental portion of the research.

3.1 Requirements Analysis

In the UWT project, users gather data from CT and MRI scans of arteries. The next stage of the project is to prepare the data for fluid flow analysis using the following multi-step process: first, the contour data, which represents cross-sections through the arteries, is converted into text format as lists of coordinates. Then, a program takes these contours as input and generates from them a triangular mesh that is an approximate reconstruction of the original artery surface. Next, a user interactively edits the mesh to remove scanning artefacts. Finally, the user exports the points of the edited surface for generating a volume mesh to be used in the ensuing fluid flow analysis.

I designed SLICE to serve as the interactive surface editing program in this process. SLICE's first obligation is thus the generation of a surface mesh from input contour data. The program must be able to generate the mesh with interactive speed, since the surface will need to be re-triangulated as the user makes changes.

The data that arrive from the artery scanning process contain high-frequency noise. This noise manifests in a rough edge of fine indentations on the contour which prevents a proper volume mesh from being created. The meshing algorithm has trouble meshing the edges, and the volume created from the mesh does not accurately portray flow within the artery. Thus, the next requirement that the UWT project imposes is that SLICE smooth the contour data.

Of primary interest in my research are the requirements involving user interaction with the data. The participants in the UWT project desire a method of interactively viewing an artery in 3D as a shaded surface, and from any position. Currently, they have no easy way to visualise the artery as a surface. They can see the artery as a set of 3D contours, but the noise and artefacts must be removed before a surface mesh can be generated using the I-DEAS package.

The second aspect of the interaction requirements is to allow users to interactively remove unwanted features from the surface mesh. This means manipulating individual points on the contours, while providing real-time visual feedback to the user. This requirement is the focus of the experimental research component of this thesis.

Finally, the program must be able to save and generate a final surface mesh that can be used to produce a tetrahedral volume mesh for later fluid flow analysis.

3.2 The SLICE solution

SLICE, the interactive contour editor I implemented for this thesis, was written to fulfil the requirements of the UWT project. SLICE is used to edit 3D surfaces that have been reconstructed from planar contours. More precisely, it is a tool for visualising the surface derived from a set of serial sections, and for removing artefacts from this surface that were created during the data-gathering process. SLICE is written in the C language for use on a Sun platform. It uses xview for the user interface and xgl for the graphics. It consists of two parts: a "back-end" for surface reconstruction, and a "front-end" user interface for surface manipulation.

The back-end of SLICE is built around Bernhard Geiger's *nuages* program, which was integrated with SLICE to perform triangular mesh generation. The *nuages* program (introduced in Section 2.1) takes as input a set of serial sections, and triangulates them into a surface mesh. I changed the *nuages* code to store the entire artery data set in memory at one time (instead of only pairs of slices). Also, instead of writing the resultant mesh to a file, as *nuages* originally did, I made modifications so that SLICE can display the mesh graphically in both a projective view (3D) and as a sequence of cross-sections (2D). SLICE reads a contour data file and then passes the point information to *nuages* functions for processing. The triangular mesh generated by *nuages* is then stored in an internal data structure in SLICE. Manipulations made to the points by the user change the contents of this data structure, which is read by *nuages* whenever the mesh needs updating. When the user has finished modifying the mesh, the changes can be saved to a file. The final mesh output is based on the stored representation of points maintained by SLICE.

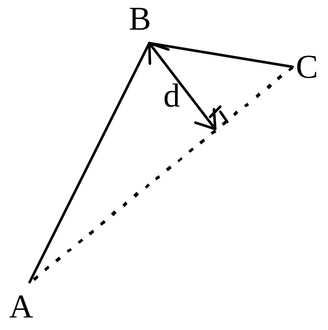


Figure 3-1: Pairwise smoothing process.

Due to the data-gathering process, the contour data is often noisy. Smoothing each individual indentation by hand would be time-consuming and tedious. An automatic method of smoothing is preferable. Geiger highly recommends using a feature of *nuages* for smoothing pixel-by-pixel contours like the ones used in UWT's research [Gei93]. The user specifies an error value for the smoothing on the command line. In *nuages*, contours are approximated with line segments, for example AB and BC in Figure 3.1. *nuages* uses a pairwise process for smoothing. That is, to decide whether to remove point B, *nuages* tests B's distance, *d*, to a new line segment, AC. If this distance is less than the user-specified error value, *nuages* removes the point. For the purposes of UWT's research, a value was chosen that yielded an error tolerance level of three percent, thus keeping the curves within three percent of their original position. For normalised data, this translates into a value of *d* of 0.03.

SLICE' s front-end consists of a graphical user-interface that provides both 2D and 3D displays of the data, as well as a means to interactively view and edit the data. I wrote the interface in Sun' s proprietary widget set, xview, since this is included with Sun hardware, and will thus be readily available to the end user in the UWT project.

The remainder of the chapter is devoted to describing how SLICE meets both the viewing and editing interaction requirements of the UWT project.

3.3 Interaction Task Analysis

Since the fundamental purpose of SLICE is to remove artefacts from artery data, the first question is one of recognising such an artefact. It is known that all contours provided by the data extraction method will be closed. All defects in the artery will therefore appear as bumps or hollows in the surface. These deformations may be the result of either points or entire contours that are misaligned. Currently, only a limited number of data sets are available and thus it is impossible to precisely describe the shape of the artefacts. By definition, however, the bumps or hollows for which a point or group of points is misaligned must take one of the following forms: a *spike*, a *spine*, a *ridge* or a *hill* (Figure 3.2). I will define a spike to be a single point that is out of place. A spine is a set of spikes along a range of contours. A ridge is a range of points along a single contour that is out of alignment. Finally, a hill is a range of ridges. Even though the deformations in Figure 3.2 all appear as bumps, the phenomena could also occur as hollows if they were inverted. It is not known how many artefacts will fall into each of these categories in real data sets. Also note that the definition of an artefact is subjective and recognition requires a trained observer.

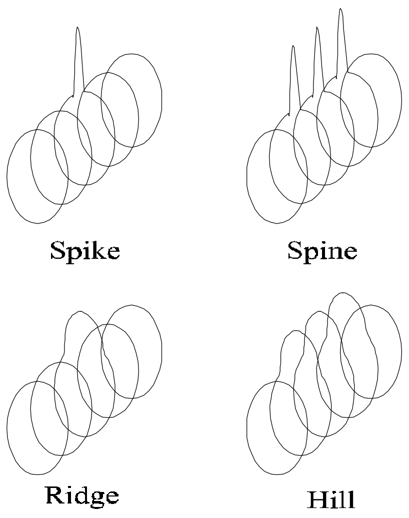


Figure 3-2: Examples of possible bump types in surface data.

In order to remove the kinds of artefacts described in the previous paragraph, interaction in the SLICE application involves making detailed changes to points on the surface. Although changes usually affect a local region of the surface, it is important to see their effect on the overall shape of the surface. Thus editing operations are integrated with changes in the view. This dual environment must be unified through a coherent interface.

The high-level tasks presented here are groups of low-level tasks that have been combined at the semantic level of user goals. They fall into two categories: viewing tasks and editing tasks. Viewing tasks include zooming, scrolling and rotating the view, viewing a cross-section of a surface, and viewing the inside of a surface. Editing tasks include removing bumps or other unwanted pieces of a surface, stretching, or disconnecting a surface, adding points, and undoing the last editing action.

3.3.1 Viewing

The user must be able to easily change the current view of the surface, either to see a global view, or to focus on a particular area. *Viewing* tasks enable the user to accomplish these goals.

The most frequently performed viewing task is that of moving the viewpoint to a new position. After the 3D view has been changed, there must also be a way to reset it to its original position. To change between a close-up view, and a view of the entire surface, the user must be able to zoom the viewpoint closer to and further from the surface. For example, if the eyepoint is positioned on the positive z axis looking toward the origin, this means moving the viewpoint along that axis while the surface remains centred at the origin. In order to better view the surface, the user must be able to scroll it within the viewing window. This means moving the viewpoint in its xy plane, if the eyepoint is positioned on the positive z axis. Finally, the user must be able to rotate a surface in 3-space with the three degrees of freedom formed by rotations.

In the surfaces that SLICE is designed to edit, that many of the points to be manipulated are hidden by other parts of the surface. Another viewing task is thus to reveal hidden parts of the surface, allowing the user to "see inside" it.

Finally, the user may wish to have a view that clearly shows the shape of contours in a particular cross-section. Since all cross-sections are planar, this can be accomplished by showing a 2D view of the cross-section of interest. In addition, there must be a way to cycle through viewing consecutive cross-sections of the object.

These viewing tasks are rigid body transformations; they do not deform the shape of the surface the way that editing tasks do. Thus surface rotation, even though it occurs by applying a transformation to the surface, as opposed to the view, will be regarded as a viewing task.

3.3.2 Editing

SLICE is designed to edit surfaces reconstructed from contours. *Editing* involves changing the shape of the surface to eliminate artefacts created during the gathering of the contour data.

The SLICE user may want to remove bumps, or other parts of a surface. Regular-shaped bumps or hollows can be eliminated by stretching or flattening the surface. Stretching or flattening is accomplished by moving groups of points in a local area. When a user moves a point on the surface, points in a local surrounding region should also move as a function of the motion of the focal point. The user must be able to restrict the area of effect of this secondary motion. This method of eradicating bumps does not delete existing points, but instead moves them to a new location. Other protrusions or unwanted pieces of the surface may be removed by deleting portions of the surface.

In addition to deleting points from a surface, the user may wish to add points to it. Because the user may choose to perform some smoothing of the contours, there may be contour segments that are sparsely populated with points. Removing a portion of the surface may also contribute to sparseness. If the user wishes to change the shape of the surface near such an area by stretching or flattening, as described above, more points may be needed on the contour to allow the user sufficient control over the shape of the curve (i.e., so that there are enough line segments to produce a reasonably smooth curve).

Editing tasks that add, move or delete points have the potential to significantly change the shape of a surface. It is therefore desirable for the user to be able to undo an action by returning to a previous state if he makes a mistake. This is especially important in light of the subjective nature of the surface editing for which SLICE is intended. In this application, it is likely that users' editing will pass though successive approximations before the final version of the surface is reached.

These high-level tasks all depend on the same low-level building blocks: selecting, adding, deleting and moving objects. These fundamental tasks might occur in any interactive 3D interface, and will be at the root of the interface design.

3.4 Design Considerations

Hardware limitations, the intended user of the software and attention to sound design principles all influence the creation of the SLICE interface. I will examine these factors now.

3.4.1 Assumptions

When designing SLICE, I made certain assumptions about the style of its interface. The following choices were based both on the focus of the research and on hardware choices imposed by the resource constraints of the UWT project. I confined the requirements of SLICE to a standard, low-end hardware configuration.

First, I limited the scope of the research to direct manipulation interfaces. This restricts the interface choices to mouse click and drag for surface manipulation. For example, a task such as rotating the view should be performed with mouse actions, rather than with scrollbars or other GUI widgets.

Second, SLICE is designed to provide only visual feedback.

Third, SLICE is designed to use a 2D input device (in the current implementation, a three-button mouse is used in conjunction with a keyboard), and output to a monitor. Other input devices were not considered.

Finally, the UWT project requires SLICE to run on a low-end Sun workstation (a Sun Sparcstation 20SX with 96 Mb of RAM and an SX + cg14 4Mb VSIMM graphics board was used). This limits the available graphics capabilities if performance is to remain at an acceptable level for interactive manipulation. Thus some options, specifically those requiring the more advanced rendering techniques, could not be explored (e.g., when objects are in motion, a wireframe rendering must be used instead of a shaded one).

3.4.2 The User

SLICE will be used by people who are accustomed to studying arteries. The users will be conversant with the scanning process used to produce the computer representations of the arteries. They will therefore be familiar with the artefacts that occur in the data, and can be expected to recognise what editing is required for a given surface representation.

Some computer skill is assumed on the part of the users. I expect them to be comfortable using both mouse and keyboard to interact with a graphical user interface. Prior experience with 3D graphical interfaces is not necessary. SLICE's 3D interface should be easy to learn by transferring skills from an existing knowledge base. The 3D interaction techniques should be simple extensions of 2D interaction methods.

I anticipate that the use of SLICE will occur in fairly infrequent but intense sessions. In other words, when new artery data is obtained, the user will sit down with SLICE, and work with one data set until all of its defects have been corrected. I expect that the user will edit each defect in sequence for a particular data set. This style of interaction will govern the use of modes in the design. I will discuss this in detail, in Section 3.4.4.

3.4.3 Consistency

Although consistency is recognised as a difficult concept to define precisely [Gru89, Kel87], it is widely perceived as a desirable goal for user interface design. Although designers may not be able to state explicitly what consistency means, it is possible to define its effects. A consistent user interface allows the user to feel familiar with the interface, and most importantly, to transfer skills and learning.

According to Grudin [Gru89], there are three types of consistency: internal, external, and external analogue. Internal consistency refers to a coherence of principles within a single interface, for example the use of the <F1> key to display help from anywhere within the interface. External consistency means consistency between different interfaces, such as the “file” menu being located at the top left of a menu bar. External analogue, or metaphoric consistency has to do with the correspondence of the interface design to features in the real world, such as the Macintosh desktop metaphor.

An interface designer should pay attention to the way in which consistency is used within an interface, for it is not a panacea. Consistency helps learning, but may sometimes impede use. For example, keeping all the household brushes in one drawer is consistent, and would help you to learn where to find them. Brushes, however, may be used for different tasks. There are paint brushes, tooth brushes, lint brushes, hair brushes, and brushes for sweeping the floor. Keeping them all in the same place would make them difficult to use. They would perhaps be better located according to their function; a tooth brush may be kept in the bathroom, a paint brush in the workshop, etc.

The interface designer may also choose to make things slightly inconsistent so that the user pays attention. Performing a particular operation, such as removing a file, may have drastic consequences to the user. If the interface for file removal is inconsistent with the rest of the interface in some way, it will draw the user’s attention making sure that he takes special care with that operation.

3.4.4 Modes

Modes in an interface refer to different interpretations of same user input according to the current state of the interface. For example, a left mouse click could mean “select a point” or “delete a point” depending on the current mode. Having modes in a user interface places additional mental load on the user since it requires that he remember the current state of the interface (or be reminded by the system), and what commands are applicable in that state. However, some use of modes cannot be avoided in most user interfaces, as there are not enough distinct brief input operations, such as single keystrokes, to map into all the commands of a system [Jac89]. The best one can hope for is to minimise the amount of mode-switching that the user must perform.

The complexity of SLICE's user interface calls for the use of modes. Since I wanted the interface to be a DMI, I wanted user input actions to come through the mouse and operate directly on the mesh. This meant that the individual input actions available for consideration included single and double click and drag with any of the left, middle or right mouse buttons, or combination of buttons (*chording*). In fact, use of the mouse while different buttons are held down might also be seen as operating in different modes. I will not, however, make this distinction in my discussion.

To determine how modes should be used in the interface, I analysed the semantic meanings that these mouse actions must take on. Table 3.1 enumerates the objects in the system and the user actions that operate on them.

Objects	Applicable Actions
view	move (for scroll and zoom)
entire mesh	move (for rotate)
point	select, deselect, move, add, delete
contour	select, deselect, move, delete
range of points	select, deselect, delete
primary point (defined in Section 3.5.1)	select, deselect, move

Table 3.1: Objects and the actions that can operate on them

The first possibility I considered was to have a mode corresponding to each type of object. For instance, depending on the mode, an action such as move would be applied to a point, a contour, or the entire mesh. One problem with this approach is that it produced too many different modes. The objects, however, can be logically grouped according to the actions that may be applied to them. This implies that actions may better correspond to modes than objects do. In addition, an action-oriented scheme of modes is more likely to correspond to the user's mental model of the system. Users are unlikely to think of the view, the mesh and its component points and contours as separate objects upon which to operate. Rather, I hope they will perceive a more unified environment.

The second design I considered thus had modes corresponding to actions. This type of interface would be similar to a paint program such as MacDraw in which actions are thought of as tools, and each tool is a mode. Thus there are modes for each of selection, deletion, addition, and motion. These modes would govern the program's interpretation of mouse input. The flaw with this design is that the user still requires a method of indicating to what object the chosen action should apply.

The final design attempted to pull together the best components of each of these designs by abstracting the objects and actions at a higher semantic level. The first design, which had objects corresponding to modes, was natural because it allowed the user to use modes that differentiate between targets for a consistent set of actions. The second design, which had actions corresponding to modes, fit better with the user's mental model of the system. More careful inspection of the objects and actions reveals natural groupings between them. The first two object-action sets listed in Table 3.1 are related to viewing. The last two are related to ranges of objects. The middle two are related to the building blocks of the mesh itself. These are the objects and actions at the heart of the interface. Ranges and viewing are really "extra" features that assist with the main tasks in this category. Thus, the actions related to points and contours should in some way be the default mode, minimising the amount of mode-switching required on the part of the user. Adding a point can be seen as a different type of action to select, move and delete since the latter manipulate existing points.

Translating these groupings into modes yields a *view* mode, a *range* mode, an *add* mode and a *default* mode. This division allows the use of modes to indicate the target of the action (e.g., the view versus an object), as well as grouping actions at a semantic level. In Section 3.5.2, I discuss the precise mapping of the various mouse actions to the semantic actions that must be performed in each mode.

Since it is expected that operations to manipulate the surface will be frequently interspersed with changes in the view, the user must be able to make rapid, fluid changes between *view* mode and *default* mode. For this reason, keyboard and mouse combinations (for example <ctrl> + mouse) were used to denote the different modes instead of using GUI widgets such as radio button. Pressing and releasing a key to switch modes can be done very quickly, and does not require a shift of attention to a new location on the screen. Using a key-mouse combination for each mode provides internal consistency to the interface. Section 3.5.2 discusses the precise mapping of keys to modes used in SLICE.

Finally, it is crucial that the user be provided with clear feedback as to the current mode. A frequent source of user error is inappropriate input due to an incorrect perception of the current mode [Wick92], for example turning the key in a car's ignition when it is already running, having incorrectly perceived the engine to be off. There are several different ways in which a mode change might be signalled such as a change in colour, or display of a label. Since the user's attention always remains focused on the object being manipulated, however, changing the cursor seemed the most effective way to indicate a mode change. The cursor is always located at the precise point of the user's interest, so a change in its shape will not go unnoticed. An open hand was used for *view* mode, a pointing hand for *range* mode, a "+" sign for *add* mode, and X11's default arrow cursor for *default* mode.

3.5 Designing the Interface

For each task in SLICE's interface, I explored a list of possible implementations. I made this list *outside* the context of any particular application. Appendix A contains this design space of interaction possibilities. In this section, I discuss how the low-level tasks are mapped into this space, in light of their intended use in the SLICE application.

Where several implementations are semantically equivalent, it will be necessary to choose between the options. I used prototyping of alternatives as well as analysis of existing interfaces to assist in making these choices. In some cases, however, it may be desirable to implement more than one method per environment of performing a task, each to be used in a different situation.

I was able to immediately reject some implementations as infeasible or inappropriate to the application. Handles, for instance, are a common tool used to implement all kinds of transformations on objects including rotation, translation and scaling, as well as more complex manipulations such as bending, twisting and tapering [Hou92, Con92, Str92]. A handle is usually some visual geometry that enforces a constraint on interaction, such as restricting motion to a single axis. I did not feel that the use of handles was appropriate in SLICE. The handles would only be useful for the viewing transformations, and not for reshaping the object. Instead, they would obscure the surface, interfering with point manipulation. Handles are better for an interface in which they realise all of the desired interaction. That way, the handles themselves represent modes. In other words, grabbing the rotation handle means mouse motion will be interpreted to spin the object, while grabbing a scaling handle has mouse motion interpreted to resize the object.

Another interface possibility that I rejected for the sake of simplicity was the use of planar boundaries to aid in selection and deletion of points. One option for selecting an area of points to move is to define the dimensions of a (rectangular) section of the plane tangent to the surface at the focal point of motion. This plane is projected onto the surface, and the region within the plane is the area affected. An ellipse may be used instead of a rectangle. A planar boundary could also be used to define an area for use as a cutting plane. It would act as a separator between the areas of the surface that are to be kept, and those that are to be discarded. In the case of removing a branch, however, deleting contours is a simpler solution. Defining a planar boundary is difficult because of the many degrees of freedom associated with an arbitrary plane. Using arbitrary planar boundaries would mean forfeiting the one advantage that the data provides: motion is constrained to a plane.

Since the high-level viewing and editing tasks share common low-level components (select, move, add and delete), I will try to provide an internally consistent interface by implementing each of the low-level tasks in the same way for all of their high-level uses.

The selection and movement of objects is at the heart of the high-level viewing and editing tasks. I discuss these two low-level tasks here in some detail, before presenting them within the interface design framework. In the rest of this section, I discuss the design of the DMI used to manipulate the surfaces.

3.5.1 Select and Move

Selecting something makes it the active object to which subsequent move or delete operations will be applied. You can select both points and contours. Only one contour at a time may be selected. However, points may be selected in a contiguous range along a contour, and in sets of ranges along several neighbouring contours (forming a contiguous area of selected points), or as a primary point (the focal point for motion of a range).

Feedback can be provided to indicate which objects(s) have been selected by highlighting them in a different colour, by having them blink on and off, or by changing their size or shape. In the SLICE interface, points and contours change colour, becoming "highlighted" when they are selected. Selecting a point as a primary point causes it to both change colour, and change shape from a circle to an asterisk.

Contour selection is used in the deletion of surfaces in whole or in part. Deleting a branch, for example, would be done by deleting its constituent contours one at a time (a potentially time-consuming operation, but one that should rarely be used). Deleting a contour in the middle of a surface will disconnect it. Contour selection in 3D may also be used to choose a cross-section of the surface for viewing in 2D. Finally, a selected contour may be moved within the plane in which it lies.

