INTERACTIVE CUTTING OF THE SKULL FOR CRANIOFACIAL SURGICAL PLANNING

Greg Pintilie*1, Tim McInerney*2,*1
*1 Department of Computer Science, University of Toronto, Canada
*2 Department of Math, Physics, Computer Science, Ryerson University, Canada
Email: gdp@cs.utoronto.ca, tmcinern@scs.ryerson.ca

ABSTRACT

We present a prototype surgical planning system that simulates cutting and positioning operations on rigid objects such as the human skull, with realistic haptic and visual feedback. Our positioning interface allows users to grab and manipulate objects while sensing the interaction forces between the object being manipulated and other objects in the environment. The cutting interfaces support planar and contour cuts. We present two algorithms that create both types of cuts through rigid objects that are represented by closed triangular meshes. Both algorithms create cuts with predictable accuracies independent of the meshes they operate on, and avoid the introduction of degenerate mesh elements. For contour cuts, the cut region may be closed or open, and whenever possible it is separated from the original object. To deal with discontinuities in the surface meshes of solid objects introduced by such operations, we have implemented an appropriate surface-reconstruction algorithm.

KEY WORDS

Modeling, Simulation and Computing, Computer-Aided Surgical Planning, Interactive Cutting and Positioning with Haptic Feedback, Triangular Mesh

1. Introduction

A computer-aided surgical planning system would be a crucial tool for craniofacial surgeons. Operations on the skull and surrounding tissues involve many patient-specific variables, making each operation unique and its results hard to predict. As a result, surgeons base their decisions mainly on surgeries they have already performed or observed. A planning system would allow a surgeon to perform the procedure ahead of time on a virtual replica of the patient's head, and to pick among surgical alternatives based on feasibility and aesthetic preferences. Such a system would enable, with the use of CT, MRI and laser range scans, the creation of an individualized model of the patient's head and face, and the determination of specific surgical operations that would produce this result based on relevant anatomical constraints governing skeletal and soft tissue transpositions.

One objective of such a system is the accurate representation of the skull and the tissues around it. The solution would involve the use of techniques to model solid and soft objects visually and dynamically as they combine and interact to create more complex and intricate entities such as the human head, as in [13]. Another objective is the development of a realistic virtual environment in which operations can be realistically performed with the aid of visual and haptic feedback. This requires the integration of advanced visualization techniques and haptic devices to create a user interface that fully immerses the surgeon in a virtual environment. Virtual saws, scalpels, drawing tools, and tissue "grabbing" tools would all contribute to the creation of such an interface. A very important interaction is bone realignment required for operations such as osteotomies. Virtual cutting tools are required to split and cut out pieces of the skull, and each resulting bone segment must then be defined as a unique object that can be shaped and manipulated independently and maneuvered into place. Collisions with other parts of the skull must be detected and their effects simulated.

In this work we limit ourselves to the modeling of and interaction with rigid objects such as the skull, represented with triangular surface meshes. We use CT scans of a patient's head and model objects using the Amira[1] software package, which provides tools to read the CT data and to convert it into triangular meshes using algorithms such as image segmentation, contour-finding, and surface reconstruction via marching cubes [14]. Our virtual environment is constructed within the Amira visualization system, which uses Open Inventor[2] for 3D graphic display. We have integrated the GHOST API[3] into this environment, allowing simultaneous visual and haptic rendering of objects, the latter through the PHANTOM[4] device. Within this environment we have developed interface tools that allow users to manipulate and cut rigid objects represented by triangular surface meshes.

Cutting algorithms for surface and solid meshes have been presented in [1-5]. Such algorithms generally operate on the target mesh by sub-dividing the mesh elements that the cut path intersects into a pre-determined number of smaller elements. The accuracy of the cuts is thus highly dependent on the geometry of the initial mesh, and the loss of small details in cuts may occur in areas of low refinement. Most of these algorithms have ways of dealing with the creation of degenerate mesh elements, i.e. elements with small area or thin shape. Such elements are largely unavoidable in cutting operations and also an important consideration, especially when the mesh is used in finite-element simulations of soft tissues.

