A survey of interactive mesh-cutting techniques and a new method for implementing generalized interactive mesh cutting using virtual tools

ARTICLE *in* THE JOURNAL OF VISUALIZATION AND COMPUTER ANIMATION · FEBRUARY 2002

DOI: 10.1002/vis.275 · Source: DBLP

6 AUTHORS, INCLUDING:

Steven Senger
University of Wisconsin - La Crosse
21 PUBLICATIONS **230** CITATIONS

SEE PROFILE

Kevin Montgomery
Stanford University
44 PUBLICATIONS **776** CITATIONS

SEE PROFILE

Simon Wildermuth
Kantonsspital St. Gallen
178 PUBLICATIONS **4,468** CITATIONS

SEE PROFILE

Richard Boyle
NASA
115 PUBLICATIONS **1,870** CITATIONS

SEE PROFILE

THE JOURNAL OF VISUALIZATION AND COMPUTER ANIMATION
J. Visual. Comput. Animat. 2002; **13:** 21–42
Published online in Wiley InterScience (www.interscience.wiley.com). **(DOI:** 10.1002 / vis.275)

A survey of interactive mesh-cutting techniques and a new method for implementing generalized interactive mesh cutting using virtual tools‡

By Cynthia D. Bruyns, Steven Senger, Anil Menon, Kevin Montgomery, Simon Wildermuth and Richard Boyle*

In our experience, mesh-cutting methods can be distinguished by how their solutions address the following major issues: definition of the cut path, primitive removal and re-meshing, number of new primitives created, when re-meshing is performed, and representation of the cutting tool. Many researches have developed schemes for interactive mesh cutting with the goals of reducing the number of new primitives created, creating new primitives with good aspect ratios, avoiding a disconnected mesh structure between primitives in the cut path, and representing the path traversed by the tool as accurately as possible. The goal of this paper is to explain how, by using a very simple framework, one can build a generalized cutting scheme. This method allows for any arbitrary cut to be made within a virtual object, and can simulate cutting surface, layered surface or tetrahedral objects using a virtual scalpel, scissors, or loop cautery tool. This method has been implemented in a real-time, haptic-rate surgical simulation system allowing arbitrary cuts to be made on high-resolution patient-specific models. Published in 2002 by John Wiley & Sons, Ltd.

Received: 10 December 2001; Revised: 11 February 2002

KEY WORDS: modeling; simulation; procedural simulation and training; haptic interfaces

Introduction

Cutting is a common manipulation encountered in simulations such as surgical training, clothing design and CAD/CAM manufacturing. The literature contains numerous methods for cutting both surface and volumetric meshes.[1–19] Developers wishing to incorporate these methods into simulation systems may find the variety of methods overwhelming and may be uncertain how to decide which method is most appropriate for their needs. This introduction will attempt to categorize these methods and give some background on the strengths and weaknesses of each approach.

In our experience, cutting methods can be distinguished by how their solutions address the following major issues:

1. definition of the cut path,
2. primitive removal and re-meshing,
3. number of new primitives created,
4. when re-meshing is performed, and
5. representation of the cutting tool.

The following sections will organize existing cutting techniques and discuss their distinctions according to how they address the above major issues. Just as the range of applications varies, the methods of cutting virtual objects can be very different. Many researchers have focused on developing cutting schemes for interactive surgical simulation requiring manipulations to be performed at haptic speeds. While this is required for some applications, other applications may find that less computationally intensive cutting techniques are more appropriate.

*Correspondence to: C. D. Bruyns, National Biocomputation Center, Stanford University, 701A Welch Road, Suite 1128, Stanford, CA 94305, USA. E-mail: bruyns@biocomp.stanford.edu

Definition of the Cut Path

Cutting a virtual object requires disconnecting one mesh primitive from another. For an interactive cutting tool, the user must begin by inspecting the object and determining where the cut should begin. After selecting this starting point, the user then defines the shape of the *cut path*. Existing cutting tools determine the cut path by either:

1. placing seed points on the mesh,
2. placing a template through the mesh, or
3. moving a virtual tool through the mesh.

In the first approach, the user selects successive points on the object's surface. These points determine the basic outline of the cut path, but must be linked together to form a continuous cut through the object. Choices for linking the points include Dijkstra's shortest-path algorithm or constructing successive planes between the points. Using Dijkstra's shortest-path algorithm to link the seed points has the drawback that only existing vertices can be used to define the cut path. Unless a regular grid is used (Linton, unpublished), the resulting contour is frequently jagged (Lee, unpublished).[6]

Using a series of planes to define the Euclidean shortest distance between the seed points has the benefit of creating new vertices at the intersection of the plane and mesh edges; however, it requires a number of planes to produce a visually smooth cut (Montgomery, unpublished). One novel approach to linking seed points is to compute the geodesic path across the surface between the points. This approach can result in a smooth curve; however, this method has the drawback of requiring a high-quality mesh in order to solve the Eikonal equation (Barth, unpublished).

In the second method, the user has a predefined shape for the cut path represented by another object in the virtual world. The user interactively positions the shape creating the desired intersection with the object to be cut. When the shape is positioned a signal is made to cut the object.

In one implementation of the third method, the user traces the contour directly onto the mesh using a point-based representation of the cutting object.[15] In another implementation, the user moves a virtual tool in the world and the cut path is determined by successive intersections of the tool with the object primitives.

Figure 1 demonstrates each of these methods for defining the cut path. Table 1 catalogs previous cutting tools based on the way the cut path is described.

Primitive Removal and Re-meshing

Cutting techniques may also be classified based on the way the cutting operation is implemented. Two common methods are:

1. removing intersected primitives,
2. re-meshing intersected primitives.

Figure 2 demonstrates the implementation of both forms of handling tool and object intersection.

The first method simply removes primitives that intersect the cutting tool. The second method recreates the path traversed by the tool through the intersected primitives by re-meshing the intersected primitives, forming a gap in the mesh. The first method has the advantage of avoiding the creation of new primitives, thereby simplifying the overall operation. However, it has the drawback of creating cuts that might be visually disturbing, especially on irregular meshes.

The second method has the additional cost of computing the intersection path and model bookkeeping, but provides a better representation of the path traversed by the cutting tool. Typically *cut cases* are used to expedite re-meshing by allowing only certain combinations of edge and face intersections within a primitive. These cut cases are stored in a look-up table allowing each example of a cutting operation to be recreated from the basic cut.

Table 2 categorizes cutting techniques that use tool and object collision to define the cut path based on the methods used to represent the cutting operation.

Number of New Primitives or Primitives Created

Cutting techniques can be further classified according to the number of new primitives created during re-meshing. Existing cutting techniques handle re-meshing by either:

(a) disallowing new primitives,
(b) allowing unnecessary new primitives, or
(c) creating a minimal number of new primitives.

In the first method, selecting a subset of the mesh traversed by the tool creates the cut path. Then the vertices of the subset are repositioned moving the edges of the traversed faces to the cut path. Finally, the edges of the subset are then doubled, creating a gap in the mesh.[17]

Cutting tools that allow unnecessary new primitives

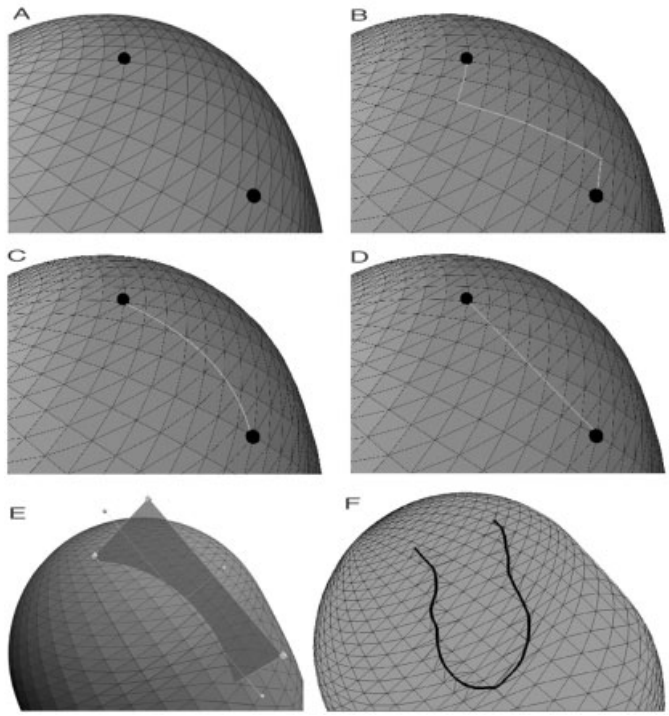

Figure 1. Definition of the cut path: (A) selection of seed points; (B) computation of Dijkstra's shortest path; (C) computation of geodesic shortest path; (D) Euclidean shortest path; (E) placing a template; (F) tracing the path directly onto the object

usually include their creation for mesh quality and symmetry purposes. As described in Bielser and Gross,[10] the additional tetrahedra come from automatically inserting an extra node into a completely cut face. The addition of this node allows for a more symmetric subdivision and additional degrees of freedom in the cut surface. Adding these nodes to two adjacent faces, however, can cause the formation of intersecting tetrahedra. These tetrahedra are handled during re-meshing by incorporating additional tests for their intersection and choosing an alternative re-meshing template that would avoid self-intersection.

Schemes that take the third approach minimize the number of new primitives by only creating new vertices at the point of intersection between the tool and primitive. Since the intersection point can occur very close to the original vertices in the mesh, it is possible to create primitives having edges of widely varying lengths. Such primitives can be problematic for subsequent numerical calculations. To mitigate this problem, one can either shift the intersection location to the nearest vertex or modify these primitives after re-meshing. Despite the additional mesh-quality steps that might be required, minimal new primitive schemes have the advantage of reducing the overall number of primitives in the mesh after the cutting operation.

Table 3 categorizes cutting techniques that re-mesh the primitive based on the number of new primitives created.