Point selection is a fundamental operation used to either delete, or more frequently, to move a point or group of points. Smoothing out a contour may require points to be selected singly or in a range. Tasks like removing a bump may require a range or area of points to be selected.

Multiple points cannot be moved without first selecting a primary point to act as the focus of the motion. As mentioned in Section 2.2.3, the user interactively manipulates the primary point, and the other selected points (secondary points) move according to some function of this motion. This is illustrated in Figure 3.3.

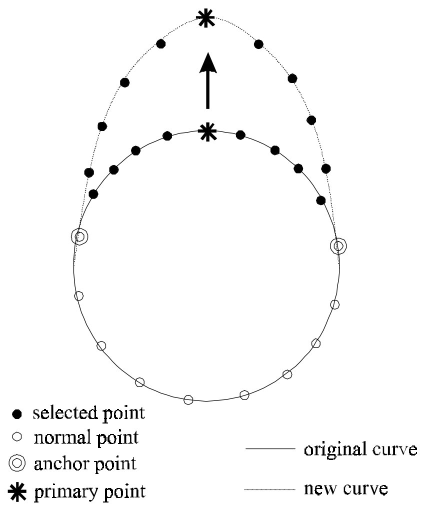


Figure 3-3: Secondary motion of points on a contour.

In the design of SLICE's interface, I considered having the user define this function. Making some assumptions about the user's goal in moving the points renders this complicated process unnecessary, however. For the UWT project, users move points to smooth out surface irregularities. SLICE, therefore, implements the function in the following way. First, a least-squares approximation is used to fit a B-spline curve to the range of selected points. (See Farin's book for a discussion of spline curves [Far93]). The order of this curve is set to be **`4 + ln( abs( range_length ))`** to provide reasonable control of the curve. In this way, the order of the curve increases logarithmically with the number of points selected (range_len). For example, the order of the curve is four when two points are selected, five when three to seven points are selected, and six when eight to twenty points are selected. A higher order curve results in more control vertices, and hence greater control of the curve. It is important to increase the order of the curve more quickly at the low end of the range length, because differences in order are more apparent when there are fewer points.

SLICE includes an additional point at either end of the selected range in the set of points to which it fits the B-spline. For this reason, at least two points on the curve must remain deselected at any time. SLICE then converts the B-spline control vertices produced by the curve-fitting to Bézier control vertices[1] because Bézier curves have the desirable property of passing through their first and last control vertices. Thus, the extra points at each end of the range become the end vertices of the Bézier control polygon, and act as anchors while the curve segment between them moves (Figure 3.3).

Because the manipulation of control vertices is an awkward, non-intuitive means of manipulating a spline curve, SLICE uses a direct manipulation technique developed by Bartels and Beatty [Bar89]. This allows the user to affect the shape of the curve in an intuitive way by dragging points on the curve (i.e., the primary points).

When interaction begins, the original Bézier evaluation of the points on the curve is determined at the locations specified by the parameters that correspond to the selected points on the contour. After the user has moved the primary point, the new locations of the control vertices are calculated so that the curve will pass through the new location of the primary point. Using these new control vertex locations, the positions are calculated for the other points on the curve.

In SLICE, it is possible for the user to select several ranges of points on adjacent contours and then move a single primary point to affect the whole region. In this case, the same process is repeated for each curve to determine the new point locations. However, since the user only selects one primary point (marked with an asterisk in Figure 3.4) for the entire area, a *pseudo-primary point* (marked with a dot in Figure 3.4) must be determined for each of the adjacent contours. This point is the closest, connected, selected point to the user-selected primary point. If such a point does not exist, the ranges cannot be moved together. This restricts the points that can be moved as a group. Adjacent ranges must contain at least one point that is connected to the primary point of motion. In SLICE, if an invalid set of ranges is chosen, the user is warned when he tries to move the primary point.

[1] The code for fitting a B-spline curve to the selected points, and for converting the B-spline control vertices to Bézier control vertices was generously provided by Richard Bartels of the Computer Graphics Lab at the University of Waterloo.

I will call the contour containing the primary point the *primary contour*, and other contours containing selected points *secondary contours*. The process of finding pseudo-primary points is repeated for all secondary contours, cascading outward from the primary contour (Figure 3.4).

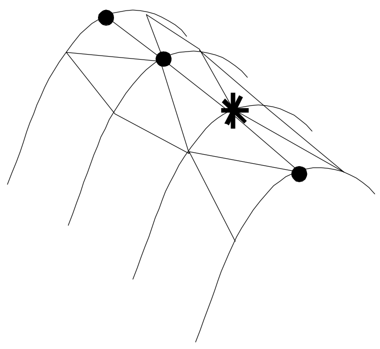


Figure 3-4: Finding pseudo-primary points.

The primary contour moves the most, while secondary contours move less according to their distance from the primary contour. Once the delta has been found for the motion of the primary point, it is applied to all the pseudo-primary points with exponential drop-off outward. This exponent can be interactively set by the user with a slider to values between one and five, with a default of two. Figure 3.5 illustrates the default case in which the pseudo-primary points in curves marked "B" move half as much as the primary point in the curve marked "A". The curves marked "C" would move one quarter as much, and so on.

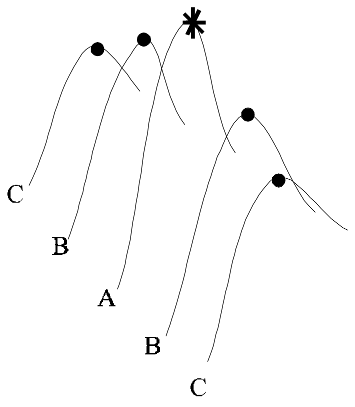


Figure 3-5: Moving an area of points.

After the delta for the pseudo-primary points is calculated, the motion can then be determined for the other points on the curve in the same fashion as described earlier for the primary contour.

3.5.2 Designing Mouse and Keyboard Interaction (The DMI)

In Section 3.4.4, I introduced SLICE's four modes (*view, range, add* and *default* modes). In this section, I discuss the keyboard actions used to implement each mode, and the mouse actions used for the tasks in each mode. All mouse actions except rotate apply to both the 3D view and 2D view, and are mapped the same way in each case for internal consistency. The following table presents the details of the mouse mappings in each mode.

	LEFT MOUSE (DOUBLE CLICK)	LEFT MOUSE (SINGLE CLICK)	MIDDLE MOUSE (SINGLE CLICK)	RIGHT MOUSE (SINGLE CLICK)
Mouse only	select contour; drag to move selected	select point; drag to move selected	deselect endpoint of range (decided after pretests)	deselect the last selection
Mouse + <Shift>	select additional contour	select additional point at end of range (decided after pretests); drag to add/delete points from range	deselect all selected objects	deselect the last selection
Mouse + <Ctrl>	N/A	drag to rotate object (3D only)	drag to translate view	drag to zoom view
Mouse + <Alt>	N/A	add point	N/A	delete the last point added

Table 3.2: Mouse mappings for SLICE

3.5.2.1 View mode

In *view* mode, the only action is “move”. There are, however, three types of motion: rotation, scroll, and zoom. The target for these actions is implied. Scroll and zoom apply to the view, and rotation applies to the entire mesh. There should not be much distinction here in the mind of the user; all actions are enabling the user to obtain a better view of the object. It is easy to map these three actions to click and drag with each of the three mouse buttons. Details are provided in Table 3.2. Using the cursor as tool to grab and move the object provides consistency through the use of an external metaphor. The user can perform scrolling and zooming in both the 3D view and the 2D view, but rotation applies only to the 3D view.

The direction of mouse movement when scrolling corresponds directly to the direction in which the surface moves.

Mouse motion upward or to the right when zooming brings the surface closer to the viewer. Mouse motion downward or to the left moves the surface further away.

Preliminary research in 3D interaction [Coa93] involved implementing rotation with a pitch, yaw and roll interface that used a combination of sliders and a dial. User testing of this implementation revealed that such an interface was awkward and unintuitive. Subjects found it very difficult to perform a simple task such as rotating the surface to swap opposite corners of an plane. For this reason, I did not consider the pitch, yaw and roll interface for this application. Instead, I used trackball rotation. This interface provides additional consistency through the use of an external metaphor. A trackball interface works by having the user imagine that the surface to be rotated is enclosed within a transparent sphere. The cursor is used to grab and roll the sphere.

3.5.2.2 Range mode

The primary purpose of *range* mode is to allow the user to select multiple points. Selection of a range of contours is not permitted because determining adjacency with a branching structure is too complex. Because only a contiguous range of points may be selected, I can use a click and drag interface similar to that for selecting and moving points (thus providing some internal consistency). A left mouse click selects the first point in the range and then dragging the mouse selects adjacent points along the contour; the further the mouse is dragged, the more points are selected. The direction of the range selected corresponds to the direction of mouse drag in the following way: dragging up and to the right means select points in a clockwise direction around the contour, and dragging down or to the left means select in a counter-clockwise direction. The model matrix containing all the viewing transformations is used to ensure that clockwise and counter-clockwise refer to how the contour appears on the screen, regardless of how the object is rotated in 3D. If the mouse is dragged to the right to select points, dragging it back to the left by a corresponding amount will deselect points, and dragging it further will begin to select points in the opposite direction.

Selecting one endpoint and then dragging out the rest of the range was chosen as the range-selection interface rather than selecting the two endpoints of the range for several reasons. Selecting two endpoints on a closed contour would ambiguously divide the points into two possible ranges, and might also require the user to perform an intermediate, time-consuming view change operation. The drag interface only requires the user to select one point with the cursor. It is also convenient since the artery data points lie very close together. Users reported employing a strategy of dragging one point past the end of the range that they wanted, and then going back. They found this a comfortable way to select exactly the desired group of points.

SLICE implements *range* mode using the <Shift> key as the mouse input modifier. Using the <Shift> key to indicate multiple selection has become an industry standard, and is used in such products as MacDraw and the drawing tool in Microsoft Word for Windows 95, thus providing further external consistency.

3.5.2.3 Default mode

Default mode is most complex. I want the user to accomplish as many of their most frequently performed tasks as possible without having to switch modes. It is natural to map the simplest mouse action to the most fundamental task. The core actions of the interface are select and move. Mapping select and move to click and drag with the left mouse button (with no key modifier) provides external consistency with other DMIs such as Microsoft Word's drawing tool and Alias Studio. In order to distinguish between points and contours, the two possible targets for these actions, single-click is used to indicate a point selection, and double-click is used to indicate a contour selection. Without different means of selecting points and contours, it might be difficult to distinguish between them on the screen when points lie close together. Double-click is reserved for contour selection since this will be less frequently used than point selection. Note that this single- versus double-click paradigm allows for a later extension to the interface to select a range of contours (i.e., double-click and drag).

The interface enforces several rules of selection. When the user makes a new selection, the selected contour or the contour to which the selected point belongs, becomes the *current contour*. It is highlighted in red in the 3D view, and the 2D view is changed to display the cross-section to which it belongs. A point and contour cannot be selected at the same time, thus a contour selection will deselect any currently selected points, and vice versa. In addition, since SLICE allows only contiguous ranges of points to be selected, a new selection on the same contour as an existing selection deselects that existing selection. Finally, since the user may select multiple ranges of points only on a contiguous range of contours, a new selection of points on a contour that is not adjacent to an existing selection deletes any currently selected points.

The middle and right mouse buttons are reserved for two kinds of deselection. Note that since deselection applies to both single points and ranges, the mapping for these mouse buttons applies in *range* mode too. The right mouse button, with the cursor located anywhere on the screen, deselects the last selection (contour, point or range of points). This operation may be applied repeatedly to deselect objects in reverse order of selection. For example, if the user had selected three adjacent ranges of points, clicking the right mouse button three times would deselect one range each for each click, from the most recent to the earliest selection. The middle mouse button is used to clear all selections.

3.5.2.4 Add mode

In *add* mode, the user can insert additional points into an existing contour. This may be useful if the user wishes to modify the local shape of a contour in an area where there is a long line segment.

One might expect *add* to be a mouse action under *default* mode. Once again, however, it is advantageous to break internal consistency to draw the user's attention to this operation as a special situation, one that more drastically modifies a mesh than simple movement of points. In addition, consistency can help learning while impeding use. Making *add* a separate mode makes *default* mode easier to use. With *add* included as an operation in *default* mode, there would be insufficient mouse actions to handle all of the tasks.

A left mouse click in *add* mode adds a point at the cursor location on a selected contour. To be consistent with the default and range modes, where the right mouse button deselects the last object selected, in *add* mode, the right mouse button deletes the last point added.

3.5.3 Designing the Look and Feel (The GUI)

The SLICE interface consists of a window that contains both a 3D and a 2D view of the surface. The 3D window is located on the left, and displays a 3D projective view of the entire surface. The 2D view is located on the right, and displays the cross-section of the surface containing the current contour (see Section 3.5.2.3 for a definition of the current contour). Interaction in one window updates the other. For example, selecting a point in 2D causes the point to be highlighted in both views, as well as causing the red-highlighted current contour to be updated in the 3D view.

Beneath each view is a text area that provides information as to the current actions available with the mouse. This information is updated according to what key modifier is depressed as well as what is currently selected.

A menu bar at the top has a "File" menu which contains the standard (and thus externally consistent) "New", "Open", "Close", "Save", "Save as…" and "Exit" options. This menu was not included in the interface used in the experiments.

Widgets providing operations that apply to both the 2D and 3D views are located between the two views in the centre of the window. These include pushbuttons for "Undo move", "Select nothing" and "Delete".

Pressing "Undo move" restores the mesh to its shape prior to the last movement of points. After undoing a move, the button label changes to "Redo move", and pressing it will restore the mesh to its state after the points were moved. SLICE provides only a single level of undo due to storage limitations imposed by the complexity of the data.

"Select nothing" duplicates the functionality of the middle mouse button, deselecting all objects. It provides some redundancy in the interface. This may be useful since the mouse mappings are fairly complex. If the user cannot remember all of the mouse mappings, there is a visible way (i.e., using a labelled button) to perform the action.

"Delete" is slightly different from an action like move or add. It is simply an operation applied to a selected point or range of points. It does not, therefore, require mapping to a mouse input action. (Note that move can also be thought of as an operation applied to a selected object, but it requires further mouse input to indicate a new position.) The interface described in Section 3.5.2.3 is used for selection of a contour, point or range of points. A GUI widget such as a push button can then implement the delete action. As per the previous discussion on consistency in Section 3.4.3, an additional benefit of making delete a special case in the interface (i.e., using a widget instead of a mouse action) is that it draws attention to an action that has the potential to do damage if applied incorrectly. A confirmation dialogue can also be displayed when the action is performed. Since no task in the experiments required the delete operation, this button was not included in the experiment version of the interface.

Buttons that apply only to the 2D view are located directly beneath it. These include "Next", "Previous" and "Reset view".

“Next” and “Previous” simply cycle through the cross-sections, displaying each one in turn in the 2D view. The current contour in the 3D view is also updated. Again, these buttons provide redundancy in the interface since the arrow keys provide the same functionality.

“Reset view” restores the view to its original state, undoing any scroll and zoom viewing transformations that were made. There is a corresponding button located under the 3D view to reset that view, which undoes all rotation, scrolling and zooming.

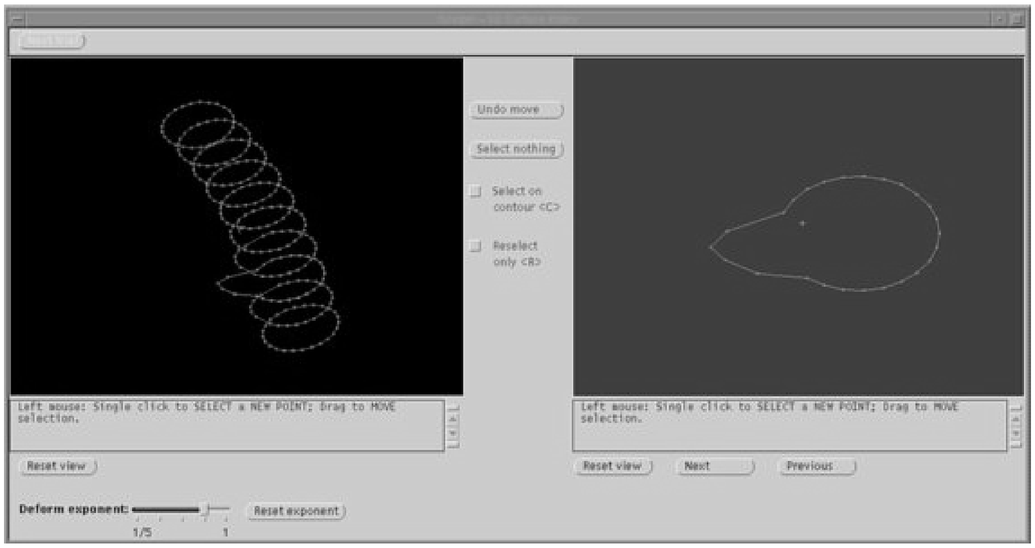
Under the 3D view are the widgets that apply only to this view. I have already discussed “Reset view”. In addition, there are radio buttons to determine the current shading style, a “Disclose Inner surface” toggle button, a slider to set the “Deform exponent” for secondary point motion (see Section 3.5.1), and a “Reset exponent” button to restore the slider to its default setting[1].

The radio buttons to change the current shading style provide options for showing only the contours, showing a wireframe view of the surface mesh, or for flat-shading the surface. A more sophisticated method of shading, such as Gouraud shading, is not used for performance reasons; even in flat-shading mode, the display changes to wireframe when the view changes or a part of the object is in motion. In Chapter 4, I will discuss a pretest that was used to determine what form of shading to use in the experiments. These radio buttons are not present in the final experiment interface.

When editing with SLICE, the portions of the surface to remove will often be on the inner side of a closed area. In these cases, it will be necessary to disclose the inner surface before editing. The “Disclose inner surface” toggle addresses this requirement by removing all front-facing polygons. This toggle button is not present in the final experiment interface.

The “Deform exponent” slider affects the surface, and the effects of its use are thus reflected in the 2D view. The slider, however, is really only useful in conjunction with the 3D view, and is therefore located with the other 3D-related widgets. If an area of points and a primary point are already selected when this slider is adjusted, it interactively changes the shape of the surface according to the selected exponent. If nothing is selected, the slider setting will apply to all subsequent motion of multiple ranges of points. A “Reset exponent” button will reset the slider to its default value. In Chapter 4, I will describe a pretest that was used to determine whether this feature should be included in the final experiment interface.

[1] Since the term “exponent” may not be familiar to medical users, this should be renamed in future versions.



Left mouse: Single click to SELECT a NEW POINT; Drag to MOVE selection.

Left mouse: Single click to SELECT a NEW POINT; Drag to MOVE selection.

Figure 3-6: SLICE interface used in the final experiment.

Figure 3.6 shows the interface used in the final experiment. There are some differences between this and the complete interfaced discussed here due to some features not being included in experiment (such as the shading style radio buttons, the button to disclose the inner surface and the delete button), and due to changes made as a result of feedback from pretests (the addition of “Select on contour” and “Reselect only” buttons) which I will discuss in Chapter 4. The interface in the experiment also contained a “Next trial” button that enabled the user to progress through a series of trials in the experiment.

4. Experiments

I conducted an experiment to empirically determine whether a 2D environment or a 3D environment is superior for editing surfaces reconstructed from contours. Participants from the University of Waterloo community ran trials in which they edited manually constructed surface data. The results of several pretests were used to fine-tune the environment for the experiment. In this chapter, I present a complete discussion of the pretests and final experiment, from the experimental design to the results and their implications.

4.1 Overview

The most important step when designing an experiment is to determine what question one hopes to answer with its results. This may seem obvious to the reader, but the kind of question posed will drive the direction of the work, and the precision with which one formulates it will determine the nature of the results. In this study, a first attempt at a question might be

"Is a 2D environment or a 3D environment best for editing surfaces reconstructed from contours?"

This question addresses the fundamental issue that I would like to investigate, but it is too vague. Several aspects need to be clarified. For instance, what does *best* mean? What is involved in *editing*? What *kinds of surfaces* should be manipulated? Each of these ambiguities relates to a different issue. The first is a lack of precision in the motivating question. The second is an experimental task that is too broad in scope. Finally, the third is an experimental environment that contains too many variables.

4.2 The Experimental Task

To make the question more precise, I first need to define "best". In the UWT project, the final edited surface will be used to make precise measurements of fluid flow through an artery. For this reason, an interface that allows accurate manipulations of the surface is important. However, it would be unrealistic to expect a user to spend a vast amount of time making a single, small adjustment. Thus the interface must allow the user to work at a reasonable speed to achieve the desired accuracy. Since speed and accuracy are the factors most important to the end user in the UWT project, I will define *best* to mean *fastest without loss of accuracy*. Speed and accuracy will be the dependent variables in my pretests and final experiment.

The second issue I must address is that of restricting the scope of the task that I investigate. I determined in Section 3.1 that the primary function of SLICE is to allow the removal of anomalies from surfaces reconstructed from contours. Because a typical editing session involves a great many steps (and thus an abundance of variables) that depend on the data set involved, any attempt to mimic a real world editing scenario in the experiment would be fraught with ambiguity. For this reason, I must choose a simplified version of the editing task that still contains the key elements of a real life situation.

The first step in making this choice is to eliminate any tasks that duplicate participants' behaviour. In Section 3.3.2, I outlined the elements of interaction, and found them to consist primarily of adding, deleting, selecting and moving objects. Contour manipulation is no different from point manipulation in terms of either the mental or physical processes of determining where to select on the screen and how to navigate the selected item in 3-space. Thus, for the experiment, I will concentrate solely on point manipulation, as it will be the more commonly performed activity in the UWT project.