We will present two cutting algorithms for creating planar and contour cuts. Both algorithms refine the mesh if necessary by adding new vertices to it so that the cut is approximated with a predictable error factor. Vertices are not added close to each other to avoid the introduction of degenerate mesh elements. The new vertices are added to the mesh using Delaunay triangulation [6], which ensures the resulting mesh is as optimal as possible. We will also describe how the cutting algorithms are extended to the cutting of meshes that represent solid objects such as the human skull.

2. Object Manipulation

2.1 'Grab and Move' Interface

Our approach to object manipulation attempts to closely mimic object manipulation in the real world. For example, to move a real object, a person would reach out to touch it, grab it, and then move it to a new position. Once grabbed, the object changes its position and orientation to follow the user's hand, with the initial conditions being maintained at the contact points.

At the root of this interaction is the sensation of contact with the object. The Phantom device is capable of point interaction with objects in a virtual environment, providing 6DOF input and 3DOF output [7]. This means that using the Phantom device, the user can sense virtual objects as they would with the tip of their finger or the tip of a pointer. When this interaction point has penetrated an object, the device outputs a force that attempts to restore it to a non-penetrating state, as if a spring were to exist between the interaction point and the nearest surface point. In this state the interaction point is said to be in a contact state with the object. In this type of interaction, the objects seem to have a soft feel, with the degree of softness depending on the stiffness of the spring.

In our interface, to start moving an object, the user makes contact with it and clicks the button on the Phantom pointer. This indicates that the object has been 'grabbed', and it is then re-positioned and re-oriented as the pointer moves such that the initial contact state is maintained (Figure 1). For the details of this computation, we refer the reader to [8]. The object is 'released' when the button is also released.

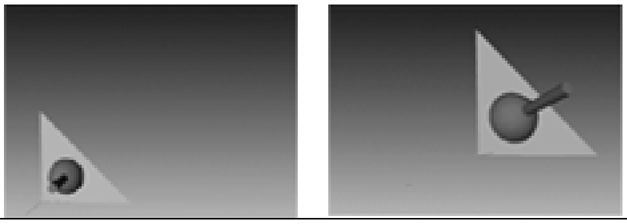
Figure 1. Two frames showing the positioning a simple object (triangle) by grabbing it with the haptic device (stick and sphere).

2.2 Haptic Feedback During Manipulation

When an object is manipulated as described in the previous section, collisions with other objects may occur. We would like to allow the user to feel the contact forces due to these interactions. This is useful in a craniofacial surgical planning environment, for example, for testing the fit of a harvested bone segment at the target site in the skull.

To detect the penetration condition of mesh objects, both point penetrations and edge intersections must be checked [9]. In our implementation a simpler approximation is needed to meet the time restrictions for computations in a haptic environment. We thus check only for point penetrations to detect object-object collisions, with the points distributed over the surface of the object being manipulated. Upon each re-positioning of the object, we check whether any of the points have penetrated other objects in the environment, this being an efficient computation provided by GHOST.

When penetrations do occur, they are handled as individual point-interactions, and spring forces are applied to try to restore the objects to a non-penetrating state (Figure 2). Because the point at which the force is created does not necessarily coincide with the point where the force is applied, torque is typically created. However the Phantom device being used for this work does not produce torque forces and consequently we omit their computation.

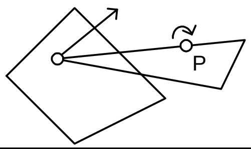

Figure 2. The Phantom pointer (P) is positioning a triangular object that intersects another object. The user feels a force that opposes the penetration (straight arrow), and also a torque (curved arrow).

The number of points that are checked for penetrations is limited, depending on the complexity of the objects in the environment and the speed of the computer, so that the computation time is within the bounds needed by the haptic device. Hence the points are distributed over the surface of the object in such a way as to best represent the object.