Placing seed points	Placing a template	Moving a tool
Linton, 95	Pieper, 92	Tanaka, 98
Lee, 98	Keeve, 96	Gibson, 98
Wong, 98	Delp, 97	Basdogan, 99
Barth, 98	Schutyser, 00	Bro-Nielson, 99
Montgomery, 00		Roux, 99
		Bielser, 00
		Cotin, 00
		Ganovelli, 00
		Neumann, 00
		Mor, 00
		Bruyns, 01
		Nienhuys, 01
		Kuhnapfel, 01

Table 1.

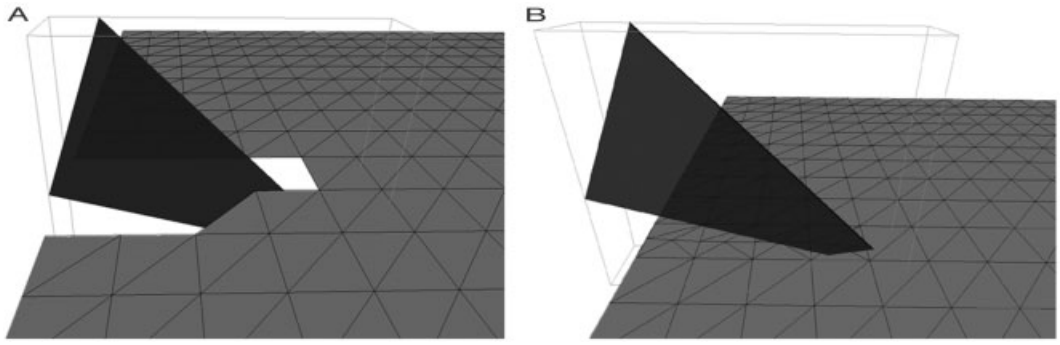

Figure 2. Handling of re-meshing: (A) removal of intersected primitives; (B) re-meshing the intersected primitives

When Re-meshing is Performed

The cutting techniques that try to minimize the number of new primitives can be further categorized based on the time that they perform the re-meshing operation. Re-meshing typically occurs:

(a) while the tool and the primitive are in intersection,
(b) after the tool is no longer intersecting the primitive, or
(c) after the tool has changed direction.

One can re-mesh a primitive along the cut path, as soon as the cutting instrument intersects it, as in progressive cutting; as soon as the instrument and primitive are no longer in collision, as with non-progressive cutting; or after the user has moved the tool out of the direction it was traveling in at the previous iteration.

Table 4 categorizes cutting techniques based on time that the primitive is re-meshed.

Representation of the Tool

The cutting techniques that simulate motion of the tool through the virtual environment and handle re-meshing can be further categorized based on the representation of the virtual tool. The tool is usually modeled as:

(a) a single point of intersection;
(b) an ideal object;
(c) an object consisting of multiple primitives.

When modeling the tool as an ideal object, usually a single edge or triangle is used, allowing for simplified intersection tests.

When modeling the cutting tool as an object with multiple primitives, a central axis is usually used for intersection tests. This reduces the tool to a simplified model for computation, but retains the complex model for rendering.

Remove intersected primitives	Re-meshing
Gibson, 99	Basdogan, 99
Tanaka, 98	Bro-Nielson, 99
Roux, 99	Bielser, 98
Cotin, 00	Ganovelli, 00
Künhapfel, 01	Mor, 00
	Neumann, 00
	Bruyns, 01
	Nienhuys, 01

Table 2.

No new primitives	Near minimal	Minimal number
Nienhuys, 01	Bielser, 00	Basdogan, 99
		Bro-Nielson, 99
		Ganovelli, 00
		Mor, 00
		Neumann, 00
		Bruyns, 01

Table 3.

Change-in-direction	Non-progressive	Progressive
Neumann, 00	Bielser, 00	Mor, 00
	Basdogan, 99	
	Bro-Nielson, 99	
	Ganovelli, 00	
	Bruyns, 01	

Table 4.

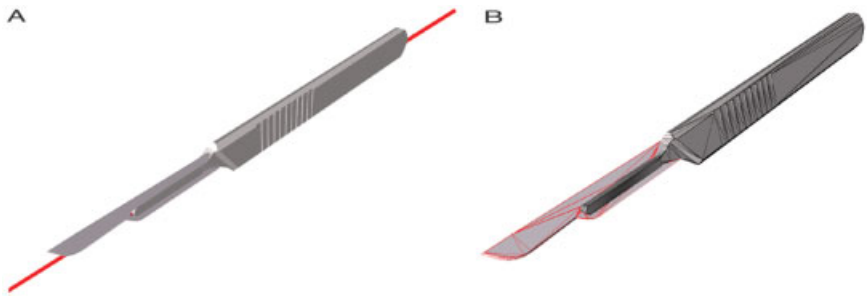

Figure 3. Representation of the cutting tool: (A) object using a medial axis; (B) object using tool features

The tool used in Figure 2 demonstrates the use of an idealized representation of a virtual instrument. Figure 3 demonstrates a complex tool using a medial axis, and a complex tool using actual tool features.

Table 5 categorizes cutting techniques that re-mesh the object and use a tool to define the cut path based on the cutting tool representation.

Summary

Many researches have developed schemes for interactive mesh cutting with the goals of reducing the number of new primitives created, creating new primitives with good aspect ratios, avoiding a disconnected mesh structure between primitives in the cut path, and representing the path traversed by the tool as accurately as possible.

Since cutting schemes are often linked with physically based models of the object being cut, there is a considerable penalty for meshes with unnecessary primitives.[20] That is because the running time of dynamic solvers is on the order of the number of primitives in the mesh. Therefore including more primitives than needed can be costly, particularly when using solvers that are already prone to long running times, such as Finite Primitive Analysis. An extreme example of avoiding the number of new elements created is presented in Nienhuys and van der Stappen.[17] Although this procedure prevents the creation of additional primitives in the mesh and avoids new primitives with poor aspect ratio, the creation of new primitives cannot always be avoided. Moreover, moving existing nodes in the mesh will inevitably change the original shape of the object.

Single point	Single cutting surface	Multiple cutting surfaces
Bruyns, 01a	Basdogan, 99 Bro-Nielson, 99 Bielser, 00 Ganovelli, 00 Mor, 00	Bruyns, 01b

Table 5.

Additionally, the stability of dynamic solvers such as Finite Element Analysis is strongly influenced by the quality of the elements in the object mesh.[20] Attempts to regulate the aspect ratios of the new primitives created have been implemented in schemes[10] where midpoint nodes are added to intersected faces. A drawback of creating self-intersecting tetrahedra when two adjacent faces are completely intersected was described in Bielser and Gross.[10] However, in our experience, by alternative placement of edges interior to the original tetrahedra, these intersections do not occur and subsequent intersection testing is not necessary. In addition, Nienhuys and van der Stappen[17] demonstrated the use of node snapping to control the shape of the new primitives. Controlling the primitive shape during re-meshing using node snapping comes at the expense of additional tests for element stability and iterations through subdivision templates until a stable configuration is found. Additionally, since cutting schemes try to represent the path traversed by the cutting instrument through the object's mesh as accurately as possible, schemes that move the tool-to-object intersection locations by node snapping do not provide an accurate representation of the path the tool followed through the mesh.

Attempts to avoid the creation of gaps formed in the mesh between adjacent primitives along the cut path have led to the development of progressive cutting schemes. The factors to consider when choosing between progressive and non-progressive cutting are the stiffness of the object and size of the primitives being cut. For example, when cutting a rigid object all

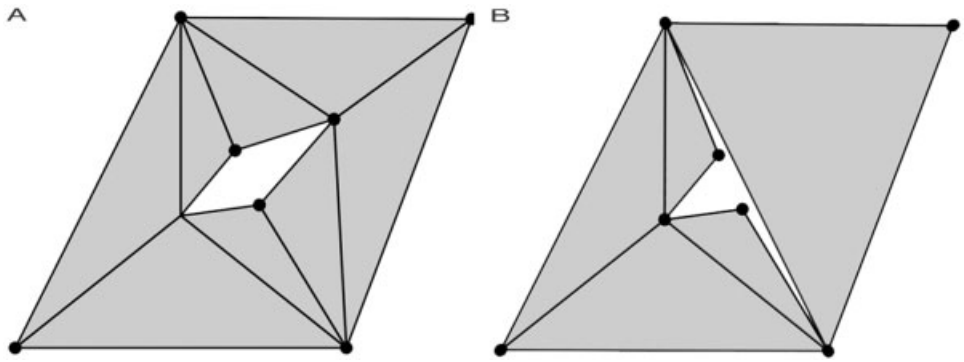

Figure 4. Time re-meshing occurs: (A) progressively; (B) non-progressively

of the above choices would produce a surface that looks connected. However, when cutting a deformable surface under high tension, gaps can be seen between primitives the tool has just moved through and primitives the tool is currently intersecting. Figure 6 demonstrates the implementation of both types of cutting method. Notice that in Figure 4(A) the mesh structure remains complete, providing an actual point that represents the cut front. However, in Figure 4(B), non-progressive-based cutting has created disconnected nodes, as seen in the triangle on the left.

Progressive cutting is also useful when trying to have the cut path follow behind the cutting instrument as closely as possible. For example, when cutting an object whose primitives are large in comparison to the overall size of the object, the delay between the intersection of a surface and creation of the cut front may be visually misleading. Figure 5 demonstrates this point. In Figure 5(A) the object's surface is tiled with triangles whose area is 0.2% of the overall object size. When cutting this object, the cut front appears to be close to the point of intersection of the tool and the surface. However, in Figure 5(B), the triangles are 3% of the overall surface area of the object. When cutting this object, the cut front appears to lag behind the point of intersection.

Progressive schemes can avoid the problems associated with meshes containing large triangles. However, the need for progressive meshing has not been demonstrated when using high-resolution patient-specific models. High-resolution models, unlike generic models, generally do not contain large primitives, so that the lag between the tool location and cut path is normally not detectable. Moreover, using a scheme similar to that in Neumann,[11] one could resolve motion inside large triangles by including a change-in-direction signal to the re-meshing algorithm. Using such a signal could inform the re-meshing algorithm that large deviations in the cut path inside an intersected primitive should be recorded, allowing the path to follow the actual user motion as closely as possible. Figure 6 shows how motion inside a large triangle can be resolved without the need for progressive cutting schemes.

