1-1-2010

Extrusioncutter : a novel system for generating interactive context-preserving cutaways of anatomical surface meshes

Patrick Crawford
Ryerson University

EXTRUSIONCUTTER: A NOVEL SYSTEM FOR GENERATING INTERACTIVE CONTEXT-PRESERVING CUTAWAYS OF ANATOMICAL SURFACE MESHES

by

Patrick Crawford, BSc, Trent University, Peterborough, Ontario, 2009

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Science

in the Program of

Computer Science

Toronto, Ontario, Canada, 2010

I hereby declare that I am the sole author of this thesis. I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Patrick Crawford	Date

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

Patrick Crawford	Date

EXTRUSIONCUTTER: A NOVEL SYSTEM FOR GENERATING INTERACTIVE CONTEXT-PRESERVING CUTAWAYS OF ANATOMICAL SURFACE MESHES

Patrick Crawford

MSc, Computer Science, Ryerson University, 2010

ABSTRACT

This thesis presents ExtrusionCutter - a novel 3D visualization tool that enables users to create complex, context-preserving cutaways of anatomical surface meshes. To accomplish this, a "paint-roller" interaction metaphor has been developed that allows users to extrude an editable cutting mesh along the natural geometry of an occluding surface. This virtual analogy of a familiar real-world action not only facilitates the removal of occluding surfaces, but also creates a user-defined region parameterization which makes it possible to also generate an effective contextual outline view of the removed material. This thesis will demonstrate how the paint roller interaction metaphor has been implemented to facilitate the creation of multiple editable cutaway types in 3D anatomical surface meshes. Additionally, it will show that the resulting cutaway views are capable of exposing occluded parts while still maintaining their context in the visualization.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my thesis supervisor Dr. Tim McInerney for the endless hours he spent helping to develop this thesis. Without his expert guidance, innovative ideas and wealth of knowledge, none of this would be possible. I could not have asked for a more patient, kind and intelligent mentor and I feel honoured having worked under his supervision for the past two years.

I would also like to thank the members of my thesis committee, Dr. Denis Hamelin, Dr. Cherie Ding, and Dr. Alex Ferworn for their time and effort in reviewing my thesis and providing helpful feedback.

A very heartfelt thanks also belongs to my fiancé Jennifer Campbell for the infinite patience and tolerance she has exhibited over the past two years. She has been nothing but kindness despite the long hours and I am an extremely lucky man to have her in my corner.

Finally, I would like to thank my parents Martin and Debra Crawford, my brother Bredon, and my sister Carrie. They have always encouraged me during every step of my education and I cannot possibly thank them enough.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter I – Introduction

The medical industry has arguably been one of the major beneficiaries of the rapid advance in imaging technology within the last 40 years. Specifically, the development of Computerized Tomography (CT) and Magnetic Resonance Imaging (MRI) have aided the healthcare industry in areas such as disease treatment and detection as well as in surgical planning [29]. The advances made in computer rendering techniques has further enhanced these procedures by providing medical professionals with the ability to construct 3D visualizations of the data acquired from such devices. However, it is not merely enough to create and render a 3D representation of a system of anatomical structures, the radiologist or clinician must have software tools to accurately examine and explore the data. For example, in computer-aided surgical planning, surgeons must ascertain the complex shape and organization of intricate anatomical systems to find optimal approaches to a target structure or to plan a series of surgical actions.

To accomplish such a task, a powerful, intuitive interaction layer must exist between the user and the rendered 3D models. For example, when inspecting and manipulating complex systems of 3D objects, it is critical that the interaction layer provides users with the ability to see through the surfaces of enclosing objects to reveal interior objects or parts. Additionally, users must be given the tools to effectively understand and measure the spatial relationship between objects in the system, or to spatially overlay two or more similar objects in order to compare their shape and size.

Designing an interaction layer to provide these capabilities is a challenging interactive 3D visualization problem. As previously mentioned, one of the most important features is to allow

the user to examine hidden regions of interest in the system easily and intuitively. However, such an action must not be achieved at the expense of a contextual view of the surrounding or occluding structures. While simply clicking and deleting an occluding structure is an effective means of focusing on an interior structure, it comes at the cost of eliminating any contextual knowledge that the occluding structure may have provided (i.e. muscles / skin over bone).

A common solution to this problem is to employ the use of semi-transparency on an outer surface in an attempt to reveal the interior geometry of an object. However, images generated in this fashion tend to be noisy and do not effectively convey the spatial relationship or depth of interior objects, especially if there are multiple layers of transparency (Section 2.1). Simply cutting away parts of the occluding objects is arguably a more effective approach, but the user is forced to mentally "fill in the gap" of the removed material and estimate its shape. Other systems have attempted techniques such as feature lines or exploded diagrams (Section 2.1). However, the ideal solution would be to cut away the outer surface while still rendering some outline of it in such a way as to simultaneously minimize occlusion and provide effective visual depth and spatial relationship cues of the interior objects. Furthermore, if this outline is aligned with the natural geometry of the outer object's surface (e.g. along primary medial axes, ridge lines, primary curvature lines etc.) then the user can more readily mentally reconstruct the missing geometry.

To facilitate the construction of such a cutaway representation, the interaction layer must translate simple input (mouse / keyboard) actions into meaningful cutter construction and editing actions relative to the surface of the occluding structure rather than to the absolute coordinate space of the 3D system. Additionally, it is imperative that the interaction layer employs a simple "interaction metaphor" that is well understood and recognizable throughout the cutaway

generation process. This requirement is due to the fact that while radiologists, surgeons and medical technicians are very knowledgeable about anatomy, their considerable workloads prohibit the use of complex and restrictive interfaces too often associated with common medical visualization software. The goal then, is to have these expert users focus on their specific visualization task and not on how to use the visualization tools. A number of current systems have attempted to solve this interaction problem by employing various strategies such as: cutting or tracing, sketching or gesturing, sculpting or widget based systems (Section 2.2 - User Interaction Models for Generating Visualizations). However, this thesis claims that all of these alternate interaction strategies are inadequate, for different reasons.

This thesis introduces a novel "paint roller" interaction metaphor that facilitates the creation of numerous cut types (i.e. freeform, straight path, wedge, rounded and freeform cuts) through the use of a single intuitive, extrusion-based action. By allowing users to simply "paint" cutaway regions on the surface of an occluding structure, the task of generating complex cutaways is simplified. Furthermore the act of "painting" also allows the system to preserve a meaningful contextual view of the removed material by interpolating ribbons or solid slices through the painted "cutter mesh". This thesis illustrates how the novel "paint roller" interaction metaphor is capable of generating multiple context-preserving cutaway types in 3D anatomical surface meshes by using simple, consistent mouse actions.

1.1 Contributions of this Thesis

This thesis presents *ExtrusionCutter*, an interactive 3D visualization tool for generating highly-descriptive 3D visualizations in a fast, intuitive manner. A major contribution of this

thesis is the development of a novel interaction metaphor that allows the user to define and edit a cutaway region in a way that respects the natural curving geometry of a target anatomical structure. The cutter tool is initially defined as a single, user initialized rectangular box situated on the surface of an occluding structure. In order to extend the cutting region, a "paint-roller" style interaction model is employed to automatically extrude the cutter in a manner very similar to that of sweeping a real hand-held paint roller across the surface of a smooth object. This sweeping action is realized by ensuring that the leading edge of the cutter always remains orthogonal to the path of the mouse as the user traces along the natural shape lines (such as ridge lines or medial axes) of the occluding object. Additionally, the cutter is constrained to remain on the surface of the occluding object, regardless of its shape. Since extruding the cutter in this manner is effectively a virtual analogy of a familiar real-world action (i.e. using the mouse to "paint" a surface-sticky cutter mesh directly on the surface of an object along an arbitrary path), the user is able to generate a wide variety of different complex cutaway regions (Section 3.3) using simple mouse movements.

A second unique, major contribution of this thesis is the development of two key descriptive visualization options for rendering an effective contextual outline of the cutaway surface. These visualization options are the use of "ribbons" and thin "solid slices" (Section 3.4) positioned automatically relative to the "painted" cutter mesh and hence along ridge lines or medial axes. These visualization options are made possible by the extrusion process itself, as a series of bounded cutting planes are constructed as the cutter is extruded by the user along an arbitrary path over the occluding surface. In addition, the cutter (including the bounded cutting planes) has an explicit representation (a mesh of quadrilateral polygons), providing the capability to edit the cutaway region (Section 3.3.3). If the user is not satisfied with the results of a cut

operation, they can directly manipulate the cutter and re-cut, rather than reconstructing the entire cutter or editing the cutaway region indirectly through widget controls.

1.2 Thesis Outline

Chapter 2 describes a variety of competing 3D visualization techniques in the literature that focus on the interactive exploration of complex, multi-part data sets. A brief outline of the basics of visualizing volumetric data, including the extraction of surface meshes, is presented. An outline of a number of key and related visualization techniques such as transparency, wireframe/lattice and cutaway renderings is then presented. Finally, the chapter concludes with a look at a number of important interaction models in the literature, used to generate 3D visualizations, including gesture, sculpting and widget-based interaction models.

Chapter 3 describes the implementation and methodology employed in the creation of the ExtrusionCutter tool. Key descriptions include: the cutter representation, the cutter initialization and extrusion process, various algorithms developed to support the initialization and extrusion process as well as cutter editing, the different types of cutaways supported, the algorithms developed to support the various descriptive visualization options, and lastly a detailed description of the ExtrusionCutter system graphical user interface (GUI).

Chapter 4 presents series of experiments and results used to evaluate the system, including the flexibility and effectiveness of the visualization options, cutter generation and editing efficiency, and the simplicity and flexibility of the interaction metaphor.

Chapter 5 concludes with final remarks including suggested improvements to the system as well as possible future work.

Chapter II – Literature Survey

As computer graphics hardware moves forward, so too does the capacity to create higher quality and more complex 3D medical visualizations capable of being rendered and explored in real time. However, it is due to this ever-increasing capacity to render and manipulate complex systems of anatomical structures that a need has arisen to develop increasingly sophisticated 3D interaction and visualization tools to aid in their understanding.

Since anatomical structures often have intricate shapes and complex spatial interrelationships, it is important to enable users to explore and *focus* on internal structures without losing the *context* provided by external (occluding) surfaces or objects. In the literature, this well-known problem is referred to, not surprisingly, as the focus+context problem [4]. More generally, the challenge is to show a given targeted element of a visualization with as much detail as possible, without losing any information about how it relates to the visualization as a whole. The problem is present not only in medical visualization, but also in scientific visualization and information visualization. In 3D medical visualizations, this problem can be addressed in a number of ways, from creating dynamic cutaways or exploded views of objects to exploring different highlighting/de-emphasis techniques. Sections 2.1.1 through 2.1.3 will describe and critique a number of leading techniques designed to address this focus+context problem in greater detail.

While the ability to create context-preserving visualizations of internal structures is an important 3D visualization task, it is equally important that the user is able to intuitively and efficiently generate these visualizations. As previously stated, the considerable workloads of radiologists, surgeons and medical technicians prohibit the use of complex and restrictive

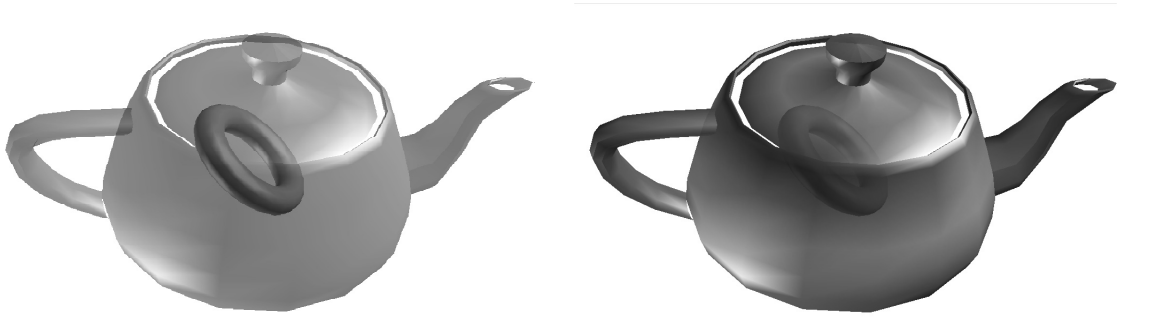
interfaces. Consequently, even if a visualization tool has the ability to create an elegant, context preserving visualization of a given anatomical structure, it is unlikely that it will be used if it is difficult, cumbersome, and time consuming to do so. Therefore, in order to fully evaluate a given visualization tool, it is vital to study its user interaction model in addition to the final visualizations that it is capable of generating. To this end, a number of key interaction models designed to generate a wide variety of 3D visualizations are outlined in Section 2.2.

The following sections describe 3D visualization techniques that utilize polygonal boundary surface meshes to represent anatomical structures. The meshes are constructed from isosurfaces, which in turn are extracted from a volumetric dataset (i.e. a stack of discreet 2D digital image "slices" obtained from an MRI/CT scanner) and can be rendered using the standard surface rendering technique. An isosurface is a surface formed from a triangulation of data points of equal value or density [31] within a volume dataset. Typically, each anatomical structure buried in a volume image is defined by a connected set of voxels[1] which exhibit a specific "density" (i.e. brightness) value within some (often not clearly defined) range. An isosurface envelopes the boundary voxels belonging to a particular anatomical structure in the volume. For example, to extract the isosurface of bone material within a volume dataset, a bone density threshold value is set, known as an isovalue. An algorithm, such as the famous Marching Cubes algorithm [31], can then be used to "visit" each voxel in the volume and compute isosurface triangles within the boundary voxels of an anatomical structure, based on the density values of neighbouring voxels. The collection of all these triangles is used to construct the anatomical structure polygonal mesh.

[1] A voxel is a 3D pixel and can be visualized as a small cube.

2.1 Visualizing Internal Structures Using Transparency

A system of anatomical surface meshes, such as the cerebral cortex of the brain and internal structures such as the lateral ventricles, caudate nucleus, and putamen, are commonly extracted from medical volume images. Examining the internal structures, i.e. "looking inside", requires the processing of occluding structures - in this case, the cerebral cortex. While simply removing occluding structures from the system of meshes is a possibility, this action effectively destroys any contextual/spatial link between the inner and outer objects. Another more effective strategy is to render the occluding structures as semi-transparent. For example, Diepstraten et al. [13] employed the intelligent use of transparency in a manner consistent with the traditional scientific visualization guidelines presented in Hodge's "The Guild Handbook of Scientific Illustration." [14]. Hodges states that the intensity of the opaque object should be set to zero at the edge of the surrounding object and slowly increase its intensity with increasing distance from the edge. In effect, this formalizes an edge highlighting method that can be used to help preserve depth cues in a semi-transparent object. The advantage of employing such a technique is evident in Figure 1b, in which the depth of the occluding structure (teapot) is more accurately preserved.



(a) (b)

Figure 1 – Rendering the Utah Teapot using: (a) transparency (b) transparency with edge highlighting [13]

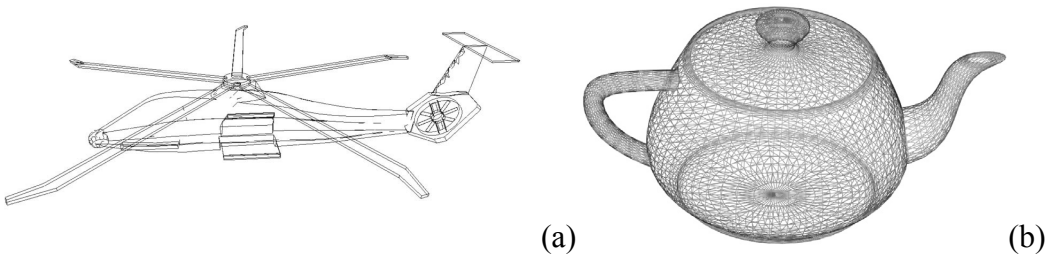
However, it is also evident from Figure 2 that transparency, when used alone, is often inadequate for the interpretation of the depth and spatial relationship of the occluded structure (the torus) with respect to the occluding structure (the teapot). Furthermore, this problem is compounded when there are several layers of semi-transparent objects – the image quickly becomes very noisy. This is the key problem with the use of transparency as a solution to the focus+context problem. While it is indeed possible to focus on an internal structure, it is difficult to judge its distance from the occluding structure in the viewing direction - a problem clearly identified in a number of similar efforts [17] [9]. To summarize, transparency is an effective means of quickly gleaning the basic organizational structure of objects within the system. However, to gain a better understanding of exactly how those objects are positioned relative to each other, further visualization techniques need to be investigated.

2.1.1 Visualizing Using Lattice Rendering and Feature Line Rendering

As discussed above, one of the major challenges in providing a contextual view of occluded objects in a visualization is the preservation of the shape outline of occluding structures. This becomes particularly difficult if transparency alone is used, as layers of transparency interfere with each other and depth information is lost. However, there are a number of techniques that attempt to preserve this depth information through the use of feature lines or shaded ribbons.

One example of the generation and rendering of feature lines is presented by Kwan-Liu Ma et. al. [23]. Unlike simple wireframe rendering (Figure 2b), a feature line representation is

capable of subtlety depicting the shape of an object without the visual noise and occlusion problems present when rendering every triangle in the mesh.



(a) (b)

Figure 2 – Wireframe rendering styles: (a) Feature Lines [23] (b) Simple wireframe

Furthermore, representing shapes as a form of "line drawing" has been suggested in works such as Marr's "Early Processing of Visual Information" [24] as being directly related to how images are stored and processed in the brain. Essentially then, if an edge can be highlighted effectively, the observer has a greater chance of mentally reconstructing the missing surface information.

In this case (Figure 2a), the highlighting can be accomplished by first looking at the edges connecting every triangle in the mesh. If the angle between the two corresponding triangles is beyond a certain threshold, then the edge is considered to be a "feature edge" and is preserved. Edges that connect triangles whose angles are within the threshold are then discarded leaving only the feature edges of the object. However, while this may work for some objects (i.e. those that exhibit sharp feature lines) it would undoubtedly fail on meshes whose geometry is smooth and rounded. Unfortunately, this is often the case with anatomical structures.

A related visualization technique that provides a further alternative take on simple wireframe rendering is lattice shading. Through the use of modern programmable graphics processing units (GPUs) [32] it is now possible to procedurally discard elements of the mesh during the rendering procedure itself. For example, texture coordinate value ranges may be set such that the mesh is rendered as a connected network of "ribbons" (Figure 3).

Figure 3 – Utah Teapot rendered using a lattice shader [8]

What makes this form of "looking inside" particularly compelling, is the lighting calculation itself. The fact that the ribbons are shaded means that the observer is able to discern their depth in the visualization more readily than using simple wireframe contours. Furthermore, the user can also shade each side of a ribbon with a different colour or lighting model. For example, the inside of the ribbons could be coloured in a darker shade than the outside. Using this technique, it would then become more evident which contours are facing away from the observer, further reinforcing the contextual shape of the occluding object.

However, while the use of shaded ribbons to convey shape is an improvement over wireframe rendering, lattice shading perhaps works best on a low resolution (i.e. low number of polygons) quadrilateral mesh, where the mesh parameterization (i.e. layout of the quads) is aligned with the natural curvature lines of the object. Anatomical surface meshes extracted from medical volume images commonly contain tens of thousands, or even hundreds of thousands, of *triangles*. Lattice shading would lose its effectiveness with these high-resolution meshes of small, randomly oriented triangles.

2.1.2 Visualizing Using Exploded Views

The previous two sections have outlined multiple attempts to solve the focus+context problem by enabling the observer to peer through occluding objects, while at the same time preserving some outline of their shape. However, in some applications, for example medical education, it is possible to expose occluded objects without the need for reduced transparency or the use of contour lines, but rather through the use of exploded (i.e. pushed apart) views [21]. If each system and subsystem of objects in the visualization is exploded in a meaningful way, a global sense of how these structures relate to each other can also be achieved. This is evidenced by Wilmot Li, et al. in their paper "Automated Generation of Interactive 3D Exploded View Diagrams" [21].