3. Cutting of Surface Meshes

3.1 Cutting Interfaces

We have implemented two cutting interfaces: a plane tool, and a blade tool. The plane tool allows the user to position a plane using the pointing device, as shown in Figure 3(a). The plane is displayed as a transparent surface, which allows the user to position it relative to objects in the environment using visual cues.

The blade tool is displayed as a thin blade extending from the pointer position, as shown in Figure 3(b). When the tip of the blade comes to intersect the surface of an object in the environment, a viscosity-like force that opposes the blade's velocity is output, as would be expected in a real cutting process. The cut contour is formed on the object surface along the path of the blade while it is intersecting the object.

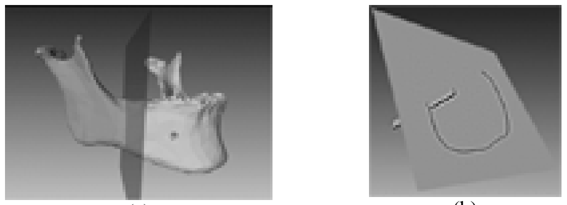

Figure 3. Tools for cutting: (a) plane tool intersecting a mandible bone (the region to be cut is shown in dark gray) and (b) blade tool intersecting a planar object and having created a 'cut contour' on its surface. The mandible bone is from the visible human data set.

3.2 Planar Cutting Algorithm

The planar cutting algorithm creates an approximate cut through a triangle mesh using the plane specified by the user. The first step is to label each vertex in the mesh as belonging to one of three categories: above the plane, below the plane, or within a distance ρ of the plane. In the last case the point is considered to be on the plane, and the distance ρ is an error tolerance value in the approximation of the cut.

The second step is to subdivide the triangles that have at least one point above the plane and one point below the plane. Such triangles have two edges that intersect the plane, and the intersection points are found. A new point set is formed consisting of the original triangle vertices and each intersection point that does not come within the distance ρ of the other triangle vertices. Given the labeling method and the triangles being considered, this new set contains at least 4 points: the three original vertices and at least one intersection point. A new mesh is created from it using Delaunay triangulation (Figure 4), which is used to replace the original triangle.

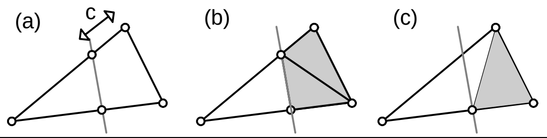

Figure 4. A triangle element intersected by a cutting plane (a). If $c > \rho$, both intersection points are added to the mesh (b). Otherwise the top point is not added to the mesh (c). The cut is formed by separating the two regions above (white) and below (gray) the plane.

The last step is to separate the mesh into two regions using the cutting plane. Since elements may intersect the plane, we find the intersection points for every element and compute their area above and below the plane. If the area above is larger than the area below, the element is considered to be above the plane, otherwise below the plane. An example of planar cutting on the mandible is shown in Figure A1.

3.3 Contour Cutting Algorithm

3.3.1 Overview

The blade tool creates a *cut contour* that is defined by a set of points that lie on the surface of the mesh. We define a *vertex contour* to be a set of points which are vertices in the mesh, and each line segment between adjacent points is an edge in the mesh.

The principle of our contour cutting algorithm is to compute a vertex contour that approximates a given cut contour to within a predictable accuracy, changing the mesh if needed, but not changing the surface shape and not introducing degenerate mesh elements. The algorithm can be broken up into four steps:

1. Refine the mesh in the region of the cut.
2. Change the cut contour so that all its segments lie on the surface of the mesh.
3. Sample the cut contour to create the vertex contour.
4. Change the mesh so that the sample points are incorporated into the mesh.

Once a vertex contour has been computed, an algorithm checks whether it separates the mesh into two disjoint regions. There are two possible ways for this to happen: either the vertex contour is open and the first and last points are vertices on the border of the mesh, e.g. Figure A2(a), or the vertex contour is closed, and the first and last points are vertices of the same triangle in the mesh, e.g. Figure A2(c). If the algorithm finds two disjoint regions, they are separated into different objects to create the cut.