Of the cutting schemes presented, the common element missing is a unifying procedure giving the user:

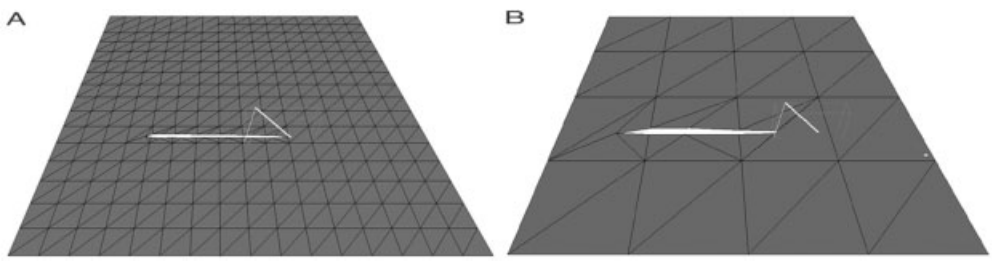

Figure 5. Appearance of lag using non-progressive cutting: (A) object tiled with small triangles; (B) object tiled with large triangles

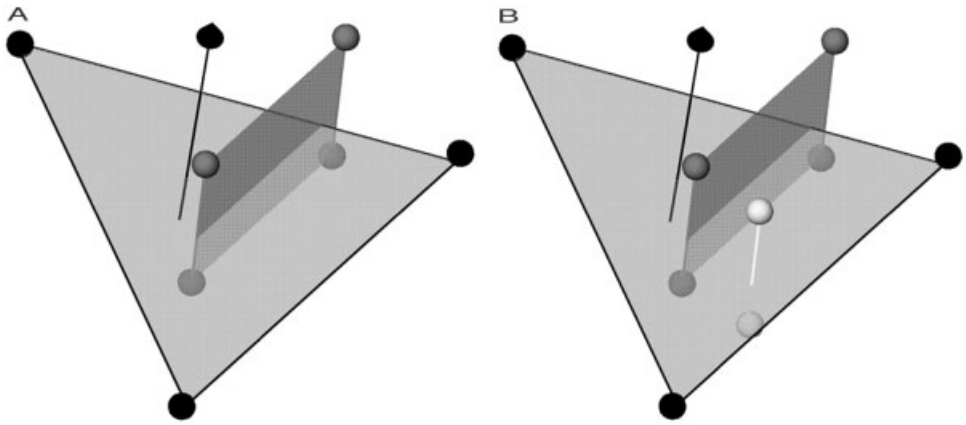

Figure 6. Change in direction interior to a primitive: (A) average plane swept between successive sample locations; (B) sample outside average plane signals change in direction

- the ability to cut various mesh primitives within the same system,
- the ability to use tools with multiple cutting surfaces, and
- demonstrated ability to cut realistic objects from patient-specific data.

The goal of this paper is to explain how, by using a very simple framework, one can build a generalized cutting scheme. This method allows for any arbitrary cut to be made within a virtual object, and can simulate a cutting surface, layered surface or tetrahedral objects using a virtual scalpel, scissors, or loop cautery tool. This method has been implemented in a real-time surgical simulation system allowing arbitrary cuts to be made on large patient-specific models.

Methods

We have implemented our cutting scheme in a real-time surgical simulation environment.[21] The basic simulation loop, which allows us to model realistic tool and tissue interaction, is outlined in Figure 7. The following sections will explain the implementation of each of the phases depicted in the figure.

Object Deformation

For deformable objects the simulation system considers the nodes as point masses and edges as springs/dampers to form a 3D mesh for mass–spring simulation. These edges can be considered as linear, piece-wise linear, or non-linear 1D springs/dampers. Each edge can have different spring and damping coefficients and the nodes can provide different mass distributions to provide for some support of anisotropic, heterogeneous tissues.

A number of numerical methods have been implemented ranging from traditional Euler to Runge-Kutta (second and fourth order). In addition, a quasi-static method, appropriate for heavily damped tissues and low interaction velocities, assumes the tissue to always

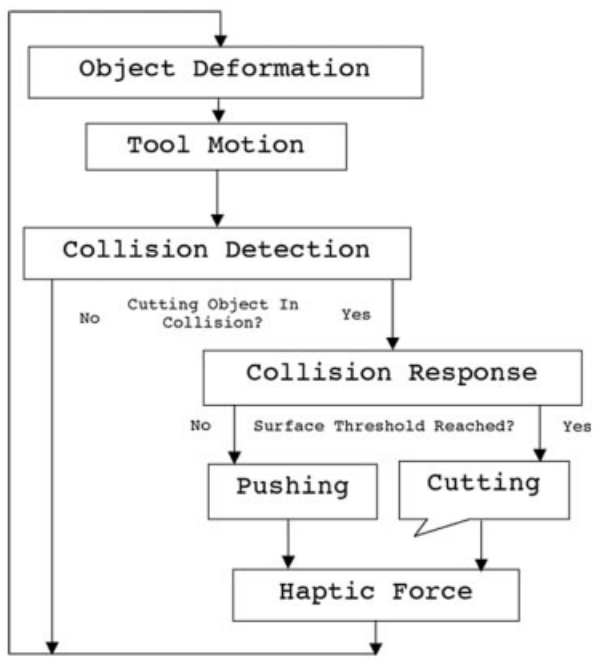

Figure 7. interaction loop

be in static equilibrium and ignores dynamic inertial and damping forces for a corresponding increase in simulation performance.[22] The ordered quasi-static method also has the benefit of processing the object local to the contact areas.

Tool Motion

We have adopted a distributed network approach to integrating various user input devices with the simulation system. Each user device (position sensor, force feedback, etc.) is directly connected to a (possibly dedicated) computer running a device control program. This program communicates with the simulation server over the network. This allows us to decouple the device interface restrictions (e.g. specific PC hardware) from the simulation itself.

When a network user interface device is instantiated on the server to correspond to a virtual device, it forms a TCP/IP socket connection with the device control program. During operation of the system, the device control program streams position, orientation and activation values to the simulation server. In return the simulation server computes, if appropriate, force information for the virtual device. This information is streamed back to the device control program for presentation to the user. A discussion on haptic force computation will be presented below. This method of network-based sensor and device control inherently provides for multi-user and multi-instrument interaction and supports collaborative procedures.

Collision Detection

Each object within the virtual environment has a tree of hierarchical bounding volumes. Deformable objects typically use bounding spheres because the update of bounding volume is simple to implement once the object has undergone deformation.[23]

The most basic cutting instrument is a straight, rigid object that has one sharp surface, such as a scalpel. When modeling this object, one can either automatically compute the sharp regions of the mesh or manually choose which areas will be allowed to cut based on inspection of the model and included as information within the object's data file. We choose to represent this information as a list of mesh primitives that are allowed to cut. In this way, virtual instruments can perform given *behaviors* such as cutting, probing, ablating, and cauterizing when in contact with a virtual tissue. We call these primitives that have been assigned behaviors *Active Primitives*.[24] For a scalpel, we choose a single edge to be active. Figure 3(B) demonstrated the selection of active edges.

Even if an object uses one primitive for rendering, for the sake of collision detection we can create bounding volumes around the other primitives to directly obtain the information important to cutting.

To avoid the possibility of missed collisions due to the discrete sampling of the tool position, we choose to detect collisions not with the tool edge but with the surface swept by consecutive positions of the tool edge.[25] By enclosing the swept surfaces of the active edges with bounding volumes, we can still exploit the benefits of a binary search tree. Figure 8 demonstrates the benefit of using a swept surface collision detection scheme for an active edge. In Figure 8(A), successive sample locations allow the edge to pass completely through the face, while in Figure 8(B), by including the swept surface, we detect the object-to-object penetration.

Currently, we store intersection information as

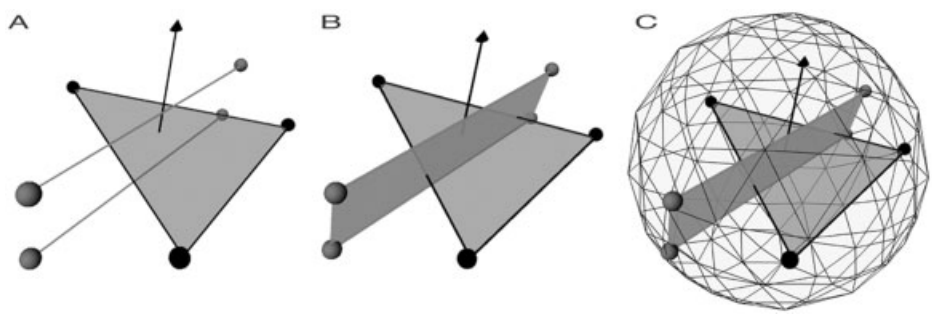

Figure 8. Edge collision detection: (A) edge passing through an object using line segment test; (B) inclusion of swept surface detects collision; (C) inclusion of the swept surface in the bounding sphere hierarchy

collision pairs containing pointers to the objects that were in collision, the point at which collision happened, and the primitives involved in the intersection. This list of collision pairs is then passed to the collision response mechanism of the tool and provides the necessary information to perform a pushing or a cutting manipulation.

Collision Response

The collision response implemented for a tool is dependent on the tool's intended behavior, its direction of movement and the dynamic state of the virtual object it collides with. For example, when a virtual tool has penetrated a deformable object, the surface displaces until the applied force becomes larger than the yield limit of the material being simulated.[26] When the breaking strength of the material is reached, further tool motion causes the object to be cut.

Designating the sharp edges of a tool automatically determines a cutting direction for the tool. For example, motion of a scalpel along the cut direction allows the cutting action to be implemented, while motion out of the allowed range causes the object to perform a pushing action. A method for applying penalty forces for a haptic interface using a tool's cutting direction is described in Bielser and Gross.[10] We extend this concept to trigger different object behaviors for virtual instruments.

Pushing. When a tool collides with a deformable object, the object's surface is moved in a direction determined from the intersection point and the direction of tool motion. We resolve the collisions in real time by displacing nodes of the deformable object until the objects are no longer in collision.

Figure 9 illustrates the ability to model initial deformation of the object before the surface yield limit is reached.