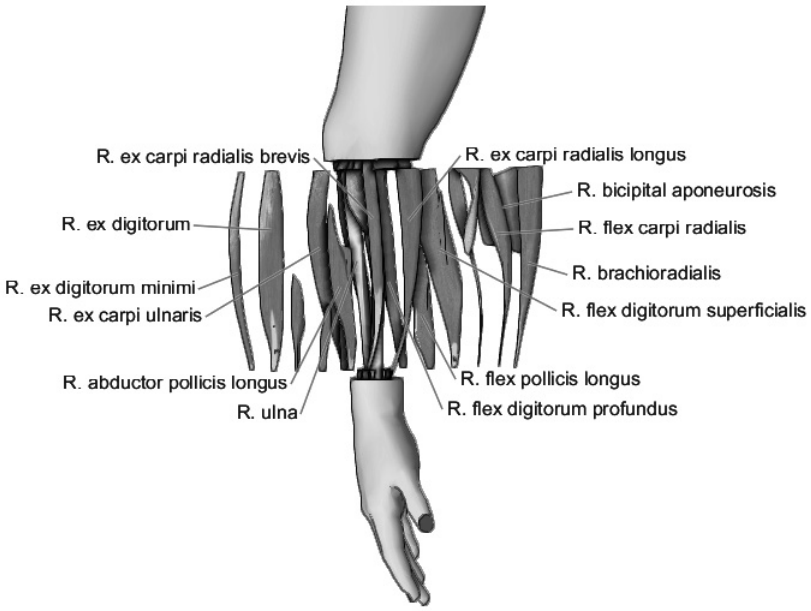


Figure 4 – Exploded view of a segmented human forearm [21]

To achieve this effect (Figure 4) a number of novel algorithms are employed to determine how the objects in the system are to explode relative to each other. For example, one algorithm first

computes low-level spatial relationships, such as which objects are containing, touching or blocking other parts, with respect to the system's coordinate axes (used as explosion directions) [2]. Using this information, an "explosion graph" can be constructed to store these relationships as well as how far each object in the graph needs to move in order to escape the bounding box encompassing the blocking or occluding structures - essentially making it visible. However, since the amount an object must move to become visible is view-dependent, additional graphs are pre-computed for many other viewpoints. Before the objects can be positioned however, a number of other algorithms need to be employed as well, including how to resolve interlocking object hierarchies and how to split or cut container objects [21].

Once all of the pre-processing (or "rigging") has been completed by an author, a viewer (for example, a medical student) can interactively explode regions of interest. The exploded views are also animated to visually reinforce the layering relationships between them. Exploded views are useful only for a limited range of object types and object interrelationships. For example, if one wanted to visually expose how a complex network of arteries or veins is situated with respect to a particular muscle, then moving the elements apart along any axis would do little to illustrate the spatial relationships between them. Furthermore, the user is unable to measure the distance between the occluding and occluded structures at a given point within the visualization.

2.1.3 Visualizing Using Cutaways

Up to this point, the techniques proposed to visualize internal or occluded objects in a complex system of anatomical structures have proven to be inadequate and/or limited in

application. For example, if transparency alone is used, it is possible to visualize occluded structures; however their position within the global visualization (context) can be extremely difficult to comprehend. If, on the other hand, ribbons or a mesh lattice is used to render occluding objects, it is much easier to determine their depth, position, orientation etc., with respect to occluding structures. Unfortunately, in order to render structures in this manner, an existing appropriate parameterization (i.e. layout of mesh polygons) of the mesh must exist or be easily computable.

Another approach to viewing occluded internal structures is to cut away portions of the occluding object surface. This can be accomplished via Constructive Solid Geometry (CSG) Boolean operations in which a "cutter" mesh (sometimes referred to as the cutter "region" or "volume") (sphere – Figure 5) is used to subtract geometry from a target structure (cube – Figure 5) to generate a cutaway view.

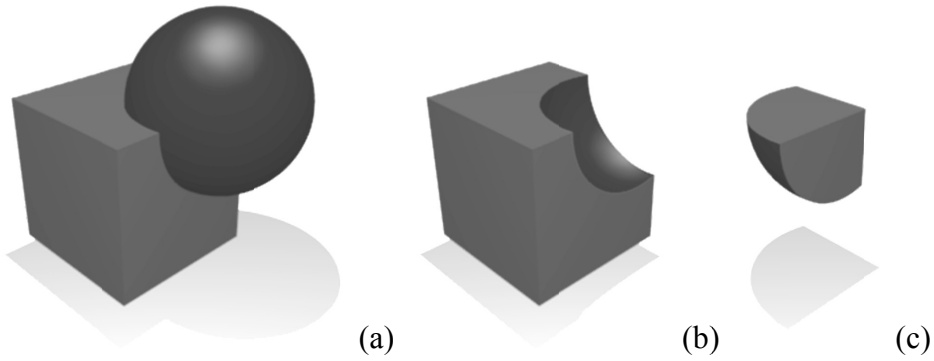


Figure 5 – CSG Boolean Operations between a Sphere and a Cube with (b) illustrating the Boolean subtraction and (c) illustrating the Boolean intersection [1]

Cutaway visualization techniques for surface meshes have been actively researched in the visualization community. For example, Coffin et al. [10] present an interactive technique to generate and manipulate cutaway views in polygonal data based on either standard or user defined cutter shapes. Likewise, Diepstraten et al. [12] demonstrate different methods to create

and depict cutaways in polygonal data that include creating "sawtooth" cuts to better distinguish between inner and outer surfaces, and creating breakaway illustrations where a hole is cut in the occluding surface that is just big enough to see the targeted internal structures. In Knodel et al. [19] users generate cutaways using simple sketching actions and then refine the shape of the cut using 3D widgets.

One of the most interesting papers is presented by Li et al. [22] in their paper "Interactive Cutaway Illustrations of Complex 3D models". The target application of this work is medical and technical education and it is not applicable as an interactive tool for radiologists and surgeons. An "author" works with a polygonal mesh dataset and uses a menu to choose among several cutaway tools. The author pre-defines the cutaway region using these tools and creates an application program. The program is then used by viewers (for example, medical students) to interactively and progressively open up the pre-defined cutaway regions in order to examine internal anatomical structures.

From a visual standpoint, Li et al. have modeled their solution to the focus+context problem on illustrative conventions outlined in such books as "The Guild Handbook of Scientific Illustration" by Elaine R. S. Hodges [16] and the Atlas of Human Anatomy by Frank H. Netter [26]. Furthermore, it is well known that a simple "naive cutaway" does not allow the viewer to properly observe the spatial interrelationships between the target internal structure(s) and its surrounding structures [22]. As such, the types of cutaways possible in Li's system extend beyond simply punching a hole through the dataset (i.e. naive cutaways) and instead focus on "respecting the geometry of the parts being cut" [22]. In addition, they employ novel rendering schemes to enhance the appearance of cutaway regions, including non-photorealistic (i.e.

simplified) rendering (NPR), edge shadows/shading, real-time depth cueing and dynamic labelling.

Using their system, a series of cuts can be designed in such a way that the viewer is capable of mentally reassembling the severed objects [22]. One of the primary ways this is achieved is by designing cuts to respect the corresponding geometry. Furthermore, Li et al. claim that to maximize the viewer's ability to mentally reconstruct cutaway regions, it is important to use an appropriate cut type. For example, intuitively exposing structures inside a cylindrical object would most likely require a wedge-cut, whereas long, narrow tubes necessitate a transverse cut. If the type of cut represents the geometry of the corresponding object in a natural way, it is far easier to visualize the position and orientation of the removed piece [22]. To this end, Li et al. have introduced 4 major categories that a segmented structure in a volumetric dataset can belong to: a rectangular parallelepiped, a long narrow tube, a short radially symmetric tube, or an extended shell. Once a structure has been identified by an author as belonging to one of these categories, an automatic process generates a cutting volume and maps its extents and orientation using an appropriate 1D, 2D, or 3D parameter space.

Rectangular Parallelepiped

Objects identified by an author as a "Rectangular Parallelepiped" are cut using an "object-aligned box" cutting volume (Figure 6). In this type of cut, the positioning and range of the cutter is automatically determined. This can be overridden and adjusted by the author. A viewer can then interactively slide the box cutter along the occluding object, performing a cutaway in real-time.

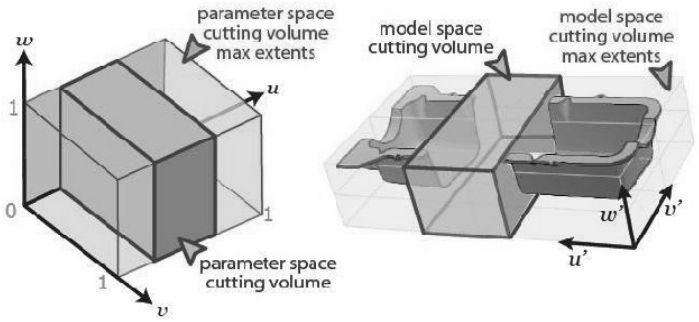


Figure 6 – Object Aligned Box Cut [22]

Long Narrow Tube

Long narrow tube structures are cut using a "transverse tube" cutting volume, automatically positioned and parameterized by automatically computing the primary centerline axis (u' in Figure 7). A viewer can, once again, interactively slide the tube cutter along the tubular object.

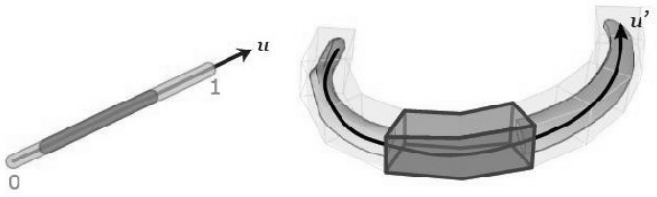


Figure 7 – Transverse Tube Cut [22]

Radially Symmetric Tube

Structures classified as radially symmetric tubes are cut using a variation of the transverse tube cut called a "wedge tube" cutting volume (Figure 8).

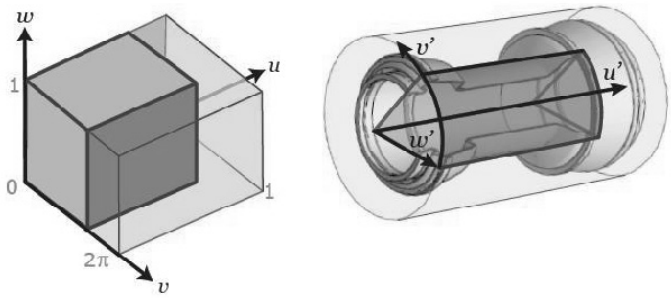


Figure 8 – Wedge Tube Cut [22]

Extended Shell

An object classified as belonging to the extended shell category is capable of being cut in one of two ways: "freeform window" and "four-sided window". As the names imply, both cutting volumes work to open a window in the occluding surface by punching a hole through the object's geometry along a closed bounding curve. To identify the curve in a freeform window cut, the author can chose to either draw directly on the surface, or let the system identify a curve automatically. If the user opts for the automatic generation of the bounding curve, the system first computes the regions on the surface that immediately occlude underlying structures. It then draws a closed line around these points with respect to the current viewpoint, creating a bounding region (window). This window is then parameterized by a single value, allowing viewers to expand or shrink the window cutaway away or towards its center (Figure 9).

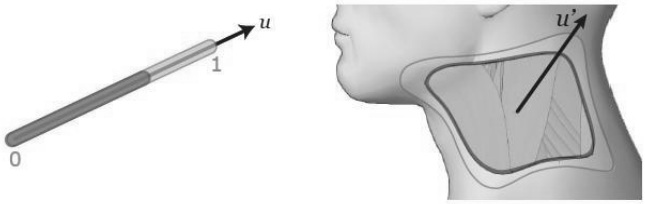


Figure 9 – Freeform Window Cut [22]

In the case of the four-sided window cut, the size of the cutaway region is created from two (roughly parallel) curves drawn across the surface of the object. The corresponding top and bottom points of these two curves are joined by automatically computed geodesic (i.e. shortest path on a surface) curves, forming a four-sided window (Figure 10). Like the freeform window, the viewer is able to expand or shrink the window within predefined limits.

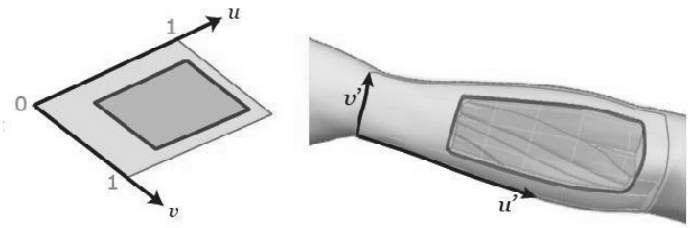


Figure 10 – Four Sided Window Cut [22]

In summary, cutaways offer an effective solution to at least part of the focus+context problem of viewing internal structures in anatomical datasets, especially if the cutaway regions are aligned with the natural geometry of the occluding structures. However, what is missing is some contextual view of the cutaway region. Furthermore, although the medical education oriented author-viewer system of Li et al. [22] cannot be directly compared to ExtrusionCutter, valuable insight can be gained by examining its authoring process. In this process, the author must first manually identify each anatomical structure as one of the four predefined shapes (i.e. rectangular parallelepiped, long narrow tube, radially symmetric tube, extended shell). However, this task puts an unnecessary cognitive load on the author of the visualization, as this identification can be difficult if a given object does not have a simple shape (Figure 11). Furthermore, some structures may not fit into this classification scheme at all.

Figure 11 – A segment of a branching artery system [37]

2.2 User Interaction Models for Generating Visualizations

Although cutaway views pose a promising solution for viewing internal structures, as is evidenced by Li et al. [22] above there is still the problem of how easy it is to for the user to generate the cutaways. That is, relying on the user to choose an appropriate cutting tool from a menu, with each tool applied in a different way, is problematic. Ideally, a *single* cutting tool would be useable for a wide range of object types. Furthermore, the tool should be simple, intuitive and efficient to use. The tool (and hence the cutaway region) should also be highly editable. In the following sections, a number of common cutaway interaction models in the literature are described and critiqued with respect to these properties.

2.2.1 Cutting or Tracing Interaction Models

Cutting or tracing-based interaction models attempt to mimic the real world action of a cutting tool, such as a scalpel, to remove a specific region of interest and expose occluded

structures [6] [22]. Essentially, this interaction metaphor is similar to the previously discussed "Freeform Window Cut" (Section 2.1.3) described in the system created by Li et al.[22]. Recall that in order to create a cutaway region in the occluding structure, the user is required to trace a contour on the object directly (Figure 12).

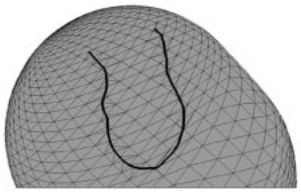


Figure 12 – Tracing a cutaway contour on a polygonal mesh [7]

Once a closed contour has been traced on the target mesh, it explicitly defines the area on the surface to be removed. However, as in similar cutting systems [6], the scalpel/tracing interaction metaphor can pose a number of problems. For example, the simple act of tracing can become tedious and difficult if the desired cutaway region covers a large, complicated area of the occluding mesh. Enabling the user to edit the cutaway is also problematic as the cutaway region is represented by an explicit curve. Consequently, if the user wishes to modify the cutaway region, they must either undo (backtrack) a portion of the curve and retrace or individually manipulate the curve's vertices. Moreover, when using a sketching or tracing metaphor alone, it is difficult to construct a more structured, symmetric or volume-based cut, typically used to cut through large volumes or thick shells.

2.2.2 Sketch or Gesture Based Interaction Models

Sketch or gesture-based interaction is another metaphor that has been used in cutaway systems to expose occluded structures [19]. In this scenario the user roughly sketches different

patterns of curves on an occluding object's surface as a means to define a certain type of volumetric cut. For example, Knödel et al. [19] have created a system in which four unique curve gestures are employed to initialize their cutter:

Line Gesture

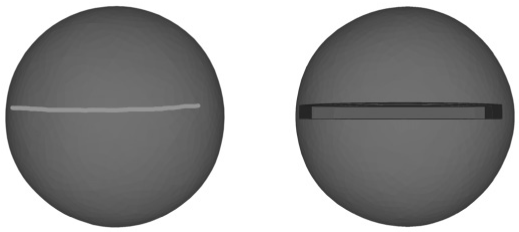


Figure 13 – Knife-like Cutaway: (left) Initial line Gesture and (right) resulting cutaway [19]

In order to initialize a narrow, knife-like cut through an object, the user simply sketches a line over the target area (Figure 13). The length, position and orientation of this line are then used as the basis to construct a narrow box volume with a fixed width. The depth of volume however, is calculated relative to the size of the target object's bounding box.

Corner Gesture

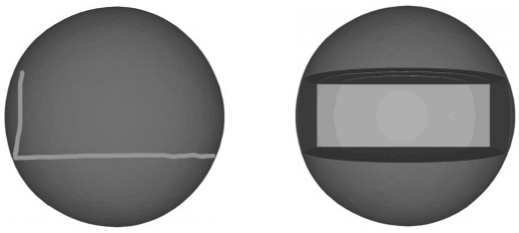


Figure 14 – Box volumetric cut: Initialization and visualization [19]

If a larger initial cutaway is desired, the system recognizes two consecutive sketches in the form of a vertical line, followed by a horizontal line (Figure 14). Similar to how a line

gesture initializes a thin cut, these two strokes provide the length, width, position and orientation of the box-shaped cutting volume. The initial depth of the box is also calculated based on the size of the target object's bounding box.

Circular Gesture

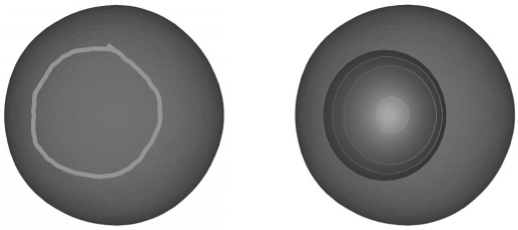


Figure 15 – Spherical volumetric cut: Initialization and visualization [19]

Under certain circumstances, a spherical volumetric cut can be effective in illustrating how an occluded internal structure relates to its surrounding structures in the visualization. To this end, a sphere-based volumetric cut can be initialized by sketching a closed, circular loop over the target object (Figure 15). In effect, the location and vertical/horizontal dimensions of the sketch are used to position a sphere of similar diameter. Unlike the previous two cuts, the depth of the cutting volume does not need to be calculated and as such, the bounding box of the target object is not referenced.

Ribbon Gesture

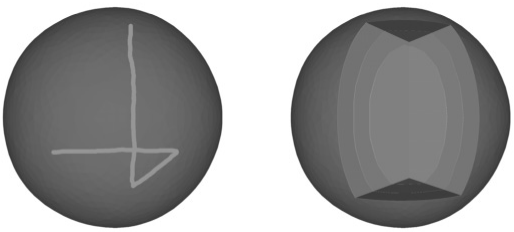


Figure 16 – Wedge volumetric cut: Initialization and visualization [19]

Wedge-shaped cuts are an important cutaway type, as they are capable of effectively illustrating the layering relationships of one or more occluding structures. In order to facilitate the initialization of such cuts through sketching, the system recognizes a single contiguous curve that intersects itself at approximately 90 degrees (Figure 16). In this way, the height, width and orientation of the wedge-shaped cutting volume can be specified in one motion. Again, the depth of the wedge is calculated based on the size of the target's bounding box.

Once the system has constructed the cutting volume based on the curve(s) sketched by the user, an image-based Boolean subtraction [27] is performed to create the cutaway view for the current viewpoint.

However, if the end-user wishes to edit the cutting volume, a simple widget-based system (further illustrated in Section 2.2.3) is employed to enable the user to perform basic manipulation tasks such as translating or resizing in the x-y plane. While this does allow the user to have greater control over the cutting volume than the cutaway system proposed by Li et al [22], it also presents a new set of problems. For example, one of the fundamental concerns with the gesture-based interaction metaphor is that users are not able to directly visualize the shape of the cutaway region as they create it. Therefore, if the shape interpreted by the system is not what the user expected, they are forced to edit the region using a second, more restrictive interaction metaphor (a widget). Furthermore, while the system has presented four key cutaway shapes, further unique shapes may be required to generate effective cutaway views for certain complex anatomical structures (i.e. Figure 11).

2.2.3 Widget Based Interaction Models

Widget-based interaction models essentially describe the use of a "widget" as a means to control objects or cutting volumes in the scene. A widget can be formally defined as: "an encapsulation of geometry and behaviour used to control or display information about application objects" [11]. Essentially, in the context of generating and manipulating cutaway views, the formal definition describes a set of explicit controls (i.e., buttons, 3D sliders etc.) to control certain physical properties of the cutting volume. For example, Figure 13 illustrates how the controls are laid out in the sketch-based interaction metaphor used by Knödel et al. [19].

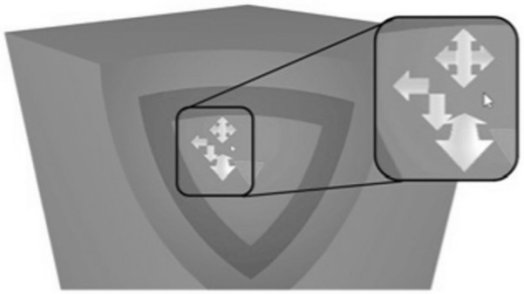


Figure 17 – Sample widget layout and functionality [19]

In this case, the user can click and drag a particular arrow to either translate or scale the current cutting volume. However, even in increasingly complex widget-based systems [25], there exists a fundamental control problem if widgets alone are used as a means to manipulate cutaways. Due to the restrictive and indirect nature of "editing by proxy" via an explicit set of controls, it can become difficult to directly alter the cutting volume appropriately. That is, the more restrictive the controls provided by the widget, the more difficult it becomes to create context-specific cutaways that are aligned with the natural geometry of the occluded surface. Essentially, this means that if one is only able to manipulate the cutting volume in a restrictive, constrained

way, the cutaway possibilities become extremely limited. Even in more advanced widget based systems such as the work presented by McGuffin et al. [25], the user is only provided a specific set of interactions to perform with a finite number of widgets. Fundamentally then, the user of the visualization tool must decide which type of widget or cutting volume to use for a given situation, rather than directly manipulating a flexible, editable cutter mesh (Section 3.3.3).