3.3.2 Mesh Refinement

To refine the mesh we use a simple mesh-subdivision procedure that maintains the general structure of the original mesh while decreasing the size of its elements. The purpose of this step is to reduce the introduction of sliver elements in a later step of the cutting procedure when sample points from the cut contour are added to the mesh.

The mesh is refined in the region of the cut by sequentially sub-dividing triangles that contain at least one cut contour point. A triangle is subdivided by introducing a new vertex at the mid-point of one of its edges, thus creating two new triangles (Figure 5). Any other triangle sharing that same edge is also divided as a result. There are three possible ways to divide a triangle using this scheme; one is picked so that the introduced edge is the one that has the smallest length.

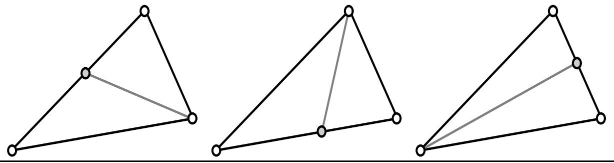
Figure 5. Subdivision schemes for a single triangle.

When a triangle is subdivided, the cut contour points on its surface are re-distributed to the newly created triangles. Subdivision continues as long as there are triangles containing cut contour points that have an area larger than a specified constant, δ. This constant is tied to the constant defining a desired cut accuracy, as described in a later section.

3.3.3 Intersection Points

Adjacent cut contour points that lie on different non-coplanar triangles form line segments that do not lie on the surface of the object. To make the cut contour between such points adhere to the surface, we replace each of these segments with two new segments that lie on the surface of the object. Both new segments have one end point from the original segment, and one point on the edge shared by the triangles that these end points lie on. The latter is found using a 3D line intersection algorithm that finds the point on the edge closest to the original line segment.

We refer to the new points added to the cut contour as intersection points, because they are where the cut contour intersects mesh edge elements. These intersection points will be incorporated into the mesh to become points in the vertex contour, as will be described later. Hence one precaution is taken at this stage: when an intersection point is found as described above, if it is within a certain distance τ of an existing vertex, the existing vertex is added to the contour instead of the intersection point (Figure 6). This is because points that are very close together would create mesh elements with very small area, i.e. degenerate triangles, which we would like to avoid.

In the case just described we might have chosen to instead move the vertex point towards the cut contour, however we have chosen to change the cut contour as we do not want the surface to lose its original shape. This means that the cut may not exactly match the specified cut, with the constant τ specifying a maximum distance that the actual cut may deviate from the specified cut. This parameter also specifies the minimum edge length of mesh elements introduced as a result of the cut, and hence there is a trade-off between the accuracy of the cut and the tolerance towards degenerate elements in the resulting mesh.

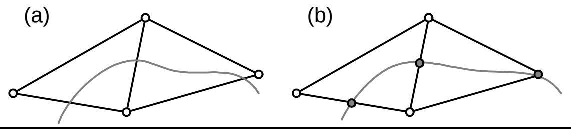

Figure 6. (a) Original mesh and a cut contour. (b) Two intersection points and a vertex are added to the cut contour to make it adhere to the surface.

3.3.4 Sampling the Cut Contour

The next step is to sample points from the cut contour, which will later be incorporated into the mesh to become the vertex contour. The objectives for this sampling are that it doesn't deviate from the cut contour by more than a constant distance, which is a measure of the cut accuracy, and that it uses as few points as possible. A constraint is that the intersection points in the cut contour must be among the sample points (Figure 7).

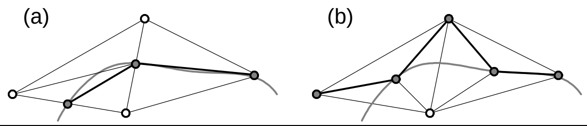

Figure 7. Resulting vertex contours (through filled points), using (a) and not using (b) intersection points. It is clear that in (a) the vertex contour more closely matches the cut contour.

The intersection points alone may not be enough to represent the cut contour accurately, for example as in Figure 9(a). We thus further sample the cut contour between intersection points using the Douglas-Peuker algorithm [10]. This algorithm finds, for a curve described by any number of points, an approximating curve with a smaller or equal number of points so that the largest distance between the two curves at any point is smaller than a given constant value υ.