Figure 10 illustrates triggering different object behaviors based on motion outside of the allowable cutting direction. In Figure 10(A), motion along the cut direction causes the tool to perform the cutting action, while in Figure 10(B) motion outside of the cutting direction causes the tool to perform the pushing action.

Cutting. The cutting operation is implemented as various stages, as indicated in Figure 11. The sequence of stages followed at each iteration depends on the primitive and active edge collision state. If the primitive and the active edge are still in collision, then the location of the intersections recorded and the haptic force are computed. If the active edge and the primitive are no longer in collision, then the primitive is

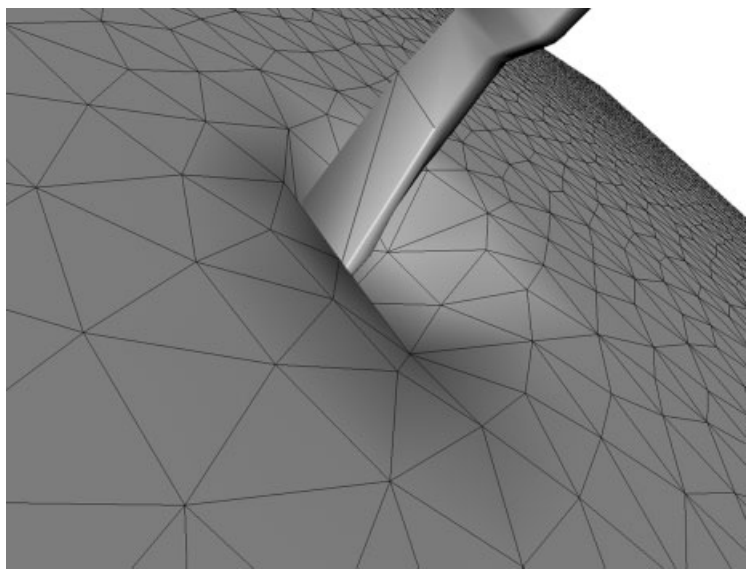

Figure 9. Initial contact with the surface

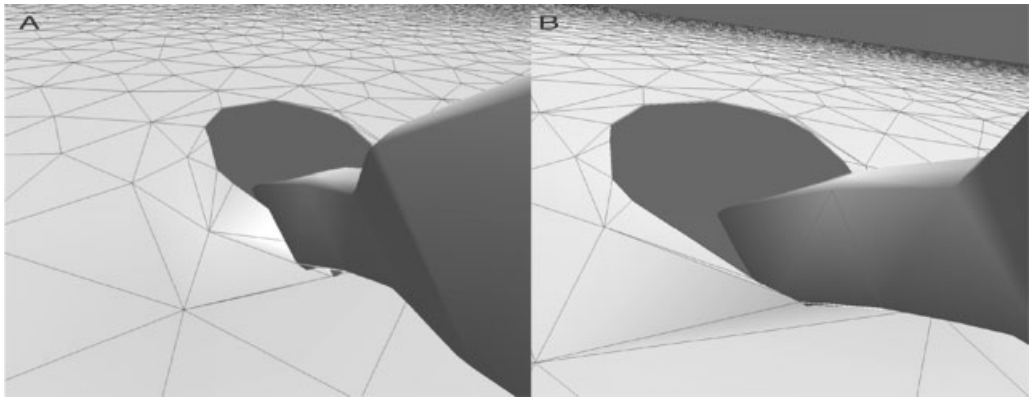

Figure 10. Tool motion: (A) out of plane does not create a cut; (b) instead it pushes the surface to the side

re-meshed. The following sections describe the implementation of the primitive testing and re-meshing phases.

Primitive Update. Each primitive stores information regarding the path traversed through it by the tool. Figure 12 shows the stages for updating the primitive.

We choose to test the edges of the primitive against the swept surface of the active edge. This plane is based on the current location of the active edge, and the location of that edge at the previous iteration. Figure 13(A) demonstrates the use of the swept plane and the intersection test with the primitive edges.

The test for collision between a primitive face and an active edge is illustrated in Figure 13(B). If an intersection is made within a primitive's face, the intersection point is recorded. Similar to the primitive edge, the position of the intersection point is updated each iteration the face is intersected. In this way the most recent intersection locations are used for re-meshing.

Each object stores a list of pointers to the primitives that were intersected by the cutting instrument at the last iteration. For each active edge in the instrument, each primitive in this list is tested for intersection and updated. In the simple case of a cutting instrument moving through a flat surface, the list is a single primitive, the primitive in contact with the active edge, which is being updated as the cutting instrument moves. However, because cutting is based on primitive states and not an object-wide state, it is possible to cut complex surfaces with folds, and multiple layers.[27] In these cases, the list of last primitives intersected might contain non-adjacent primitives and primitives with varying properties.

When cutting a volumetric object, tool motion is usually not as straightforward as moving across a

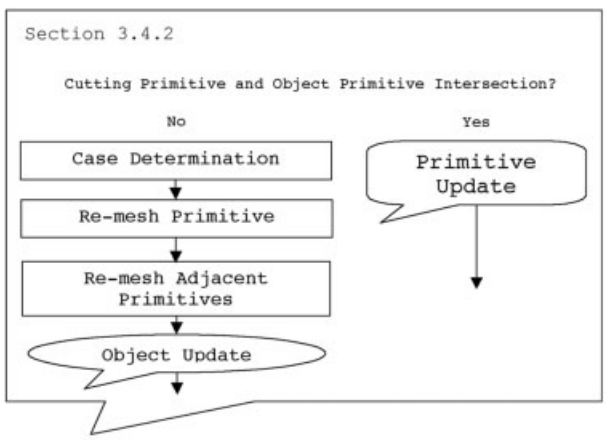

Figure 11. Cutting loop

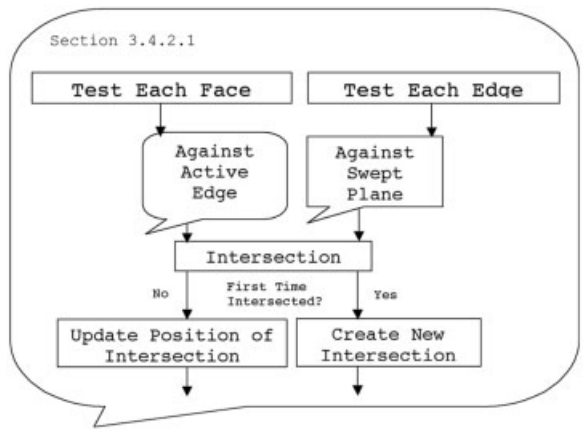

Figure 12. Primitive update

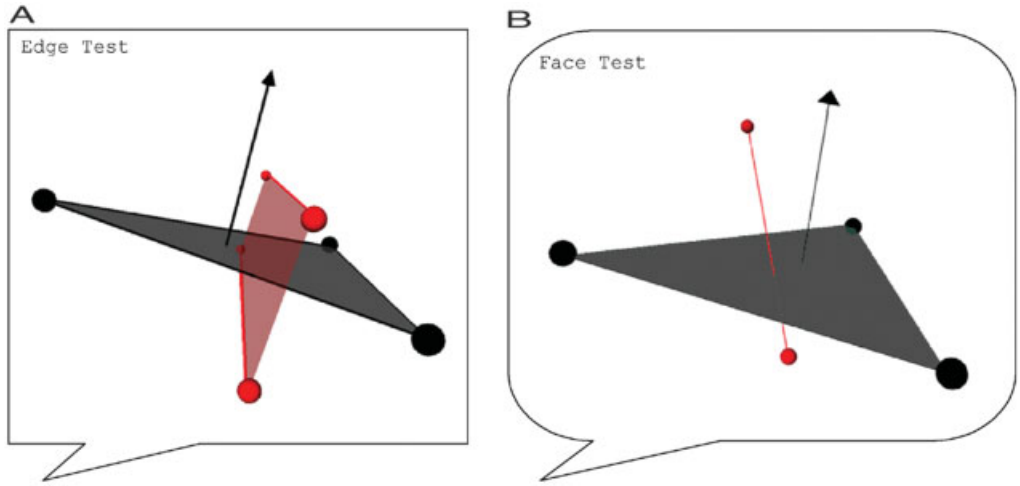

Figure 13. Primitive tests: (A) edge test; (b) face test

surface or layered surface. In general the list of intersected primitives will contain several volumetric primitives.

Case Determination. After the tool has left the primitive, the last configuration of edge and face intersections is evaluated and the primitive is re-meshed based on the allowable cutting template cases.

For objects composed of triangles, Figure 14 shows the allowed cases. In Figure 14(B) the mid-node created between the two edge intersections is not necessary; however, creating these mid-nodes allows for a common implementation between triangle and tetrahedral re-meshing.

Figure 15 demonstrates the tetrahedral re-meshing cases. Notice that the combination of face cases determines the tetrahedral cases. These cases are similar to those in Bielser and Gross[10] but do not have the problem of creating intersecting tetrahedra, eliminating the need for additional self-intersection tests.

Re-mesh Primitive. Once a valid case has been determined, nodes are created at the intersection points. For intersections interior to a face, one node is created. For intersections on an edge, two nodes are created. New edges and faces are then created, replacing the original primitive from the object mesh.