Chapter III – Methodology and Implementation

Through careful analysis of the various strategies employed to solve the focus+context problem that occurs when exploring 3D triangular meshes, a number of important issues have emerged. Firstly, in order to successfully expose targeted internal structures while maintaining the context provided by the occluding and neighbouring structures, it is vital that occluding structures are only minimally altered. As such, a cutaway visualization tool has been developed that combines the use of CSG Boolean subtraction and a carefully constructed cutter mesh to selectively remove parts of a specified occluding anatomical structure mesh. Secondly, as motivated in the previous chapters, the method to generate and manipulate/edit the cutter mesh (i.e. the interaction metaphor) must be extremely flexible, while still being simple and intuitive to use. Finally, the cutter mesh itself must not be limited to simple shapes; complex anatomical structures require unique cutaways that are aligned with the geometry of the occluding structure – an essential feature that greatly increases the observer's ability to mentally reconstruct the removed geometry [22]. This chapter details the ExtrusionCutter implementation in order to show how the system meets each of these three requirements.

3.1 Loading and Viewing Data

Before any cuts of 3D anatomical surface meshes can be made, they must first be loaded into memory and rendered on screen. To this end, an open source software toolkit known as the Visualization Toolkit (VTK) [39] was used as the underlying software platform of ExtrusionCutter. VTK is a large set of C++ classes, built on top of the standard Open Graphics

Library (OpenGL) [28] (standard open-source computer graphics API), and it provides extensive functionality for the creation and rendering of objects needed in complex 3D data visualizations. VTK facilitates the rapid construction of custom visualization systems by providing, among other features, numerous classes that implement common and essential algorithms. For example, VTK provides classes to perform all basic object rendering and loading of 3D data from disk files into the ExtrusionCutter system. To load an anatomical surface mesh, convenient methods are provided in the vtkDataSetReader class to read edge, face and vertex information stored in an ascii or binary (.vtk) file. This mesh data is then passed to a number of other rendering objects to determine both vertex and face normals, as well as to assign lighting properties such as ambient (overall background light), specular (i.e. shininess) and diffuse (matte) values to the mesh. This rendering data, as well as the mesh data itself, is then referenced by a "vtkActor" instance, effectively transforming the mesh data into an actor on the rendering stage (Figure 18a).

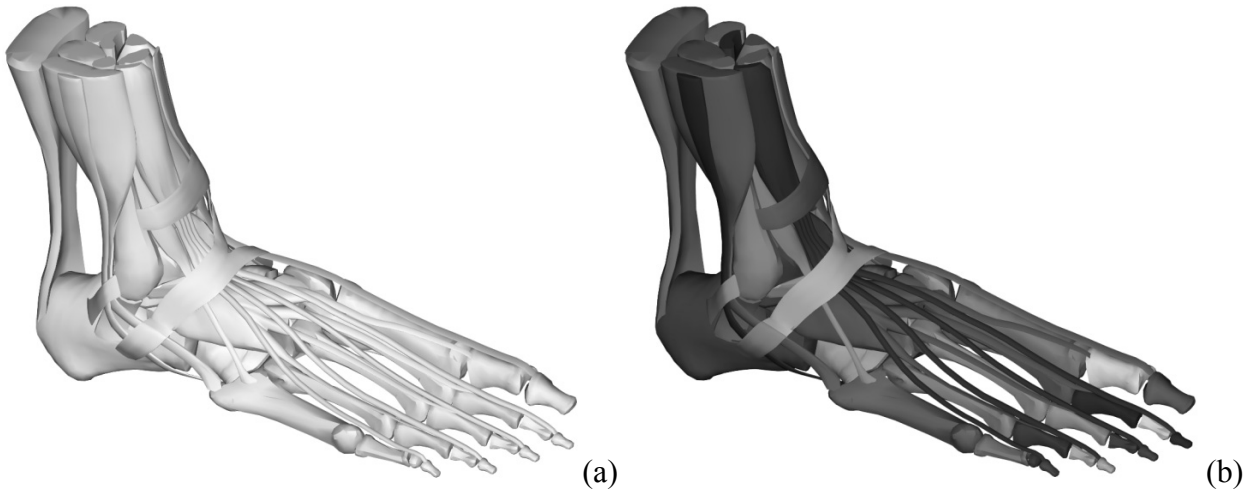


Figure 18 – Rendering of foot dataset made up of 45964 triangles (faces). (a) No individual parts identified (b) 53 parts identified and rendered using individual actors.

Often a single VTK file will contain a number of separate, segmented anatomical structures. In this case, the ExtrusionCutter system automatically identifies each structure by interpreting "regions" or parts (i.e. closed sub-meshes) within the mesh data read from the file. Each region is then explicitly stored as a separate mesh and associated with its own actor and rendered with a random colour to easily identify it within the visualization (Figure 18b). Lastly, as collision detection between object meshes is widely used by the algorithms employed in the ExtrusionCutter system, each actor in the scene is built with an associated Oriented Bounding Box Tree (OBBTree) [15] spatial data structure. Essentially, by building an OBBTree for every actor in the visualization as it is loaded, the time required to perform the numerous collision-detection calculations (required when positioning a cutter mesh within or around an anatomical structure mesh for example) are greatly reduced.

3.2 Cutter Representation

As previously stated, it is important that cutters in the ExtrusionCutter system are able to take on more complex shapes than simple convex cutters - such as a sphere, cylinder or box – commonly found in other systems. Therefore, each individual cutter is represented in VTK as a fully editable 3D closed mesh of quadrilaterals. It can be visualized as a connected series of quadrilateral polyhedra (loosely referred to as "boxes") (Figure 19a).

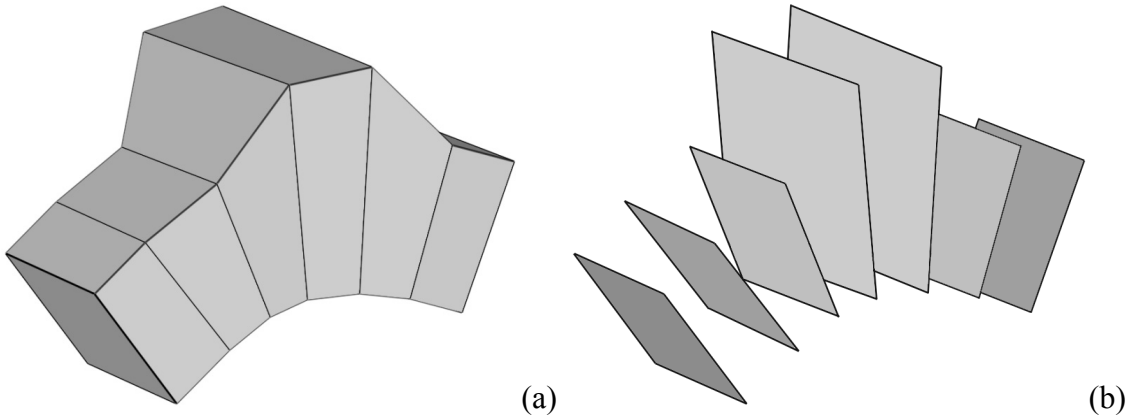


Figure 19 – (a) Simple, opaque view of the cutter mesh representation used in ExtrusionCutter (b) Bounded cutting plane representation of the cutter mesh representation

From an implementation perspective, it can also be considered as a linear list of bounded cutting planes, situated at right angles to the medial axis (i.e. center line) of the cutter (Figure 19b). Each box of the rectangular mesh itself is constructed by connecting the four vertices of the previous bounded cutting plane with the new bounded cutting plane (created during extrusion) to form the top, bottom, front and back faces. That is, the left and right sides of each box are the actual bounded cutting planes themselves. Essentially then, the cutter can also be thought of as a series of editable planes that intersect the anatomical structure mesh at given points – a visual analogy that is further reinforced by the novel interaction metaphor employed to generate and edit these planes (outlined in Section 3.3).

3.3 Generating the Cutter – The Extrusion Process

The following sections describe how the cutter mesh is generated and how it can be edited (i.e. the "interaction metaphor"). Since the cutter mesh used in ExtrusionCutter is a connected series of fully editable bounded cutting planes, close attention must be paid to how these planes can be positioned simply and accurately. To this end, a number of smoothing and

constraint-based algorithms have been developed to facilitate the use of simple, two-dimensional mouse movements for the placement and modification of these planes in three-dimensional space so that the cutter intersects or surrounds an occluding anatomical structure mesh.

3.3.1 Initializing a Cutter

Creating a cutter along (or surrounding) the surface of an occluding object involves several simple, intuitive user input actions. However, before these input actions take place, the ExtrusionCutter system automatically calculates a common "increment" value (δ). This value is used as the smallest increment that any object can move in the 3D world relative to the size of all of anatomical surface meshes in the visualization. Since the scale of these meshes can vary considerably from one visualization to the next, (δ) must be determined relative to the coordinate system used by objects on a visualization-case basis. Fortunately, the virtual camera can be used as a frame of reference for approximating scale. For example, once every structure has been loaded into the visualization, the camera distance is automatically reset from the default viewpoint using a native VTK method. Essentially, the algorithm works by first positioning the virtual camera at the center point of all actors in the scene. It is then translated along the view plane normal (i.e. viewpoint direction) far enough to allow every actor to be seen in full. In effect, this distance can then be used as an approximation of the maximum space occupied by the objects in the scene, relative to their coordinate system. Determining (δ) is then a simple matter of dividing the virtual camera distance by a constant value. In the case of ExtrusionCutter, a constant fraction of $1/2000^{th}$ of this distance (determined through experiment) is employed as the standard increment measure for every visualization. Once (δ) has been calculated, it is used to

determine a number of important cutter properties during initialization such as the initial cutter width and depth.

To begin the cutaway process, the user first selects a target data object by double-clicking with the mouse to highlight the object. In this way, the user has full control over which object will be directly affected by each newly constructed cutter. Furthermore, this also ensures that subsequent cutter edits will not be hindered by other objects in the scene as they are ignored by the system once a target object has been identified. This leaves the user free to manipulate the cutter with respect to the target object without worrying about accidentally altering the appearance of other objects in the visualization.

Once a target object has been selected for cutting, the user initializes the cutter at the desired location on its surface. The initial positioning of the cutter is accomplished by clicking a point that appears to be on (or near) the surface of the target object (point "A" in Figure 21), and then holding down the mouse button and dragging a cursor point that appears to be slightly above the object surface (point "B" in Figure 21), continuously extending a narrow rectangular strip.

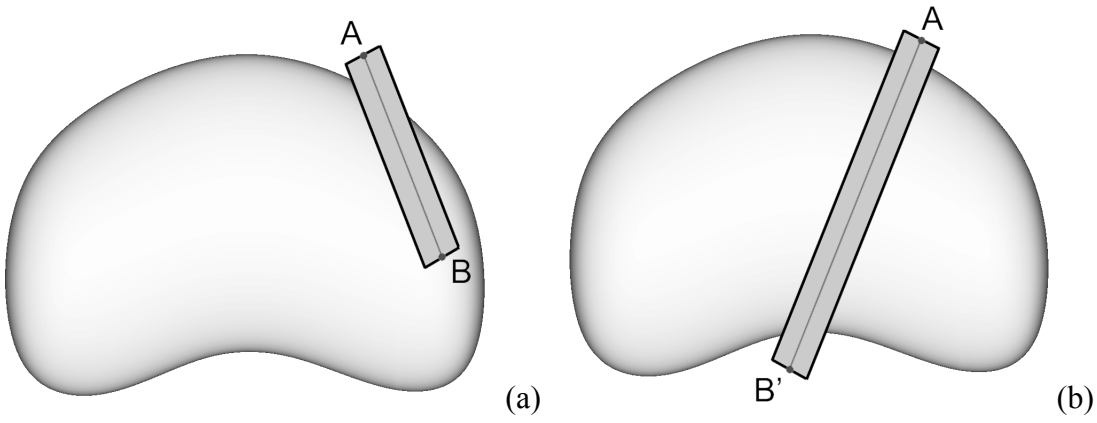


Figure 20 – A cutter is initialized by clicking on an initial point on or near the target occluding object surface, and then dragging a second point that appears along the surface to form a narrow rectangular strip.

The narrow strip is assigned a width of 10δ. The length and orientation of the strip can be continually and smoothly changed until the user is satisfied, whereupon the user releases the mouse button. The user is also able to click on and change the initial point ("A"), if desired. This simple process enables the user to establish and view the position, orientation, and height (i.e. vector AB' in Figure 20b) of the start of the cutaway region, as well as to make slight adjustments to the height or orientation if necessary.

From the user's perspective, the clicking, dragging, and extending of the initial cutter strip appears visually to take place on or slightly above the surface of the target object. Rather than picking points directly on the object surface and then constructing the strip in 3D space, the strip is constructed on the 2D view plane[2]. This indirect initialization process provides the user with the ability to click on a point in space above/below the object and create a cutter that surrounds the object. It also ensures that if the user wishes to create a cutter that completely surrounds a region of the object (i.e. placing the initial point above (or below) the region), mesh points that are far away in depth from the region of interest will not be inadvertently chosen.

Once the strip has been extended and the mouse button has been released, the view-plane constrained strip is automatically projected in depth towards the closest point on the target object mesh. This process can be thought of as simply pushing the rectangle towards the object mesh until it collides with the object (Figure 21).

[2] A plane defined slightly in front of the camera and perpendicular to the vector defined from the position of the camera- the viewpoint - toward a reference point in the scene (i.e. the viewing direction).

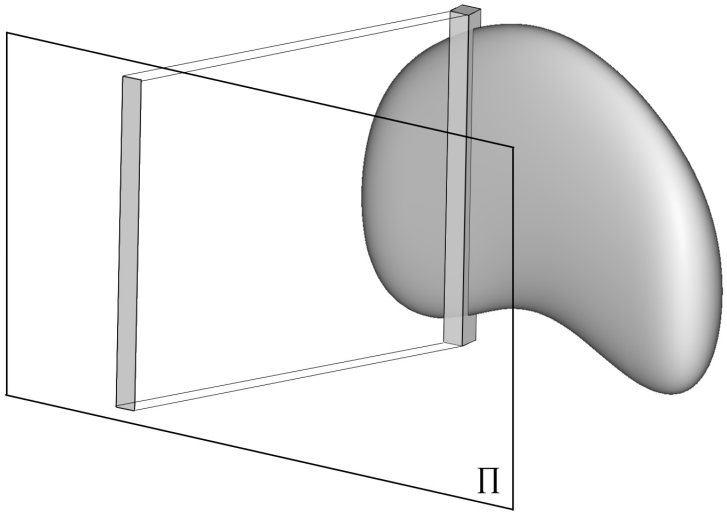


Figure 21 – Positioning of the initial cutter strip in 3D from the 2D initialization in the viewing plane ∏ by projecting ("pushing") the strip onto the target object.

To detect the collision (i.e. intersection) between the initial strip and the target object surface (and hence to find the closest point on the object surface), collision detection algorithms are needed. In the ExtrusionCutter system, all collision detection algorithms are performed using a VTK library called vtkCollisionDetectionFilter [20], which utilizes OBBTree data structures to quickly compute a list of contact points where two meshes intersect. Therefore, in order to find the closest point (C) on the object mesh directly underneath the user-created rectangle strip, a projection of the strip is performed by extruding it away from the camera along the view plane normal (i.e. viewing direction) to form a deep rectangular box (Figure 22).

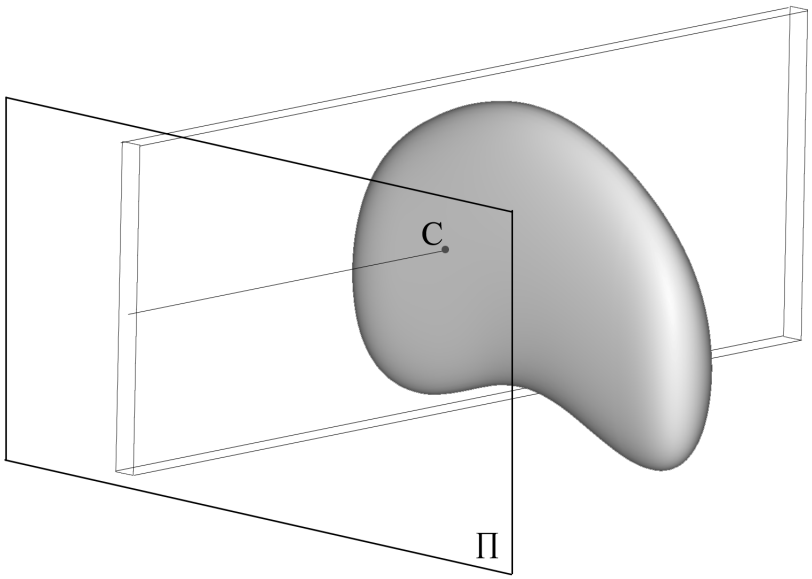


Figure 22 – Finding the closest point (C) on the target mesh by extruding the initial rectangle and computing the collision points.

The vtkCollisionDetectionFilter is used to compute a small set of intersection points. A distance calculation is then performed for every intersection point in the set to determine which of these points is closest to the view plane (∏). Finally, the view plane normal is used to translate the four points that constitute the original user-defined rectangle strip until the strip reaches the closest point (C) on the mesh (i.e. AEHD – Figure 23).

As a final step to conclude the initialization of the cutter, an initial rectangular box is constructed (Figure 25) by copying the rectangle strip and using the view plane normal to translate this second rectangle (BFGC in Figure 27 - the back face of the box) a distance of 5δ further than the first rectangle (AEHD in Figure 27 – the front face of the box).

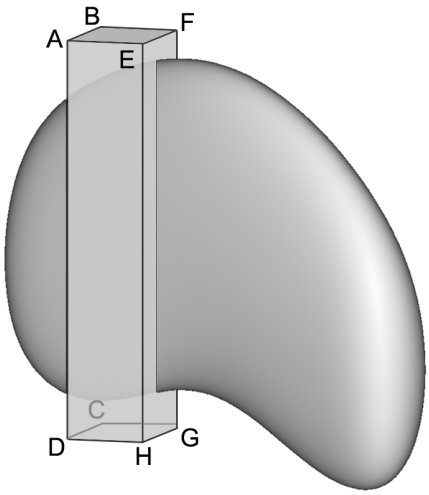


Figure 23 – Bounded cutting planes ABCD and EFGH in initial cutting region (NOTE: initial cutter size increased for illustrative purposes)

These two rectangles are connected to form the initial box. Furthermore, the left side rectangle ABCD (the left face) and right side EFGH (the right face) are explicitly stored as bounded cutting planes 0 and 1 respectively.

3.3.2 Extruding a Cutter

In many cutaway algorithms, the ability to easily, efficiently and effectively construct, define and modify the cutaway region to obtain the optimal view of the interior is often inadequate. The contention of this thesis is that this is primarily due to the choice of interaction model (see Chapter 4 for a discussion of interaction models). Therefore, the simple and intuitive click-drag-extend style input action used to initialize the cutter mesh also forms the basis of the robust extrusion-based interaction model used to expand or grow the cutter mesh (and thereby define the cutaway region). As mentioned in the thesis introduction, the extrusion-based interface

employs a “Paint Roller” interaction *metaphor;* the user extrudes (grows) the cutter in the same way that one would pull a paint roller along a smooth surface (Figure 24).