This sampling algorithm is applied to each section of the cut-contour between intersection points as follows: if the contour deviates from the straight line formed by its two end points by more than a distance υ, a new sample point is added at the point on the contour with the largest distance from this line. Then the contour is divided into two new sections at this point, and the process is repeated on both sections (Figure 8).

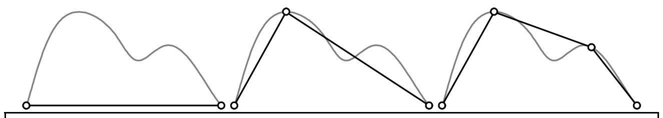

Figure 8. Three steps of the Douglas-Peuker sampling algorithm.

The sample points found at this step and the intersection points are added to the vertex contour in the order that they occur in the cut contour. If the start and end points of the cut contour are found to be close together, extra sample points are added between them if necessary, so as to create a closed vertex contour for closed cuts (Figure A2-c).

The constant υ used in this step is similar to the constant τ of the previous section, in that it specifies a maximum allowed deviation distance between the cut contour and the vertex contour. Hence we generally set them to have the same value.

3.3.5 Adding the Sample Points to the Mesh

In this step the sample and intersection points in the cut contour are introduced into the mesh. To do this, every triangle affected by the cut is replaced by a new set of triangles that include the intersection points on its edges and every sample point in between (Figure 9). The new triangle set is constructed using Delaunay triangulation [6].

As a result of this process, vertex contour segments will in most cases have a corresponding edge in the new mesh. For a more detailed analysis, see [8]. Two exceptions are when the cut contour intersects itself and when the sampling between intersection points yields a large number of sample points. The latter is more likely to happen when there are many sample points inside a triangle, as in Figure 9. This can be detected and fixed by 'flipping' an edge between two triangles, though at the price of using a less optimal mesh structure. This is illustrated in Figure 9(b), where the edge intersecting the dotted line is removed and the dotted line is added as a new edge.

The mesh refinement process generally decreases the occurrence of cases when an edge flip is needed since it reduces the size of each triangle affected by the cut. This ties the intersection and sampling constants (τ and υ), to the mesh-refinement constant δ. In practice we set δ to a value that reduces the number of sample points needed inside any one triangle to a small predictable number. To find a value for δ we note that δ refers to area whereas τ and υ to length; a formulation that we've found works well in practice is:

$$\delta = 5\tau^2, \text{ (and } \tau = \upsilon\text{)}.$$

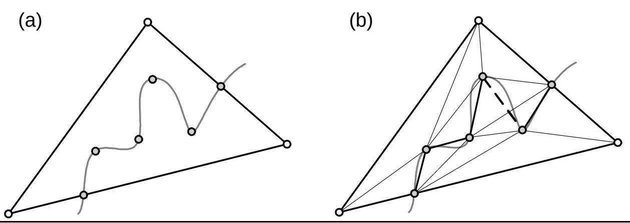

Figure 9. (a) Sample and intersection points for a cut contour inside a triangle. (b) The resulting mesh after Delaunay triangulation, which does not contain a required edge (dashed line) for the vertex contour.

3.3.6 Finding the Cut Region

We now present an algorithm that determines whether a vertex contour creates a separable cut. The algorithm attempts to find two disjoint regions in the mesh that are separated by the vertex contour, the smaller of which is considered to be the cut region.

We assume that we have already identified for each segment in the vertex contour a corresponding edge in the mesh. The algorithm performs two simultaneous traversals over the triangles in the mesh, each starting with one of the two triangles that share the corresponding edge of the first segment in the vertex contour. A queue is used by each traversal, which holds triangle elements that are yet to be visited. A triangle is added to the queue if it is a neighbour of the triangle being visited, unless the edge shared by the triangles has a corresponding segment in the vertex contour.