Re-mesh Adjacent Primitives. In order to avoid the gaps created when using non-progressive meshing, we also re-mesh the primitives adjacent to the original

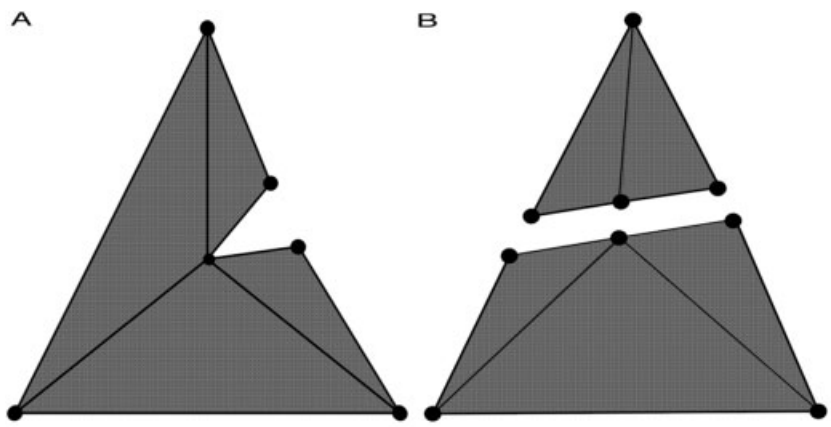

Figure 14. Face cases

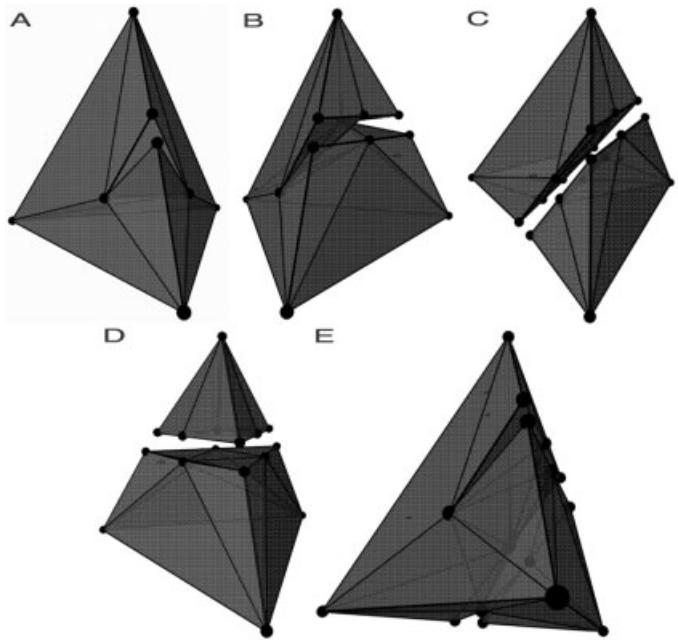

Figure 15. Tetrahedral cases

primitive. Because the tool will most likely be in collision with one or more of these adjacent primitives at the next iteration, we flag the 'non-original' features in the temporary elements so that they are not tested for intersection. Because we do not re-mesh these temporary primitives at each iteration, this method is not implemented as previous progressive schemes. However, we solve the problem of creating gaps in the mesh by re-meshing the primitives that share the split edges and faces. In this way, the cut path remains *stalled* at the edges of the primitive the tool just left, until the tool completely passes through the adjacent primitives, at which time the cut path advances as before. Should the tool leave contact with the object at the location of the adjacent primitives, they are left in their temporary re-meshing configuration, as they would normally be re-meshed to close the cut path. Figure 16 illustrates the effect of automatically re-meshing the adjacent primitives in order to keep the mesh structure intact. Figure 16 further details the features of the re-meshed adjacent primitives that are flagged for exclusion from subsequent intersection testing.

Object Update. Because cutting adds new nodes to the object mesh, there are several other steps that must occur in order to maintain a realistic object representation. These steps are outlined in Figure 17.

Mass Redistribution. We assign nodal masses to the object when the simulation begins. This mass is assigned based on the fraction of Voronoi area (or volume) adjacent to the node to the overall surface area (or volume) of the object. When creating new nodes, we preserve mass distribution by reassigning nodal masses based on the change in Voronoi region of the nodes enclosing the newly created nodes. In this manner the original nodes *donate* their mass to the newly created nodes. Equation (1) describes the method used for reassigning nodal masses to existing nodes and assigning mass to newly created nodes. Figure 18 demonstrates the computation of these quantities.

$$n_i.adjacent\_v\_area = \sum_{j=0}^{n_i.num\_adjacent\_triangles} n_i.triangle_j.contribution\_to\_v\_area$$

$$n_i.change\_in\_v\_area = \frac{n_i.v\_area\_before\_retriangulation - n_i.v\_area\_after\_retriangulation}{n_i.v\_area\_before\_retriangulation}$$

$$n_i.new\_mass = 1 - n_i.change\_in\_v\_area * n_i.mass$$

$$n_{new}.mass = \sum_{i=0}^{n_{new}.num\_adjacent\_triangles} n_i.change\_in\_v\_area * n_i.mass \quad (1)$$

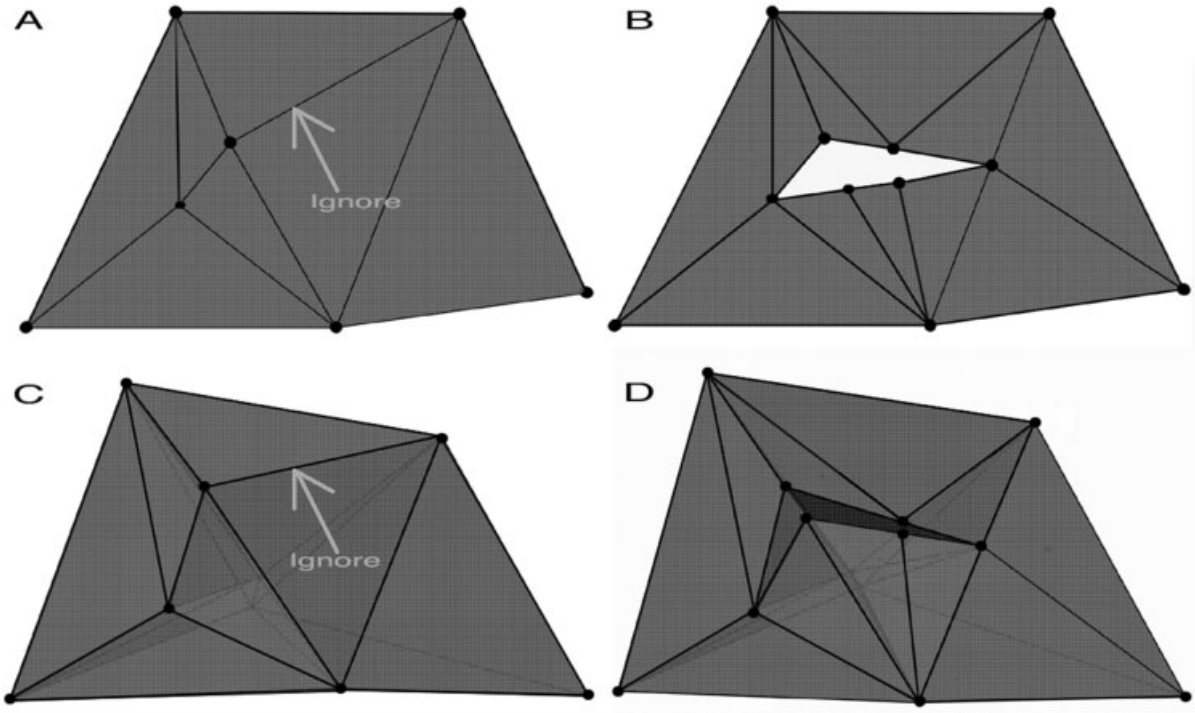

Figure 16. Meshing adjacent primitives and features marked for exclusion from intersection testing: (A) triangle on the left is re-meshed along with the adjacent triangle; (B) once the tool has exited the adjacent triangle, the split becomes apparent; (C) tetrahedra on the left is re-meshed along with adjacent tetrahedra; (D) once the tool has exited the adjacent tetrahedra, the split becomes apparent

If mass is not redistributed, but simply added with the creation of new nodes, then the mass of the object would increase proportionally with the size of the cut. In addition to violating basic physical laws, the contours of the cuts would look jagged because of the unbalanced distribution of mass in the mesh. We perform this operation locally, avoiding the need to iterate over all of the primitives in the object when a primitive is cut.

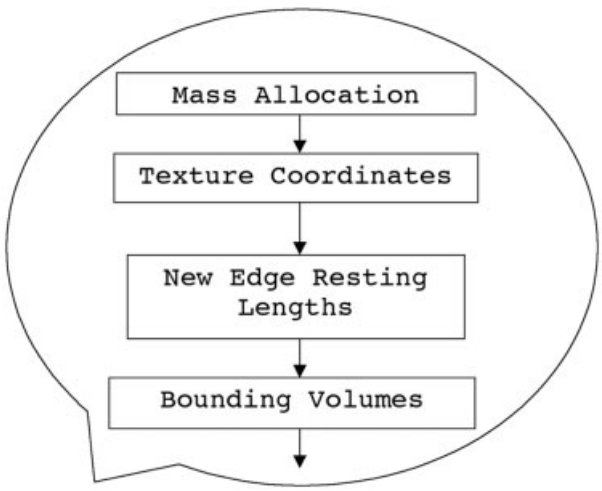

Figure 17. Object update

Texture Coordinate Interpolation. Similarly, coordinates for the newly created nodes are interpolated using a weighted linear interpolation scheme. Equation (2) describes the calculation of texture coordinates for the newly created nodes in the center of a triangle. For nodes created on an edge, only the nodes that make up the edge need to be included in the interpolation calculation.

$$node_{new}.s,t = \frac{\sum_{i=0}^{3} node_i.s,t * (1 - |dist(node_{new}, node_i)|)}{\sum_{i=0}^{3} |(1 - dist(node_{new}, node_i)|} \quad (2)$$

For volumetric primitives, texture coordinate creation can be handled if 3D texture data is available. Using

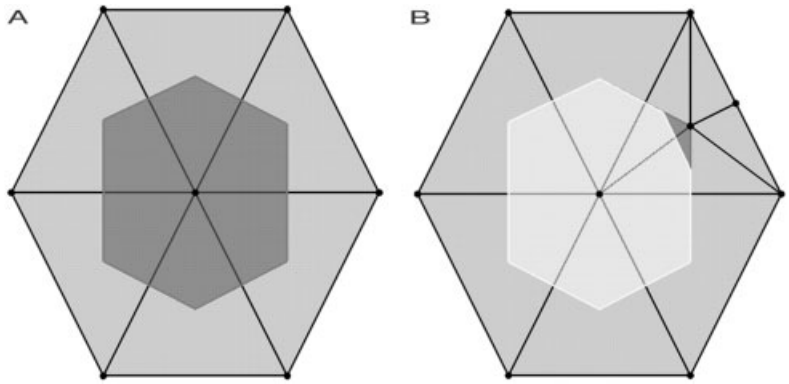

Figure 18. Mass redistribution: (A) Initial Voronoi area adjacent to a node; (B) new area after cutting determines change in mass of the node

this data one can obtain the new texture coordinates for the newly created nodes in the regular parametric fashion.