Since deleting points is simply applying an operation to a selected point, and adding a point is simply selecting a new location in which to place a point, the tasks of selecting and moving a point encompass all of the interactions involved in point manipulation.

The task chosen for the experiment was to remove a bump or hollow from a surface, using only the existing points. In other words, points may not be added or deleted. The participant must reshape the surface by selecting and moving existing points. I used the same task in both the pretests and the final experiment. This task covers all aspects of interaction necessary in the SLICE interface including changing the view, selecting and moving points, and making three dimensional spatial judgements.

The third and final issue to address is what type of surface to use for the editing task. The participants for the experiment had no training or experience recognising artefacts in artery data. It was therefore infeasible to present a real data set to a participant for repair. Such a task would not have a clearly identifiable goal for the chosen participants. Instead, simplified data sets were manufactured for the experiment. I used a cylindrical tube as the surface, which enabled me to provide a goal that the participant could easily identify: returning the tube to a regular, cylindrical shape. This had the further benefit of guaranteeing a degree of similarity between trials. The cylindrical tube does, however, maintain some realism as it is an idealised artery shape.

I first considered having the cylindrical goal surface shown on the screen for the user. However, this made the experiment into a matching task, which is different from a real world situation where there is nothing to which the surface can be compared. It is likely to be easier to match an existing surface than to create a new one. The presence of the goal surface on the screen would provide depth and positional cues not ordinarily present in the scene, and so one was not displayed.

I considered several options for defining the end of a trial. One possibility would be to have the trial end when the participant brought the surface within a prescribed tolerance of the goal surface. This, however, leads to the measuring of speed, and not accuracy. This is undesirable for two reasons. First, I want to see if users will achieve the same accuracy in 3D as in 2D, so I must not force a particular level of accuracy in the trial. Second, I would be unable to differentiate between a slow user who is good at 3D interaction, and a fast user who struggles for a long time because he has trouble with 3D interaction.

Another option for determining the end of a trial was to let the participant manipulate the data for a fixed amount of time, and then to measure the accuracy at the end of the elapsed period. This, however, would lead to greater inter-participant variation within each environment in the key factor of interest: accuracy. Some participants would be inherently slower than others, and thus be interrupted at different stages of their repair strategies. This would make it more difficult to obtain meaningful statistical results when comparing differences between environments. In addition, people might behave differently, and use different strategies when under artificial time pressure.

The option that I chose was to have the participant determine for himself when the editing task is complete by making a subjective judgement that the surface has been restored to the goal cylinder. Thus, the participant is matching the experimental surface to an implied target surface. Such a technique has been used in previous experiments in the form of colour [Sch85] and shape [Bos87, Rue89] matching to imitate a creation task. However, unlike a creation ask, matching allows the experimenter to measure accuracy as well as speed. Using the matching technique, I was also able test whether accuracy judgements are difficult to make with the SLICE interface.

Permitting the participant to determine his own stopping criterion allows an inherently slow participant to be distinguished from one who is having trouble perceiving accuracy. If I used a tolerance-level stopping criterion, both types of participant would have long trial times and an equal (by enforcement) measure of accuracy. With my method however, participants having difficulty making accuracy judgements may have faster trial times, but will score lower on accuracy. In addition, letting the participant determine the end of the experiment eliminates any trials where the user might become frustrated after a long period of struggling to arrive at the desired shape. If the user gives up on an editing task, such trials will be reflected in a poor accuracy result.

The subjective factor that a participant-determined stopping condition introduces into the experiment is somewhat mitigated by the instructions that I, as the experimenter, provided to the participant. These included a rough guide as to the level of accuracy and speed that were sought. In Section 4.3.7, I discuss the instructions provided to the participants in more detail.

4.3 Method

Once I had determined the question, I had to decide how best to answer it. The basic experimental design is the same for the pretest experiments and the final experiment. In this section, I present the elements of subject selection, data design and procedure common to all the experiments. I discuss the those aspects that pertain to a particular experiment in Sections 4.4 through 4.7.

4.3.1 Participants

Participants for the pretests were all recruited from the University of Waterloo's Computer Graphics Lab (CGL) and, as such, had some previous exposure to interactive 3D graphics. I hoped to receive useful feedback from these participants for fine-tuning of the final experiment.

All participants in the pretests, except me, were male. My own results were typically examined separately as those of a trained user. Only one participant in the pretests was left-handed, but used the mouse with his right hand. Participants were all in their mid-twenties to early thirties. They received no compensation for their participation in the pretests.

For the final experiment, there were sixteen participants in all. Of these, thirteen were male and three were female. The participants were recruited by a news posting to the computer graphics course newsgroup, the CGL newsgroup, and individual email requests.

One participant was a fourth year undergraduate student, three were professors, and the rest were graduate students. All sixteen were from the Department of Computer Science. I wanted to use subjects who were familiar with using computers since the experimental task was fairly complex. In addition, future users of the SLICE system, who represent the target population, have experience using computers.

Two participants were left-handed, only one of whom used the mouse with the left hand. Participants had a range of experience in 3D interactive graphics, ranging from none at all to extensive training. CGL members were not compensated for their participation in the study as such contribution is expected as a condition of lab membership. Non-members were paid ten dollars for their contribution.

After using the SLICE interface extensively, I and one other participant became "expert users". Our results were handled separately from the regular participants.

4.3.2 Data Creation

I created the data set for each trial by adding a deformation to a cylindrical tube of unit radius comprised of ten cross-sections, with each cross-section having 25 points. I will refer to this as the *base cylinder*. Instead of creating several deformations in one data set, I deformed the base cylinder with a single bump or hollow for each individual trial to control for editing style. In other words, participants are forced to tackle each deformation in sequence, rather than switching back and forth between separate bump removal tasks. This provides an element of consistency between participants, and is in fact the way in which expert users are expected to use the system. Having only one bump for each trial instead of many, as might occur in reality, also makes it easier to measure the time taken to remove the bump.

Because I have few data sets from the UWT project, there is no clear pattern to the appearance of the scanning artefacts. Thus, for the experiment, I attempted to reproduce all possible classes of bump. The deformation added to the base cylinder for each data set was one of a spike, a spine, a ridge or a hill (refer to Section 3.3 for a description of each type of deformation) according to the pretest or experiment in question. I provide details pertaining to individual experiments in Sections 4.4 through 4.7.

The deformations were created using the SLICE editor. Starting with the base cylinder, points were moved to create a bump or hollow. To avoid always having a simple inverse operation of the data-creation steps to return the surface to its base cylindrical shape, deformations were created in a number of ways. Some deformations were made by selecting and moving a single group of points. These data sets are termed *default* or *simple*, depending whether they use the "Deform exponent" slider setting (refer to Section 3.5.1 for a description of the slider that controls the amount of movement of secondary contours) of ½ (default), or another setting (simple). Other data sets were made by selecting and moving one group of points, and then selecting and moving a second, overlapping set of points (thus influencing the position of some points by a two-step process). The data sets created with this two-step method are termed *complex*, since a different slider setting was used when moving each of the two point selections.

When I chose the data sets to be used in each experiment, I ensured that for each deformation type, there were an equal number of bumps and hollows, that the bumps appeared equally often at the front of the surface, the middle and the back (with respect to the orientation of the bump at the start of a trial), that the deformation lay equally often completely on-screen and partially off-screen (in the 2D view), and that there were an equal number of deformations with an odd and even number of points, and in the case of spines and hills, an odd and even number of contours.

Recall from Section 3.5.1 that if a user selects a primary point for motion that is not connected to an adjacent range, a warning is issued and no action takes place. In the cases of spine and hill data sets, I ensured that the displaced points could all be selected and moved together so that this situation would be unlikely to arise during the experiments.

4.3.3 Experiment Design

In Section 4.2, I determined that speed and accuracy should be the measures of the "best" editing environment. Thus, speed and accuracy are the dependent variables for all experiments. For the final experiment, the environment is the independent variable, and has two "levels:" 2D and 3D. Some of the pretests have a different independent variable, such as surface type. I will discuss such variations on a per-experiment basis in Sections 4.4 through 4.7. In addition, the nature of the deformation (spike, spine, bump or hill, as defined in Section 3.3) is a second independent variable in some pretests, making those factorial design experiments.

I chose a within-subject, or "repeated measures" design for the experiments (i.e., each subject receives all conditions) for two reasons. First, it would have been difficult to recruit the large numbers of participants required for a between-subjects design (i.e., a different group of subjects receives each condition). Second, each participant needs significant time to learn the interface through instructions and practice trials. A between-subjects design would involve extra training time.

To minimise the effects of learning, I used a counterbalancing scheme. I made "order" a between-subject variable by giving each participant a different random order of levels of the independent variable. Since there are two independent variables in all experiments, I must combine every level of one with every level of the other. Each such combination is termed a *condition*. Thus, I randomise the order in which the conditions are presented to each participant. Even the order in which practice trials are presented is randomised, in case subjects choose to do only the first one. I ensured that an equal number of each condition was presented in each session. In fact, each participant was presented with the same data sets during the experiment, but in a pseudo-random order. However, each trial used a different data set, even when conditions were repeated.

Each participant completed a single session comprising two or three practice trials followed by 12 to 24 actual trials, depending on the experiment. The number of trials given to each participant was governed both by the length of time it takes to complete one trial, as well as the number of conditions required for the experiment. One hour was divided by the number of conditions multiplied by the time per condition (i.e., time for one trial) to determine how many times the complete set of conditions could be repeated for the experiment. I did not wish to make the entire session longer than one hour for fear the participant would lose concentration, or become fatigued. This meant I had to use a fairly large participant pool to obtain sufficient data since the collection of a single data point took approximately two minutes.

4.3.4 Procedure

The experiment is run from a Unix shell script. Each trial is a new invocation of the SLICE program, with command line parameters to specify the conditions for the trial, such as the data file used and the surface rendering type. The shell scripts were generated from a C program that used **`srand()`** and **`drand48()`**to pseudo-randomly generate the parameters that create the trial conditions. This script generation program was run once for each participant, and a script file was output that had the participant's initials as a suffix.

On each trial, the participant is presented with the SLICE interface as depicted in Figure 3.5. Note that the appearance of the interface was slightly different for the pretests; such differences are described in Sections 4.4 to 4.7. In the 3D view, the tube data set appears in the centre of the window, from an eyepoint looking down the z-axis. In the 2D view, a single contour lies in the centre of the window. After the participant has completed the interactions necessary to eliminate the deformation, he presses the "Next Trial" button to stop the timing, end the current trial, and begin the next.

Participants may pause between trials at this point if they desire, as timing for the new trial does not begin until the mouse is moved. Between trials, the interface disappears and reappears for the new trial, since each trial is a new invocation of SLICE.

4.3.5 Data Collection

The philosophy behind the data-gathering process was to collect as much information as possible, in case it might prove useful in the analysis to follow. By gathering all information thought to be relevant, I hoped not to have to repeat any experiments should some trend be revealed in the initial analysis that provoked a look at unanticipated factors.

The experiment program recorded information for every window entry or exit, mouse click and button press event generated by the participant. For these events, a timestamp, event code and description were stored. The information was written out to four files. The naming convention for these data files used the participants initials as a prefix to the file names. The files were called abc.accuracy.info, abc.event.info, abc.event.data and abc.move.info, where "abc" were the participant's initials. Examples of these files and details of their contents appear in Appendix B.

The accuracy file recorded the final distance of each point from the corresponding point on the base surface. The two event files stored information about the start time, duration, and nature of every significant user interaction with the interface. Each time a point was moved, the movement file kept track of its new location, the distance it was translated, the current selection type, and whether the move was done in 2D or 3D.

The information contained in the four files described above was distilled into a single file for analysis by SAS (a statistical analysis software package). A C program was used to process the data files, generating a file abc.out that was read by SAS. This file contained one line per trial, with the subject's initials, trial number, information on the data set used, total elapsed time, total manipulation time, total viewing time, accuracy and other optional information, depending on the particular pretest or experiment. The accuracy was calculated as the total error summed over each point and each contour. An example of this file is included in Appendix B.

4.3.6 Data Analysis

I used an analysis of variance (ANOVA) test to determine the means, variance and whether there was any significant difference in the means between groups. An ANOVA is used for interval or ratio data when the underlying distributions are approximately normal, as is expected of my speed and accuracy measurements. I used a single variable ANOVA for the experiments with only one independent variable, and a two variable, no replacement ANOVA for my factorial design experiments, which have two independent variables.

The F value of the ANOVA test reported for each experiment is the ratio of variance between groups to variance within groups. A ratio close to 1 indicates that the groups sampled come from the same population and the independent variable has no effect. As F gets larger, one becomes increasingly confident that differences among groups were due to the effect of the independent variable rather than to chance. As a standard level at which a result can be said to be statistically significant, scientists have agreed that the likelihood of obtaining the observed differences in samples due to chance should be less than 1 in 20. Some scientists are more strict, requiring that the test indicate a difference less than 1 time in 100. This is called testing at the .05 or the .01 level of significance and is referred to as $p < .05$ or $p < .01$. I will use $p < .05$ for testing my results.

4.3.7 Instructions

Each participant received both written and verbal instructions at the beginning of the experiment. I gave the same verbal instructions regarding the experiment procedure to all participants, but also permitted them to ask questions if they had difficulty with the interface. After the interface was displayed on the screen with the first sample data set, I described the format of the experiment. I told each participant that his task was to move points on the surface to eliminate the single bump or hollow that would appear, thus returning the cylinder to a smooth, regular shape.

I informed participants that their accuracy in performing the experimental task was more important than their speed. Because my final goal is to create a surface suitable for generating a precise volume mesh, accuracy is more important to me than speed (provided, of course, that the difference in speed is not extreme).

I indicated that they should strive to make each surface presented into a surface as close to a cylinder as possible in a reasonable amount of time, and without worrying about minute details. In order to provide a guideline for what a "reasonable amount of time" should be, I estimated that an average trial should take roughly two minutes. This was determined by previously timing several people for some sample tasks. I instructed participants that I would measure accuracy by the final shape of the surface, and not by the position of individual points. This reflects what is important in fluid-flow analysis for the UWT project.

I told participants that they had three practice trials before the real experiment began, and that they should use these trials to familiarise themselves with the interface. To this end, I left written instructions with the participant that included a tutorial to guide them through the use of all the features of the interface. The written instructions also included a letter explaining the purpose of the experiment. The full written instructions provided to the participant for the final experiment appear in Appendix C. Instructions for the pretests were similar.

During the fifteen minutes following my initial instructions, I visited the participant in the experiment room twice to see if there were any questions about the interface. I then left the participant alone to do the experiment, and then fill out a questionnaire.

4.3.8 Pretests

Because the SLICE interface is so complex, there are many variables in the experimental environment that have the potential to affect the results. It is therefore desirable to eliminate as many variables as possible so that I can be more certain that any experimental result obtained is due to the independent variable, and not to some factor in the environment. Consequently, I used several pretests in an effort to eliminate some of the variables such as the surface used for rendering, and the controls of the GUI. The pretests also served to fine-tune the experimental design. They helped to validate the task, the data-gathering process and the reliability of the results.

I used the first pretest to determine the most effective surface type (both objectively and subjectively) for displaying the data among contours only, wireframe mesh or flat-shaded. By using only one surface type in the final experiment, I eliminated one variable in the environment.

I used the second pretest to determine which views (2D, 3D or both) should be presented to user. By using only the best arrangement of views for the final experiment, I eliminated another environmental variable.

I used the final pretest to determine whether any bump types could be eliminated from consideration for the final experiment in a further effort to reduce variables.

4.4 Surface Type Pretest

To reduce the number of variables in the experiment, I wanted to have only one surface type for rendering objects. This pretest was designed as an informal means of determining the best surface type to use, both from a subjective and an objective point of view. The objective measure of "best" was that described previously in Section 4.1: fastest for editing without loss of accuracy. The three surface types among which I had to choose were contours only, wireframe mesh and flat-shaded. Due to performance limitations, I was unable to consider such options as Gouraud-shaded or texture-mapped surfaces.

4.4.1 Experimental Design

In this pretest, I was looking for a trend, rather than a rigorous result. Consequently, there were only four subjects, including myself. Each subject received three practice trials, all using hill data sets.

This pretest had a factorial design, since both surface type and bump type were manipulated. For each of the three surface types, all of the four bump types (spike, spine, ridge and hill) were tested, resulting in a total of $3 \times 4 = 12$ conditions. Two repetitions of each condition were used for a total of 24 trials in this pretest. For the spine and hill data sets, default, simple and complex slider settings were present in equal proportions.

Only the 3D interface, with no 2D display, was used for this experiment.

4.4.2 Participant Feedback

Participant feedback from this pretest served to uncover the bulk of the problems with the SLICE interface. Participants found that they often accidentally selected a point that was two contours away from their current selection, especially in wireframe and shaded modes. This caused them to lose their entire selection, as it was replaced with the newly-selected point. These comments led to an augmentation of the interface, whereby selections made with the <Shift> key depressed are restricted to a contours adjacent to any existing selection.

Participants reported similar difficulties when re-selecting a point to make it the primary point for motion. If an unselected point were accidentally chosen, the current selection would be lost. To reduce the occurrence of this problem, a second feature was added to the interface. When set, the “Reselect only” toggle restricts the points that can be picked to those that are already selected. This assists in the selection of primary points.

Further participant feedback led to the improvement of the “Select on Contour” toggle. Instead of restricting selection to points on the contour on which the last selection was made, I changed the functionality so that when set, selection can occur only on the current, active contour (see Section 3.5.2.3). This contour can be changed using the arrow keys. It is highlighted in red, thus providing a cue to the user as it is “moved” through the object. Now, the “Select on contour” toggle makes it easy for the user to locate and select points on any given contour by restricting the degrees of freedom inherent in the problem.

Participants experienced further frustration when they selected a range of points, and “missed” the first point, i.e., they started the range selection one point too early or one point too late, and then dragged out the rest of the range. Because all range selection was done by dragging the mouse, there was no way to add or deselect a point from the end of a range; the whole range had to be selected again. To remedy this, I added the ability to deselect the endpoint of a range using the middle mouse button, and to select an additional point at the end of a range using shift plus the left mouse button.

Participants also pointed out the confusion caused by having the direction of range selection fixed with respect to the object instead of the world. When the object was rotated 180 degrees about the y axis, the direction of range selection would reverse from the user's point of view. This inconsistency was eliminated by adjusting the direction of selection according to the current degree of rotation obtained from the model transformation matrix. For later experiments, the direction of rotation was always counter-clockwise *on the screen* as the user moved the mouse to the right. Now, depending on the 3D rotation, the direction of range selection in the 3D view may not always correspond with the direction in the 2D view. This may cause problems if the user is selecting in one view while watching the other. On the other hand, the discrepancy can provide some feedback as to the current 3D rotation.

A final detail of the interface was corrected at this point by having the interface window appear in the same place on the screen for each trial.

Perhaps a more serious problem than those encountered with the interface involved the nature of the data. Some participants found themselves simply zooming in from the default view, and performing no rotation of the surface. This made the cylindrical surface appear as a set of concentric circles, and hence the points that formed the deformation were immediately obvious (see Figure 4.1).

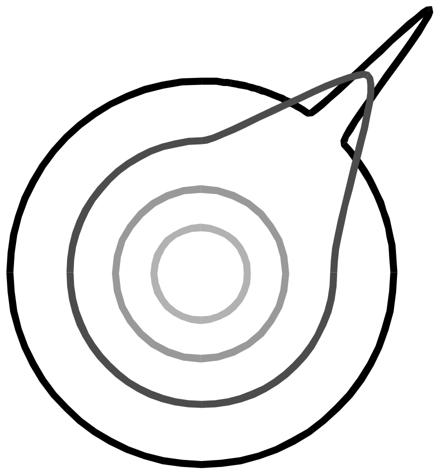


Figure 4-1: Cylinder perceived as concentric circles.

The arrangement of the contours as a cylinder made the deformation into an emergent feature, i.e., a global property of the data not visible in isolation [Wic92]. In other words, the bump became apparent as soon as it was placed in close proximity to the other non-deformed contours. In addition, the human perceptual system is quite adept at perceiving circles and detecting when a point lies slightly off the circle [Arn65]. Both of these global processing and feature detection phenomena tend to be pre-attentive and automatic. Thus, my choice of data set produced an experimental task that participants could perform with great ease and accuracy: aligning points to match a circle. Such a task fails to mimic the ambiguity of the real world task of removing deformations from an artery surface.

I decided to modify the test data sets used in the experiments in case the effect of concentric circles introduced a confounding variable into the study. I felt that simply squashing the circular contours to form ellipses would not be sufficient to eliminate the effect. My next thought was to change the surface from a cylinder to a Y-shaped tube. I still believed, however, that this would still be prone to the same pitfalls as a cylinder when viewed from certain angles.

The solution I implemented was to make the tube into a spiral shape. In other words, the contours are laid out along a spiral path rather than along the straight line axis of a cylinder. A partial spiral added sufficient curvature to the surface so that there was no angle from which the user could line up the contours.

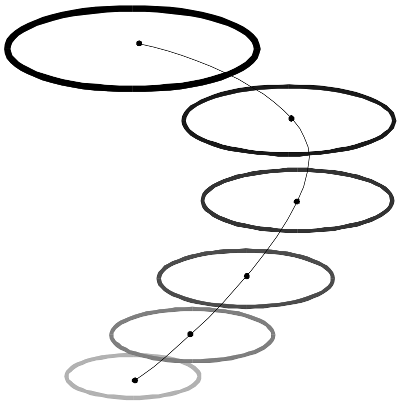


Figure 4-2: Bending the cylinder into a spiral.