The algorithm stops when either of the two searches has finished, and the elements traversed by that search define the cut region. At every traversal step if it is detected that the two traversals have visited the same triangle, the algorithm stops and fails because then the regions are not disjoint. In case of success the two regions are separated to create the cut. Examples of closed and open cuts on surface meshes are shown in Figure A2.

4. Cutting of Solid Objects

4.1 Solid Objects as Surface Meshes

Solid objects reconstructed from medical data using Amira can be represented either by closed triangular surface meshes or tetrahedral meshes. We use the surface mesh representation for the human skull, which has tremendous surface variation but relatively small thickness in regions where a craniofacial surgeon is most interested in repairing or harvesting bone. In such regions the objects can be considered to be bounded by inner and outer surfaces.

4.2 Planar Cuts through Solids

Planar cuts through solid objects create a gap at the intersection of the plane and the object, as illustrated in

Figure 10(a). The boundary vertices may or may not lie on the cutting plane, due to the approximating nature of the planar cutting algorithm. Assuming they are very close to the cutting plane, we project the vertices onto the plane and create a new mesh using a constrained planar triangulation algorithm [11]. This mesh structure is then added to the mesh, mapped to the original mesh vertices, to close the gap, as shown in Figure 10(b).

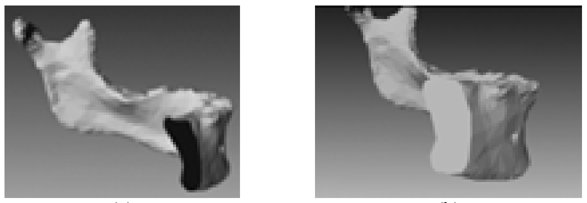

Figure 10. (a) Mandible bone before, and (b) after adding of mesh to close the gap resulting from a planar cut.

4.3 Contour Cuts through Solids

To create a contour cut through a solid object, we extend the cut so that cut contours are simultaneously created on both the inner and outer surfaces. As the user creates a contour cut on the outer surface, a ray-object intersection is performed to create the cut contour on the inner surface. The ray starts at the cut contour point on the outer surface and extends into the object in the direction of the blade. The inner cut contour point results from the intersection of this ray and the inner surface. For simplicity we use a single direction for this ray for the entire cut contour, and this is picked to be the direction of the blade when the cut is initiated.

Once an outer cut contour and an inner cut contour have been acquired, the contour cutting algorithm is applied independently to each surface, resulting in inner and outer cut surfaces. These two surfaces represent a single solid object, however they are discontinuous at the cut boundary, as shown in Figure 11(a).

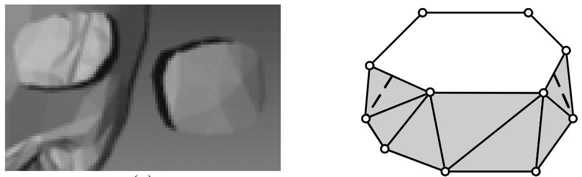

Figure 11. (a) A cut through a solid object, showing discontinuity between inner and outer surfaces at the cut boundary. (b) A diagram of the reconstructed mesh (in gray) used to fill such a gap.

To close the resulting gap, we use the surface reconstruction approach described in [12]. This creates a new mesh that becomes part of the cut object, as diagrammed in Figure 11(b). The same mesh (but with opposite normals) is also added to the original object to close the gap left there as a result of the cut. The process of creating a cut through a solid object is illustrated in Figure A3.

5. Conclusion

In this work we have presented user interfaces and algorithms for object manipulation and cutting for the purpose of surgery planning on the human skull. The interfaces are similar to their real-life counterparts as they provide similar visual and haptic feedback. The cutting algorithms we have developed create planar and contour cuts through surface meshes with predictable accuracy. The mesh is refined when necessary to create a cut within this accuracy measure. An accuracy parameter specified by the user trades off between the maximum deviation between the specified and actual cuts and the tolerance to degenerate triangles created as a result of mesh refinement. Using the surface cutting algorithms, it is possible to simulate the cutting and extraction of pieces from a solid object that is represented as a surface mesh.