Resting Length Assignment. In order to preserve original edge dynamics, the resting lengths of the newly created springs are interpolated according to the relative location of intersection along the original edge length.

$$edge_{new}.rest\_length = \left(\frac{edge_{new}.length}{edge_{original}.length}\right) * edge_{original}.rest\_length \quad (3)$$

Similar to mass redistribution, if the resting lengths of the new edges are not computed as a function of the original edge resting lengths but merely assigned to be the length of the edge created by the point of intersection, then the edges would not split apart at the cut location. Instead relaxation of the cut would occur in the surrounding primitives under tension.

Bounding Volume Update. As primitives are removed from the mesh, their associated bounding volume is removed from the bounding volume hierarchy and is not included in subsequent collision detection tests. For newly created primitives, leaf nodes are created and added to the tree locally. Because each primitive has a pointer to its bounding volume leaf, we do not need to traverse the entire tree to determine where to add these new leafs, since the new primitives will replace the primitive removed. We locally sort these new primitives and add them to the tree starting at the parent node of the primitive removed. We do not process the update of the bounding volume hierarchy at each cut operation, since addition of new primitives that fill the same area (volume) of the original primitive that was removed from the tree will not change the size of the parent nodes above these leafs. As a result we can forgo updating the bounding hierarchy until the cutting operation is finished.

Rendering Haptic Force

Because each instrument in the virtual environment is controlled by the user, the method of user interface and the presence of haptic feedback has a large bearing on the reality of a given simulation. For example, the PHANTOM is a natural interface for simulating the control of a straight rigid object such as a scalpel, while a Laparoscopic Impulse Engine (LapIE) is a natural interface for controlling an instrument with a handle that can open and close, such as a pair of scissors. More importantly, each device can render a force to the user as they are interacting with the environment, allowing them to feel the result of tool and tissue contact. Our simulation supports many haptic feedback interface devices (SensAble PHANTOM and Immersion LapIE, 3GM, and BiManual LapIE).

In the previous section we described how we could allow the instrument to perform manipulations on the virtual object at multiple points of contact; however, conveying the result of multi-point contact is still a limitation for haptic feedback devices. Because of this, even though the tool makes contact with the model at multiple points of contact, we render a single resultant force to the user. To do this, we compute the force vectors at each point of contact and average them to apply a single force vector.

Results

The results presented in this section are derived from running the simulation on one processor of a Sun Microsystems (Mountain View, CA) E3500 8 × 400 MHz UltraSparc station. Since each tool is an extension of an instrument with one active edge, we provide examples of cutting various mesh representations with a scalpel. We present the results of interactive cutting according to the time to perform the various stages of the cutting operation using a scalpel on a surface, a multi-layered surface and volumetric objects. We also report the number of elements cut and the number of elements created for comparison with other methods. To give examples of using multiple active edges, we present the results of cutting a surface-based mesh with scissors and with a loop cautery tool.

Scalpel Cutting

Single Surface. Figure 19 demonstrates the use of a single sharp edge of a scalpel consisting of 772 triangles to cut a single deformable shoulder composed of 12,000 triangles. Table 6 shows the time to compute the various stages of the interaction loop. The first column describes the cumulative number of primitives that have been cut for a given iteration, the second shows the number of permanent primitives that have been created. The third column reports the **time to perform the interaction loop while cutting** one primitive at a time.

Number cut	Number created (permanent)	Time during cutting (ms)
0	0	242
1	10	20
25	79	26
50	151	36
75	230	42
100	308	54

Table 6.

Table 7 provides a further description of this value based on time to update the deformable object, tool motion and update of the collision data structures time to perform collision detection and collision response. The second column provides more detail as to the actual number of nodes the equations of motion are computed for. The last two columns provide more detail about the time to perform collision response partitioned to time to perform the cutting operation and time to order the nodes for use with the ordered quasi-static method.

Multiple Surfaces. Figure 20 demonstrates the use of a single primitive to cut multiple surfaces. In this

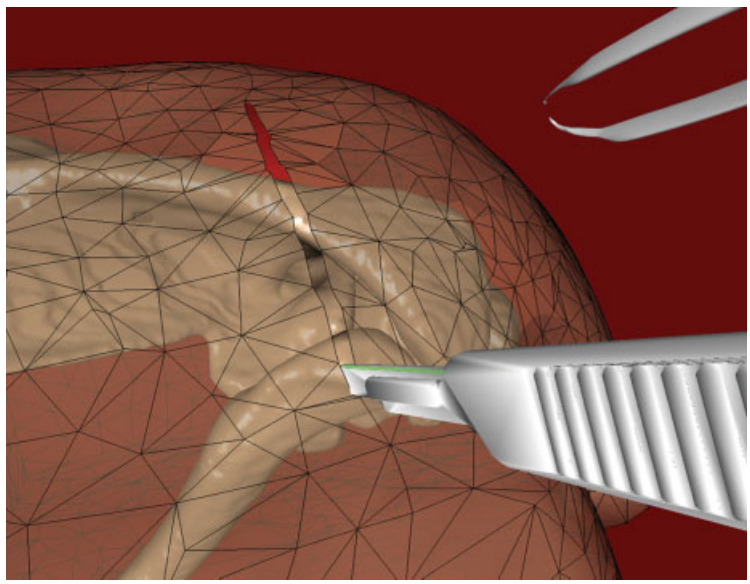

Figure 19. Cutting a single surface with a scalpel

Time during cutting						
					Collision response	
Number cut	Object update (seconds)	Number of nodes updated	Collision detection (seconds)	Collision response (seconds)	Time to cut (seconds)	Time to order nodes (seconds)
0	9.830E-04	0	2.409E-01	9.40E-05		
1	2.008E-03	83	7.161E-03	1.113E-02	1.603E-03	0.952E-02
25	7.004E-03	732	6.878E-03	1.178E-02	1.791E-03	0.999E-02
50	1.572E-02	2084	6.860E-03	1.313E-02	1.863E-03	1.126E-02
75	2.207E-02	3059	6.797E-03	1.283E-02	1.922E-03	1.091E-02
100	3.601E-02	5240	6.918E-03	1.093E-02	1.820E-03	0.911E-02

Table 7.

example, the 12,000-triangle shoulder mesh has been extruded, creating two layers connected by springs. The resulting multi-layered object is composed of 24,000 triangles.

Table 8 provides the results for cutting the multi-layered object.

Table 9 provides more detail about the time to perform the cutting operation.

Volume. Figure 21 shows a scalpel cutting a male pelvis composed of 25,000 irregular tetrahedra. The surface created by re-meshing the tetrahedra as the tool passes through the volume is highlighted. The fidelity of this surface demonstrates that the re-meshing algorithm accurately represents the path the tool followed through the mesh, and it gives an example of creating deeps cuts into the deformable tissue.

Number cut	Number created (permanent)	Time during cutting (ms)
0	0	510
1	4	87
25	80	86
50	155	103
75	232	111
100	305	128

Table 8.

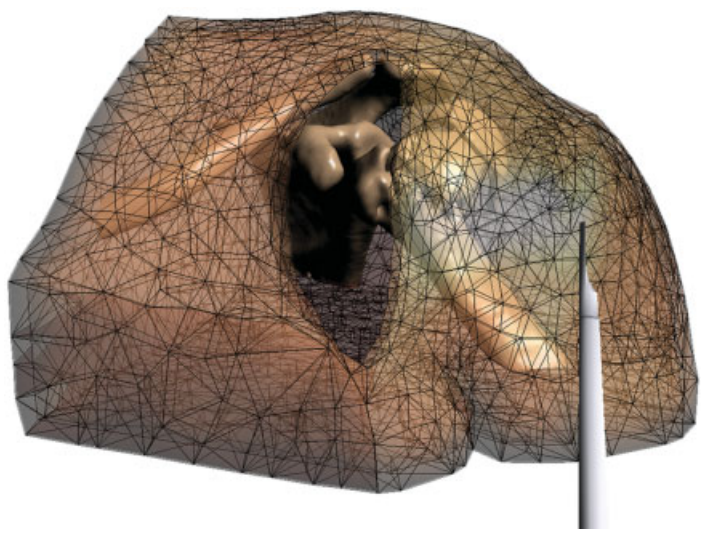

Figure 20. Cutting a multi-layered surface with a scalpel

Time during cutting						
					Collision response	
Number cut	Object update (seconds)	Number of nodes updated	Collision detection (seconds)	Collision response (seconds)	Time to cut (seconds)	Time to order nodes (seconds)
0	6.570E-04	0	5.089E-01	1.960E-04		
1	2.304E-03	62	2.697E-02	5.797E-02	4.797E-03	5.317E-02
25	1.149E-02	1304	2.441E-02	5.054E-02	5.733E-03	4.480E-02
50	2.925E-02	3125	2.598E-02	4.807E-02	5.586E-03	4.249E-02
75	4.132E-02	4586	2.230E-02	4.763E-02	5.524E-03	4.211E-02
100	4.770E-02	5331	2.987E-02	4.996E-02	5.774E-03	4.419E-02

Table 9.