To further mitigate the effect, I used elliptical, rather than circular contours. Figure 4.2 illustrates the resulting surface. To create the new data, I warped my existing data sets with the following equations:

$$x = (1 + \varepsilon)\, x + \alpha(z), \text{ where } \alpha(z) = \sin(z / k)$$

$$y = y + \beta(z), \text{ where } \beta(z) = \cos(z / k)$$

'k' was chosen as 4 to sweep out ¼ of a circle and 'ε' was chosen as 0.4 to stretch the unit circle to an ellipse with a major axis of 2.8 (i.e., $2 * 1.4$), and a minor axis of 2.

Since all contours of the base surface are still identical under this mapping, the smooth, regular shape of the surface is preserved. This allows participants to easily identify irregularities and recognise when they have been corrected.

In addition, because the data sets were warped after the deformations had been added, the deformation exponent slider no longer performs a perfect inverse operation when the user tries to eliminate the bump. Without this effect of the warping, the slider might be more useful in the experimental situation than it would be in the real world. With the warping of the data, however, the effectiveness of the slider is more realistic.

4.4.3 Results

Because there were so many changes to the interface as a result of the feedback from the first group of participants, the experiment was repeated. Two fresh participants in addition to one participant from the first version of the pretest performed the experiment with the new interface and warped surfaces. I used the data gathered during this run of the experiment to determine the most effective surface type to use in the rest of the experiments. I present the main results in the tables below, but a full analysis can be found in Appendix D.

In the tables in this and the following sections, the letters in the "Significant difference?" column are used to indicate whether or not the result in the second column (mean speed or mean accuracy) is statistically significant at the .05 significance level (i.e., a large value of F and *p* < .05). If two rows have the same letters in the "Significant difference" column, then there is no significant difference, whereas means marked with different letters are significantly different. For example, in the table below, we see that the *hill* bump type is marked with an "A," while the *ridge*, *spine* and *spike* bumps types are marked with a "B." This means that there was a statistically significant difference at the .05 level in speed between the hill and the rest of the bump types (F = 6.84, *p* = 0.0005), but no significant difference in speed among the other bumps types.

Bump type	Mean speed (seconds)	Standard deviation	Significant difference?	No. of repetitions
hill	169.12	142.89	A	18
ridge	96.58	80.37	B	18
spine	87.00	62.41	B	18
spike	37.98	33.46	B	18

Table 4.1: Mean speed for each bump type

The first test I did on the data was a "sanity check" to see that it took longer to correct the deformation the more points there were to move. This trend is indicated by the mean speeds in Table 4.1. The spike (only one point displaced) was faster than the spine (single points along a range of contours), which was in turn faster than the ridge (points along a single contour). The most complex bump type, the hill (an area of points across multiple contours), was slowest of all. Although the differences are not statistically significant, it is likely that they would become so with data from more participants.

Surface type	Mean speed (seconds)	Standard deviation	Significant difference?	No. of repetitions
shaded	112.23	70.61	A	24
wireframe	103.12	129.23	A	24
contours	77.66	89.94	A	24

Table 4.2: Mean speed for each surface type

The mean speeds for each surface type in Table 4.2 show that participants were fastest with the contour surface, and slowest with the shaded. These differences were statistically non-significant (F = 1.00, *p* = 0.37), although a larger sample size may have made the result more pronounced.

Participants unanimously found the surface drawn with only contours to be the easiest to manipulate, and preferred it over the wireframe and shaded surfaces. They reported that they were much more willing to spend the time to place the points precisely with the contour surface since that surface type was much less frustrating to use than the other two. In spite of the extra care spent with the contour surface, it still had the shortest associated mean time.

Bump type	Mean accuracy	Standard deviation	Significant difference?	No. of repetitions
hill	1.7722	0.6788	A	18
ridge	0.9962	0.9984	B	18
spine	0.2696	0.2524	C	18
spike	0.0593	0.0742	C	18

Table 4.3: Mean accuracy for each bump type

To secure confidence in my data and my analysis process, I compared the accuracy for each bump type as an additional check. Recall from Section 4.3.5 that accuracy is measured as the total displacement of the points from the base surface, and thus a lower number means higher accuracy. The units used to measure accuracy are the same as those used to measure the size of the ellipses in the base cylinder, which had a major axis of 2.8 units and a minor axis of 2 units.

As one might expect, the mean accuracy data in Table 4.3 shows that there was a statistically significant difference in accuracy at the .05 level between some of the bump types (F = 36.20, $p = 0.0001$). Not surprisingly, accuracy, like speed, was best for the simplest data type (the spike) and worst for the most complex (the hill). The fact that further statistical significance emerges between the various bump types when accuracy is measured supports my previous assumption that the results for speed differences between bump types would become more significant with more participants.

Surface type	Mean accuracy	Standard deviation	Significant difference?	No. of repetitions
shaded	1.0698	1.0945	A	24
wireframe	0.8118	0.9328	B	24
contours	0.4406	0.5193	C	24

Table 4.4: Mean accuracy for each surface type

There was a statistically significant difference in accuracy at the .05 level with each of the three different surface types (F = 8.00, $p = 0.0008$). Accuracy was highest with the contour surface and lowest with the shaded surface. The mean accuracy for each surface type is reported in Table 4.4.

Users were fastest and most accurate with the contour surface. These results were consistent with participants' verbal feedback. Contours only was found to be the easiest surface type to manipulate, and was preferred by all subjects.

As a final check, I looked for an interaction between the bump type and the surface type to ensure that the contour surface did in fact globally produce higher accuracy. I found no tendency toward such an interaction (for accuracy, F = 1.74, $p = 0.13$, and for speed F = 1.40, $p = 0.23$). Thus, the contour surface was used for all subsequent pretests and experiments. This also allowed more direct comparisons between the SLICE interface and the IDEAS interface, which uses only contours to display the surface for editing.

4.5 Additional View Pretest

In the final experiment, I am interested in comparing editing in 2D and 3D. When editing in a 2D environment, however, the presence or absence of a 3D view may affect the user's performance (and likewise when editing in 3D). To avoid testing all combinations of 2D and 3D views in the final experiment, I used a pretest to determine whether in fact an additional view was helpful to the user.

4.5.1 Experimental Design

Since I was again looking only for a trend, only four subjects in addition to myself and the other expert user performed this pretest. I ran it in two separate sessions. One session compared editing in 2D using only the 2D display with editing in 2D using an additional 3D display for surface viewing only. The other session made the equivalent comparison for editing in 3D, comparing performance with and without an additional 2D display for viewing only. Each subject received three practice trials and sixteen actual trials in each session.

This pretest had a single variable design: the editing environment was the independent variable manipulated. For both of the editing environments (i.e., with and without an additional view), all of the four bump types (spike, spine, ridge and hill) were tested, resulting in a total of $2 * 4 = 8$ conditions. Two repetitions of each condition were used for a total of 16 trials in this pretest. As in the surface type pretest, default, simple and complex spine and hill data sets were present in equal proportions.

When both displays were present, the user was able to differentiate between the display for editing and the display for viewing by the background colour. The display for editing had a black background, whereas the display for viewing only was grey.

4.5.2 Participant Feedback

As expected, participants found the 3D view more difficult to use than the 2D view. They reported that since they found the 2D view easier to use, they felt more inclined to take extra care with accuracy when they were using it. In a real world scenario, however, one can assume that users would be more motivated to be precise in whichever view they were using.

I added the instruction to the participants to stress accuracy over speed as a result of the feedback from this pretest. I hoped that it would urge users to take as much care with accuracy in 3D as in 2D. With this instruction, I now hoped to measure differences in effectiveness of the interface, rather than in motivation. The fact that users prefer to work in 2D, however, is already a telling sign that the 3D interface is arduous to use.

After this pretest, I uncovered another problem with my data sets unrelated to the use of additional views. From participants' reports, I found that it was still to easy to remove the deformations from the cylinders by performing the inverse process to their creation. If a participant happened upon the right group of points, he could smooth out the bump perfectly in a single movement. Since this is unrealistic in terms of the real data, I introduced some random jitter into the points according to the following equation:

```
jitter = 0.85 + 0.3 * drand48()
x = x * jitter; y = y * jitter
```

After the points had been stretched from a circle to an ellipse, I jittered a random point on each bump by a small, random amount within a certain range. The effect of this was such that if the participant did select and move the right group of points (i.e., the same ones that had been moved in the creation of the deformation), the points would not fall exactly into place. Some further fine-tuning of single points would be required to smooth the surface properly. This effect better reflected interaction with actual artery data.

4.5.3 Results

I used the data from this pretest to determine whether or not an additional view should be present in the interface for the final experiment. Since I wanted to use the same interface for all trials of the final experiment, I did not differentiate between bump types in the data. A single factor ANOVA was performed on both the 2D and 3D data to find out whether an additional view was useful over all data types. Once again, I present the main results in the tables below, with a full analysis in Appendix D.

View type	Mean speed (seconds)	Standard deviation	Significant difference?	No. of repetitions
2D only	55.84	45.42	A	46
2D with 3D view	59.23	60.19	A	46

Table 4.5: Speed for editing in the 2D views.

The presence of the 3D view when editing in 2D had no statistically significant impact on participants' speed (F = 0.093, *p* = 0.76), although the speed with the 3D view was slightly higher than without it (Table 4.5). It does seem reasonable that a user would be slower with the additional 3D view since it provides extra information to process.

View type	Mean speed (seconds)	Standard deviation	Significant difference?	No. of repetitions
3D only	70.25	62.13	A	49
3D with 2D view	73.64	57.64	A	49

Table 4.6: Speed for editing in the 3D views.

When editing in 3D, the presence of the 2D view had a statistically non-significant effect on speed (F = 0.079, *p* = 0.78). As in the 2D session, the mean speed with the additional view was slightly greater than without it (Table 4.6). From verbal reports from the participants, it seems that the 2D view was used to aid with precision. The extra effort that participants made to be precise when the 2D view was present may have slowed them slightly.

View type	Mean accuracy	Standard deviation	Significant difference?	No. of repetitions
2D only	0.3441	0.4106	A	46
2D with 3D view	0.3049	0.4038	A	46

Table 4.7: Accuracy achieved with the 2D views.

Table 4.7 reports the mean accuracy for editing in 2D with and without the additional 3D view. When editing in the 2D view, there was no statistically significant difference in accuracy with the presence of the 3D view (F = 0.21, *p* = 0.64). The 3D view did not seem to provide the kind of information that assisted users with precise point adjustments, since there was no significant improvement in accuracy with its presence. There was also no decrease in accuracy, however, since the 3D view provides additional information to complement the 2D view.

View type	Mean accuracy	Standard deviation	Significant difference?	No. of repetitions
3D only	0.4736	0.7334	A	49
3D with 2D view	0.3241	0.4042	A	49

Table 4.8: Accuracy achieved with the 3D views.

As anticipated from the verbal feedback, accuracy was slightly better when the 2D view was present when editing in 3D (Table 4.8), although this result was also statistically non-significant ($F = 1.56$, $p = 0.21$). Users consistently reported that the 2D view helped them to be more precise. This may be because the 2D view is, by construction, parallel to the plane of the contour. By the nature of the 3D view, the contour being edited is often rotated so that it is no longer parallel to the viewing plane.

There was a wide variation between participants in terms of which view combinations were most useful. Some subjects never used the extra view; others some used it extensively. Those participants who used the extra view were adamant about the need for its inclusion. Aside from the one participant who found the extra view distracting, those who did not use it were generally unconcerned about its presence. Thus, participants found the extra view helpful at best, and redundant but harmless at worst. It provided additional information for those who wanted it, and was ignored by those who didn't. The interface was therefore constructed with the additional view for all subsequent experiments.

4.6 Deformation Type Pretest

At the outset of my inquiry into 2D and 3D environments for surface editing, I expected that users would perform better in a 2D environment for tasks of a 2D nature. In other words, where the deformation was restricted to a single contour, users would do better correcting it in a strictly 2D environment. My hope was that a 3D environment would be better for editing deformations that were 3D in nature, i.e., spanning multiple contours. With this pretest, I thus wished to eliminate the spike and ridge deformations as conditions for the final experiment since they affect only a single contour. If participants perform better in 2D for these types of deformations, then only spine and hill data need be tested in the final experiment.

4.6.1 Experimental Design

Instead of running more participants through an experiment, the data used to determine whether the spike and ridge could be eliminated from the final experiment were taken from the additional view pretest, and simply analysed differently. Only the data from the sessions with both views present were considered (i.e., sessions having 2D editing with a 3D view and 3D editing with a 2D view). For each of these two environments, all four bump types had been tested, and two repetitions had been made of each condition, resulting in 16 trials per participant ($4 \times 2 \times 2 = 16$). Six participants had performed the pretest. This meant that there were 96 data points in total, consisting of 24 for each bump type, with half from each environment. For this pretest, each of these groups of 24 data points was analysed separately using environment type, i.e., 2D versus 3D, as the independent variable.

4.6.2 Results

The following tables present the main results, with a complete analysis in Appendix D. The conditions analysed in this pretest were the same as those used in the final experiment, so I could look at trends to anticipate the results of the final experiment. The spine and hill data did not produce any statistically significant results with only the small number of participants used in the pretest, however, and so merit further study with more people in the final experiment.

Environment	Mean accuracy	Standard deviation	Significant difference?	No. of repetitions
2D	0.0743	0.0876	A	12
3D	0.1002	0.1408	A	12

Table 4.9: Mean accuracy for spine deformation.

Environment	Mean speed (seconds)	Standard deviation	Significant difference?	No. of repetitions
2D	52.49	44.90	A	12
3D	59.01	28.78	A	12

Table 4.10: Mean speed for spine deformation.

The means reported in Tables 4.9 and 4.10 reveal no statistically significant difference in speed or accuracy for spine deformations (for speed, F = 0.18 and *p* = 0.68, and for accuracy, F = 0.29 and *p* = 0.59). Participants were a little slower and a little less accurate in 3D. Further study with more participants in the final experiment will show whether or not this trend is significant.

Environment	Mean accuracy	Standard deviation	Significant difference?	No. of repetitions
2D	0.8467	0.4361	A	12
3D	0.8239	0.4275	A	13

Table 4.11: Mean accuracy for hill deformation.

Environment	Mean speed (seconds)	Standard deviation	Significant difference?	No. of repetitions
2D	131.70	64.80	A	12
3D	145.50	58.61	A	13

Table 4.12: Mean speed for hill deformation.

As illustrated by Tables 4.11 and 4.12, the same trend in speed occurred for hill data as for the spine data, i.e., participants were slightly faster in 2D than in 3D. Accuracy, however was slightly better in 3D. Neither result was statistically significant (for speed, F = 0.31, and *p* = 0.58, and for accuracy, F = 0.017 and *p* = 0.90). Again, this merits further study with more participants.

Environment	Mean accuracy	Standard deviation	Significant difference?	No. of repetitions
2D	0.0255	0.0207	A	11
3D	0.0212	0.0223	A	12

Table 4.13: Mean accuracy for spike deformation.

Environment	Mean speed (seconds)	Standard deviation	Significant difference?	No. of repetitions
2D	16.29	7.57	A	11
3D	26.29	10.66	B	12

Table 4.14: Mean speed for spike deformation.

Table 4.13 shows that there was no significant difference in accuracy between 2D and 3D (F = 0.23, *p* = 0.64) for the spike data. As predicted, however, participants were significantly faster at the .05 level in 2D than in 3D (F = 6.61, *p* = 0.02). According to the definition of "best" from Section 4.2, this means that 2D is the best environment for removing spike deformations.

Environment	Mean accuracy	Standard deviation	Significant difference?	No. of repetitions
2D	0.2443	0.0976	A	11
3D	0.3095	0.2072	A	12

Table 4.15: Mean accuracy for ridge deformation.

Environment	Mean speed (seconds)	Standard deviation	Significant difference?	No. of repetitions
2D	30.47	10.75	A	11
3D	57.78	27.39	B	12

Table 4.16: Mean speed for ridge deformation.

Just as the spike data confirmed my predictions about 2D being a better environment for editing 2D deformations, so did the ridge data. Table 4.15 presents the mean accuracy for the two environments; there was no significant difference between them (F = 0.90, $p = 0.35$). The mean speeds in Table 4.16, however, show that participants were again significantly faster at the .05 level at editing in 2D (F = 9.56, $p = 0.006$).

Spikes and ridges were eliminated from consideration in the final experiment, leaving only spine and hill data sets to be tested. This cut the number of experimental conditions in half, allowing more relevant data to be gathered during the limited duration of the final experiment.

4.7 The Final Experiment

Once a sound experimental environment was established, I was ready to test for my main result. This experiment was used to determine whether a 2D environment or a 3D environment was better for editing deformations of a 3D nature, namely spine and hill data sets.

4.7.1 Experimental Design

The final experiment was conducted over a period of one week. Each participant was brought into a small, quiet room to perform the experiment alone on the room's single workstation. Each session lasted approximately one hour, and consisted of three practice trials, twelve real trials, and a short questionnaire.

There were an equal number of trials for the 2D and 3D environments, and an equal number of spine and hill data sets, distributed evenly over the two environments. In order that the participants could finish the trials in about half an hour, three repetitions of each condition were included, since a trial takes approximately two minutes to complete ($2 \times 2 \times 3 = 12$). An equal number of data sets had bumps and hollows, and equal numbers were constructed using simple, default and complex slider settings.

In the first practice trial, the participant was free to edit in both the 2D and the 3D windows so that he could try everything in the instructions. The second two practice trials were presented in random order. One allowed editing only in 2D and the other allowed editing only in 3D.

The same data sets were presented to all of the participants, but each participant received them in a different random order.

4.7.2 Results

I present the main results in the tables and charts below. A complete statistical analysis can be found in Appendix D. Spine and hill data were grouped together in this experiment, and a single variable ANOVA was used to test for differences in the mean speed and accuracy between 2D and 3D.

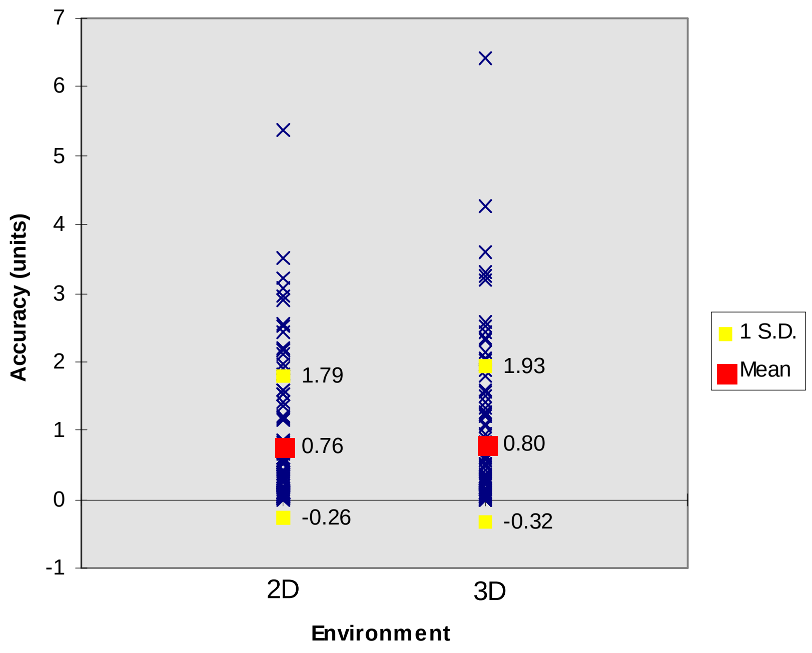


Figure 4-3: Accuracy for 2D and 3D environments

Figure 4.3 compares the mean accuracy for the 2D and 3D environments (indicated by red squares). The yellow squares mark one standard deviation away from the mean. There is very little difference in the means between the two environments.

Environment	Mean accuracy	Standard deviation	Significant difference?	No. of repetitions
2D	0.7622	1.0231	A	94
3D	0.8045	1.1279	A	96

Table 4.17: Mean accuracy for 2D and 3D environments.

Indeed, the means in Table 4.17 show no statistically significant difference in accuracy between the two environments (F = 0.073, *p* = 0.79). This bodes well for the viability of a 3D interface because it shows that users are able to be equally precise using either 2D or 3D for editing.

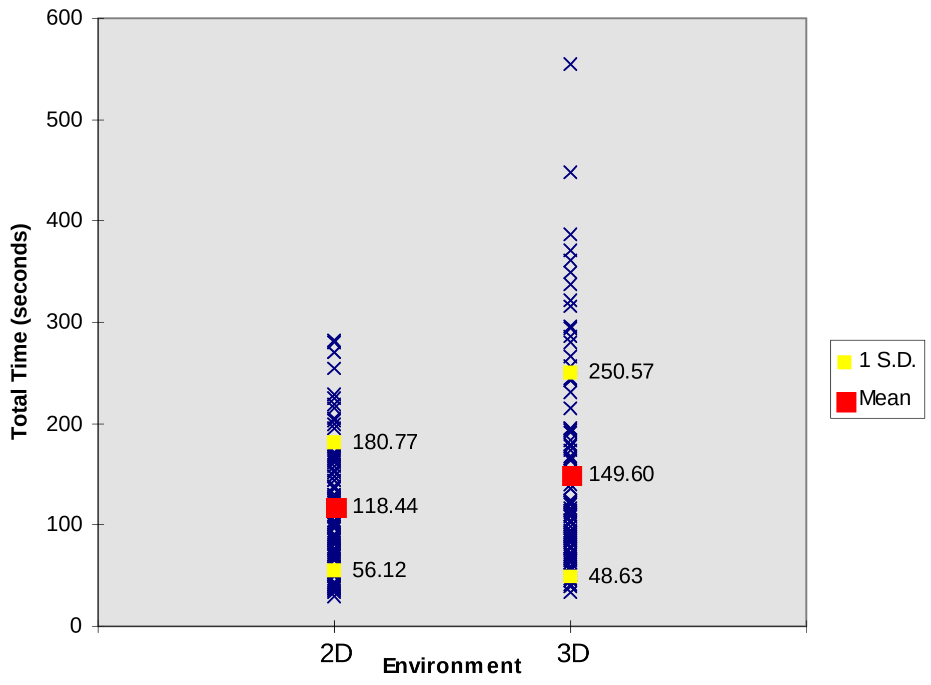


Figure 4-4: Time to edit for 2D and 3D environments.