REFERENCES

[1] P.F. Neumann, L.L. Sadler, & J. Gieser, Virtual Reality Vitrectomy Simulator, *Medical Image Computing and Computer-Assisted Intervention*, 1998, 910-917

[2] D. Bielser & M.H. Gross, Interactive Simulation of Surgical Cuts, *Proceedings of Pacific Graphics*, 2000, IEEE Computer Society Press, 116-125

[3] H. Nienhuys & A. Frank van der Stappen, Combining Finite Element Deformation With Cutting for Surgery Simulations, *Proceedings of Eurographics 2000, short presentations*, 43-52

[4] F. Ganovelli & C. O'Sullivan, Animating Cuts with on-the-Fly Re-Meshing, *Proceedings of Eurographics 2000, short presentations*, 243-247

[5] H. Delingette, S. Cotin, & N. Ayache, A Hybrid Elastic Model Allowing Real-Time Cutting, Deformations and Force-Feedback for Surgery Training and Simulation, *CAS99 Proceedings*, 1999, 70–81

[6] M. Bern & D. Eppstein, Mesh Generation and Optimal Triangulation, *Computing in Euclidean Geometry*, edited by Ding-Zhu Du and Frank Hwang, (Singapore: World Scientific Publishing Co. Ltd. 1995) 47-123

[7] C. Zilles & J. Salisbury, Constraint-based God-object Method for Haptic Display, ASME Haptic Interfaces for Virtual Environment and Teleoperator Systems, *Dynamic Systems and Control*, vol. 1, 1994, 146-150.

[8] G. Pintilie, *Interactive cutting of surface meshes for computer-aided surgical planning*, M.Sc. Thesis, Department of Computer Science, University of Toronto, 2001

[9] M. Moore & J. Wilhelms, Collision detection and response for computer animation, *Computer Graphics, 22*(4), 1988, 289-298

[10] D.H. Douglas and T.K. Peuker, Algorithms for the Reduction of the Number of Points Required to Represent a Line or its Caricature, *The Canadian Cartographer, 10*(2), 1973, 112-122.

[11] J. O'Rourke, *Computational geometry in C, Second Edition*, (Cambridge, UK: Cambridge University Press, 1998).

[12] H. Fuchs, Z.M. Kedem, & S.P. Uselton, Optimal Surface Reconstruction from Planar Contours, *Communications of the ACM, 20*(10), 1977, 693-702.

[13] Y. Lee, D. Terzopoulos, & K. Waters, Realistic modeling for facial animation, *Proc. ACM SIGGRAPH'95 Conference*, Los Angeles, CA, August, 1995, 55-62.

[14] W. E. Lorensen & H. E. Cline, Marching Cubes: A High Resolution 3D Surface Construction Algorithm, *Computer Graphics, 21*(4), July 1987, 163-169.

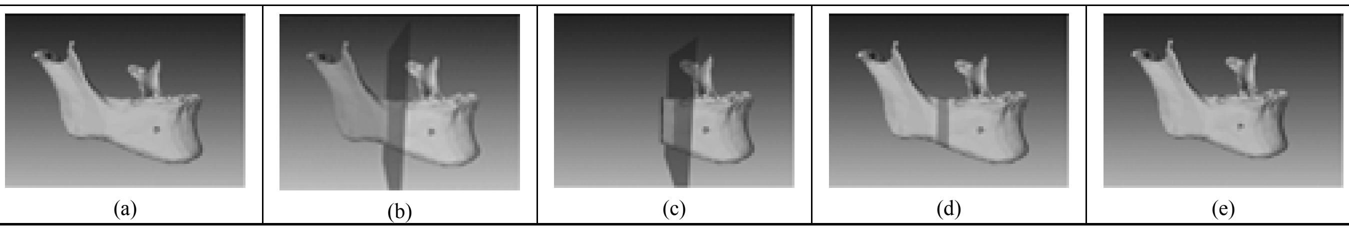

Figure A1: Use of planar cutting and manipulation tools on the human mandible from the visible human data set. The initial object is shown in (a). The planar cutting tool is used to cut it (b) and (c). The separated objects are shown together in (d), with the middle segment highlighted in a darker colour. In (e), the middle segment is removed and the front portion is re-positioned to close the gap.