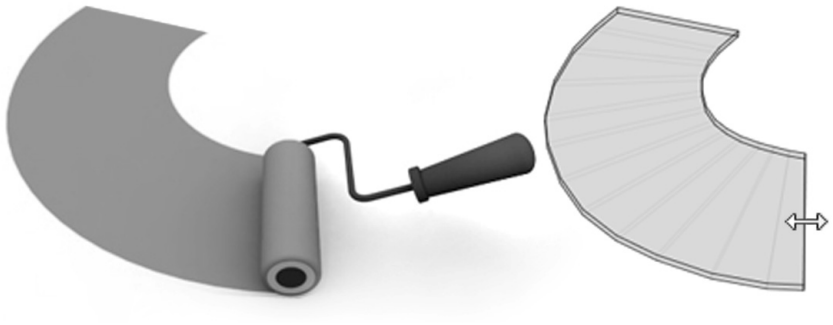
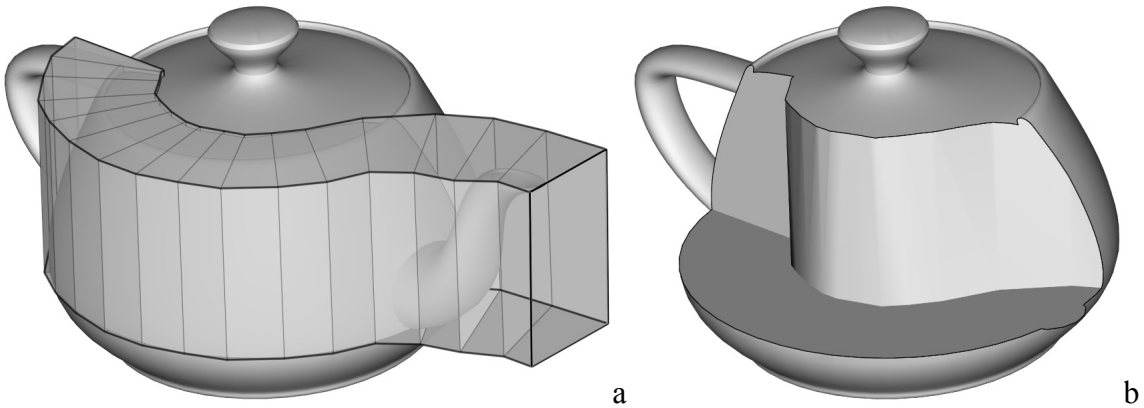


Figure 24 – “Paint Roller” extrusion-based interaction metaphor

Clicking on the edge of the initial cutter and sliding the cursor along the object surface is analogous to grabbing the paint roller “handle” and sweeping it along a curving path on a wall or other smooth surface. Much like a real paint roller, dragging the cutter in this fashion allows the user to create a wide swath of adjoining quadrilaterals along a curving path. Like the roller and handle, the right edge of the lead quadrilateral is constrained to remain orthogonal to the direction of travel defined by the cursor (Figure 24 right). As a result, the extrusion path can be made to coincide with medial axes, ridge lines, or other curving feature lines that accurately reflect the natural geometry of the target object. Furthermore, the user also has the option to stop at any time and alter the shape, size, orientation and number of bounded cutting planes defining the cutting region during the extrusion process. This feature presents a significant advantage over other interaction models described in the literature; immediate visual feedback is given to the user as the cutaway region (via the cutter) is created. The following sections will outline how the paint roller interaction metaphor is both implemented and extended in ExtrusionCutter to produce a wide range of cutaway regions.

3.3.2.1 Constructing Curving Path Cuts

Curving path cuts are essentially the concrete implementation of the paint-roller interaction metaphor. By extruding the initial cutter along a curving path on the surface of the target object, curving cuts can be made along any portion of the target object (Figure 25a-b).



a b

Figure 25 – Curving Path Cut: (a) Cutter Mesh (b) Cutaway View

In order to facilitate the fast and simple creation of such a complex cutaway region, a number of important algorithms were developed. For example, before the extrusion process begins, the system needs to know that the user does indeed wish to extrude the cutter, rather than simply edit one of the existing bounded cutting planes. This is determined by looking at which part of the cutter the cursor is hovering over before it is clicked (further explained in Section 3.3.3). Essentially, if the cursor is positioned somewhere above the middle third of the rightmost bounded cutting plane (i.e. the right face of the leading box), the system assumes the user is trying to grab the cutter "handle" to begin the extrusion process. Once the user presses and holds the left mouse button, the system automatically repositions the cursor directly above the middle of the rightmost bounded cutting plane's front edge. Moving the cursor to this position results in more predictable behaviour (i.e. more like a real paint roller).

While holding down the left mouse button, the user traces a path over the target mesh and the system extrudes the cutter mesh to follow this path (Figure 26a-c).

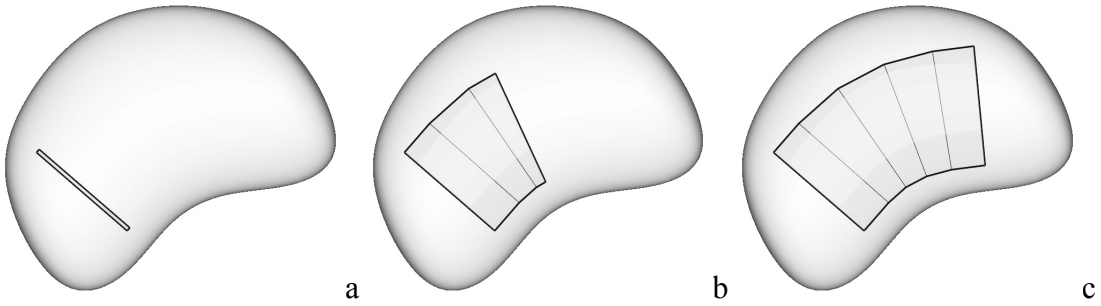


Figure 26 – Extrusion of the cutting region along a curving path.

The system extrudes the cutter such that the right face of the leading box (the lead bounded cutting plane) is always constrained to be orthogonal to the user-defined path. The leading box is expanded or widened until a user-defined width is reached. At this point, a new box is constructed and added to the cutter mesh. Additional constraints are used to ensure each box is well-formed and no local self-intersection occurs with the previous box. This extrusion process happens quickly and seamlessly, with the set of constraints providing the smooth paint roller "feel" to the cutter.

As mentioned, the right face of the leading box must be continually repositioned in order to maintain orthogonality with the user-defined path. This constraint is implemented as follows. The first step is to determine the mouse location in 3D space. This is done by projecting the location of the mouse cursor in the view plane (∏) onto the plane defined by vectors AE and AD (Figure 27a) using the view plane normal. This can simply be thought of as pushing the mouse point away from the screen until it lies on the same plane as quadrilateral AEHD (i.e. the front face of the lead box) – hereafter referred to as the "front plane".

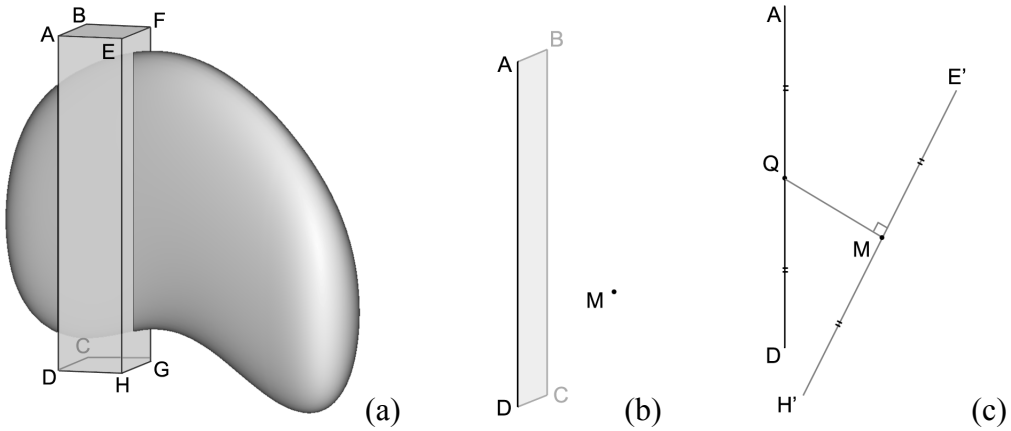


Figure 27 – Steps a through c of repositioning the leading bounded cutting plane during extrusion to follow the mouse point M

Once this point (M – Figure 27b) has been defined, it is used as the basis to construct a new rightmost front-face edge E'H'. The endpoints E' and H' are determined by first connecting the mouse point M with the midpoint of edge AD (point Q in Figure 27c). Vector MQ is an approximation to the path defined by the mouse and lies in the front plane. This vector is rotated 90 degrees in both the clockwise and counter clockwise directions in the front plane to form two new vectors: ME' and MH' respectively. Next, ME' and MH' are scaled to the same magnitude as vectors QA and QD respectively to ensure that the new right edge E'H' of the front face is the same magnitude as the left edge AD.

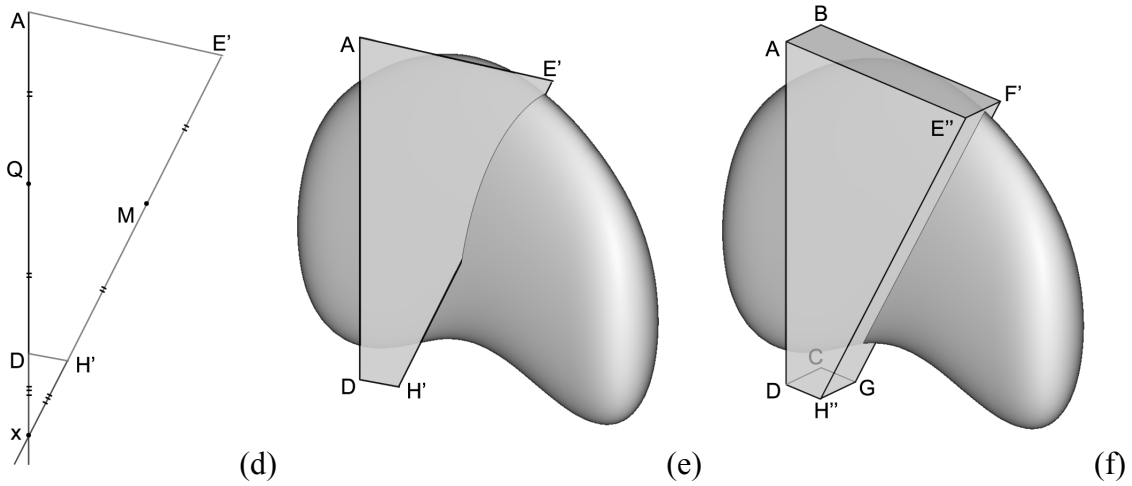


Figure 28 – Steps d through f of repositioning the leading bounded cutting plane during extrusion

A correction process is then invoked to guarantee that the top and bottom edges of the front face, vectors AE' and DH' respectively, will be parallel. This constraint ensures boxes remain well-formed and smoothes the transition between one box and the next, thereby counteracting noisy user input due to hand jitter. The correction is accomplished by extending the vectors AD and E'H' to find their point of intersection (point x – Figure 28d). The magnitude of the vector xD is then used to translate H' and E' an equal distance along the vector xM. Fundamentally, this process creates an isosceles triangle xAE' which guarantees that DH' is parallel to AE' (Figure 28).

Unfortunately, since this process works entirely in the front plane, the resultant front face AE'H'D may not be sitting directly on the target object surface (Figure 28e). For example, there may be a bump or ridge in the object surface directly adjacent to the initial cutter region and the act of extending the initial front face (AEHD) may have caused it to intersect the bump or ridge. Similarly, there may be a valley in the object surface and the new front face (AE'H'D) is now sitting too far above the surface. To resolve these problems and ensure that AE'H'D is situated correctly on the object surface, an iterative face rotation algorithm is employed (Figure 28e-f).

Each iteration of the algorithm rotates the points E' and H' (i.e. effectively rotating the face AE''H''D) clockwise (outward) about the vector DA in order to displace them by a distance of 2(δ) relative to their starting location. After each rotation, the new front face (AE''H''D) is then fed into the vtkCollisionDetectionFilter to determine if it still intersects the object surface. If it still intersects, another iteration is performed. Similarly, if the front face is too far above the object surface, the algorithm is again performed with the points E'' and H'' rotated counter clockwise (inward) iteratively until an intersection occurs. Once an intersection has occurred, they are then rotated clockwise (outward) one last time to displace them by a distance of 2(δ) above from the target mesh (Figure 28f).

Now that the front face has been rotated to ensure that it is sitting properly on the object surface, the new right-face back edge points G' and F' can be positioned to create the final repositioned bounded cutting plane (E''F'G''H). This is accomplished by translating them a constant distance along the negative normal of the newly rotated front face (i.e. AE'' × AD) to ensure that edges E''F' and H''G' are at right angles to it (Figure 28f). As a final step, the previous bounded cutting plane (ABCD) is joined up to the newly constructed bounded cutting plane (E''F'G''H).

In order to create fully curving cuts however, this entire box expansion and repositioning process is continually executed as the user drags the mouse along the object surface. The process is invisible to the user – the user sees only a smooth expansion of the lead box of the cutter that sticks closely to the object surface.

Once the lead box reaches a user-defined width, the lead box is locked into place and a new box is constructed, with a very narrow initial width. This new lead box is added to the cutter mesh. The frequency at which this happens (ω) is controlled via a simple slider bar located

within the GUI of the ExtrusionCutter system (Section 3.5 - User Interface Specification). For example, if the user moves the slider to a setting of "20", then each time the lead box width a magnitude of 20(δ), a new box is created. Once again, this process is automatic – the user sees only the smooth expansion and creation of new boxes as they drag the mouse over the object surface. This allows the user the flexibility to create detailed, highly curved regions or large, sweeping volumetric regions by altering only a single value (ω) with a slider.

3.3.2.2 Constructing Straight Path Cuts

While curving path cuts offer a unique opportunity to create complex cutaway views, the user may also want to use the same interaction metaphor to construct a more volumetric cut, such as a box (Figure 29).

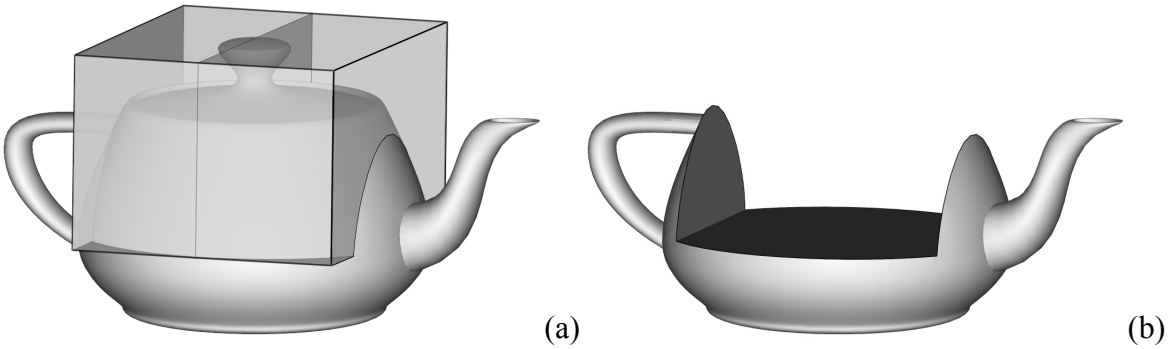


Figure 29 – Straight Path Cut: (a) Cutter Mesh (b) Cutaway View

In this case, ExtrusionCutter allows the user to enforce/disable a straight-line path constraint at any point during the extrusion process by simply holding/releasing the "shift" key. Recall that in order to extrude the mesh, the mouse point is projected onto the "front plane" and the vector MQ is constructed (Figure 30a). However, if the shift key is currently down, the M point is rotated

such that the MQ vector is orthogonal to the AD vector (Figure 30b) before the rest of the bounded cutting plane extrusion algorithm (Figure 28) is executed.

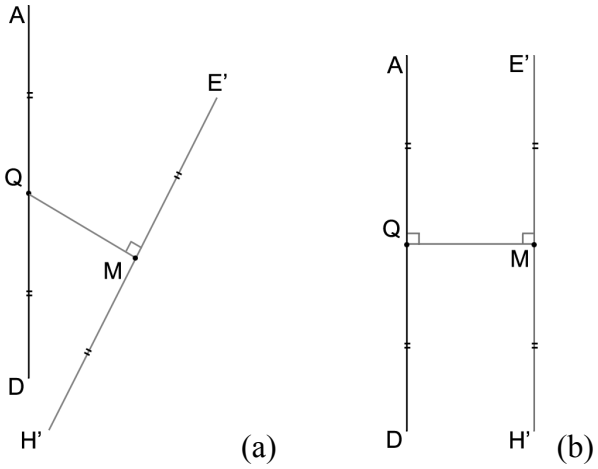


Figure 30 – Implementation of the horizontal restraint during cutting region extrusion

By dynamically repositioning the MQ vector in this way, the user is able to extrude out a perfectly 3D straight line path without painstakingly guiding the mouse across the target object. If this constraint is then enforced while a large (ω) (frequency of new boxes added during extrusion) value is selected, a "box" cut (Figure 29) can be created using the same extrusion-based interaction metaphor as the curving cut.

3.3.2.3 Constructing Wedge Cuts

Many anatomical structures are disc-shaped or tubular, with a well-defined single primary medial axis. As such, the user may wish to create a wedge-shaped volumetric cut (Figure 31a,b), instead of a box cut or complex curving cut. This is also possible in the ExtrusionCutter system through another simple alteration to the basic extrusion algorithm.

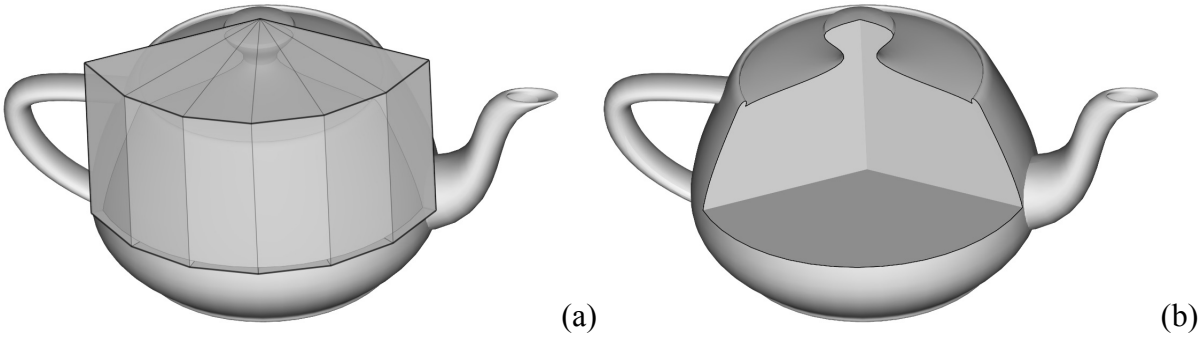


Figure 31 – Wedge Cut: (a) Cutter Mesh (b) Cutaway View

If the user clicks the "Circle" button on the ExtrusionCutter GUI (Section 3.5 - User Interface Specification) before initializing a new cut, they effectively drag out an initial triangle strip instead of an initial rectangle strip in the view plane (∏). That is, the cutter initialization process is fundamentally identical to dragging out a rectangle, except that the resulting two projected bounded cutting planes ABCD and EFGH are joined at the top (Figure 32a,b) to form a prism-shaped cutting region rather than a rectangular box.

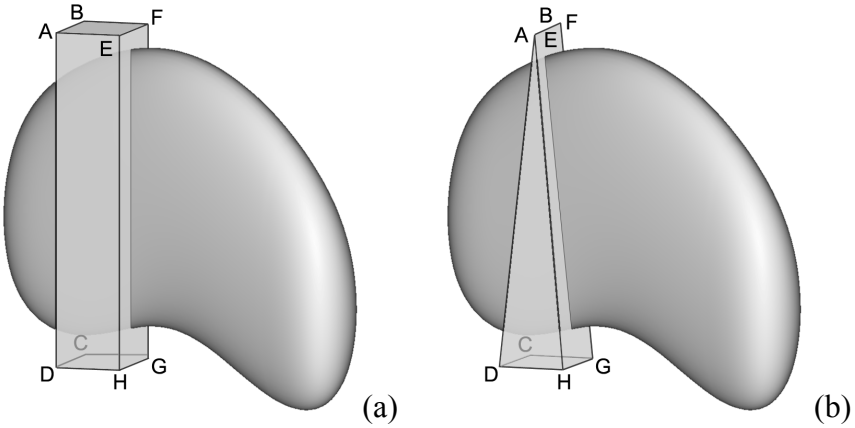


Figure 32 – Initial bounded cutting plane orientations and initial polyhedron used to construct: (a) curving/straight path cuts (b) wedge cuts

In effect, the vector defining the top of the prism (either AB or EF) also defines a single edge shared by all prisms of the entire 3D wedge-shaped cutting region. This edge also acts as a

rotation axis. This simplifies the extrusion process as the projected cursor point (M) no longer defines the center point of E'H', but rather defines how far E'H' needs to be rotated about the center edge (Figure 33).

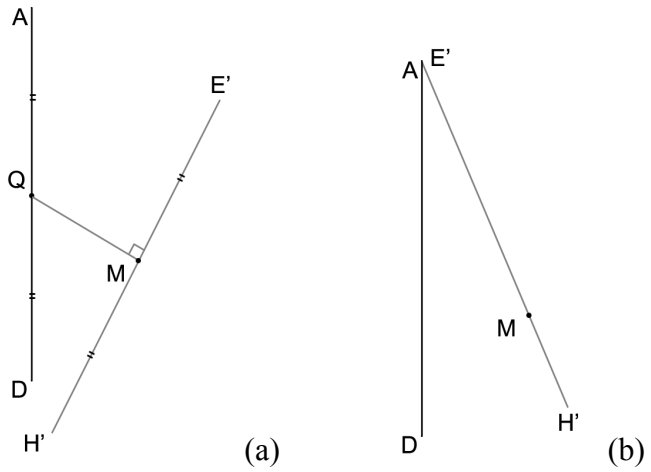


Figure 33 – Implementation of the circular restraint during wedge cut region extrusion

Therefore, point E' is constrained to be at the same spot as A, while the position of H' is determined by scaling the vector E'M such that it is equal in length to the magnitude of vector AD. If a collision is detected however, rather than rotating the front face about the vector DA, the entire cutter mesh is extruded toward away from the object surface by a factor of 2(δ) along the vector BA. This is done to ensure that the wedge remains a consistent "pie" shape throughout the extrusion process despite noisy user input, or complicated target object geometry.