A glance at Figure 4.4 shows that the performance of the 3D interface suffered with respect to its speed component. The mean speed for 3D is noticeably slower than for 2D, with a wider spread of the data.

Environment	Mean speed (seconds)	Standard deviation	Significant difference?	No. of repetitions
2D	118.44	62.33	A	94
3D	149.60	100.97	B	96

Table 4.18: Mean speed for 2D and 3D environments.

This difference in speed was statistically significant (F = 6.52, *p* = 0.012), as illustrated in Table 4.18.

To better understand why participants might be so much slower at editing in 3D than in 2D, I looked at a breakdown of how they spent their time in each environment.

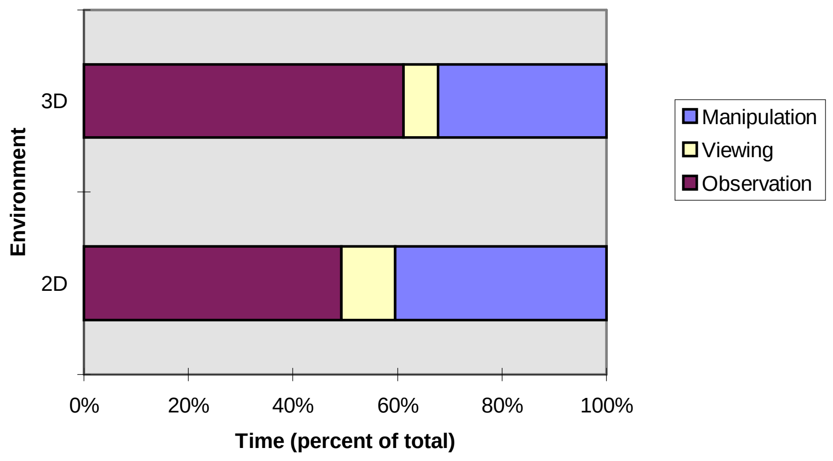


Figure 4-5: Breakdown of total time to edit.

In Figure 4.5, all segments represent a percentage of the total time editing in each environment. The burgundy band indicates time spent observing. This measured time spent either simply looking at the screen, or performing actions such as picking points in preparation for editing. The yellow band shows time spend in viewing the surface, which included time spent rotating, translating or zooming the view. Finally, the blue band shows the time spent manipulating, or actually moving, the points.

The overall pattern of interaction was the same in each environment. It is interesting to note, however, that a larger percentage of time was spent on observation time in 3D. This is presumably because there is more information to process in the 3D environment, and it takes longer to orient oneself in 3D.

In absolute terms, more time was spent in 3D than in 2D for each of the three editing actions. In particular, 20 percent of the extra time was spent on viewing, 40 percent of it on manipulation and 40 percent of it on observation.

In hoping that 3D would be better than 2D for tasks of a 3D nature, I had hypothesised that the ability to manipulate all the points of the 3D deformation as a single group would give 3D an advantage over 2D. After discovering that 3D was in fact slower than 2D for the editing, I looked at the data more closely to see whether the editing strategy chosen by the user had any impact on performance. In particular, I compared editing speed of participants who used a "multi-range" strategy (i.e., selected and moved areas of points spanning multiple contours as a single group) with editing speed of participants who used a "single-range" strategy (i.e., selecting and moving ranges of points one contour at a time). I found that in fact, multi-range editing was a significantly *slower* strategy than the single-range technique.

The more 3D in nature the task became, the slower were the participants. Not only were they slower when using a full-blown 3D editing technique, but when I analysed the spine and hill bump-types separately, I found that the significant difference in editing speeds between 2D and

3D actually came from the hill data, with the spine data showing the same trend without a statistically significant difference.

The last factor I investigated was the effect of previous experience with 3D graphics on performance. By chance, participants were evenly distributed into three categories of experience according to a self-reported level of experience obtained from the questionnaire administered at the end of the experiment. Five participants were deemed novice users, having never used a 3D graphics package before. Six participants were intermediate users with limited experience with 3D graphics, such as taking a fourth-year university computer graphics course. The other five participants were experts conducting full-time research related to 3D graphics.

While both novice and expert user groups performed with similar speed and accuracy in 2D and 3D, intermediate users were less accurate and slower in 3D than in 2D.

Experience	2D mean speed (seconds)	2D standard deviation	3D mean speed (seconds)	3D standard deviation
Novice	128.90	70.86	149.28	93.24
Intermediate	115.02	52.87	166.67	102.40
Expert	112.79	65.34	129.42	106.14

Table 4.19: The effect of experience with 3D graphics on performance.

The mean speeds reported in Table 4.19 reveal that all users were faster in 2D than in 3D. In fact, however, this difference is only significant for the intermediate level participants (F = 7.23, *p* = 0.009).

Experience	2D mean accuracy	2D standard deviation	3D mean accuracy	3D standard deviation
Novice	1.00	1.30	0.92	1.37
Intermediate	0.58	0.85	0.78	1.04
Expert	0.97	0.90	0.72	0.97

Table 4.20: Accuracy in 3D versus experience.

It should also be noted that there was no significant difference in accuracy between the groups (F = 0.26, *p* = 0.77). In fact, the intermediate users were slightly less accurate than the expert users, although slightly better than the novices (Table 4.20). This means that the reduced speed for the intermediate users cannot be accounted for by an increase in precision. The only theory I can offer for this result is that the novices were faster in 3D than the intermediate users because they were less precise, and that the intermediate users were slower than the experts as a result of their lack of experience. This suggests that previous experience with 3D graphics does improve performance with SLICE's 3D editing interface. It does not show, however, that prior experience with 3D graphics narrows the gap between performance in 2D and 3D with SLICE.

In addition to the participants in the final experiment, two expert users, including myself, used the SLICE interface extensively. As is suggested by the results with general experience in 3D graphics, prior experience with SLICE improved performance with that interface. This was true for both 2D and 3D, however, and expert users were still much faster at 2D than at 3D, with similar accuracy in both environments.

4.7.3 Participant Feedback

In addition to the quantitative results I obtained from the 2D and 3D trials, I gathered some subjective data through a questionnaire that was filled out by participants upon completion of the trials. The text of questionnaire appears in Appendix E along with a complete analysis of the results from every question. Here, I will report only on the overall results and significant findings.

For each viewing, point selection and point movement task, the questionnaire asked participants whether they found 2D or 3D more enjoyable, easiest to use and most effective for that task. Users could also indicate no preference by marking a category designated "Equal." The questionnaire asked about viewing the bump, viewing the tube as a whole, and selecting and moving single points, ranges of points and areas of points.

The final question asked for the users overall preference between 2D and 3D. In this case, no "Equal" category was provided. In two cases, participants chose to create their own "Equal" category. Figure 4.6 below shows the results.

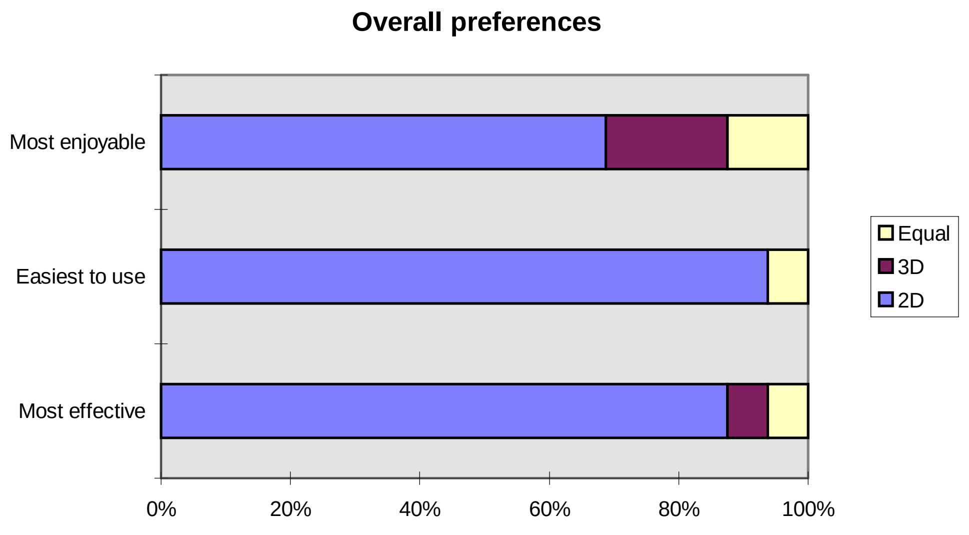


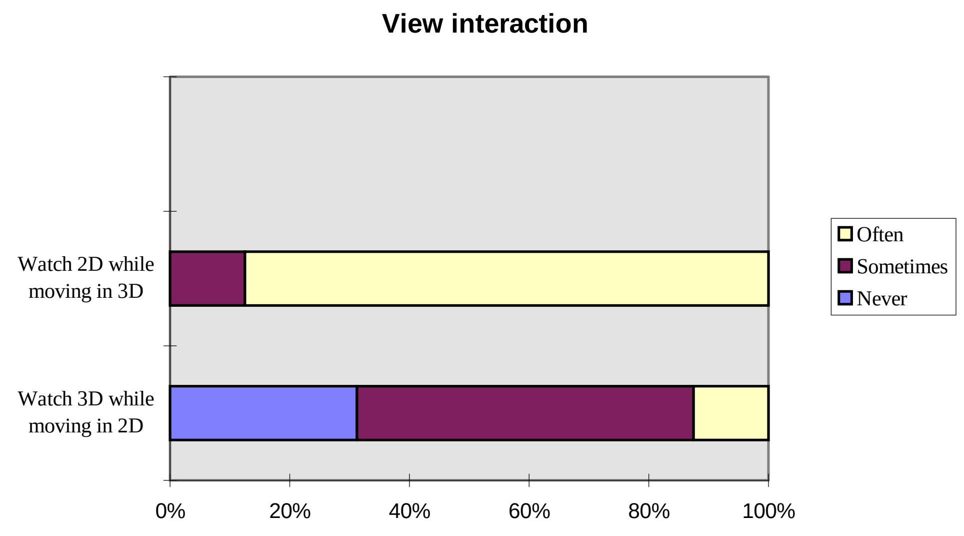
Figure 4-6: Overall environment preferences.

2D was clearly preferred by users as the most enjoyable, easiest to use and most effective interface. Some participants even told me that given a chance, they would have performed the entire experiment using the 2D environment. In the questionnaire results, 3D came out ahead in all categories for viewing the object as a whole. This is not surprising, as I assumed from the start that 3D was better than 2D for the visualisation of entire objects.

3D also won out over 2D for moving ranges of points on multiple adjacent contours. This does not correlate with the quantitative results, but it is interesting that users *perceived* 3D to be a better environment for that task. This leads me to believe that it might have been details of the interface's implementation, rather than the fact of editing in 3D that led to the slower performance with 3D. At the very least, there seems to be some role for 3D in this type of editing process.

Finally, 3D was perceived as more effective than 2D for selecting ranges of points on multiple adjacent contours. In terms of the more enjoyable and easy to use interface, 2D and 3D were fairly evenly matched. For every other interaction task, 2D was chosen as the better interface, hence the overall results.

The other interesting result to emerge from the questionnaire was in response to a question I posed about the interaction between the 2D and 3D views. I added this question as a result of observing the way in which I used the SLICE views myself. Figure 4.7 reports the results of asking whether or not participants watched one view while moving points in the other.



i re sin t e an vie s in concert.

When asked whether or not they watched the 2D view while moving points in 3D, everyone reported that they did at least some of the time, and most people reported doing this frequently. When questioned further, participants said that watching the 2D view helped them to place points more precisely.

People also watched the 3D view while editing in 2D, although this occurred less often. Participants said that the 3D view provided them with some context to see how the change they were making to a particular contour related to the adjacent contours, and the surface as a whole.

5. Conclusions and Future Work

At the outset of this study, I predicted that 2D would be better than 3D for editing deformations of a 2D nature, specifically spikes and ridges. That prediction was borne out by my experimental results. I had also hoped, however, that 3D would be a superior to 2D as an editing environment for correcting deformations of a 3D nature, namely spines and hills. The 2D nature of the data, however, had a stronger effect than the 3D nature of the deformation. In spite of, and perhaps *because of*, the inherent constraints in the surfaces that restricted point motion to a plane, participants were faster at editing in 2D for all types of deformations, while maintaining a consistent accuracy between 2D and 3D.

In determining the applicability of these results to the real world, I must assess how well they generalise to the target setting: the UWT project. There are a two key differences between the data created for the experiments and actual artery datasets. First, the points on the actual datasets are much closer together than the points on the hand-generated data used in the experiment. This should not have a significant impact on the results, however, since the click and drag range selection interface (see Section 3.5.2.2) was designed with the tightly-spaced points of actual artery data in mind. Second, the experimental data contained no branches, which are key elements of the data being studied in the UWT project. Whether or not this impacts the results depends on where the artefacts in the scanned data are located. Unfortunately, I have little information on either the shape or the location of the deformations. As far as the shape is concerned, the experimental data covered all of the basic types of bumps and hollows possible, and the surface itself is similar in shape and curvature to actual artery data.

Since the experimental environment that I used probably represented a reasonable interface for a general contour-editing application running on low-end hardware (i.e., systems using a monitor and mouse as opposed to a stereoscopic display and 3D input device), one might conclude that editing of surfaces reconstructed from contours should be performed in a strictly 2D environment. Participant feedback, however, revealed that each environment had its strengths and weaknesses. 2D was clearly best for precise movement of points. The 3D view did have its place in the interaction process too, however, and was found to be best for tasks that dealt with the surface in a more global way.

Because the 3D view allowed users to see the surface as a whole, it afforded an easy way to locate a deformation and determine its magnitude. In addition, 3D provided a context in which to evaluate the deformation. One contour could be compared to the next to determine how changes to it would affect the global shape of the surface.

During the experiment, participants were restricted to editing in only one of the 2D or 3D environments, and using the other just for viewing. When the expert users of the interface had both views available for editing, however, they found it natural to use the 2D and 3D views in conjunction. An effective editing procedure was to use the 3D environment to locate and select a contour for editing, and the 2D environment to actually move the points. Furthermore, coarse changes could be made in 3D before proceeding to 2D for fine-tuning.

Further studies on using the two views in concert are warranted. In determining what features to include in the final SLICE interface, it would be interesting to investigate in more detail for which tasks each of the two environments (2D and 3D) is best suited. At the very least, however, it seems that an interface for editing surfaces from contours should provide a 3D environment for viewing the entire surface, and selecting individual contours for editing in a 2D environment.

When considering the slower performance results in 3D, one must ask how much impact the particular implementation of the interface had on the outcome. I feel that the result was strong enough that although improvements to the 3D interface might narrow the performance gap between 2D and 3D, 2D will always outperform 3D for editing surfaces reconstructed from contours. It would be instructive to test this with further studies. These could investigate such factors as better rendering techniques, new interface styles, automatic feature detection and using improved hardware.

It might be informative to relax the hardware restrictions imposed by the UWT project to test whether better rendering techniques make 3D editing a more viable option. Texture maps could be added to the flat-shaded surfaces to provide better depth cues. Introducing a better shading model such as Phong or Gouraud shading (see Foley and vanDam for a discussion of these shading methods [Fol90]) might also enhance depth cues, thereby improving the 3D environment. Translucent shading of the surfaces might provide an effective means of displaying the contours and the shape of the surface simultaneously which might also give 3D an edge.

It would be also be interesting to combine the current SLICE style of 3D interface with a traditional constraint-based tool like a handle widget [Hou92, Con92, Str92] that emphasised the inherent planar constraint of the surface data.

Breaking away from the current interface paradigm, future work could incorporate some automatic processing of the data into the interface. Either the program could have some intelligence built in to detect anomalous features, or the intelligence to correct such anomalies. Because of the distinction between artefacts in the data and actual features, some human intervention in the editing process is always likely to be required. For the auto-correction of defects, the user could select the points that form comprise the defect, and have the program fit a curve or curves to remaining points on the contours. It could then fill in a new curve segment for the selected area to match the fitted curves, thus smoothing out the bump or hollow. The user could then accept that as the final surface, or intervene with further manual editing.

Finally, an entirely different tack could be taken in future research. Instead of attempting to make the mapping problem less troublesome in the 3D interface, improved hardware could be used to bypass it altogether. Surface editing experiments could be conducted using stereoscopic views and bats (3D mice).

Pursuing one or more of these avenues definitely merits some attention. I think it is significant that users perceived the 3D environment to be better than 2D for selecting and moving points that spanned multiple contours, even though this was not reflected in the quantitative results. The intuitive preference for 3D in this situation leads me to believe that it is worth continuing the search for a natural and effective interface for editing surfaces in a 3D environment.

Appendix A: Design Space

The following is a description of the design space of possibilities I considered in the creation of the SLICE interface. I present only the most basic tasks, namely selection, rotation, translation, deletion, insertion. Further details depend on the particular implementation chosen. For each task, I give an application-independent list of possible implementations.

Select

In order to add, delete or move an object, a position must first be selected. Points, contours and entire surfaces may be selected, singly or in groups.

Possible implementations include the following:

- Rubber-band a volume around the desired object or objects in a 3D view.
- Rubber-band an area around the desired object or objects in a 2D view.
- Press a mouse button (single or double click) while the locator is over the object to be selected. If this selection is ambiguous, then have the user cycle through the possibilities until the desired choice is reached. For a range of objects, hold down the <Shift> key[1] while making selections after the first. The mouse may be pressed for each new selection, or simply dragged to select neighbours in a direction corresponding to the mouse motion.
- To select a range of objects around an already selected focal object, a slider, dial, or text entry (specifying the number of affected points) could be used to select neighbours in a radial or axial direction from the focus.
- Click on the end objects of a range to select all objects in between.
- Interactively drag two lines to intersect them over the desired object in a 2D or 3D view of the surface; each line has one degree of freedom for movement in a 2D view, or two degrees of freedom in 3D.
- Define a planar boundary in 3D, such as a rectangle or ellipse, and
    1. select all objects that lie within its projection on some surface, or
    2. use it to divide an object into two parts, selecting only those parts that lie on one side of the bounding plane and within its confines.

[1] Note that the <Shift> key is chosen here for multiple selections because of its fairly widespread use in other graphical applications, but any non-repeating key could be used just as easily.

- Using the spacebar, arrow keys, mouse buttons or a button widget, cycle through all objects marking the desired ones by means of a mouse or key press.
- Choose a named object using a text entry, scrolled list, or radio buttons (check boxes for multiple objects).

Rotate

Some viewing tasks, and possibly some editing tasks, require rotation. The user must be able to rotate the entire surface in order to view it from different angles. In addition, if planar boundaries are used, they must be rotated to the correct orientation for tasks such as specifying a cutting plane and defining an area of effect.

Possible implementations include the following :

- Use a slider or dial for rotation about each of the three coordinate axes.
- Use a slider or dial for each of pitch, yaw and roll.
- Use trackball rotation while a mouse button is held down, or while in a “rotate mode”.
- Each of the three mouse buttons is mapped to rotation about one of the coordinate axes. Either or both of mouse x and mouse y motion while the button is held down produces rotation about the corresponding axis.
- Use a text entry to specify an angle of rotation about each axis.
- Use graphical handles on the object with which to rotate it about each axis.

Translate

Both scrolling and zooming of the view require translation of the viewpoint. In the 2D view, the zoom effect is created by scaling the cross-section. All translation of the view is constrained to a plane. Scrolling occurs parallel to the xy plane. Zooming is further constrained to a single axis (z).

Points, contours and planar boundaries can also be translated. The only translation task for which there are three degrees of freedom is that of moving a planar boundary in 3-space. Like the view, translation of points and contours is constrained to a plane. As a result, the same object motion to mouse motion mappings for translation of points and contours can be used in either a 2D or a 3D view.

Possible implementations include the following:

- Use a slider or dial for translation parallel to each coordinate axis.

- Use text entry for amount of translation parallel to each coordinate axis.
- Use text to enter coordinates of a new position in space. (Besides not being intuitive, this method makes it difficult for the user to keep the point in an arbitrary plane in 3-space).
- Map the motion of the object in a single direction to either or both of mouse x and mouse y motion while one of the mouse buttons is held down, or while in a "Move mode".
- Map the motion of the object in two perpendicular directions to mouse x or mouse y motion while a button is held down. One button maps to object motion in one direction and the other button maps to object motion in the orthogonal direction.
- Map the motion of the object in two orthogonal directions to mouse x and mouse y motion respectively, while a button is held down, or while in a mode.
- Indicate the new location for an object by intersecting two lines at the desired point in a 2D or 3D view. Each line has one degree of freedom for movement in a 2D view, or two degrees of freedom in 3D.
- Use the arrow keys to move the object in 2D, or in the plane to which it is constrained in 3D.
- Move objects as a function of the motion of another specific object (for example move a range of selected points as a function of the motion of one of the points).
- Use graphical handles on the object with which to translate it in each direction.