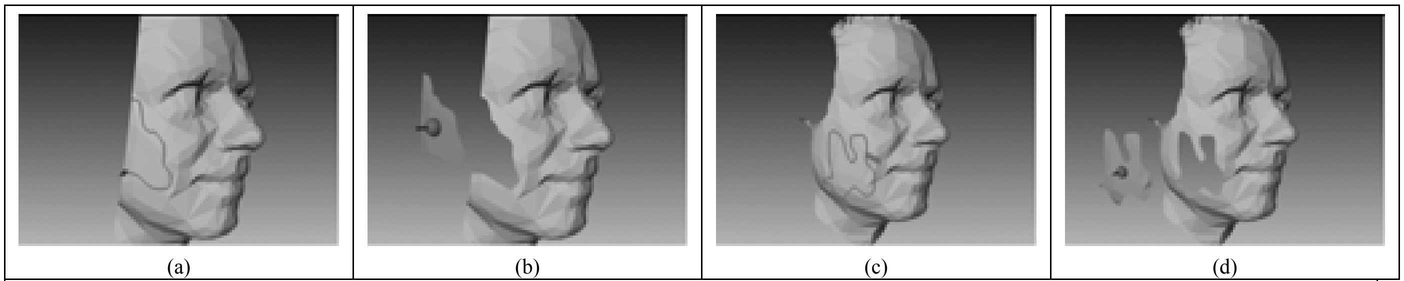

Figure A2: Examples of separable cuts which are: (a,b) open, and (c,d) closed. The mesh is the face of Ken Salisbury, provided with the GHOST API. Figures (a) and (c) show the cut contour on the mesh, and (b) and (d) show the separated cut fragments.

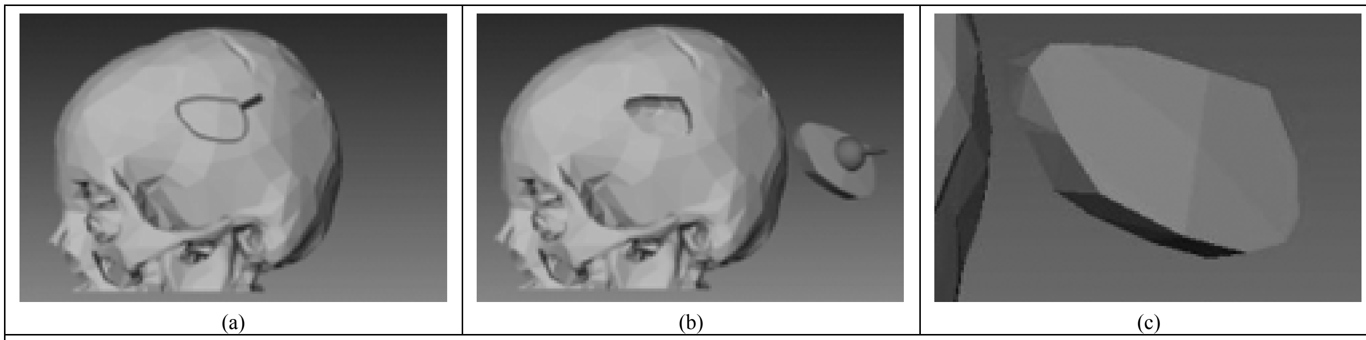

Figure A3: A solid contour cut through a skull bone. The bone structure was obtained from medical image data using Amira. Image (a) shows the traced cut contour. Image (b) shows the separated fragment being manipulated away from the skull. Image (c) shows a close-up of the separated fragment.

[1] *Amira* is a registered trademark of *Konrad-Zuse-Zentrum fur Infor-mationstechnik Berlin.*

[2] *Open Inventor* is a trademark of *Silicon Graphics Incorporated*.

[3] *GHOST* is a registered trademark of *SensAble Technologies, Inc*.

[4] *PHANTOM* is a trademark of *SensAble Technologies, Inc*.