3.3.2.4 Constructing Rounded Cuts

Certain anatomical structures may need to be cut using a curving or "rounded" cutter back so that internal structures are entirely exposed without cutting all the way through the

occluding anatomical structure. In this case, ExtrusionCutter can once again be extended to create such cuts (Figure 34a,b).

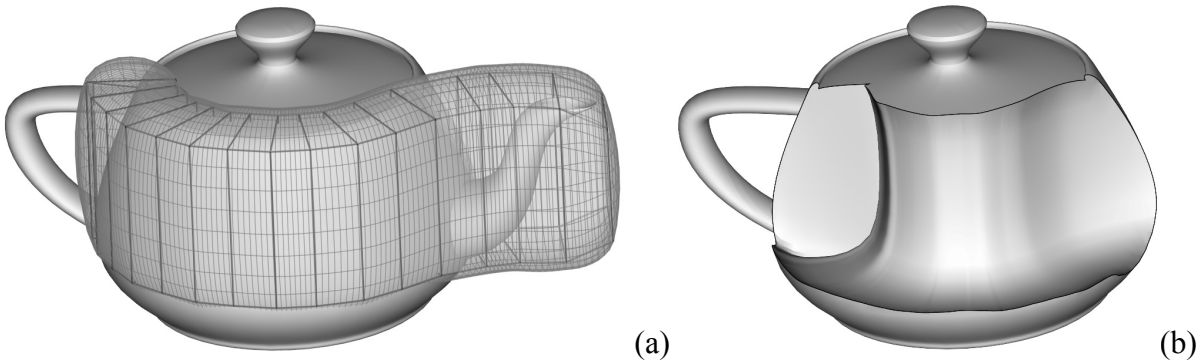


Figure 34 –Rounded Cut: (a) Cutter Mesh (b) "Rounded" Cutaway View

By simply pressing a button on the ExtrusionCutter GUI (Section 3.5 - User Interface Specification) the user can generate a smooth, more rounded cutter mesh. This smoothing and rounding process is accomplished through the use of an interpolating subdivision surface algorithm [33] , designed specifically for quadrilateral meshes. Since the entire cutter mesh used in ExtrusionCutter is represented as a mesh of quadrilaterals, it can be input into the subdivision surface algorithm [33] to generate a smoother, more rounded cutter mesh that still passes through (interpolates) the vertices of the original cutter mesh. This interpolation property allows the user to have full control over a smooth subdivided cutter mesh by manipulating the original bounded cutting planes as usual (Section 3.3.3).

3.3.2.5 Constructing Freeform Cuts

Shell-like anatomical structures can also be cut using the smooth subdivided cutter mesh (Figure 34)

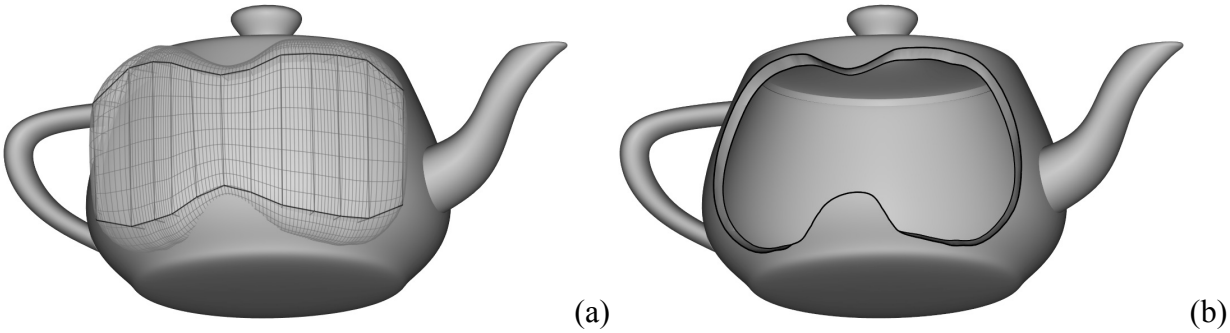


Figure 35 – Freeform Cut in a hollow Utah Teapot dataset: (a) Cutter Mesh (b) "Freeform" Cutaway View

By extruding the cutter and manipulating the cutting planes in the usual manner, the user is able to define smooth windows in the occluding surface along a curving path. Since the boundary of the window is controlled by a subdivision surface (Section 3.3.2.4), the curves are easily edited without the need to redefine the cutting region or explicitly retrace the curve on the surface.

3.3.3 Editing the Cutter

The previous section has demonstrated that through the use of a common, simple interaction metaphor, the user is able to create flexable, complex-shaped cutters – and hence complex cutaways of 3D surface meshes. However, while the initial generation of such cutaway views is vitally important, is equally important that the user can effectively edit them.

Recall that the cutter can be thought of as: 1) a quadrilateral mesh, 2) a series of connected "boxes", or 3) a series of bounded cutting planes regularly positioned along user-defined path. Furthermore, since the user can view the bounded cutting planes during the extrusion process (i.e. the cutter is typically rendered as semi-transparent), it is natural to allow the user to "reach out" and reposition them as a natural part of the ExtrusionCutter interaction metaphor.

The user is able to stop extruding at any time and use the mouse to select a specific cutting plane (Figure 36). Once selected, the user can perform the following constrained editing operations:

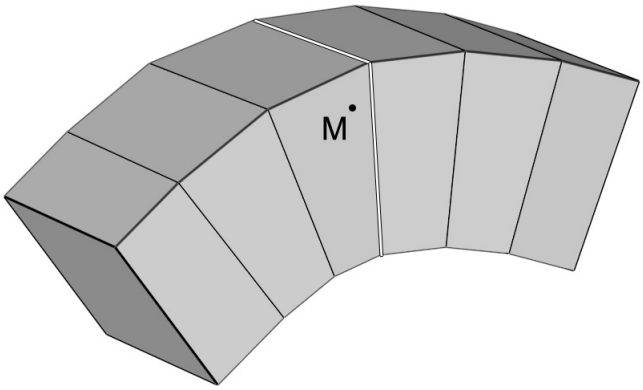


Figure 36 – Highlighted bounded cutting plane as a result of its proximity to user-picked point M

Repositioning Cutter Edges

One of the most basic edits that can be performed on a selected bounded cutting plane is to click-and-drag one of its front vertices and change the location of the edge defined by this front vertex and the back vertex. Only edges can be repositioned (rather than individual vertices) to prevent ill-formed boxes and non-planar cutting planes. The user can drag the edge around such that the front vertex is always constrained to be on the plane defined with a normal vector calculated by subtracting the two vertices (Figure 37).

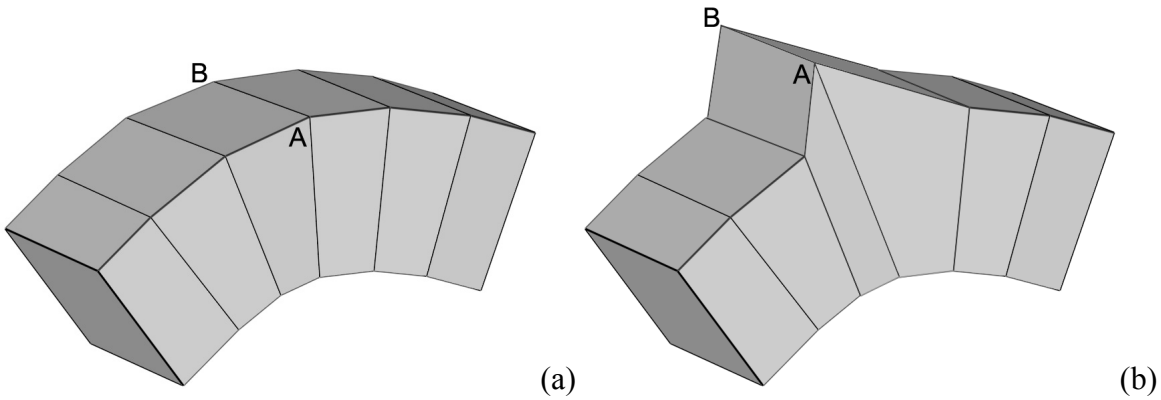


Figure 37 – Repositioning the top edge in the 4th bounding plane by dragging point A along the plane defined by vector AB

Increasing the Depth of a Bounded Cutting Plane(s)

The initial depth of the cutter is quite thin - a constant 5δ. To increase the cutter thickness (and hence make a deeper cut into the target object), the user simply moves the cursor such that it is not over any part of the cutter and moves the mouse scroll wheel up. Conversely, moving the scroll wheel down decreases the cutter thickness.

When increasing the cutter thickness, it is possible that two or more bounded cutting planes may collide and intersect (Figure 38b).

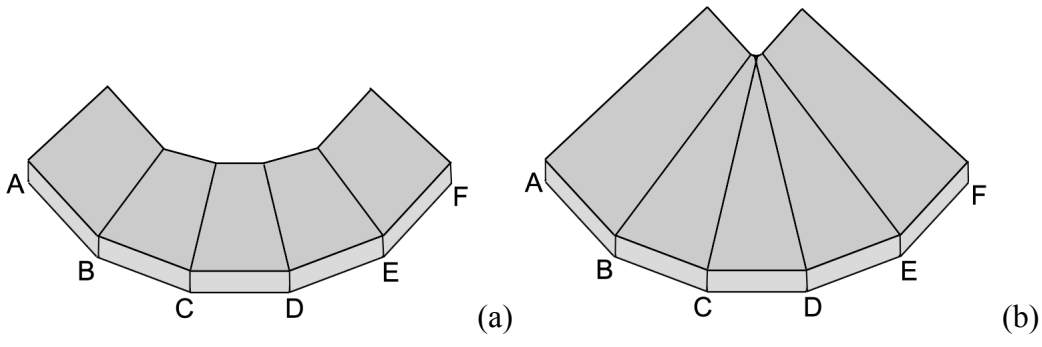


Figure 38 – Top view of a cutter thickness increase that results in a collision between bounded cutting planes

For example, in Figure 42b bounded cutting planes C and D have collided, resulting in a "self-intersection" in the cutter mesh. This self-intersection results in a degenerate cutter mesh which cannot be used by the GTS library to perform the Boolean subtraction operation. The self-intersection can resolved relatively simply through the use of an iterative algorithm. Essentially, after every incremental increase in depth, the system detects "groups" of intersected bounding planes by iteratively traversing through the cutter and performing intersection tests between each bounded cutting plane. For example, if plane 0 is intersecting planes 1 and 2 and plane 4 is intersecting planes 5 and 7 two groups will be identified (Group A [0,1,2] and Group B [4,5,6,7]). The "center point" of each group is then defined by the median of the list of bounded cutting planes belonging to each group (i.e the median of Group A is 1, while Group B is 5.5). Each bounded cutting plane belonging to each group is then rotated a single increment value about its front edge either: clockwise (if its index is above the median), counter clockwise (if its index is below the median) or not at all (if its index is equal to the median). Once each rotation has been executed for each identified group, the cutter is once again traversed and new groups are established. If however, the system does not detect any collisions then it can be said that the cutter is no longer self-intersecting and the algorithm is complete.

The user may also select a single bounded cutting plane and increase/decrease its depth with the mouse scroll wheel. This edit action is useful when the user wishes to cut more deeply or more shallow in some parts of the cutaway.

Rotating a Bounded cutting plane

While the system automatically rotates cutting planes as a means of resolving collisions, there may be situations where the user wishes to alter the orientation of a bounding plane manually (Figure 39).

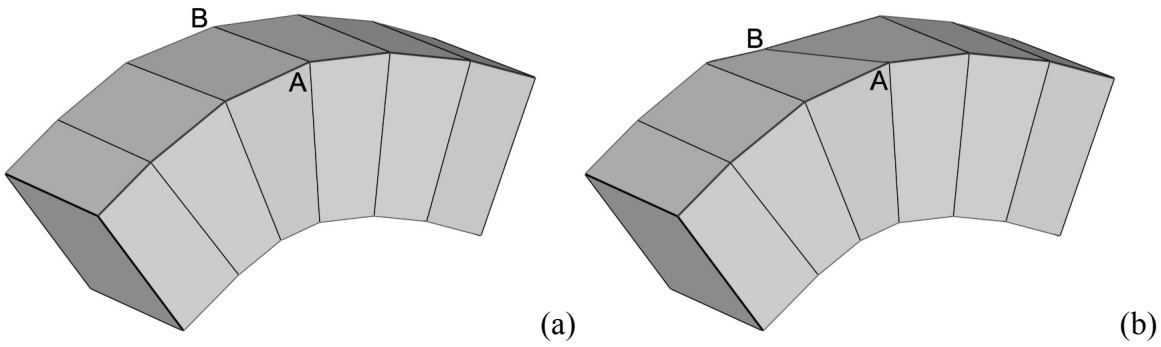


Figure 39 – Altering the angle of the selected bounded cutting plane by rotating it about the front edge

In this case, the scroll wheel is again employed with the additional requirement of having the "ctrl" button pressed as well. In this way, the system knows the difference between scrolling the wheel to manipulate the depth of the currently selected bounded cutting plane and scrolling the wheel to manipulate its angle. If the user does indeed wish to manipulate the angle, scrolling the wheel *up*/*down* has the effect of rotating the face clockwise/counter clockwise about the front edge (Figure 39a,b).

Removing a Box

Lastly, if the user has extruded the cutting region too far or in an incorrect direction, they have the option of sequentially removing the most recently added box of the cutter (Figure 40a,b).

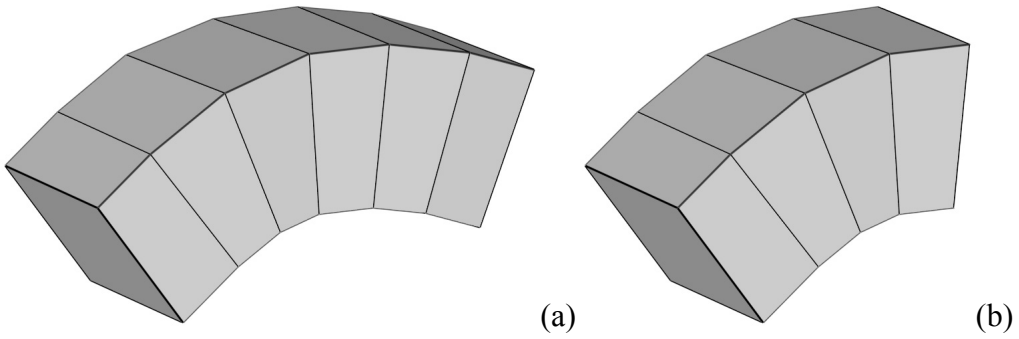


Figure 40 – (a) Complete cutter mesh (b) Cutter mesh with two bounded cutting planes removed

By simply pressing the delete key when the mouse is over any part of the cutter, the user is able to continually remove boxes one by one until only one remains – essentially resetting the cutter mesh to its initial state. The user is free to once again extrude the mesh in any direction along the geometry of the target object.

3.4 Rendering the Cutaway Region

The previous sections have demonstrated that through the use of a simple, consistent interaction metaphor, the user is able to create a variety of different cutters. These cutter meshes can then be used in a Boolean subtraction operation to generate a corresponding cutaway view.

However, unlike many cutaway tools [22] [19], ExtrusionCutter physically removes and subsequently re-triangulates the geometry of the target anatomical structure mesh affected by the cutting region. This can be seen as a "true cut" in that the geometry (triangles, edges, vertices) of the target structure is physically altered, as opposed to simply not rendered (Figure 41)

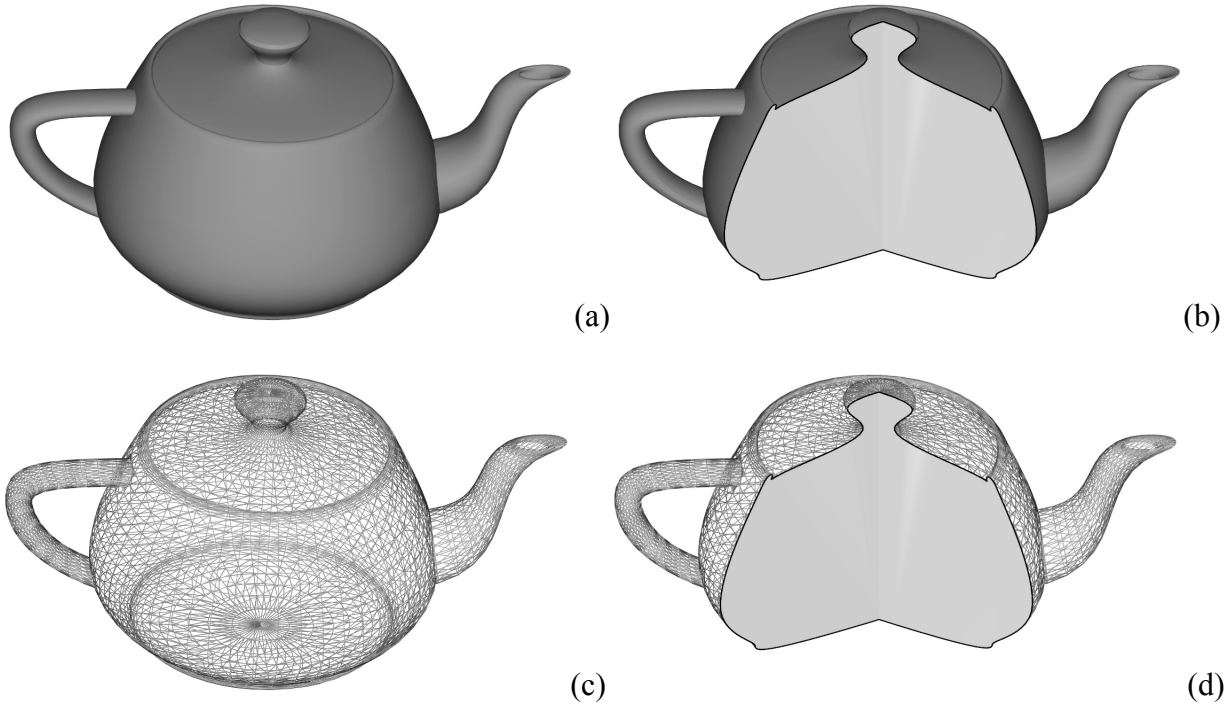


Figure 41 – Wireframe representation of a wedge cut in the Utah Teapot. The triangles of the cutaway region are physically removed by ExtrusionCutter and a triangulated "wall" mesh (gray-shaded region) is constructed and added to the remaining part of the teapot mesh.

Furthermore, ExtrusionCutter is also capable of computing, via GTS, the intersection surface of the cutter mesh and the target object mesh (Figure 41d). This intersection surface mesh can be rendered using a separate colour or lighting model to effectively highlight the cutaway view in the target object. Furthermore, as an additional highlighting measure, the collision detection algorithm provided by vtkCollisionDetectionFilter outputs an intersection contour that can be used to generate a thick boundary line encompassing the entire cutaway region (Figure 41b,d).

In order to perform such complex cutting/triangulation algorithms efficiently and robustly, the Gnu Triangulated Surface Library (GTS) has been wrapped with a simple VTK interface. Essentially, when the user is ready to render the cutaway view, they click "Preview" (Section 3.5 - User Interface Specification) and both meshes (the cutter and target object) are converted into "GtsSurfaces". An efficient software-based GTS-native boolean subtraction is then performed between both meshes to generate three additional GtsSurfaces (Figure 42).

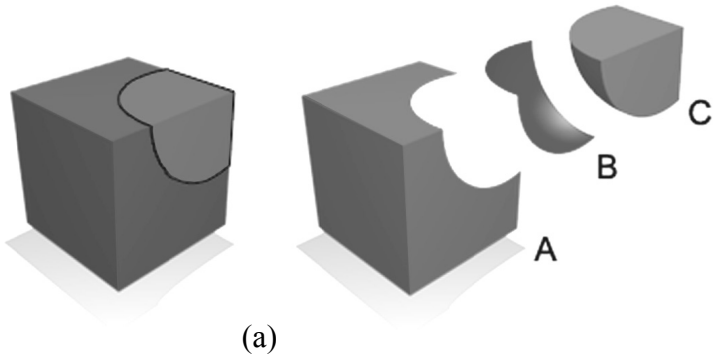
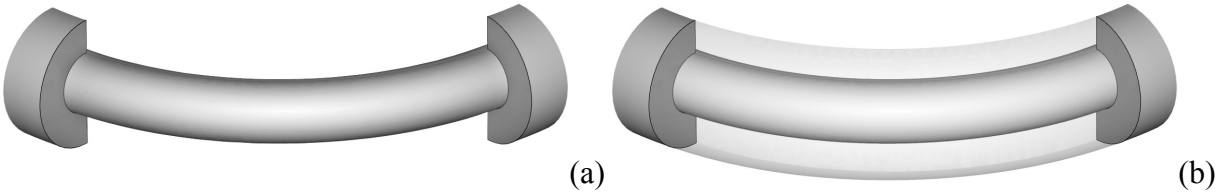


Figure 42 – Geometry computed by the Gnu Triangulated Surface Library (GTS) during boolean subtraction. (A) illustrates the cut object, (B) illustrates the intersection surface and (C) illustrates the cutaway region,

These include: re-triangulated geometry of the target object mesh that lies directly *outside* the cutter mesh (i.e. the target object mesh with the cutaway region removed) (A), the triangulated inner "walls" that represent the intersection surface mesh between the target object and the cutter (B), and the re-triangulated geometry of the target object mesh that lies directly *inside* the cutter mesh (i.e. the "removed" piece(s)) (C). These three separate GtsSurfaces are then explicitly converted back to VTK-native meshes (i.e. using class vtkPolyData) and rendered as separate actors on the stage. It is at this point that each actor can be rendered using a different colour or lighting/rendering technique.