All of the above methods can be used in a 2D or a 3D view

Insert

Points may need to be inserted on a sparsely populated contour in order to facilitate fine adjustments to its shape. Contours must always be planar, so insertion of a point is restricted to the plane of the contour to which it is being added. When inserting in 3D, a grid representing the plane of the contour may be used to provide feedback to the user for positioning. Since contours are evenly spaced in the input data, contour insertion will not be permitted.

Possible implementations include the following:

- Click on the location for the new point with the mouse. This may be done in a 2D view, or in a 3D view where position is constrained to a plane.
- Use text entry to specify the location of the new point. This may be done in 2D or 3D.

- Use the arrow keys or sliders to move a marker around in 2D or 3D constrained to a plane. A button widget, key press or mouse press indicates when the marker has reached the desired location for the new point.

- Indicate the new location for an object by intersecting two lines at the desired point in a 2D or 3D view. Each line has one degree of freedom for movement in a 2D view, or two degrees of freedom in 3D.

Delete

Points, contours and surfaces may be deleted. If deletion is implemented as an operation on a selected item, then all of these cases would be handled in the same way. On the other hand, if it is implemented as a mode, then deletion would work just as select (which could be another mode).

Feedback to the user to indicate *delete* mode could be a cursor change to, for example, the standard skull and crossbones cursor.

An undo feature, which could be provided for all editing operations, is especially desirable for 'delete' since it has potentially destructive consequences. Another option, which could be combined with 'undo', is for the delete operation to request confirmation from the user.

Possible implementations include the following:

- Apply delete as an operation.

- Drag an item to an icon. This could only be used for surfaces since other objects like points and contours are attached.

Apply Operation

Operations that do not require the user to specify any parameters, or for which sufficient information has already been provided, may be applied with a single action. Such operations include deleting selected objects, turning off front-facing polygons, or undoing the previous action.

Possible implementations include the following:

- Press a button widget to perform the operation.

- Press a key to perform the operation.

- Select a menu item to perform the operation.

- Double-click the mouse to perform the operation.

Appendix B: Sample Data Files

Accuracy is based on matching the shape of the base cylinder, rather than matching corresponding points. Thus, accuracy is measured by determining the distance from each point on the participant's final surface to the corresponding circle on the base cylinder. The file abc.accuracy.info contains a numbered label for each point beside its distance from the "true" circle. The list of points for each section begins with numbered section label. Each trial is also demarc7ated with a label.

Each user interaction event was recorded with a timestamp. The file abc.event.info contains this information. This file records the name of the data set for each trial, along with descriptive details about the type of surface it contains, a new trial demarcation, the view type for the trial (i.e., 2D or 3D), and an entry for each event. The start time, end time and duration of each event is recorded, along with a textual description of the event, such as "move point." The event is recorded when it ends. There are two types of events. Some events are single operations, such as picking a point. These are deemed to occur while the participant is observing. The second type of event measures the time spent in different modes, including observing. The single events that occur during observation are indented in the file. By convention, they all have the same start time as the "parent" observation event during which they occur. Their end times mark when they finish, and the duration measures the amount of time since the beginning of the observation event. Other events, such as "rotate" or "move point," are treated equivalently to "observe", and have their start, end and duration times recorded upon completion. At the end of each trial, the total elapsed time, total view time, total manipulation time and total observation time are recorded. The SLICE program keeps track of these with internal counters. Total view time includes all time for rotation of the cylinder, and translation and zooming of the view. Total manipulation time includes all time that the participant spent moving points. Total observation time encompasses the rest of the time when the user was simply observing the data, making decisions about what points to move, selecting or deselecting points, changing the current SLICE with the arrow keys etc. The total elapsed time is simply the sum of the total viewing, manipulation and observation times.

Timing was done within the SLICE program using calls to the C library function **`gettimeofday()`**. Timestamps were recorded for events by getting the current clock time with this function. Timing was done in milliseconds. I did not worry about the time taken to write to record the events in the files since this time is negligible in terms of the elapsed times of roughly two minutes per trial.

The file abc.event.data contains exactly the same information as abc.event.info, but in a form that is less human-readable, and easier to process with a C program. The event descriptions are replaced with numerical codes.

In case it became important to study the movement of individual points, and not just their final position, information was recorded each time points were moved. The file abc.move.info contains the data set used for each trial, and a new trial demarcation. For each trial, the points that were moved are listed one per line. A blank line separates each group of points that was moved. The start of the line is the word *Single* (if the point was moved alone), *Range* (if the point was moved as part of a range; this is secondary motion), *Primary* (if the point was the primary one moved by the user when moving a range, or multiple ranges) or *Multi-range* (if the point was moved as part of a group of ranges; this is secondary motion). The next item on the line indicates the context, i.e., whether the move occurred in 2D or in 3D. The section number and point number follow this. Finally, the distance that the point moved since its last position is recorded. This data was not used in the final analysis.

The following are samples from the data files gathered during the running of a trial:

abc.accuracy.info

```
/p/arteries/code/scalpel/data/hill5b.cnt

New trial:

Accuracy:

{…}

section 5:
point  0: distance: 0.000000
point  1: distance: 0.018504
point  2: distance: -0.017115
point  3: distance: -0.025646
point  4: distance: 1.850630
point  5: distance: 2.948860
point  6: distance: 2.391859
point  7: distance: 1.151355
point  8: distance: 0.000000
point  9: distance: 0.000000
point 10: distance: 0.000000
point 11: distance: 0.000000
point 12: distance: 0.000000
point 13: distance: 0.000000
point 14: distance: 0.000000
point 15: distance: 0.000000
point 16: distance: 0.000000
point 17: distance: 0.000000
point 18: distance: 0.000000
point 19: distance: 0.000000
point 20: distance: 0.000000
point 21: distance: 0.000000
point 22: distance: 0.000000
point 23: distance: 0.000000
point 24: distance: 0.000000

{…}
```

abc.event.info

```
/p/arteries/code/scalpel/data/hill3a.cnt
New trial:
data type 4
sub type 3
view type 3
start time:                 817244373.082452
    enter 3D: 3D            817244373.082452    817244373.448569    366
observe: 3D                 817244373.082452    817244375.779678    2697
rotate: 3D                  817244375.779746    817244381.258229    5478
{…}
range select: 3D            817244404.778251    817244408.793335    4015
    arrow back: 3D          817244408.793405    817244410.012301    1218
    valid mouse pick: 3D    817244408.793405    817244412.288025    3494
observe: 3D                 817244408.793405    817244412.406018    3612
{…}
observe: 3D                 817244414.264295    817244422.315661    8051
adjust def exp: 3D          817244422.315749    817244423.794884    1479
    enter 3D: 3D            817244423.794951    817244424.164793    369
    mouse pick primary: 3D  817244423.794951    817244426.457958    2663
observe: 3D                 817244423.794951    817244426.577315    2782
move multi range: 3D        817244426.577424    817244431.515485    4938
    mouse deselect: 3D      817244431.515554    817244433.506066    1990
    mouse deselect: 3D      817244431.515554    817244433.655771    2140
observe: 3D                 817244431.515554    817244435.565958    4050
rotate: 3D                  817244435.566024    817244438.532604    2966
    mouse pick primary: 3D  817244438.532690    817244439.670193    1137
    mouse deselect: 3D      817244438.532690    817244440.076499    1543
    mouse deselect: 3D      817244438.532690    817244440.226456    1693
    mouse deselect: 3D      817244438.532690    817244440.886517    2353
    valid mouse pick: 3D    817244438.532690    817244442.335187    3802
observe: 3D                 817244438.532690    817244442.449034    3916
move point: 3D              817244442.449225    817244446.927055    4477
    arrow fwd: 3D           817244446.927124    817244448.187133    1260
    valid mouse pick: 3D    817244446.927124    817244450.523069    3595
observe: 3D                 817244446.927124    817244451.377661    4450
move point: 3D              817244451.377729    817244454.497892    3120
    {…}
    valid mouse pick: 3D    817244454.497960    817244464.562188    10064
    invalid mouse pick: 3D  817244454.497960    817244465.474053    10976
    invalid mouse pick: 3D  817244454.497960    817244466.004571    11506
    valid mouse pick: 3D    817244454.497960    817244468.047012    13549
{…}
observe: 3D                 817244558.478922    817244560.749194    2270
move point: 3D              817244560.749259    817244561.769139    1019
    arrow fwd: 3D           817244561.769207    817244563.295328    1526
    arrow fwd: 3D           817244561.769207    817244563.470684    1701
End trial:                  817244561.769207    817244564.737489    2968
Total elapsed time 191.655037
Total view time 20.934568
Total manip time 60.365459
Total observe time 110.349399
```

abc.move.info

```
/p/arteries/code/scalpel/data/hill5b.cnt
New trial:
Single          context: 2 section: 4 point 12: dist moved: 1.028524

{…}

Single          context: 2 section: 5 point 3: dist moved: 0.410723

Single          context: 2 section: 5 point 4: dist moved: 0.240950

Range           context: 2 section: 5 point 2: dist moved: 0.163566
Primary         context: 2 section: 5 point 3: dist moved: 0.286301
Range           context: 2 section: 5 point 4: dist moved: 0.211602
Range           context: 2 section: 5 point 5: dist moved: 0.064857

Primary         context: 2 section: 5 point 2: dist moved: 0.044644
Range           context: 2 section: 5 point 3: dist moved: 0.047716
Range           context: 2 section: 5 point 4: dist moved: 0.018846
Range           context: 2 section: 5 point 5: dist moved: 0.002399

Single          context: 2 section: 5 point 1: dist moved: 0.000000

Range           context: 2 section: 5 point 2: dist moved: 0.187020
Primary         context: 2 section: 5 point 1: dist moved: 0.291178

Single          context: 2 section: 5 point 3: dist moved: 0.000000

Range           context: 2 section: 5 point 4: dist moved: 0.040100
Primary         context: 2 section: 5 point 3: dist moved: 0.062624

Single          context: 2 section: 5 point 5: dist moved: 0.031894

{…}

Multi-range     context: 2 section: 5 point 7: dist moved: 0.214153
Multi-range     context: 2 section: 5 point 6: dist moved: 0.699068
Multi-range     context: 2 section: 5 point 5: dist moved: 0.945488
Multi-range     context: 2 section: 5 point 4: dist moved: 0.541113
Multi-range     context: 2 section: 4 point 16: dist moved: 0.857317
Multi-range     context: 2 section: 4 point 15: dist moved: 2.798349
Primary         context: 2 section: 4 point 14: dist moved: 3.781952
Multi-range     context: 2 section: 4 point 13: dist moved: 2.166448

By exponent     context: 2 section: 5 point 7: dist moved: 0.856610
By exponent     context: 2 section: 5 point 6: dist moved: 2.796271
By exponent     context: 2 section: 5 point 5: dist moved: 3.781953
By exponent     context: 2 section: 5 point 4: dist moved: 2.164452

{…}
```

The following is an example of the file produced from processing the previous data files. It served as input into the statistical analysis package:

abc.out

```
Suj/tri   data/sub  viewt elaps        view         manip        obsv         acc        mr



jrh  1    4    5    2    1642.573880  160.701431   294.469201   1187.383091 31.368387 1

jrh  2    2    7    2    56.325769    1.409902     12.136408    42.778671    0.017623  1

jrh  3    4    3    3    257.768463   28.221481    72.658821    156.881829   0.945007  1

jrh  4    4    1    3    173.349806   7.486496     64.609234    101.250629   0.303381  1

jrh  5    2    1    2    37.518057    1.049873     13.695989    22.771365    0.000000  1

jrh  6    4    3    3    191.655037   20.934568    60.365459    110.349399   1.572414  1

jrh  7    2    4    3    110.466812   10.032031    42.121329    58.310868    0.057716  1

jrh  8    4    6    2    202.440859   11.710618    70.777607    119.947292   2.157498  0

jrh  9    4    5    3    140.405402   20.250786    43.717115    76.434207    1.232670  1

jrh  10   4    2    2    168.916645   17.166891    57.973815    93.770738    0.609180  1

jrh  11   4    4    2    137.649984   9.100651     39.202469    89.342976    1.589894  0

jrh  12   2    5    2    54.839138    11.578115    14.114977    29.144774    0.066103  1

jrh  13   2    6    3    99.449770    13.804576    17.181083    68.461957    0.227878  1

jrh  14   2    3    2    110.059874   15.770696    27.841905    66.443630    0.130213  1

jrh  15   2    2    3    40.018902    8.480365     12.311272    19.225994    0.041205  1
```

Appendix C: Experiment Instructions

The following are the instructions given to participants at the start of the final experiment:

Department of Computer Science
Faculty of Mathematics
University of Waterloo

September 26, 1995

Dear Student:

I am a graduate student in the Department of Computer Science at the University of Waterloo. I am conducting research under the supervision of Professors Rick Kazman and Stephen Mann on the manipulation of computer representations of surface meshes. Three dimensional object manipulation is becoming more and more prevalent with the advent of 'virtual reality'. Thus there is the need to determine where the strengths and pitfalls of 3D interaction lie. As a student of computer science yourself, your views and abilities could be of importance to this study.

I would appreciate if you could participate in an experiment and fill out a brief questionnaire. The experiment will require you to move points on a deformed tube to return it to its original regular shape. The manipulations will be done in both 2D and 3D environments. Instructions will be given (both written and oral) as to the exact methods of your experiment. You will be timed for the duration of the experiment, and your interactions with the computer will be recorded (ie messages will be automatically written to a file as you view and move the tube). The questionnaire will ask you to rate your experiences of the experiment.

You will be paid $10 for participating in the research study.

It is expected that the experiment and questionnaire will take approximately one hour of your time. Although it is desired that you complete the experiment and questionnaire, you can end your participation at any time. Please note that any information that you provide is considered confidential and would be seen only by myself and my supervisors. Furthermore, I am interested in general results rather than specific results of an individual, so you will not be identified by name in any report.

This project has been reviewed and approved for ethics through the Office of Human Research & Animal Care at the University of Waterloo. If you have any questions or concerns resulting from your participation in this study, please contact this office at 885-1211 Ext. 6005.

Thank you in advance for your assistance. If after reading this letter and participating in the experiment and questionnaire, you have any questions about this project, please feel free to contact Professor Kazman at 885-1211 Ext. 4870, or Professor Mann at 885-1211 Ext. 4526.

Yours sincerely,
Julie Waterhouse

Department of Computer Science
Faculty of Mathematics
University of Waterloo

Surface Editing Experiment Instructions

The surface editing experiment will consist of a series of trials. In each trial, two views will be presented to you containing a tube that has a deformation (bump or hollow). Your task is to remove the deformation, and turn the object back into a regular elliptical tube. You will do this by selecting and moving points on the tube.

On each trial, both a 2D view and a 3D view of the tube will be shown. The 2D view is a cross-section through the tube showing a contour. It is possible to cycle through each of the contours in order. This 2D view can *always* be changed through translation and zoom operations (described in the attached Reference section). The 3D view is a perspective projection of the whole tube. This 3D view can *always* be changed through rotation, translation and zoom operations (described in the attached Reference section).

Although the view can always be changed in either 2D or 3D, you will only be able to move points in one of the two views. Which view this is will vary from trial to trial. The view in which you can move the points will have a **black** background. The other view will have a **grey** background. Points can be selected and moved singly or in groups in order to eliminate a deformation. A group of points can be a continuous range of points along a single contour, or an area of points made up of a set of adjacent ranges across multiple contours.

When you have completed a trial, click on the ``Next trial'' button in the top left-hand corner of the interface, or press the F1 key, to move on. The first three trials are just for practice. Use this time to get familiar with the interface, and take as long as you want on each practice trial. The first practice trial allows you to move points in both 2D *and* 3D. You should use this trial to go through the reference guide and learn all of the interface operations. The other two practice trials will be just like real ones: you will only be able to move points in 2D *or* in 3D (but not both). You will be signalled with a popup window when the real experiment begins.

Your interactions with the program will be recorded and timed. If you need a break, you can rest at the *start* of a trial (right after you have pressed ``Next trial'' or the F1 key), because timing does not begin until you move the mouse or press a button during a trial. Both your speed and accuracy will be measured. Accuracy is most important. Try to be as accurate as you can, but don't spend too long making micro-adjustments to the points. It is expected that a trial will take roughly two minutes on average.

If you have any questions after reading these instructions, please feel free to ask me now.

Department of Computer Science
Faculty of Mathematics
University of Waterloo

Surface Editing Interface Reference

The following is a description of how the interface works. All methods described work in both the 2D view and the 3D view unless otherwise specified.

In order to **change the view**, hold down the *control* key. Moving the mouse while holding down the left mouse button then provides trackball rotation. Trackball rotation is explained in a section at the end of this document. Rotation is only available in the 3D view. The middle mouse button is used to zoom the view, and the right mouse button is used to translate the view. This can be done in 2D or 3D.

To **undo viewing changes**, click on the ``Reset view'' button for a window. This will undo any rotation, translation or zoom changes that were made. There are separate buttons for the 2D and the 3D views. There is **no** redo option with this button. The view is reset to its starting position, and any changes that you made to the view are lost.

Select a point by clicking on it with the left mouse button. Dragging the mouse after a selection will **move the point**.

The *shift* key is used when **selecting multiple points**. Simply hold down the *shift* key for selections after the first. To select a set of adjacent points (a **range**), select the first point, and then **hold down the shift key** while dragging the mouse. Dragging the mouse to the right selects in a clockwise direction on the screen; dragging to the left selects in a counter-clockwise direction on the screen.

In order to **add an extra point onto the end of a range**, shift click on it. It must be adjacent to an existing range.

In order to **delete a point from a range**, select it with the middle mouse button. It must be one of the endpoints of the range.

Multiple ranges are selected by repeating the range selection process while the Shift key is depressed. In this way, you can select an area of points.

Three points worth noting about selection:

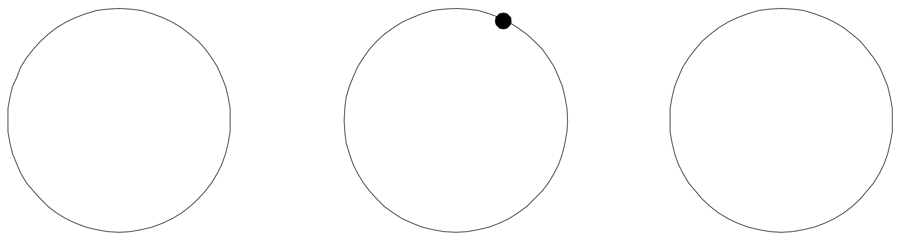
1. Every point you select in a range moves when a primary point is picked. Thus when moving a range, you should only select those points that are “out of place”, and that you wish to move (ie don't select any “anchor” points.)

2. When selecting multiple ranges, additional ranges must be *adjacent* to existing ranges of selected points.

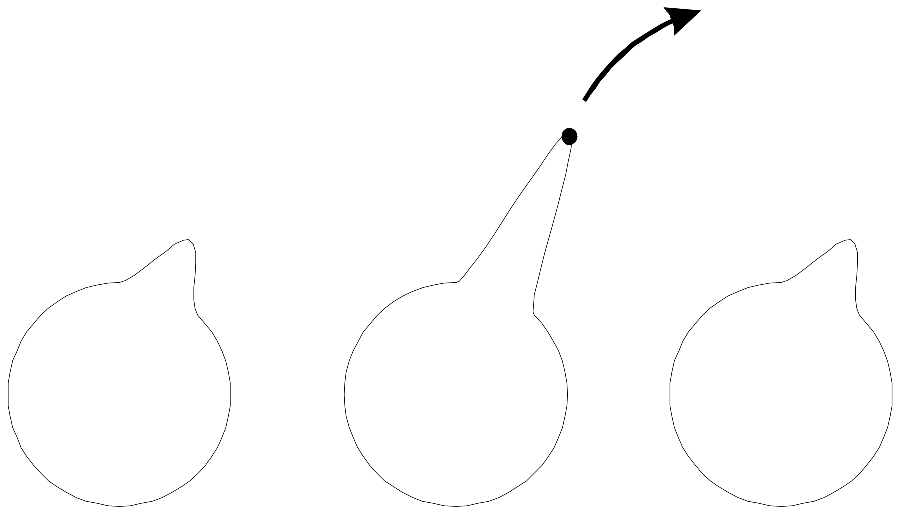
3. When selecting with the shift key depressed, selection is restricted to new selections on contours with already selected points, or the contours adjacent to these.

In order to **move multiple points**, a *primary* point of motion must be defined by clicking on one of the already selected points with the left mouse button. Dragging the mouse after this selection will cause the group of selected points to move. One primary point can be selected for a single range, or for multiple ranges. In the latter case, moving the primary point creates or flattens a mound-like shape.