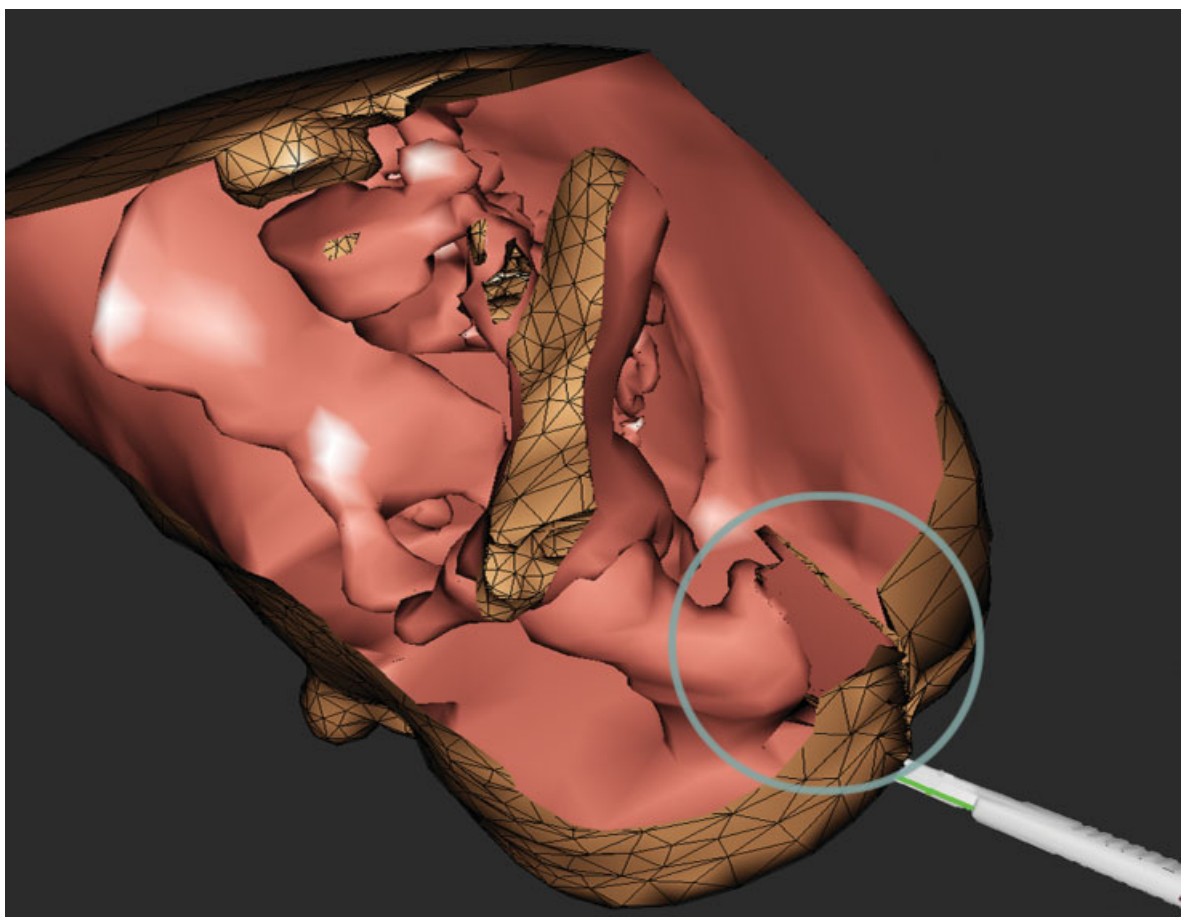

Figure 21. Cutting a volumetric with a scalpel with surface created by cutting tool passing through tetrahedra (circled)

Number cut	Number created	Time during cutting (ms)
0	0	593
1	12	48
25	212	75
50	502	86
75	798	98
100	1074	110

Table 10.

Table 10 provides the time to cut the tetrahedral mesh.

Table 11 (below) provides detail about the time to perform the cutting operation.

Scissors Cutting

When implementing scissors cutting, one must choose at least two edges as sharp. The allowable cutting direction is further restricted to permit cutting only when the cutting edges are moving towards each other. These two edges can either straddle the surface using the bottom edge to stabilize the surface while the top edge closes downward; or if the edges are on the same side of the surface, the edges pinch the surface together until the two edges form a junction with the simulated tissue. Figure 22 shows a pair of virtual scissors cutting a deformable model consisting of 5000 triangles. The figure also demonstrates how the surface relaxes as the scissors are opened.

Time during cutting						
					Collision response	
Number cut	Object update (seconds)	Number of nodes updated	Collision detection (seconds)	Collision response (seconds)	Time to cut (seconds)	Time to order nodes (seconds)
0	9.770E-04	0	5.913E-01	1.038E-03		
1	2.283E-03	365	9.137E-03	3.629E-02	0.946E-02	2.683E-02
25	2.092E-02	1555	9.353E-03	4.485E-02	1.902E-02	2.582E-02
50	3.335E-02	2553	9.973E-03	4.219E-02	1.628E-02	2.592E-02
75	4.508E-02	3536	9.952E-03	4.270E-02	1.616E-02	2.654E-02
100	5.540E-02	4434	1.000E-02	4.431E-02	1.763E-02	2.668E-02

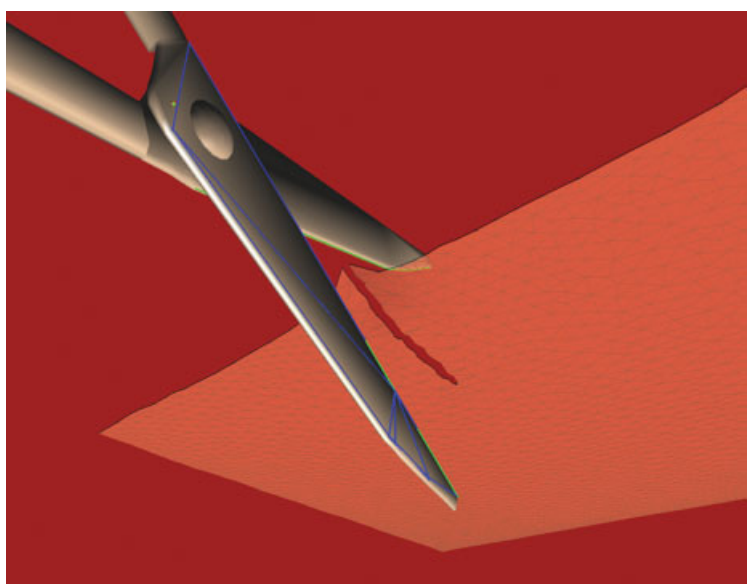

Figure 22. Cutting a surface with a pair of scissors with multiple active edges

Loop Cautery

When modeling a loop cautery tool, one needs to choose several edges as sharp. These edges have their own non-rigid dynamics that must be modeled as well. Figure 23 shows a wire that has been assigned 20 cutting edges and is cutting a virtual polyp consisting of 5000 triangles.

Discussion

In our simulations, we exploit spatial coherence as we typically move through the mesh by passing through primitives local to one another in the bounding sphere tree. We also prune the collision detection for the virtual instrument by including only the active primitives in the bounding hierarchy. This allows us to obtain the intersection information of interest while maintaining high update rates for large meshes. However, the time to perform collision detection before initial contact is made is several times longer than the time to detect collisions once a cut has started. In addition, moving the tool through many primitives invalidates the local collision detection tests. In these cases we traverse through the entire tree as usual, which also increases the time to detect collisions. The results also show that detecting collisions for layered surfaces can be several times longer than for single surfaces or even tetrahedral objects. That is because a tool in the same configuration when cutting a single surface is now intersecting multiple primitives at once, requiring further searching through the bounding volume hierarchy.

As seen in the Results section, the time to detect collisions influences the overall simulation speed until the first contact between the tool and tissue is made. After contact is made, detecting collisions is a small portion of overall iteration speed, and average speed of the simulation is governed by the time to update the deformable object. As seen in the Results section, the time to resolve collisions increases the overall iteration time. The results above show that the time to perform the cutting operation is a small portion of the overall time to resolve collisions. Instead, most of the time to resolve collisions during a cutting operation is spent ordering the nodes for subsequent updating of the deformable object. This is because the ordering operation requires iterating through every node in the mesh even though it models interactions only at areas of contact.

The results also demonstrate that the time to update the object increases with the number of elements cut. That is because as the object is cut, more of the object requires updating. Even though the ordered quasi-static method models tool and tissue interaction locally, the creation of a very large cut (as shown in the Results section) makes this operation less local. See Brown *et al.*[22] for a discussion on how the cut-off force

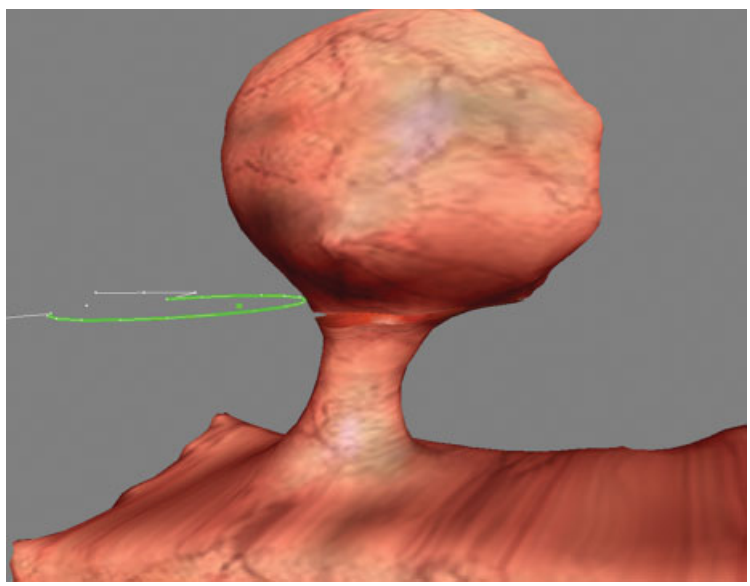

Figure 23. Cutting a surface with a loop cautery wire with multiple active edges

is computed; it should be noted, however, that the stiffness of the object will have some influence on the number of nodes that are processed at each iteration. In addition, when using highly tessellated meshes, it is possible to move through several primitives for a given user motion. This is because of the disparity in size of the tool and the primitives passed through. Figures 19–23 give an accurate representation of the tool and primitive size. In these cases, the cut operation is called several times during one iteration. Because the ordering of the nodes is performed for each primitive cut, moving through several primitives increases the time to perform the cutting operation. Given the impact ordering nodes has on performance it might be advantageous to order the nodes for only the last primitive cut or determine an alternative method for ordering nodes during cutting. Despite the requirements for node ordering, the ordered quasi-static method still provided the best results when compared to other methods such as non-ordered quasi-static.[22]

We present results for interactive cutting of meshes composed of a large number of primitives using realistic tools on a moderate processor. While these results may not appear to provide haptic compatible rates, we can identify the areas in the simulation that can be optimized. For example, the use of node ordering at each cutting operation is not necessary. If we omit the node ordering step we can decrease the time to perform a cutting operation for 25 primitives cut to 16 ms for the 12,000 triangle surface and 49 ms for the 25,000 tetrahedra volume. In addition, since the tool is undergoing a rigid transformation, we process the update of the associated bounding sphere tree at each iteration. Updating of the entire tree is not necessary since we are only detecting collisions for the single active edge. Future extensions will address these two optimizations.