3.4.1 Rendering Using Transparency

As previously stated, the Boolean operations provided by the GTS library are capable of explicitly outputting a mesh representation of the cutaway region. The cutaway region mesh can be rendered semi-transparently (Figure 43b).



(a) (b)

Figure 43 – Curving Cut in a Tube dataset: (a) without transparency (b) with transparency

By rendering the cutaway region as semi-transparent, the observer is able to see inside the target object while still retaining a contextual view of the entire structure.

3.4.2 Rendering Using Ribbons

Despite the contextual view provided by the semi-transparent rendering of the cutaway region, transparency alone is often an ineffective technique for conveying depth (Section 2.1) and can often generate noisy views if multiple layers of transparency are employed. Some opaque outline of the cutaway region is required to provide depth information. The ExtrusionCutter was designed to meet this requirement. Since the cutter in the ExtrusionCutter is represented as a series of connected bounded cutting planes, positioned along a single axis (i.e. the extrusion "path"), it is simple to construct a series of thin rectangular box meshes centered on each cutting plane, or centered at regular intervals along the extrusion path with a height and

depth interpolated from the cutting planes on either side. The width of these meshes, is under user control. This series of separate "sub-cutters" can be intersected with the target object mesh, resulting in a series of ribbon-like intersection surface meshes (Figure 44).

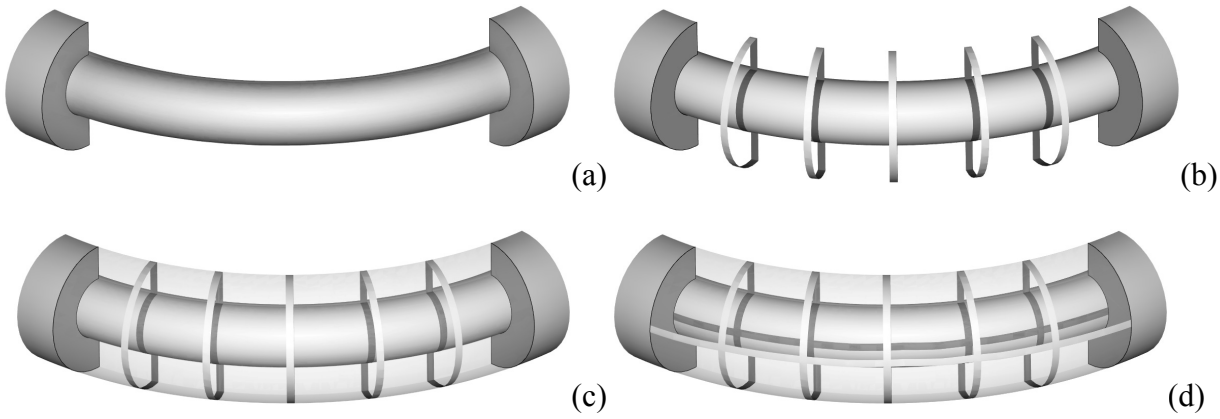


Figure 44 – Curving Cut in a Tube dataset: (a) without ribbons (b) with ribbons (c) with vertical ribbons and transparency (d) with vertical ribbons, a horizontal ribbon and transparency

For example, in Figure 44, five sub-cutters have been interpolated at regular intervals along the extrusion path (the center axis of the cutter). These sub-cutters are then intersected with the target object and a Boolean subtraction is calculated for each cutter. However, unlike the Boolean subtraction required to generate the initial cutaway view, the "inner walls" (i.e. surface B in Figure 42) are not computed. This ribbon outline of the cutaway region surface not only provides an effective contextual view of the cutaway front surface but also provides a contextual view of the cutaway bottom, back, and top surfaces.

3.4.3 Rendering Using Solid Vertical Slices

To further visually reinforce the depth of the geometry removed when generating a cutaway view, the user may wish to generate *solid* slices in place of ribbons (Figure 45).

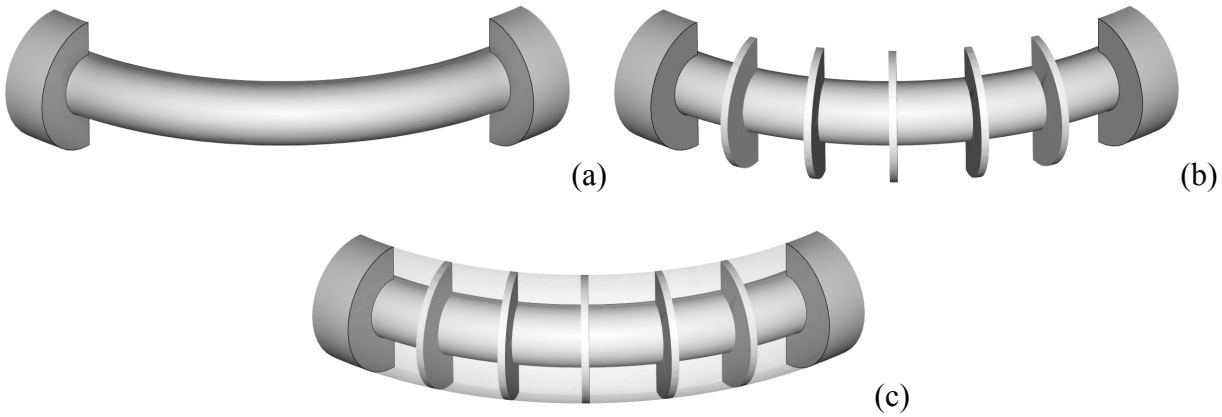


Figure 45– Curving Cut in a Tube dataset: (a) without solid vertical slices (b) with solid vertical slices (c) with solid vertical slices and transparency

Essentially, the logic to create this view is identical to that of the vertical ribbons, except the "inner walls" (i.e structure B – Figure 42) are also computed. By rendering the inner walls, the user is able to more readily estimate the depth of the removed geometry along the cutaway extrusion path and can more readily "fill in" the depth in between the slices. Furthermore, these walls will intersect any occluded structures, essentially creating a visual representation of the distance between the outer surface of the occluding target object and the inner occluded object.

3.5 User Interface Specification

The ExtrusionCutter system is built using a combination of Microsoft Visual C# and C++ programming languages to generate both the "front end" graphical user interface (GUI) and "back end" logic, respectively. The GUI consists of a set of components (buttons, sliders menus, etc.) for controlling various attributes of the visualization, such as the colours or rendering styles of the actors (anatomical structure meshes) in the scene. The following paragraphs describe the various GUI components:

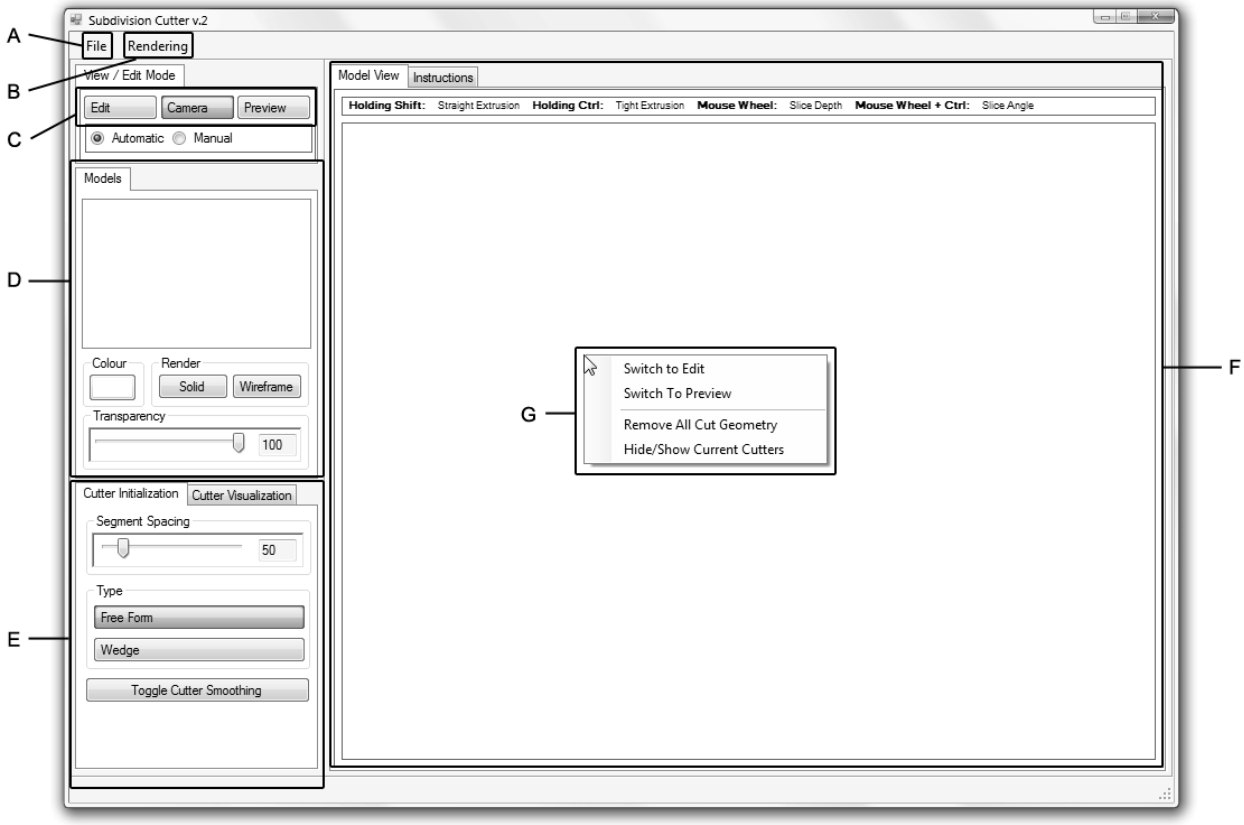


Figure 46 – Graphical User Interface employed by the ExtrusionCutter system

A – File Menu

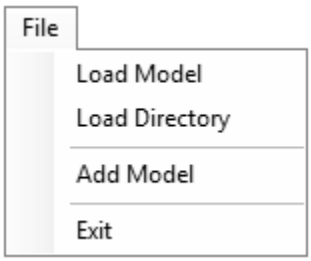


Figure 47 – File Menu

The File menu provides standard file input methods such as loading or adding an input mesh model or directory of models to the scene. To load or add a model, a standard "open" dialog is used with a default ".vtk" extension filter. If the model has been separated into separate files, the user has the option of simply browsing to the directory containing one or more .vtk files

using the “Browse For Folder” dialog. In this case, any “.vtk” files found in the chosen directory will be loaded automatically into the scene.

B – Rendering Menu

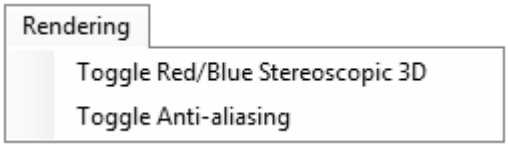


Figure 48 – Rendering Menu

As a small extension to the default rendering scheme, the user also has the option of toggling “Red/Blue Stereoscopic 3D” mode, as well as enabling/disabling anti-aliasing (smoothing) (Figure 48). This is typically only enabled when an image of the 3D rendering is saved, as anti- aliasing tends to slow down the rendering time.

C – Mode Buttons

Figure 49 – Mode Buttons

Since the mouse is used for both the construction/editing of cutters and camera manipulation (i.e. zoom, pan, rotate) multiple “modes” are necessary to determine the desired action (Figure 54). In camera mode, clicking and dragging the mouse has the effect of rotating or (with Shift held down) panning the camera in the viewport. Additionally, the mouse scroll wheel is used for zooming the camera in/out respectively. If “Edit” mode is selected, mouse input is interpreted as cutter initialization, extrusion, and editing actions (Section 3.3). Lastly, if the “Preview” mode is selected, the cuts are executed and the cutter mesh is hidden before the

cutaway views are rendered to the screen. In this mode, the mouse is put into camera mode and can be used to zoom, pan or rotate the camera. If the user wishes to modify any cutaways, they must switch back to Edit mode.

D – Models List

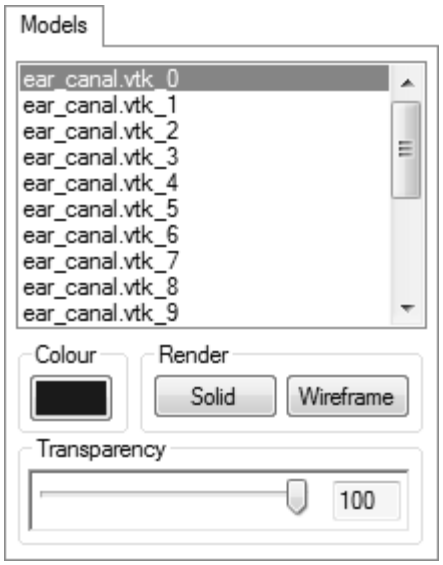


Figure 50 – Models list

Once a model has been loaded and its "regions" (parts) have been identified and given a random colour (Section 3.1), a text representation of each region or model is added to the master "Models" list (Figure 50). The naming convention used is simply the name of the .vtk file loaded with a number appended at the end to specify a given region. In this case, the ear_canal.vtk mesh model has been loaded and a series of regions (sub-meshes) have been identified. Upon selecting a given region in the list, the "Colour" button changes to match its working colour within the scene. If the button is pressed, a standard colour picker dialog is evoked and set to match the working colour (Figure 51).

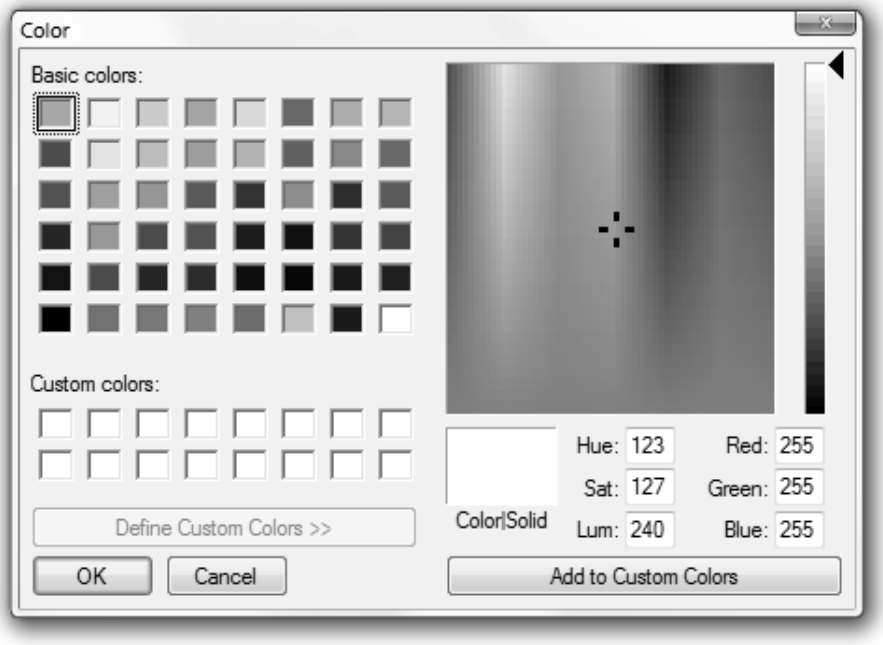


Figure 51 – Standard colour picker

The user also has the option of rendering the selected region using either a solid or wireframe representation by simply toggling one of the two "Render" buttons (Figure 50). Lastly, the user has the option of modifying the opacity of the selected region through the use of a simple transparency slider.

E – Cutter Initialization / Visualization

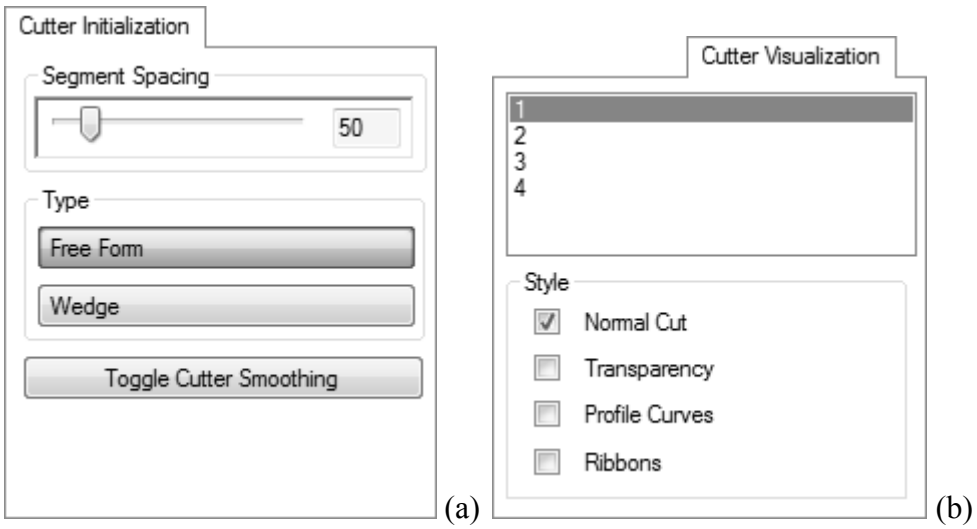


Figure 52 – Cutter Initlization (a) and Visualization (b) tabs

In order to specify cutter parameters, the "Cutter Initialization" tab is employed. The user is able to explicitly set the distance that a cutter box must be extended before a new box is created (i.e the "Segment Spacing"). Additionally, the user can switch from the default free form extrusion mode, to the wedge-constrained mode necessary for the construction of wedge cuts (Section 3.3.2.3) by toggling the "Type" buttons (Figure 52a). Lastly, the user can enable/disable the cutter smoothing option used in the creation of freeform cuts (Section 3.3.2.4)

In order to specify the type of context-preserving cutaway technique used in the final visualization (i.e. "Preview" mode), the user must explicitly check one or more of the techniques for each cutaway created. This is accomplished by simply selecting a cutter number from the list and checking one (or more) of: "Normal Cut" (Section 3.4), "Transparency" (Section 3.4.1), "Profile Curves" (Section 3.4.3), or "Ribbons" (Section 3.4.2). In this way, the user has full control over the combination of context-preserving views used to render the cutaway view.