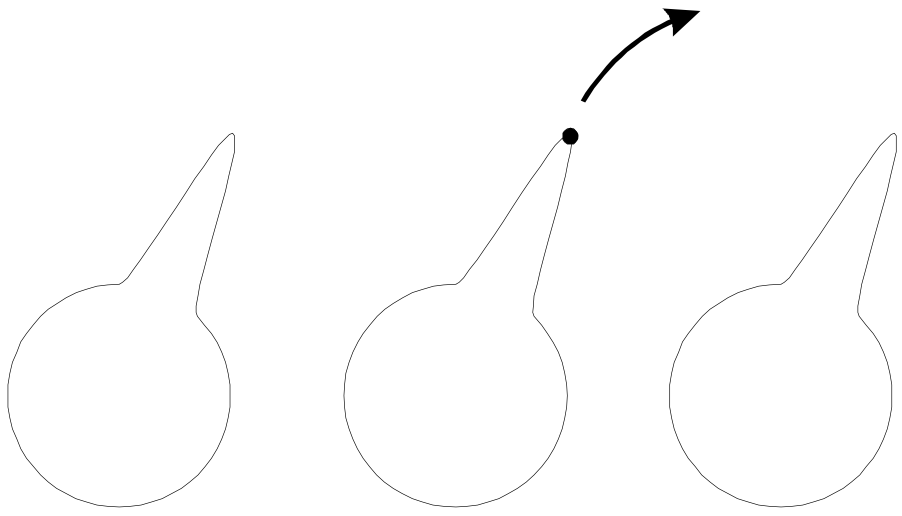
The ``Deform exponent'' slider **changes the shape of deformation**. It can be used when multiple ranges of points are being moved on adjacent contours. When the primary point is dragged, the slider controls how much the points in the other contours move. The slider value represents exponential dropoff. For example, if the slider is set to one, points on adjacent contours move the same amount as the primary point. If the slider is set to 1/5, the points in the contours adjacent to the one containing the primary point move one fifth as much as the primary point, and the contours adjacent to those (one more level out), move 1/5 as much again. The ``Reset exponent'' button is used to reset the slider to its default value of 1/2. Please refer to the diagram on the next page.



Starting contours



Dragging some selected points with slider set to ½.



Dragging some selected points with slider set to 1.

Figure C.1: Using the “deform exponent” slider.

Clicking anywhere in the 2D or 3D window with the right mouse button will **deselect** the last selection. It can be used repeatedly to deselect items in the reverse order of selection. To deselect everything, either click repeatedly with the right mouse button, or click the "Select nothing" button on the interface.

Click on the "Undo move" button on the interface to **undo the last move**. The button will then be labelled "Redo move", and can be used to redo the move that was undone.

"Next" and Previous" are used to change the 2D view to the next contour of the cylinder in each direction. This also changes the "current contour" in the 3D view. The left and right or up and down arrow keys can also be used to **cycle through the contours** (ie change the current contour).

In order to **restrict selection** to a contour, set the "Select on contour" toggle. This means that points can only be selected on the highlighted, "current" contour. This is useful when selecting in 3D where certain views can make selection ambiguous.

In order to **restrict selection** to an already selected point, set the "Reselect only" toggle. This is useful when selecting a primary point, to ensure that only an already selected point is targeted as the primary point. It helps to avoid the problem of missing the desired point and starting an entirely new selection.

Trackball rotation

Author for this section: Michael Hardy, August 12, 1993.

Holding down the Control key and left mouse button and moving the mouse will rotate the scene using trackball rotations. The best way to think of this type of interaction is to pretend that the scene is in a large glass sphere that is sticking out of the window. The mouse can be thought of as a hand that strokes the sphere. Where and in what direction the sphere is stroked will determine how the scene rotates.

Moving the mouse from left to right THROUGH THE CENTER OF THE WINDOW will rotate the scene from left to right.

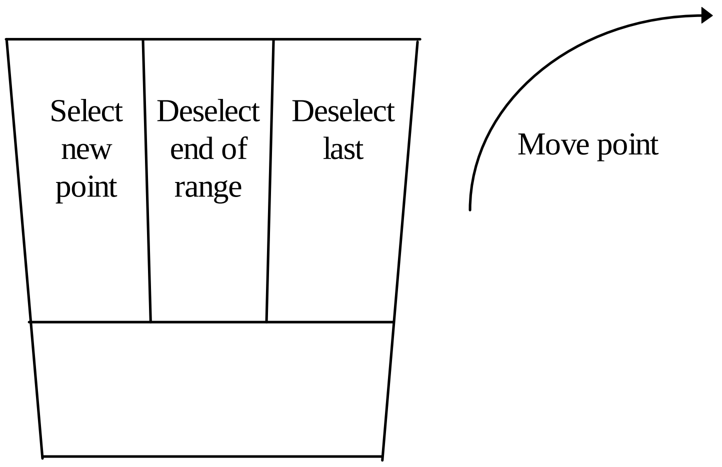
Moving the mouse from top to bottom THROUGH THE CENTER OF THE WINDOW will rotate the scene from top to bottom.

Moving the mouse from the upper right to the lower left THROUGH THE CENTER OF THE WINDOW will rotate the back right corner of the scene up and over to the front left of the scene. This is like stroking the sphere from the upper right corner towards the lower left corner.

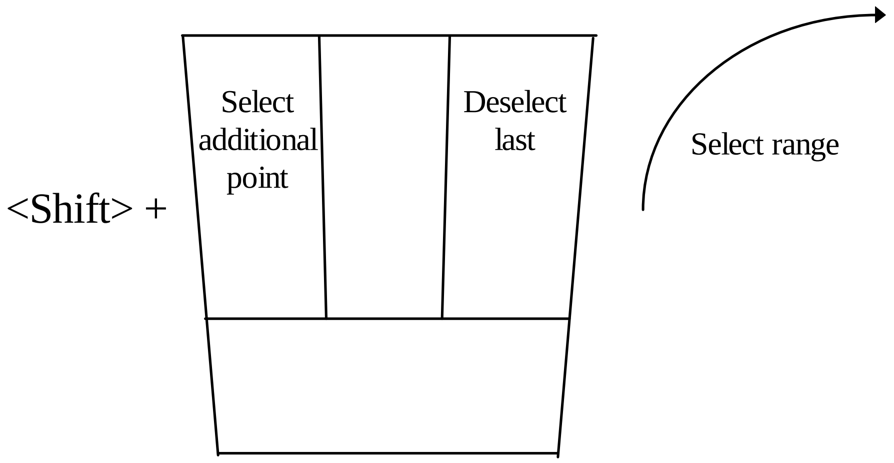
Moving the mouse clockwise around the window WHILE STAYING NEAR THE BORDER OF THE WINDOW will rotate the scene clockwise. This is like turning the sphere clockwise.

Mouse mapping

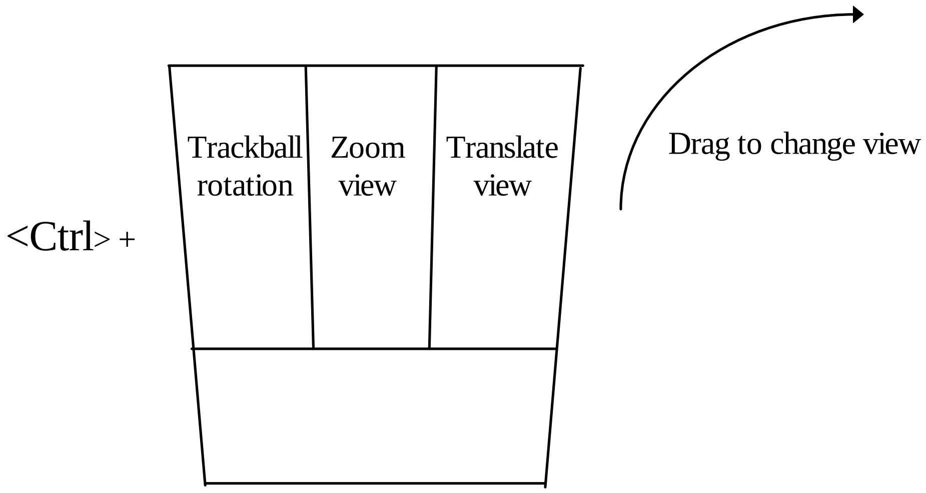
The current mouse mapping for a window is always described in the text box located directly below that window. Also feel free to refer to the following diagrams at any time.



Default mouse mapping.



Mouse mapping when shift key is pressed.



Mouse mapping when control key is pressed.

Appendix D: Experiment Results

Following is the complete statistical analysis of the results of the pretests and final experiment.

Surface type pretest results:

Speed: Surface type and Bump type Interaction

Anova: Two-Factor With Replication

SUMMARY	Wireframe	Shaded	Contours	Total
Spike				
Count	6	6	6	18
Sum	195.5916	380.293	107.7022	683.5867
Average	32.59859	63.38217	17.95036	37.97704
Variance	306.685	2192.58	16.95612	1119.624
S.D.				33.46078
Spine				
Count	6	6	6	18
Sum	387.3114	790.7117	387.9868	1566.01
Average	64.55189	131.7853	64.66446	87.00055
Variance	278.7209	8454.133	901.1557	3895.361
S.D.				62.41283
Ridge				
Count	6	6	6	18
Sum	678.9234	819.8758	239.6623	1738.461
Average	113.1539	136.646	39.94372	96.58119
Variance	10333.29	5173.33	347.5631	6458.634
S.D.				80.36563
Hill				
Count	6	6	6	18
Sum	1212.952	702.6906	1128.589	3044.232
Average	202.1587	117.1151	188.0982	169.124
Variance	46262.96	3047.907	15123.99	20418.33
S.D.				142.8927
Total				
Count	24	24	24	
Sum	2474.779	2693.571	1863.94	
Average	103.1158	112.2321	77.66418	
Variance	16701.26	4985.691	8089.9	
S.D.	129.2334	70.60942	89.94387	

ANOVA

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Bump type	158111	3	52703.68	6.841726	0.000485	2.758078
Surface type	15406.68	2	7703.338	1.000009	0.373924	3.150411
Interaction	64560.2	6	10760.03	1.396813	0.230814	2.254055
Within	462196.3	60	7703.272			

Accuracy: Surface type and Bump type Interaction

Anova: Two-Factor With Replication

SUMMARY	Wireframe	Shaded	Contours	Total
Spike				
Count	6	6	6	18
Sum	0.173555	0.730429	0.163274	1.067258
Average	0.028926	0.121738	0.027212	0.059292
Variance	0.00026	0.01036	0.001065	0.005502
S.D.				0.074174
Spine				
Count	6	6	6	18
Sum	1.515016	2.357709	0.962454	4.835179
Average	0.252503	0.392952	0.160409	0.268621
Variance	0.125187	0.046582	0.011976	0.063723
S.D.				0.252434
Ridge				
Count	6	6	6	18
Sum	6.208968	9.821954	1.901004	17.93193
Average	1.034828	1.636992	0.316834	0.996218
Variance	1.052984	1.250472	0.037066	0.996734
S.D.				0.998366
Hill				
Count	6	6	6	18
Sum	11.5855	12.76596	7.54748	31.89893
Average	1.930916	2.12766	1.257913	1.772163
Variance	0.150389	0.845843	0.071328	0.460823
S.D.				0.67884
Total				
Count	24	24	24	
Sum	19.48303	25.67605	10.57421	
Average	0.811793	1.069835	0.440592	
Variance	0.870057	1.197939	0.269716	
S.D.	0.932768	1.094504	0.519342	

ANOVA

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Bump type	32.61466	3	10.87155	36.20319	1.75E-13	2.758078
Surface type	4.802585	2	2.401292	7.996507	0.000834	3.150411
Interaction	3.135155	6	0.522526	1.740056	0.127271	2.254055
Within	18.01756	60	0.300293			
Total	58.56996	71				

Additional view pretest results:

2D Speed
Anova: Single Factor

SUMMARY

Groups	Count	Sum	Average	Variance	S.D.
Both	46	2724.591	59.23023	3623.044	60.19173
2D only	46	2568.862	55.84483	2062.7	45.41696

ANOVA

Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	263.6008	1	263.6008	0.092723	0.761446	3.946866
Within Groups	255858.5	90	2842.872			
Total	256122.1	91				

2D Accuracy
Anova: Single Factor

SUMMARY

Groups	Count	Sum	Average	Variance	S.D.
Both	46	14.02001	0.304783	0.163017	0.403754
2D only	46	15.83046	0.34414	0.168612	0.410624

ANOVA

Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	0.035627	1	0.035627	0.214863	0.644101	3.946866
Within Groups	14.92332	90	0.165815			
Total	14.95895	91				

3D Speed
Anova: Single Factor

SUMMARY

Groups	Count	Sum	Average	Variance	S.D.
Both	49	3608.413	73.64107	3321.938	57.63625
3D only	49	3442.106	70.24706	3860.513	62.13303

ANOVA

Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	282.2238	1	282.2238	0.078587	0.779825	3.940158
Within Groups	344757.6	96	3591.226			
Total	345039.9	97				

3D Accuracy
Anova: Single Factor

SUMMARY

Groups	*Count*	*Sum*	*Average*	*Variance*	*S.D.*
Both	49	15.88239	0.32413	0.163379	0.404202
3D only	49	23.20663	0.473605	0.537875	0.7334

ANOVA

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	0.547393	1	0.547393	1.561181	0.21453	3.940158
Within Groups	33.66021	96	0.350627			
Total	34.2076	97				

<u>Bump type pretest results:</u>

Spike: Speed
Anova: Single Factor

SUMMARY

Groups	*Count*	*Sum*	*Average*	*Variance*	*S.D.*
2D	11	179.179	16.289	57.23406	7.565319
3D	12	315.4412	26.28677	113.6699	10.66161

ANOVA

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	573.657	1	573.657	6.60928	0.017816	4.324789
Within Groups	1822.71	21	86.7957			
Total	2396.367	22				

Spike: Accuracy
Anova: Single Factor

SUMMARY

Groups	*Count*	*Sum*	*Average*	*Variance*	*S.D.*
2D	11	0.280902	0.025537	0.000427	0.020653
3D	12	0.254884	0.02124	0.000497	0.022302

ANOVA

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	0.000106	1	0.000106	0.228462	0.637605	4.324789
Within Groups	0.009737	21	0.000464			
Total	0.009843	22				

Ridge: Speed
Anova: Single Factor

SUMMARY

Groups	Count	Sum	Average	Variance	S.D.
2D	11	335.1637	30.46943	115.581	10.75086
3D	12	693.3819	57.78183	749.9729	27.38563

ANOVA

Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	4281.203	1	4281.203	9.558785	0.005527	4.324789
Within Groups	9405.511	21	447.8815			
Total	13686.71	22				

Ridge: Accuracy
Anova: Single Factor

SUMMARY

Groups	Count	Sum	Average	Variance	S.D.
2D	11	2.68698	0.244271	0.009517	0.097553
3D	12	3.713743	0.309479	0.042951	0.207246

ANOVA

Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	0.024403	1	0.024403	0.902814	0.352836	4.324789
Within Groups	0.567629	21	0.02703			
Total	0.592032	22				

Spine: Speed
Anova: Single Factor

SUMMARY

Groups	Count	Sum	Average	Variance	S.D.
2D	12	629.9031	52.49193	2015.678	44.8963
3D	12	708.0858	59.00715	828.1498	28.77759

ANOVA

Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	254.6887	1	254.6887	0.179117	0.676242	4.300944
Within Groups	31282.1	22	1421.914			
Total	31536.79	23				

Spine:Accuracy
Anova: Single Factor

SUMMARY

Groups	*Count*	*Sum*	*Average*	*Variance*	*S.D.*
2D	12	0.891309	0.074276	0.007682	0.087649
3D	12	1.202506	0.100209	0.019821	0.140787

ANOVA

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	0.004035	1	0.004035	0.29343	0.593475	4.300944
Within Groups	0.302536	22	0.013752			
Total	0.306572	23				

Hill: Speed
Anova: Single Factor

SUMMARY

Groups	*Count*	*Sum*	*Average*	*Variance*	*s.D.*
2D	12	1580.345	131.6954	4199.512	64.80364
3D	13	1891.504	145.5003	3434.786	58.60704

ANOVA

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	1189.186	1	1189.186	0.312901	0.581313	4.279343
Within Groups	87412.06	23	3800.524			
Total	88601.25	24				

Hill: Accuracy
Anova: Single Factor

SUMMARY

Groups	*Count*	*Sum*	*Average*	*Variance*	*S.D.*
2D	12	10.16082	0.846735	0.190151	0.436063
3D	13	10.71126	0.823943	0.182792	0.427542

ANOVA

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	0.003241	1	0.003241	0.017398	0.896209	4.279343
Within Groups	4.285166	23	0.186312			
Total	4.288407	24				

Final experiment results:

Speed

Anova: Single Factor

SUMMARY

Groups	Count	Sum	Average	Variance	STD
2D	94	11133.53	118.4418	3884.497	62.32574
3D	96	14361.26	149.5965	10195.05	100.9706

ANOVA

Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	46098.98	1	46098.98	6.517284	0.011477	3.891401
Within Groups	1329788	188	7073.343			
Total	1375887	189				

Accuracy

Anova: Single Factor

SUMMARY

Groups	Count	Sum	Average	Variance	STD
2D	94	71.64813	0.762214	1.046797	1.023131
3D	96	77.22798	0.804458	1.272085	1.127867

ANOVA

Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	0.084757	1	0.084757	0.073026	0.787277	3.891401
Within Groups	218.2002	188	1.160639			
Total	218.285	189				

Appendix E: Questionnaire and Results

The following is the questionnaire that was administered to participants upon completion of the final experiment.

Department of Computer Science
Faculty of Mathematics
University of Waterloo

For the following tasks, please indicate which environment was most effective for completing that task, which was easiest to use, and which was most enjoyable to use. Circle the appropriate response.

viewing the tube as a whole:

most effective	2D	3D	Equal
most easy to use	2D	3D	Equal
most enjoyable to use	2D	3D	Equal

viewing a bump or hollow::

most effective	2D	3D	Equal
most easy to use	2D	3D	Equal
most enjoyable to use	2D	3D	Equal

selecting a single point:

most effective	2D	3D	Equal
most easy to use	2D	3D	Equal
most enjoyable to use	2D	3D	Equal

selecting a range of points on a single contour:

most effective	2D	3D	Equal
most easy to use	2D	3D	Equal
most enjoyable to use	2D	3D	Equal

selecting ranges of points on multiple adjacent contours:

most effective	2D	3D	Equal
most easy to use	2D	3D	Equal
most enjoyable to use	2D	3D	Equal

moving a single point:

most effective	2D	3D	Equal
most easy to use	2D	3D	Equal
most enjoyable to use	2D	3D	Equal

moving a range of points on a single contour:

most effective	2D	3D	Equal
most easy to use	2D	3D	Equal
most enjoyable to use	2D	3D	Equal

moving ranges of points on multiple adjacent contours:

most effective	2D	3D	Equal
most easy to use	2D	3D	Equal
most enjoyable to use	2D	3D	Equal

Please answer the following questions by circling the appropriate response:

Overall, which environment did you find *most effective* in performing the tasks?

2D	3D

Overall, which environment did you find *easiest to use* when performing the tasks?

2D	3D

Overall, which environment did you find *most enjoyable* when performing the tasks?

2D	3D

Did you watch the 3D view while moving points in the 2D view?

A) Never

B) Sometimes

C) Often

Did you watch the 2D view while moving points in the 3D view?

A) Never

B) Sometimes

C) Often

Have you ever used any graphical modelling packages (such as a CAD package) before?