Conclusions

We have provided a summary of the literature on mesh-cutting schemes. We have also presented ways in which to categorize these previous methods and compared the strengths and weaknesses of each. We have identified areas where previous cutting methods were lacking and presented methods to address the needs in the state of the art in interactive mesh cutting.

Creating cuts through large meshes is extremely simple using the schemes presented in this paper. We have demonstrated the implementation of these schemes allowing us to model various surgical instruments and provided metrics about the performance of these methods within a surgical simulation environment. These schemes have been employed in a rat dissection simulation system,[28] a virtual polypectomy simulator,[29] and a virtual hysteroscopy simulation system[30] utilizing a haptic feedback interface.

ACKNOWLEDGEMENTS

We wish to thank the BioVis Lab at the NASA Ames Research Center and past collaborators Sam Linton, Aaron Wai Lee and Tim Barth for their support of this research. Special thanks to the National Biocomputation Center and including Joel Brown, Steven Sorkin, Fredric Mazzella, Jeremie Roux, Julien Durand, Bharat Beedu and Guillaume Thonier. Thanks also to Jean Claude Latombe of the Stanford University Robotics Department. This work was supported by grants from NASA (NCC2-1010), NIH (NLM-3506, HD38223), NSF (IIS-9907060), and a generous donation from Sun Microsystems.

References

1. Pieper S, Rosen J, Zeltzer D. Interactive graphics for plastic surgery: a task-level analysis and implementation. In *Symposium on Interactive 3D Graphics*. ACM Press: New York, 1992; 127–134.
2. Song G, Reddy N. Tissue cutting in virtual environments. In *Medicine Meets Virtual Reality*, Westwood J *et al*. (eds). IOS Press: Amsterdam, 1995; 359–364.
3. Keeve E, Girod S, Girod B. Computer-aided craniofacial surgery. *Journal of the International Society for Computer Aided Surgery* 1996; **3**: 6–10.
4. Gibson S. Volumetric object modeling for surgical simulation. *Medical Image Analysis* 1998; **2**: 121–132.
5. Mazura A, Seifert S. Virtual cutting in medical data. In *Medicine Meets Virtual Reality*, Westwood J *et al*. (eds). IOS Press: Amsterdam, 1997; 420–429.
6. Wong KC, Sin TY, Heng P. Interactive volume cutting. Technical report, Department of Computer Science and Engineering, Chinese University of Hong Kong, 1998.
7. Basdogan C, Ho C, Srinivasan MA. Simulation of tissue cutting and bleeding for laparoscopic surgery using auxiliary surfaces. In *Medicine Meets Virtual Reality*, Westwood J *et al*. (eds). IOS Press: Amsterdam, 1999; 38–44.
8. Bro-Nielsen M, Helfrick D, Glass B, Zeng X, Connacher H. VR simulation of abdominal trauma surgery. In *Medicine Meets Virtual Reality*, Westwood J *et al*. (eds). IOS Press: Amsterdam, 1999; 117–123.
9. Voss G, Hahn JK, Muller W, Lineman RW. Virtual cutting of anatomical structures. In *Medicine Meets Virtual Reality*, Westwood J *et al*. (eds). IOS Press: Amsterdam, 1999; 381–383.

10. Bielser D, Gross M. Interactive simulation of surgical cuts. *Proceedings of Pacific Graphics* 2000; 116–125.
11. Neumann P. Near real-time cutting. In *SIGGRAPH 2000, Sketches and Applications*, New Orleans, LA, 2000.
12. Ganovelli F, Cignoni P, Montani C, Scopigno R. A multiresolution model for soft objects supporting interactive cuts and lacerations. *Proceedings of the 21st European Conference on Computer Graphics* 2000; 271–282.
13. Mor A, Kanade T. Modifying soft tissue models: progressive cutting with minimal new element creation. In *MICCAI 2000*, Niessen WJ *et al*. (eds). Springer: Berlin, 2000; 598–607.
14. Schutyser F, Van Cleyenbreugel J, Nadjmi N, Schoenaers J, Suetens P. 3D image-based planning for unilateral mandibular distraction. In *Computer Assisted Radiology and Surgery*, Heinz U *et al*. (eds). Elsevier: Amsterdam, 2000; 899–904.
15. Bruyns C, Senger S. Interactive cutting of 3D surface meshes. *Computer and Graphics* 2001; **25**: 635–642.
16. Bruyns C, Senger S, Wildermuth S, Montgomery K, Boyle R. Real-time interactions using virtual tools. In *MICCAI 2001*, Niessen WJ *et al*. (eds). Springer: Berlin, 2001; 1349–1351.
17. Nienhuys HW, van der Stappen AF. A surgery simulation supporting cuts and finite element deformation. In *MICCAI 2001*, Niessen WJ *et al*. (eds). Springer: Berlin, 2001; 145–152.
18. Meier U, Monserrat C, Parr NC, Garcia FJ, Gil JA. Real-time simulation of minimally-invasive surgery with cutting based in boundary element methods. In *MICCAI 2001*, Niessen WJ *et al*. (eds). Springer: Berlin, 2001; 1263–1264.
19. Serby D, Harders M, Szekely G. A new approach to cutting into finite element models. In *MICCAI 2001*, Niessen WJ *et al*. (eds). Springer: Berlin, 2001; 425–433.
20. Mitchell SA, Vavasis SA. Quality mesh generation in higher dimensions. *SIAM Journal of Computing* 2000; **29**(4): 1334–1370.
21. Montgomery K, Bruyns C, Brown J, Sorkin S, Mazzella F, Thonier G, Tellier A, Lerman B, Menon A. Spring: a general framework for collaborative, real-time surgical simulation. In *Medicine Meets Virtual Reality*, Westwood J *et al*. (eds). IOS Press: Amsterdam, 2002; 296–303.
22. Brown J, Sorkin S, Bruyns C, Latombe JC, Montgomery K, Stephanides M. Real-time simulation of deformable objects: tools and application. *Computer Animation* 2001; 6–8.
23. Sorkin S. Distance computing between deformable objects. Honors thesis, Computer Science Department, Stanford University, 2000.
24. Bruyns C, Wildermuth S, Montgomery K. Active edges: on interacting in a virtual environment. *Vision, Modeling and Visualization* 2001; 21–23.
25. Boyse JW. Interference collision detection among solids and surfaces. *Communications of the ACM* 1979; **22**: 3–9.
26. Terzopolous D, Fleicher K. Modeling inelastic deformation: viscoelasticity, plasticity, fracture. *Computer Graphics Proceedings of SIGGRAPH* 1988; 269–278.
27. Mazzella F. Auto acquisition of elastic properties for surgical simulation. http://biocomp.stanford.edu/papers/ [Last accessed 1/11/01].
28. Bruyns C, Montgomery K, Wildermuth S. A virtual environment for simulated rat dissection. In *Medicine Meets Virtual Reality*, Westwood J *et al*. (eds). IOS Press: Amsterdam, 2001; 75–81.
29. Wildermuth S, Bruyns C, Montgomery K, Marincek B. Virtual colon polyp extraction (simulation and preoperative planning). In *MICCAI 2001*, Niessen WJ *et al*. (eds). Springer: Berlin, 2001; 1347–1348.
30. Montgomery K, Heinrichs L, Bruyns C, Wildermuth S, Hasser C, Ozenne S, Bailey D. Surgical simulator for operative hysteroscopy and endometrial ablation. In *Computer-Aided Radiology and Surgery*, Lemke H *et al*. (eds). Elsevier: Amsterdam, 2001; 79–84.

Authors' biographies:

Cynthia Bruyns is currently a senior research scientist working at the BioVis Lab at the NASA Ames Research Center and National Biocomputation Center at Stanford University. Her research interests include physically based modeling, computational geometry and computer graphics. She received her MS in mechanical engineering from Stanford University in 1997 and her BS in mechanical engineering from Rensselaer Polytechnic Institute in 1995. She is a member of the ACM and ASME.

Steven Senger is a professor of computer science and mathematics at the University of Wisconsin–La Crosse.

His research interests include visualization and applications involving high-performance computing and network architectures. He received his BS and PhD degrees in mathematics from Purdue University in 1977 and 1982 respectively. He is a member of the IEEE and ACM.

Anil Menon is pursuing an MD at Stanford University. His research interests include electrical engineering and computer science. He is currently working on volume preservation methods for real-time interactions, as well as mesh refinement techniques. He earned his BA in neuroscience at Harvard University researching Huntington's disease at Massachusetts General Hospital.

Kevin Montgomery is the Technical Director of the National Biocomputation Center, a joint NASA–Stanford center performing research and development of applications in computer-based surgical planning, intraoperative assistance systems, surgical simulation, and telemedicine/wearable computing. He received his PhD in computer engineering at the University of California, Santa Cruz.

Simon Wildermuth is a senior staff radiologist at the Institute for Diagnostic Radiology at the University Hospital Zurich. He is section chief of computer tomography and 3D imaging. His research interests include segmentation for 3D medical visualization, virtual endoscopy and surgical planning/simulation. He received his degrees from the University of Zurich. He is a member of the Swiss and the European Society of Radiology.

Richard Boyle is Director of the BioVis Lab at NASA Ames Research Center. He received his BA in psychology from University of Colorado, MS in physiology at McGill University, and PhD in biological sciences from the Scuola Normale Superiore, Pisa, Italy. His main research involves study of the efferent vestibular system. Dr Boyle is a member of the International Brain Research Organization of UNESCO, the Bárány Society, and Honorary Member of the Vestibular Disorders Association.