F – Rendering Window

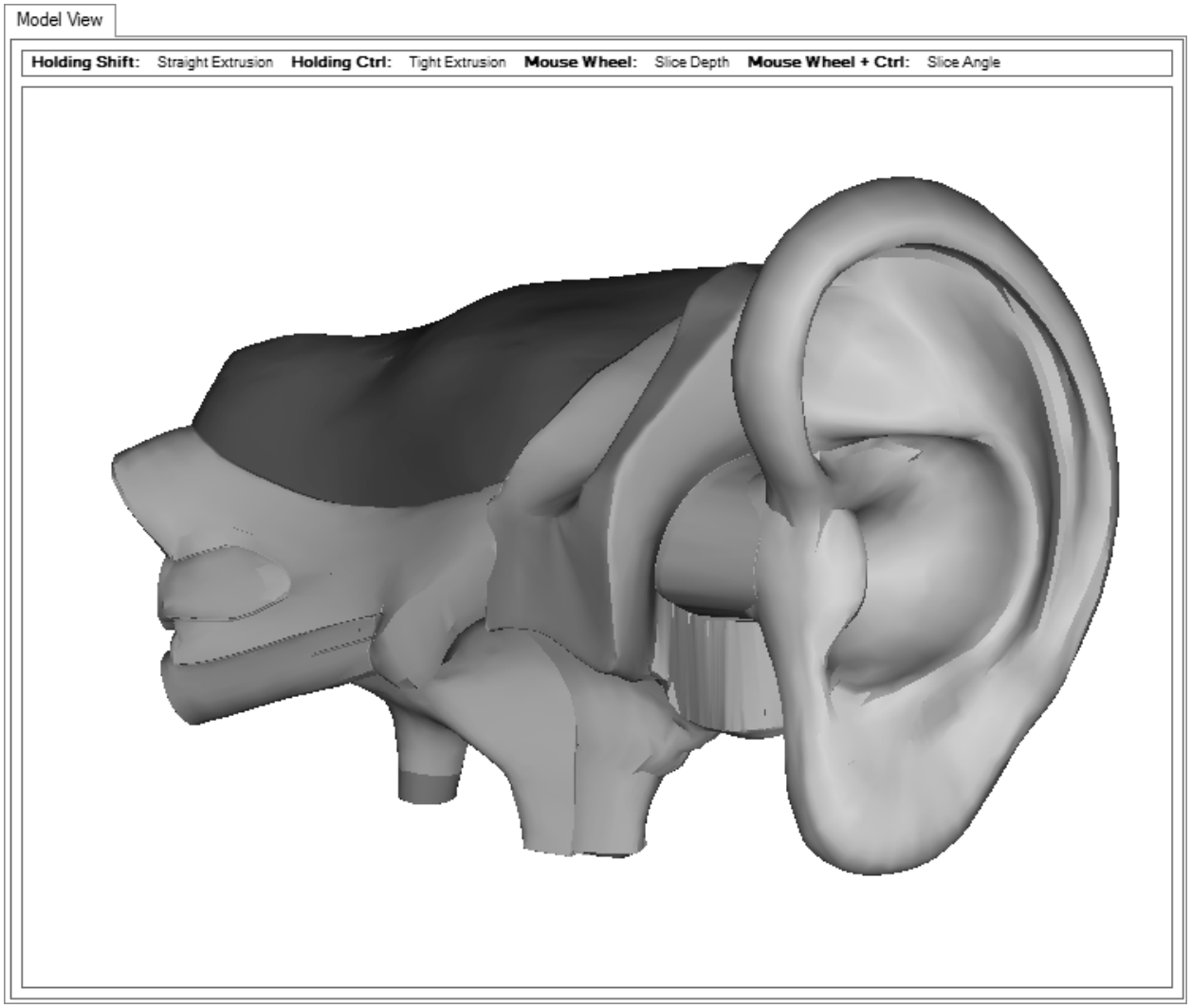


Figure 53 – VTK viewport

The model view window is used for all model interaction and visualization and as such is the primary focus of the interface. As a small reference key, the mouse/keyboard commands used in the editing of a cutaway region are displayed above the VTK viewport.

G – Context Menu

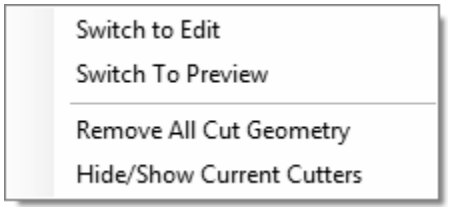


Figure 54 – Context menu

It may not always be efficient to move the mouse away from the visualization in order to click a button to switch modes. As a convenience, a simple context menu can be accessed at any time by clicking the right mouse button anywhere within the model render window (Figure 59). Two other operations are introduced in this menu to aid in the exploration of the visualization when in edit mode. For example, by clicking "Remove All Cut Geometry" the system executes simple cutaways for each cutter created. This allows the user to then create further cuts in objects that would not otherwise be visible. In a similar vein, the "Hide/Show Current Cutters" simply toggles the visibility of all cutters when in edit mode. This option allows users to create additional cutting regions without accidentally editing the wrong one.

Chapter IV – Results

At its core, the ExtrusionCutter system is the concrete implementation of a novel, region-selection interaction metaphor developed to facilitate the simple, fast generation of complex cutaways in 3D triangular mesh representations of anatomical structures. It has been shown that in order to create cutaway views in triangular mesh structures, well-formed cutters need to be generated and used as a means to encapsulate regions within a target mesh. Using the cutter tool must be intuitive, fatigue-free and require minimal labour as the primary users (radiologists, surgeons and other clinicians) are extraordinarily busy professionals with little time to spend on mastering complex visualization software interfaces. Additionally, in order to further reinforce the spatial relationships between occluding and occluded objects, the cutter must facilitate the construction of context-preserving views. For these reasons, the ExtrusionCutter system has been designed with the ability to construct five unique, editable cutaway types with a single, intuitive "paint roller" interaction metaphor. Additionally, the cutaway system has been implemented such that context-preserving rendering techniques are also possible. The following sections illustrate the "painting" ability of the interaction metaphor to generate cuts, the five basic cutaway types possible, and the context preserving rendering techniques supported, on a series of 3D triangular mesh structures.

4.1 Paint Roller Interaction Metaphor

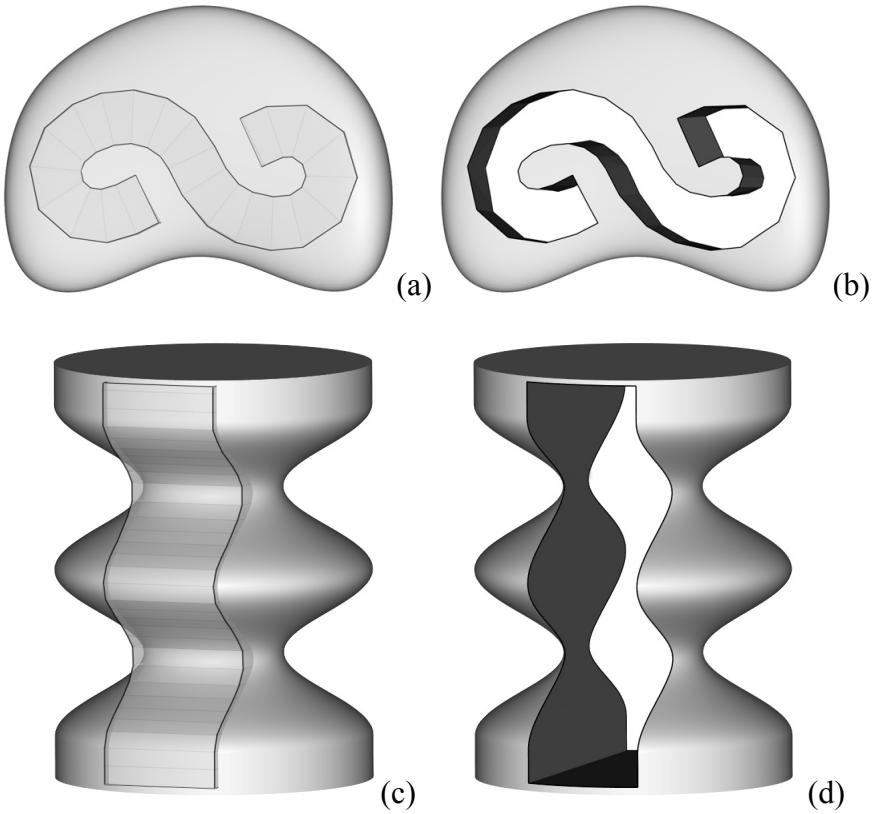


Figure 55 – (a),(b) Cutting mesh "painted" on a synthetic object by tracing an "S" shape with the mouse. (c),(d) Cutting mesh "painted" over an object with hills and valleys

It has been stated that the software realization of a paint roller interaction metaphor allows the cutter mesh to be constructed in a way that follows the natural geometry of the target structure. Two examples of this ability are presented in Figure 55. Figure 55(a,b) illustrates an "S" shaped cutter mesh that has been *painted* on the target structure by tracing a corresponding path with the mouse. Similarly, Figure 55(c,d) illustrates the ability of the cutter mesh to "stick" to the target structure over hills and valleys as it is "*painted"*. By mimicking, through a series of geometric constraints and algorithms, the curving and surface-sticky motion of a real paint roller, complex cutaway views can be constructed using simple mouse actions.

4.2 Cutaway Types Demonstrated

While the realization of the paint roller interaction metaphor naturally allows for the creation of curving cuts, it has also been demonstrated that through the use of simple constraints, straight path, wedge, rounded and freeform cuts are also possible. Since the creation of the cutter mesh employs a single interaction metaphor throughout the entire application (regardless of cutaway type) the user is not forced to learn additional creation/editing actions. To demonstrate the ability of the ExtrusionCutter system to generate complex cutaways in 3D triangular mesh representations of anatomical structures using this action, each of the above cut types have been performed on either a human skull/skin dataset, or a human spine dataset. The following sections illustrate both the "before" and "after" images of each cutaway type performed to expose a specific internal (occluded) region in a system of anatomical structures.

Curving Path Cut

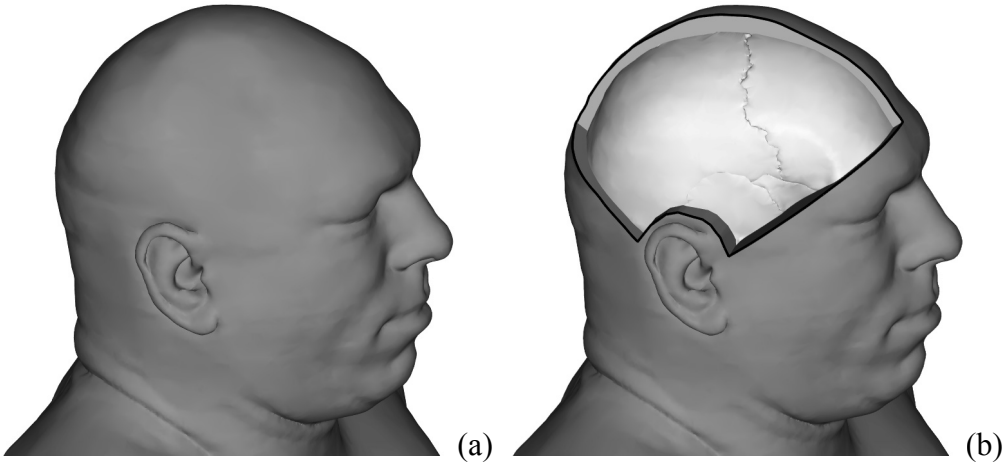


Figure 56 – Freeform cut exposing parts of the right parietal, frontal and temporal bones in the skull

A curving path cut (Section 3.3.2.1) has been performed on the outer skin of the upper right region of the skull to expose parts of the right parietal, frontal and temporal bones.

Straight Path Cut

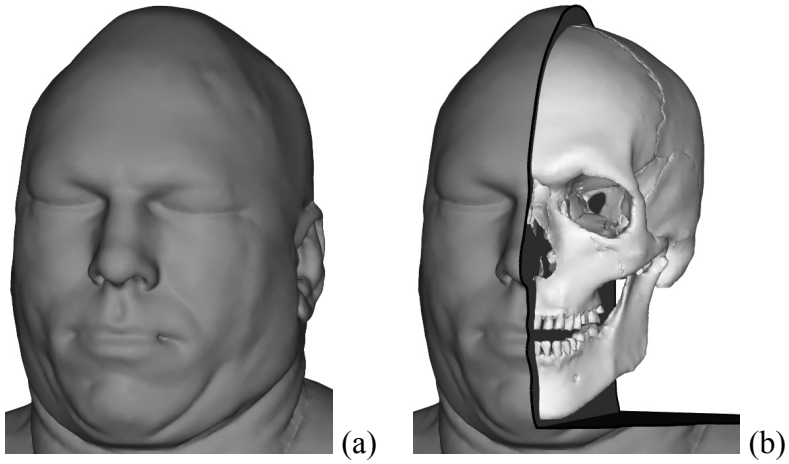


Figure 57 – Straight path cut exposing the left side of the skull

In this example, a straight path cut (Section 3.3.2.2) has been performed on the outer skin to expose the left half of the skull

Wedge Cut

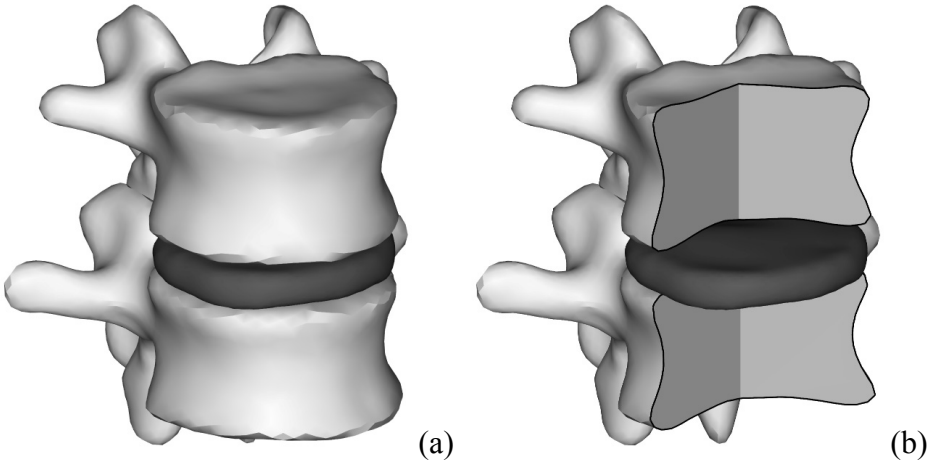


Figure 58 – Two wedge cuts in the upper and lower vertebra exposing the intervertebral disc

To illustrate wedge cuts (Section 3.3.2.3), two cuts have been performed on the surrounding vertebra in order to expose an intervertebral disc on a human spine dataset.

Rounded Cut

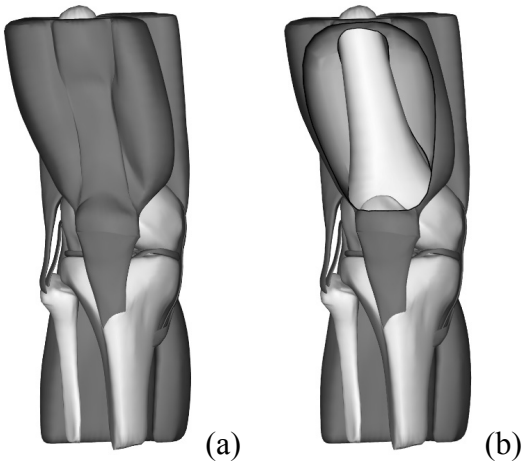


Figure 59 – Rounded cut in the quadriceps to expose the femur

A rounded cut (Section 3.3.2.4) has been performed over the front of the quadriceps to expose the femur.

Freeform Cut

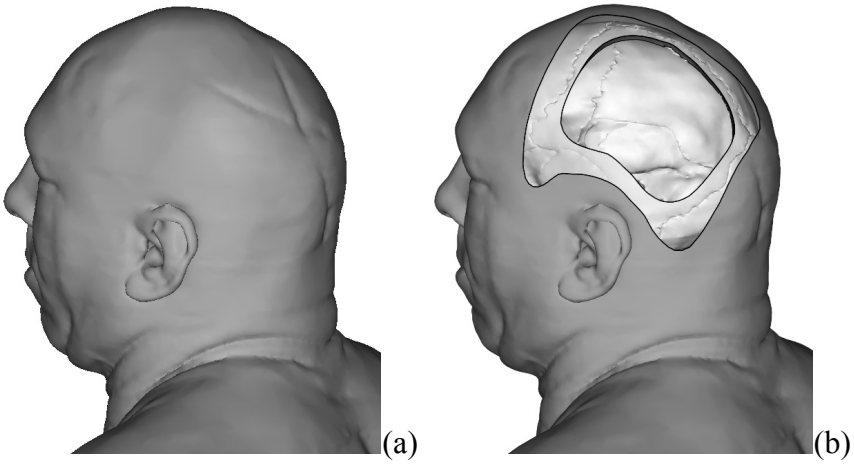

(a) (b)

Figure 60 – Freeform cuts exposing the inside of the skull beneath the left parietal bone

Lastly, two freeform cuts (Section 3.3.2.5) have been performed on both the outer skin and left parietal bone to expose the inside of the skull.

4.3 Context-Preserving Rendering – Visual Comparison

One of the key benefits of the extrusion-based interaction metaphor used in the ExtrusionCutter tool is the ability to position a curving cutter along the primary curvature lines of a target object. By constructing thin sub-cutters at regular intervals along the cutter length, the system can create effective visual cues of the removed surface, the depth of internal occluded structures with respect to the outer surface, and the spatial relationship between the back surface of the occluding structure and the front surface of the internal structure. These context-enhancing visual cues take the form of "ribbons" (polygonal strips of the occluding structure) and "slices" (polygonal mesh "disks" of the occluding structure. If these visual cues are also used in conjunction with a transparent representation of the cutaway surface, the viewer is able to more readily mentally reconstruct the shape of the removed material. The following cutaway views provide compelling visual evidence of the effectiveness of the ribbon and slice cues.

4.3.1 Knee Dataset – Context Preserving Rendering

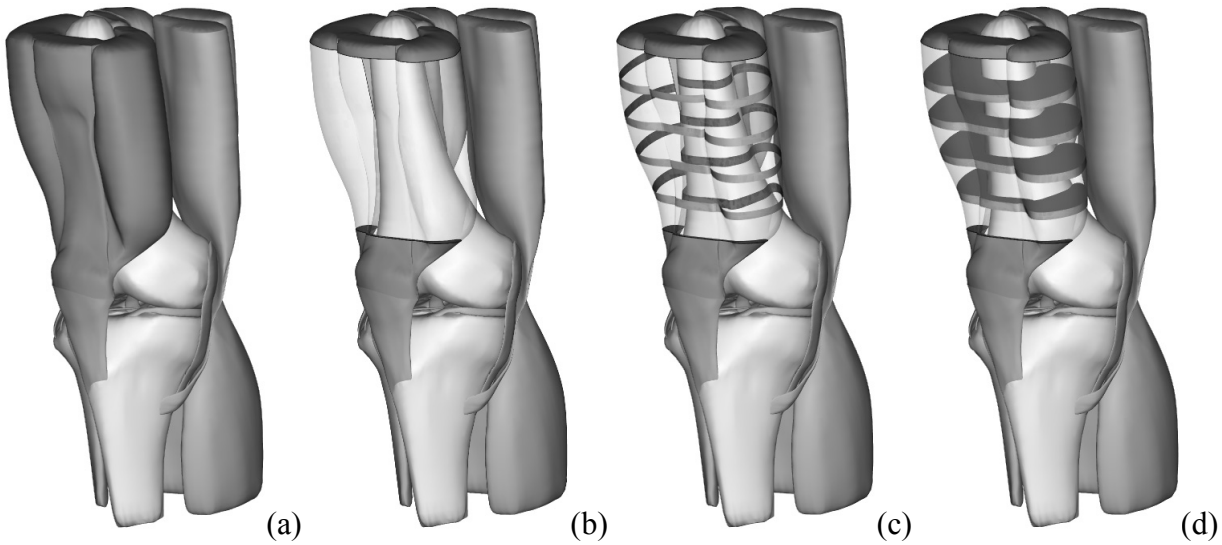


Figure 61 – Knee dataset cutaway rendered using: (b) simple transparency (c) context-preserving ribbons (c) context-preserving slices

A knee dataset (Figure 61) has been cut using a curving cutter extruded along the front of the quadriceps muscle. Notice the clear difference between employing transparency alone (Figure 61b) and employing transparency with ribbons or solid slices (Figure 61c, d). While the use of transparency does indeed provide some clue to the approximate shape of the removed geometry, it does not effectively illustrate the position of the quadriceps muscle relative to the femur. In addition, the curving shape of the back of the quadriceps (as it wraps around the femur) is lost completely. However, by wrapping ribbons around the quadriceps at regular intervals, their "U" shape is instantly recognizable as well as their position relative to the femur. Furthermore, by employing solid slices, the volume occupied by the quadriceps is easily perceived at a glance.