A) Never

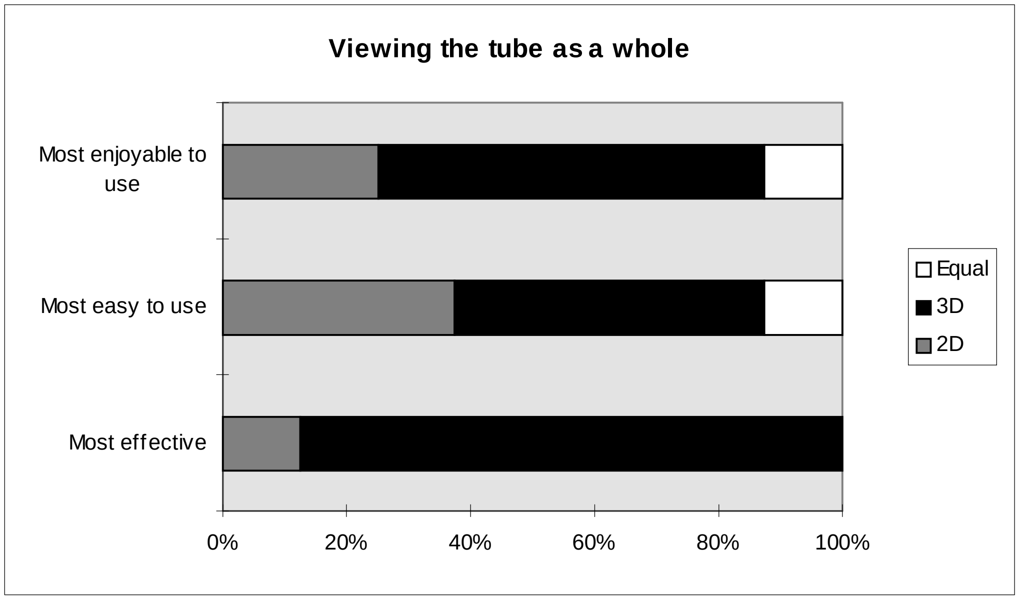
B) Sometimes (please specify): ____

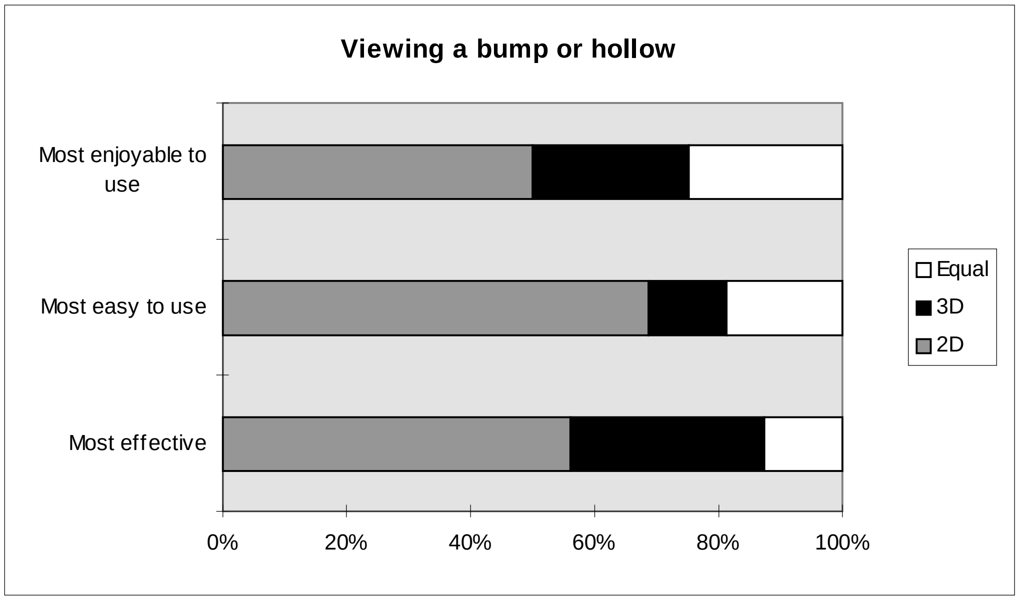
C) Often (please specify): ____

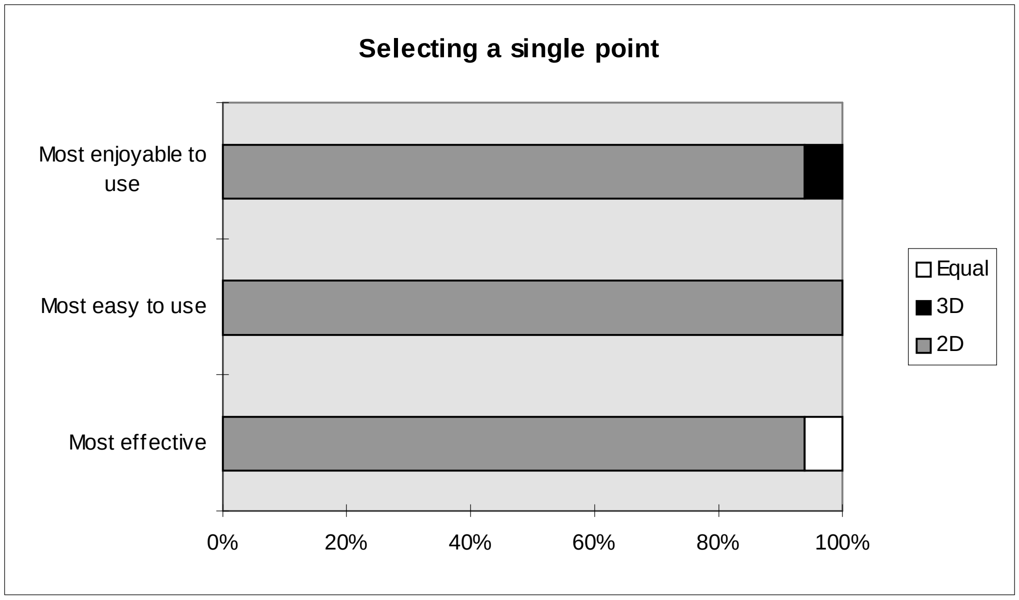
Please comment on any problems you had using the interface:

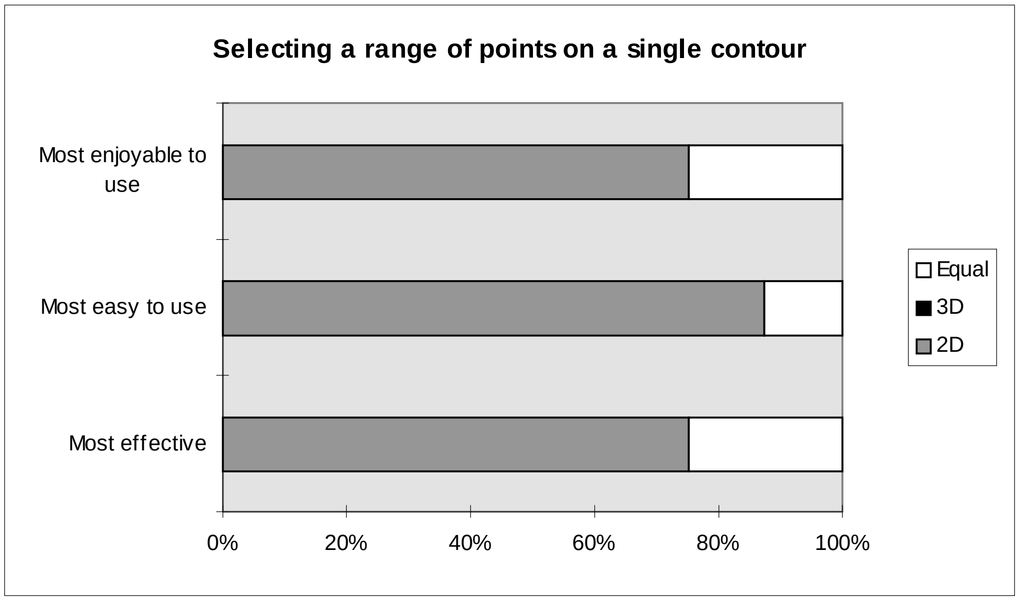
Please provide any additional comments you feel might be useful:

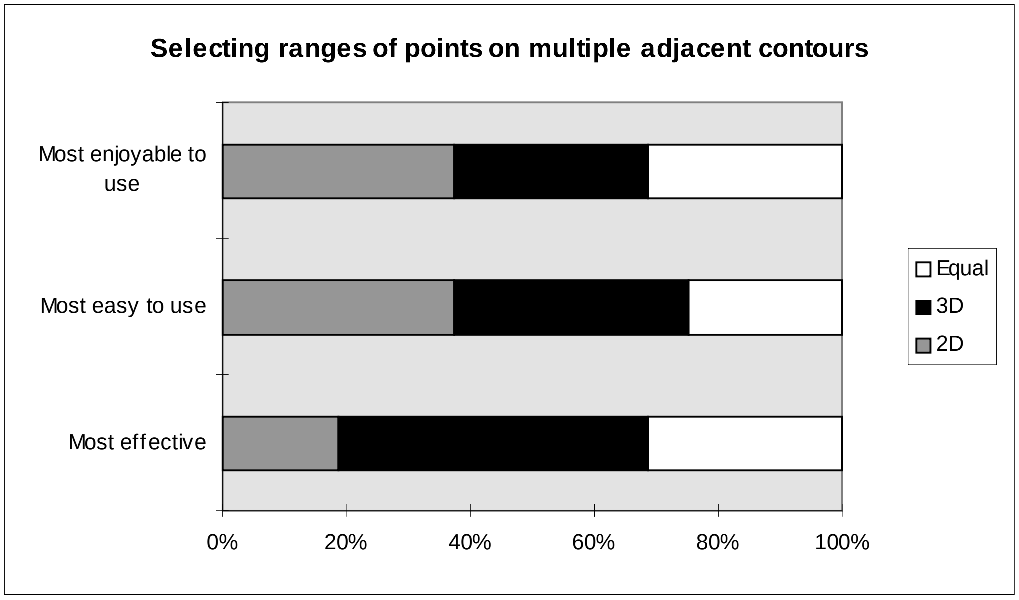
The following charts illustrate the results from the questionnaire.

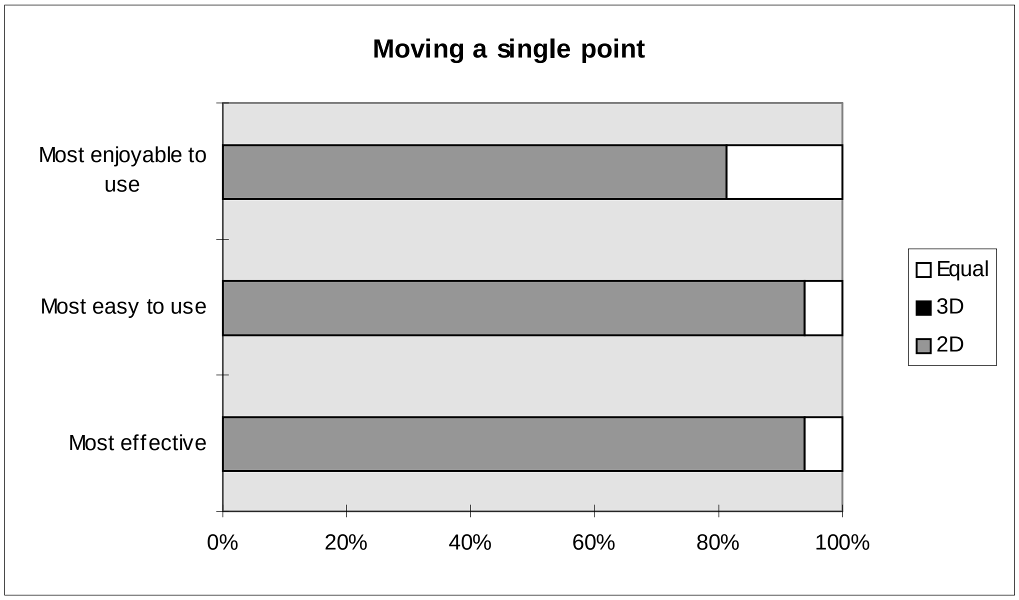


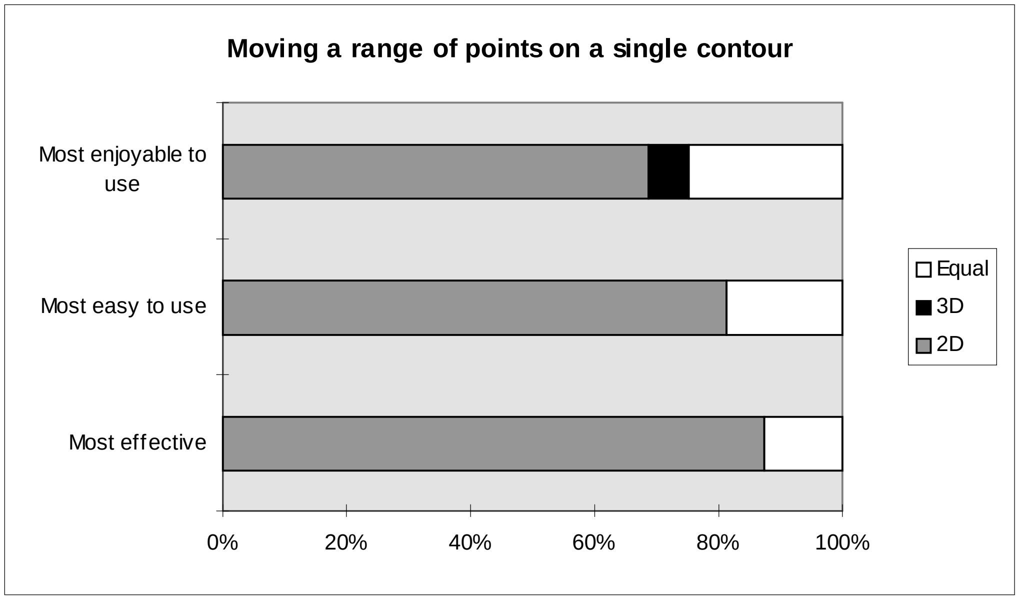


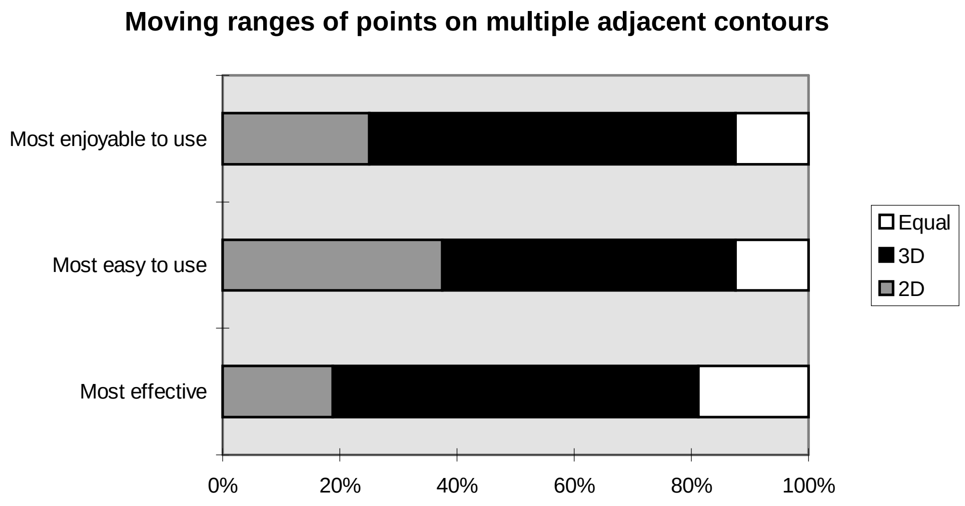

Selecting a single point
Most enjoyable to use
Most easy to use
Most effective
0%
20%
40%
60%
80%
100%
Equal
3D
2D


Selecting a range of points on a single contour
Most enjoyable to use
Most easy to use
Most effective
0%
20%
40%
60%
80%
100%
Equal
3D
2D


Selecting ranges of points on multiple adjacent contours
Most enjoyable to use
Most easy to use
Most effective
0%
20%
40%
60%
80%
100%
□ Equal
■ 3D
▨ 2D


Moving a single point
Most enjoyable to use
Most easy to use
Most effective
0%
20%
40%
60%
80%
100%
□ Equal
■ 3D
▨ 2D


Moving a range of points on a single contour
Most enjoyable to use
Most easy to use
Most effective
0%
20%
40%
60%
80%
100%
Equal
3D
2D


Moving ranges of points on multiple adjacent contours
Most enjoyable to use
Most easy to use
Most effective
0%
20%
40%
60%
80%
100%
Equal
3D
2D

Bibliography

[Arn65] Rudolf Arnheim. *Art and visual perception; a psychology of the creative eye*. 2nd edition, University of California Press, Berkeley, 1965.

[Bar89] Richard Bartels and John C. Beatty. A Technique for the Direct Manipulation of Spline Curves. In *Proceedings Graphics Interface '89*, pp. 33-39, London, Canada, June 1989.

[Bor86] Alan Borning and Robert Duisberg. Constraint-Based Tools for Building User Interfaces. In *ACM Transactions on Graphics,* Vol. 5, No. 4, pp. 345-374, October 1986.

[Bos87] Eric Bosch. *Workstation-Based Shape Matching Experiments.* Master's thesis, University of Waterloo, 1987.

[Car78] C.G. Caro, T.J. Pedley, R.C. Schroter and W.A. Seed. *The Mechanics of the Circulation.* Oxford University Press, Oxford, 1978.

[Che88] Michael Chen, S. Joy Mountford and Abigail Sellen. A Study in Interactive 3-D Rotation Using 2-D Control Devices. In *SIGGRAPH '88 Conference Proceedings,* pp. 121-129, 1988.

[Coa93] Sean Coady, Andrew Park and Julie Waterhouse. *ArchiTech: Final Report*. April 1993, unpublished.

[Con92] D. Brookshire Conner, Scott S. Snibbe, Kenneth P. Herndon, Daniel C. Robbins, Robert C. Zeleznik, and Andries van Dam. Three-Dimensional Widgets. In Levoy and Catmull [Lev92], pp. 183-188.

[Coo84] Lynn A. Cooper and Roger N. Shepard. Turning Something Over in the Mind. In *Scientific American*, Vol. 251, No. 6, pp. 106-114, 1984.

[Cuv86] C. Cuvelier, A. Segal, A.A. van Steenhoven. *Finite Element Methods and Navier-Stokes Equations*. D. Reidel Publishing Company, Boston, 1986.

[Dob94] Philip B. Dobrin. *Intimal Hyperplasia*. R.G. Landes Co., Austin, 1994.

[Eth92] Ross Ethier. "Hemodynamic Effects in Bypass Graft Failure: Phase I", grant proposal submitted to Heart and Stroke Foundation of Canada in 1992.

[Fol90] James Foley, Andries vanDam, Steven Feiner, and John Hughes. *Computer Graphics: Principles and Practice*, 2nd edition. Addison-Wesley, Reading, Massachusetts, 1990.

[Fuc77] H. Fuchs, Z.M. Kedem, and S.P. Uselton. Optimal Surface Reconstruction from Planar Contours. In *Communications of the ACM*, Vol. 20, No. 10, pp. 693-702, October 1977.

[Gei93] B. Geiger. Three-dimensional modeling of human organs and its application to diagnosis and surgical planning. Technical Report 2105, Institut National de Recherche en Informatique et Automatique. France, Dec 1993.

[Gru89] Jonathan Grudin. The case against user-interface consistency. In *Communications of the ACM*. Vol. 32:10, pp. 1164-1173, October, 1989.

[Hec92] *Graphics Gems IV*. Paul Heckbert, editor. Academic Press, 1994.

[Her92] Kenneth P. Herndon, Robert C. Zeleznik, Daniel C. Robbins, D. Brookshire Conner, Scott S. Snibbe, and Andries van Dam. Interactive Shadows. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pp. 1-6, 1992.

[Hou92] Stephanie Houde. Iterative design of an interface for easy 3D direct manipulation. In *Proceedings of ACM CHI'92 Conference on Human Factors in Computing Systems*, pp. 135-142, 1992.

[How92] David C. Howell. *Statistical Methods for Psychology*, 3rd edition. Duxbury Press, Belmont, California, 1992.

[Hsu92] William M. Hsu. Direct Manipulation of Free-Form Deformations. In *ACM SIGGRAPH Video Review*, Issue 86, no. 8, 1992.

[Hut85] E.L. Hutchins, J. D. Hollan and D. A. Norman. Direct Manupulation Interfaces. In *Human-Computer Interaction*, Vol. 1, No. 4, pp. 311-338.

[Jac89] Robert J.K. Jacob. Direct Manipulation in the Intelligent Interface. In P.A. Hancock and M.H. Chignell, editors, *Intelligent Interfaces: Theory, research and design*, North-Holland, Netherlands: Elsevier Science Publishers B.V., pp. 165-212.

[Jau95] Fabrice Jaubert. A study of Delays and De-Synchronisation in a Multiple-View Direct Manipulation Task. Master’s thesis, University of Waterloo, 1995.

[Jon94] Mark W. Jones and Min Chen. A New Approach to the Construction of Surfaces from Contour Data. In *Eurographics ’94*, Eurographics Association, 1994.

[Kel87] Wendy A. Kellog. Conceptual Consistency in the User Interface: Effects on User Performance. In *Human-Computer Interaction - INTERACT ’87*. North-Holland, 1987.

[Lam85] Leslie Lamport. *LaTeX: A Document Preparation System.* Addison-Wesley, Reading, Massachusetts, 1985.

[Lev92] Marc Levoy and Edwin E. Catmull, editors. *Proceedings of the 1992 Symposium on Interactive Three-dimensional Graphics.* ACM SIGGRAPH. March, 1992.

[Mey92] Meyers, David and Shelley Skinner. Surfaces from Contours. In *ACM Transactions on Graphics*, Vol. 11, No. 3, July 1992, pp. 228-258.

[Moo95] Personal communication with Jennifer Moore, Ph.D. graduate working on the UWT project at the University of Toronto. Email, June 6, 1995.

[NAS93] *ROSS: Reconstruction of Serial Sections, User Guide*, Version 2.3. National Aeronautics and Space Administration/Ames Research Center, 1993.

[Phi88] Cary B. Phillips and Norman I. Badler. Jack: a toolkit for manipulating articulated figures. In *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software*, pp. 221-229, 1988.

[Phi92] Cary B. Phillips, Norman I. Badler and John Granieri. Automatic viewing control for 3D direct manipulation. In Levoy and Catmull [Lecv92], pp. 31-38.

[Rue89] Paul Ruest. *An Evaluation of Tension Within an Extensible Spline Testing Facility*. Master's thesis, University of Waterloo, 1989.

[Sch85] Michael Schwarz. *An Empirical Evaluation of Interactive Colour Selection Techniques*. Master's thesis, University of Waterloo, 1985.

[SDR93] *I-DEAS Shape Design Modeling Guide*, for I-DEAS Master Series 1.0, Document number P-10055, published by SDRC, 1993.

[Sni92a] Scott S. Snibbe, Kenneth P. Herndon, Daniel C. Robbins, D. Brookshire Conner, and Andries van Dam. Using Deformations to Explore 3D Widget Design. In *SIGGRAPH '92 Conference Proceedings.* ACM SIGGRAPH, Addison-Wesley, pp. 351-351, July 1992.

[Sni92b] Scott S. Snibbe, Kenneth P. Herndon, Daniel C. Robbins, D. Brookshire Conner, and Andries van Dam. Using Deformations to Explore 3D Widget Design. In *ACM SIGGRAPH Video Review*, Issue 86, no. 15,1992.

[Str92] Paul S. Strauss and Rikk Carey. An Object-Oriented 3D graphics Toolkit. In *SIGGRAPH '92 Conference Proceedings*, pp. 341-349, 1992.

[War90] Colin Ware and Steven Osborne. Exploration and Virtual Camera Control in Virtual Three Dimensional Environments. In *Interactive 3D Graphics: Proceedings of the 1990 Symposium, Computer Graphics* vol. 24, no. 2, pp. 175-183, 1990.

[Wic92] Christopher D. Wickens. *Engineering Psychology*, 2nd edition, Harper-Collins, New York, 1992.

[Wic89] Christopher D. Wickens, Ian Haskell and Karen Harte. Ergonomic Design for Perspective Flight-Path Displays. In *IEEE Control Systems Magazine*, Vol. 9, No. 4, pp. 3-8, June 1989.

[Woo84] David D. Woods. Visual Momentum: A Concept to Improve the Cognitive Coupling of Person and Computer. In *International Journal of Man-Machine Studies*, Vol. 21, No. 3, pp. 229-244.