4.3.2 Mandible Dataset – Context Preserving Rendering

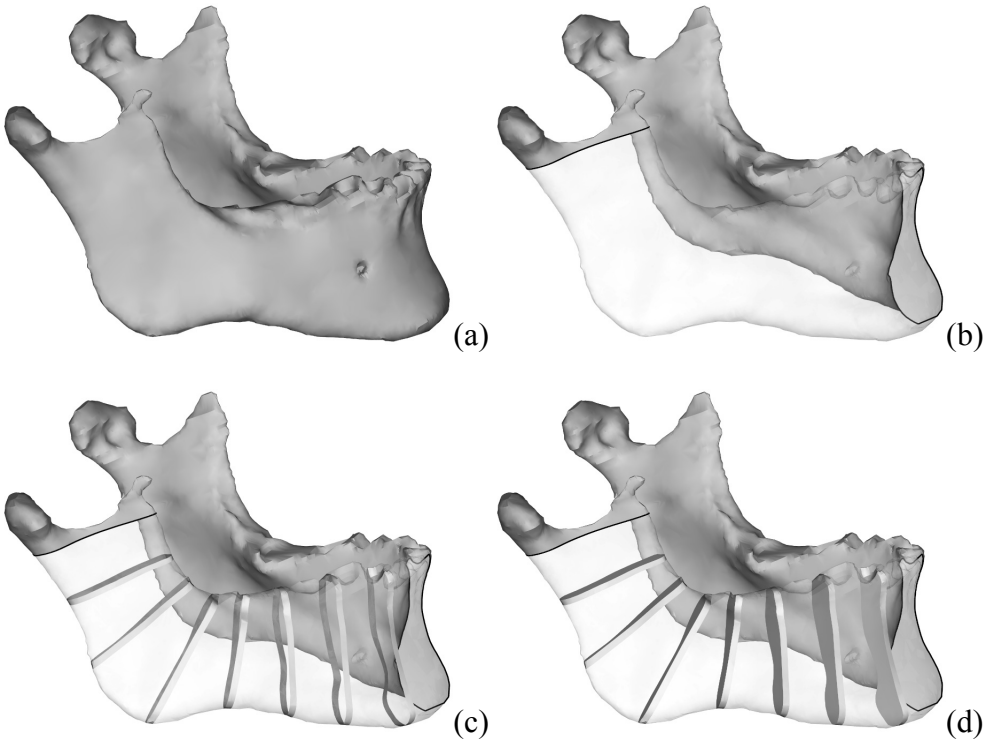


Figure 62 – Mandible dataset cutaway rendered using: (b) simple transparency (c) context-preserving ribbons (d) context-preserving slices

In this example, a single mandible has been cut using a curving cutter extruded along the natural "L" shape of the geometry. Notice how transparency can be applied to the removed region to help preserve the curving nature of the mandible (Figure 62b). However, as in the knee dataset cutaway, there is a problem perceiving its depth along the cutaway region. This is remedied by positioning ribbons or solid slices at regular intervals along the mandible to visually reinforce its shape (Figure 62c, d). Furthermore, by observing the angles of the ribbons/slices relative to each other, it is possible to follow the path of the jaw as it rotates inward from one end of the cutaway to the other.

4.3.3 Vertebrae/Spinal Disc Dataset – Context Preserving Rendering

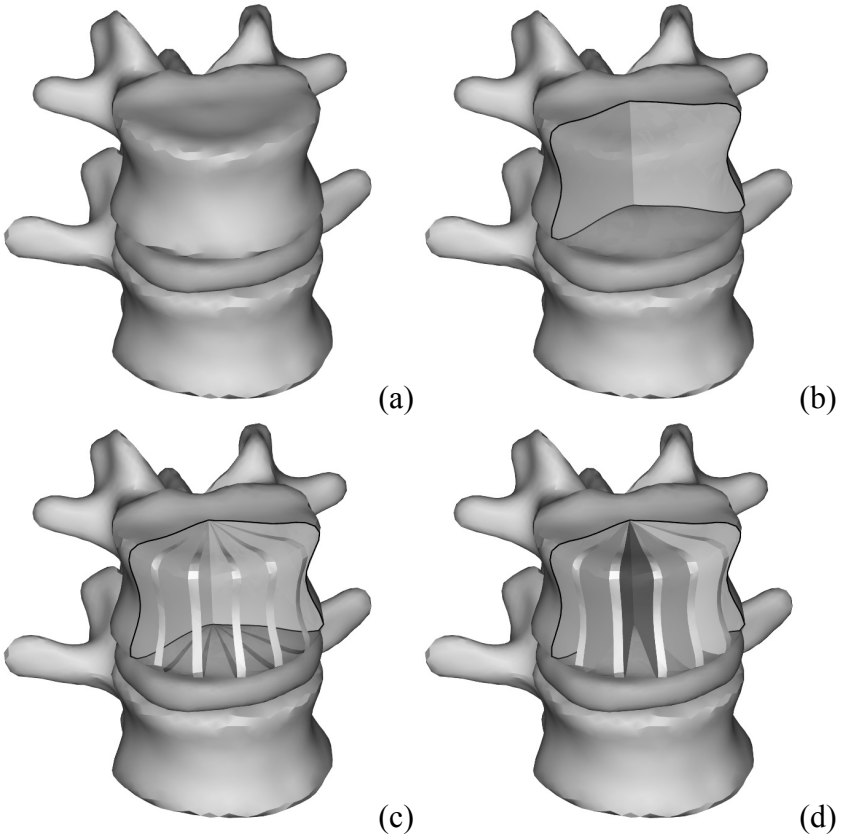


Figure 63 – Vertebrae/Spinal disc dataset cutaway rendered using: (b) simple transparency (c) context-preserving ribbons (d) context-preserving slices

In this example, a single vertebra is cut using a wedge-shaped cutter to expose the intervertebral disc directly beneath it. In this case, due to the angle of the camera relative to the spine, transparency does little to hint at the shape of the missing geometry of the cutaway vertebra (Figure 63b). However, by employing ribbons/slices (Figure 63c,d) the natural shape of the vertebra is more accurately represented. For example, the distinctive radial indentation can now be perceived by observing the shape of the corresponding ribbons (Figure 63c) or slices (Figure 63d). From the ribbons, it is also clear how the vertebra sits on the disc (i.e. the spatial interrelationship between the vertebra and the disc is clearly depicted).

4.3.4 Foot Dataset – Context Preserving Rendering

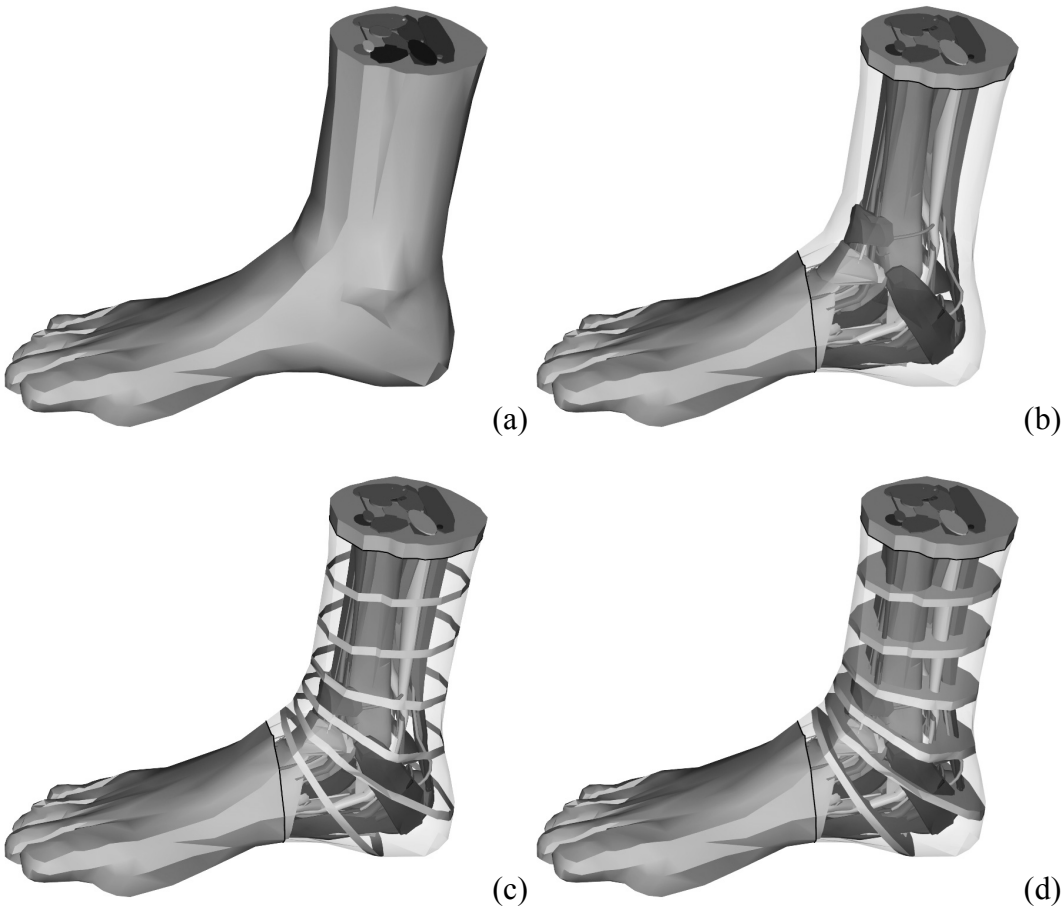


Figure 64 – Foot dataset cutaway rendered using: (b) simple transparency (c) context-preserving ribbons (d) context-preserving slices

In this final example, the outer layer of skin encompassing the internal veins, arteries, muscles, ligaments and bones of the foot has been cut away (using a curving cut) over the heel and achilles tendon. Transparency has once again been applied to Figure 64b as a visual clue to help discern the location of the internal objects relative to the outer skin. However, while transparency does offer the greatest reduction in occlusion, it unfortunately does little beyond providing a silhouette of the skin of the foot and is viewing angle dependent. Ribbons (Figure 64b) offer a greater depiction of the 3D shape of the removed material, but the exact distance between the outer skin and the inner structures is still difficult to perceive. It is not until solid

slices are employed (Figure 64c) that the physical gap between the outer skin and the inner occluded structures is accurately and effectively represented.

Chapter V – Summary, Conclusion and Future Work

A number of novel rendering techniques in the literature have been discussed as a means to expose internal structures in a 3D mesh visualization while still maintaining a sense of their context relative to outer occluding objects. Reducing the opacity of occluding objects is a simple example of how this can be accomplished. However, if reduced opacity (transparency) alone is used to expose internal structures, it is often difficult or impossible to perceive complex layering or spatial relationships between the inner (occluded) and outer (occluding) structures. In an attempt to gain a greater understanding of spatial interrelationships between occluded and occluding objects, lattice or feature line rendering has also been examined. However, due to the complexity, size (i.e. number of polygons), and triangular-polygon nature of reconstructed anatomical surface meshes, such automatic rendering techniques are inadequate for conveying useful information. Cutaway views have been examined as a means of exposing occluded objects and providing depth information of internal structures relative to external structures.

However, while cutaways present a promising technique for exposing internal structures, this thesis has argued that in many systems the ability to efficiently and effectively construct, define and modify a corresponding cutaway region is often lacking. This is primarily due to the design of the *interaction model* used in defining the cutaway region. For example, tracing-based interaction models attempt to mimic the real world action of using a pen to outline a region of interest. While this type of surface-constrained interaction may be useful to construct freeform "window" cuts, it is difficult to perform more structured, symmetric or volume-based cuts. Furthermore, editing or modifying the cut region is also problematic as the user is forced to backtrack and/or retrace the desired outline.

Sketch- or gesture-based interaction is another metaphor that has been implemented in cutaway systems. In this scenario the user draws different curve shapes on the object surface and each curve is then interpreted by the algorithm to generate different cutaway regions. For example, Knödel et al. [19] define four different curve gestures: line, circle, corner, and ribbon. One of the basic problems with this interaction metaphor is that users cannot immediately see the shape of the cutaway region they are creating. Another problem is after the initial cutting phase, the modification of a cutaway region is typically executed using a widget to perform basic actions such as translating or uniformly scaling the cutaway region. In addition to potentially limiting the editing flexibility of the cutter, introducing widgets as an editing interface forces the user to master a second, distinct interaction metaphor.

The ExtrusionCutter tool employs a unique "paint roller" interaction metaphor that has been designed to minimize these problems by providing two key pieces of functionality: the ability to position the cutter along natural shape features of the object using simple constrained input actions, and the ability to easily modify/fine tune the cut. Furthermore, by introducing simple constraints, five distinct types of cutaways can be created using the same extrusion process. Additionally, ExtrusionCutter has been designed to facilitate unique "context-preserving" cutaways in addition to traditional cutaways. For example, by rendering ribbons or solid slices at regular intervals throughout the removed cutaway material, it is possible to visualize its shape and spatial orientation relative to the inner, occluded objects.

5.1 Conclusion

This thesis has argued that using either semi-transparency or cutaways separately to examine occluded interior structures often results in an ineffective visual communication style. Some synthesis of the two techniques is often necessary in order to understand the spatial interrelationships of a complex system of anatomical structures. ExtrusionCutter introduces shaded ribbons and solid slices to achieve this synthesis. The shaded ribbons coupled with a semi-transparent cutaway region not only maintains a contextual view but also clearly shows the spatial relationship of the back of the occluding surface and the front of the interior surface. Additionally, solid slices positioned at key points within cutaway region are capable of clearly illustrating the distance between occluding and occluded objects in 3D space. Furthermore, it has been demonstrated that the simple, intuitive generation of a desired data view is equally as important as the visual communication style itself. Therefore, ExtrusionCutter has introduced a novel extrusion-based "paint roller" interaction model capable of generating a number of different cutaway styles including curving, straight path, wedge, rounded and freeform cuts. This design allows the user to create a variety of complex, context preserving cutaways that are aligned with the natural shape features of an occluding object by using consistent, familiar mouse movements.

5.2 Future Work

Currently, there are a number of interesting improvements and extensions to the ExtrusionCutter system being explored. For example, to further increase the types of cuts

possible, the extrusion model can be extended to allow any edge of the cutter mesh to be extruded, rather than just the leading one. This would create a more complex arrangement of cutting planes, making cutaways in multiple directions along branching structures possible. Additionally, multiple strategies for minimizing editing time are also being investigated, including automatically fitting the height of the cutter to surround the target region as it is extruded. Similarly, the use of a multi-touch input device is being actively researched to enable the user to sweep a given cutting region over a target object with their hands, potentially allowing users to vary the height of the cutaway region during the sweeping action itself. Lastly, there are plans to accelerate the cutting operation by porting the CSG processing to the GPU and consequently enabling real-time cutting during cutting region extrusion.

References

[1] AbsoluteAstronomy.com. (n.d.). *Constructive solid geometry: Facts, Discussion Forum, and Encyclopedia Article*. Retrieved September 22, 2010, from http://www.absoluteastronomy.com/topics/Constructive_solid_geometry

[2] Agrawala, M., Phan, D., Heiser, J., Haymaker, J., Klingner, J., Hanrahan, P., and Tversky, B. (2003). *Designing effective step-by-step assembly instructions*. In SIGGRAPH '03: ACM SIGGRAPH 2003 Papers, pages 828-837, New York, NY, USA. ACM.

[3] ARM Information Center. (n.d.). *Mali™ GPU Shader Library User Guide Version: 1.0*. Retrieved September 22, 2010, from http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0510a/index.html

[4] Björk, S., & Redström, J. (2000). *Redefining the focus and context of Focus+Context visualizations*. INFOVIS '00: Proceedings of the IEEE Symposium on Information Vizualization 2000, 85.

[5] Bruckner, S., Grimm, S., Kanitsar, A., & Groller, M. E. (2006). *Visualization and Computer Graphics*, IEEE Transactions on Visualization and Computer Graphics, 12(6), 1559.

[6] Bruyns, C. D., & Senger, S. (2001). *Interactive cutting of 3D surface meshes*. Computers & Graphics, 25(4), 635.

[7] Bruyns, C. D., Senger, S., Menon, A., Montgomery, K., Wildermuth, S., and Boyle, R. (2002). *A survey of interactive mesh-cutting techniques and a new method for implementing generalized interactive mesh cutting using virtual tools*. The Journal of Visualization and Computer Animation, 13(1):21-42.

[8] Candyshopsoftware.com. (n.d). *Eugene's Shader Presentation*. Retrieved August 22, 2010, from www.candyshopsoftware.com/GPU/index.html

[9] Chan, M., Wu, Y., Mak, W., Chen, W., & Qu, H. (2009). *Visualization and Computer Graphics*, IEEE Transactions on Visualization and Computer Graphics, 15(6), 1283.

[10] Coffin, C., Höllerer, T.*, Interactive Perspective Cut-away views for general 3D scenes*, In Proc. of IEEE Symposium on 3D User Interfaces, Alexandria, VA, 2006, 25–28.

[11] Conner, B. D., Snibbe, S. S., Herndon, K. P., Robbins, D. C., Zeleznik, R. C., & van Dam, A. (1992). *Three-dimensional widgets*. I3D '92: Proceedings of the 1992 Symposium on Interactive 3D Graphics, Cambridge, Massachusetts, United States. 183-188.

[12] Diepstraten, J., Weiskopf, D., Ertl, T., *Interactive Cutaway Illustrations*, Computer Graphics Forum 21, v22, n3, 2003, 523–532.

[13] Diepstraten, J., Weiskopf, D., Ertl, T., *Transparency in Interactive Technical Illustrations*, Computer Graphics Forum 21, 2002, 317–326.

[14] E. Hodges., *The Guild Handbook of Scientific Illustration*, Van Nostrand Reinhold, New York, 1989.

[15] Gottschalk, S., Lin, M. C., *OBBTree: A Hierarchical Structure for Rapid Interference Detection*, Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, pp 171-180, 1996.

[16] Hodges, E.R.S., *Guild Handbook Of Scientific Illustration*, New York: John Wiley And Sons Ltd, 2003.

[17] Huang, J., Carter B. M., *Interactive Transparency Rendering for Large CAD Models*, IEEE Transactions on Visualization and Computer Graphics, vol. 11, no. 5, 2005

[18] International Symposium on Visual Computing. (n.d.). *International Symposium on Visual Computing*. Retrieved September 22, 2010, from http://isvc.net

[19] Knödel, S., Hachet, M., Guitton, P.*, Interactive Generation and Modification of Cutaway Illustrations for Polygonal Models* , In Proc. Of the 10th Int. Symposium on Smart Graphics, Salamanca, Spain, 2009, 140–151.

[20] Lawlor, G. (n.d.). *Bioengineering Research*. Retrieved September 08, 2009, from http://www.bioengineering-research.com

[21] Li, W., Agrawala, M., Curless, B., & Salesin, D. (2008). *Automated generation of interactive 3D exploded view diagrams*. SIGGRAPH '08: ACM SIGGRAPH 2008 Papers, Los Angeles, California. 1-7.

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

[23] Ma, K., & Interrante, V. (1997). *Extracting feature lines from 3D unstructured grids*. VIS '97: Proceedings of the 8th Conference on Visualization '97, Phoenix, Arizona, United States. 285-ff.

[24] Marr, D., *Early Processing of Visual Information*, Philosophical Transactions of the Royal Society of London, Series B. Biological Sciences, vol. 275,(October 1976), pp. 483-519.

[25] McGuffin, M.J., Tancau, R., Balakrishnan, R., *Using Deformations for Browsing Volumetric Data*, In proceedings of IEEE Visualization, Seattle, Wash., 2003, 401–408.

[26] Netter, F. H. (2006). *Atlas of Human Anatomy, Professional Edition* (4 ed.). St. Louis: Saunders.

[27] OpenCSG - The CSG rendering library. (n.d.). *OpenCSG - The CSG rendering library*. Retrieved September 22, 2010, from http://www.opencsg.org

[28] OpenGL. (n.d.). *OpenGL - The Industry Standard for High Performance Graphics*. Retrieved September 22, 2010, from http://www.opengl.org

[29] OsiriX. (n.d.). *OsiriX - DICOM Viewer*. Retrieved September 22, 2010, from http://www.osirix-viewer.com

[30] Power, K. (n.d.). *Projection & Viewing Transformations*. Retrieved September 22, 2010, from http://glasnost.itcarlow.ie/~powerk/GeneralGraphicsNotes/projection_viewing.html

[31] Roberts, J. C. (1993). *An overview of rendering from volume data --- including surface and volume rendering*, University of Kent, Canterbury, UK: University of Kent, Computing Laboratory.

[32] Rost, Randi J. (2006). *OpenGL(R) Shading Language (2nd Edition)* (OpenGL). 2 ed. New York: Addison-wesley Professional.

[33] Schaefer, S, Warren, J., *A Factored Interpolatory Subdivision Scheme for Quadrilateral Surfaces*, Curve and Surface Fitting: Saint Melo 2002, 2003, 373–382.

[34] SourceForge. (n.d.). *GTS - The GNU Triangulated Surface Library*. Retrieved September 22, 2010, from http://gts.sourceforge.net

[35] Stewart, J. (n.d.). *Volume Visualization & The Marching Cubes Algorithm*. Dr. Jaromczyk's Research Group Summer 2005. Retrieved September 22, 2010, from csurs1.csr.uky.edu/~pthacker/volvis.ppt

[36] Tamir, D., Komogortsev, O. V., & Mueller, C. J. (2008). *An effort and time based measure of usability*. WoSQ '08: Proceedings of the 6th International Workshop on Software Quality, Leipzig, Germany. 47-52.

[37] Tyszka, J. M. (n.d.). *Creation of spiral, CT-derived, three-dimensional VRML objects*. FBS, A Journal And Virtual Library . Retrieved September 22, 2010, from http://www.bioscience.org/1997/v2/f/tyszka1/htmls/2-3.htm

[38] Verroust, A., Lazarus, F., *Extracting skeletal curves from 3d scattered data*, The Visual Computer 16, 1, 15–25, 2000.

[39] VTK - The Visualization Toolkit. (n.d.). *VTK - The Visualization Toolkit*. Retrieved September 22, 2010, from http://www.vtk.org

[40] VTK Journal. (n.d.). *VTK Journal - Boolean Operations on Surfaces for VTK*. Retrieved September 22, 2010, from hdl.handle.net/10380/3169

[41] Ward, M., *Overview of the Marching Cubes Algorithm*, <http://web.cs.wpi.edu/~matt/courses/cs563/talks/march_cub.html>, 2010.