Algorithmic Tools for Real-Time Microsurgery Simulation

Joel Brown [a], Stephen Sorkin [a], Jean-Claude Latombe [a], Kevin Montgomery [b], and Michael Stephanides [b]

[a] *Computer Science Department, Stanford University*

[b] *Stanford-NASA National Biocomputation Center*

Abstract

Today, there is growing interest in computer surgical simulation to enhance surgeons' training. This paper presents a simulation system based on novel algorithms for animating instruments interacting with deformable tissue in real-time. The focus is on computing the deformation of a tissue subject to external forces, and detecting collisions among deformable and rigid objects. To achieve real-time performance, the algorithms take advantage of several characteristics of surgical training: (1) visual realism is more important than accurate, patient-specific simulation; (2) most tissue deformations are local; (3) human-body tissues are well damped; and (4) surgical instruments have relatively slow motions. Each key algorithm is described in detail and quantitative performance-evaluation results are given. The specific application considered in this paper is microsurgery, in which the user repairs a virtual severed blood vessel using forceps and a suture (micro-anastomosis). Microsurgery makes it possible to demonstrate several facets of the simulation algorithms, including the deformations of the blood vessel and the suture, and the collisions and interactions between the vessel, the forceps, and the suture. Validation of the overall microsurgery system is based on subjective analysis of the simulation's visual realism by different users.

1 Introduction

As computer power and graphics capabilities continue to increase, there is growing interest in surgical simulation as a technique to enhance surgeons' training. Such training currently requires cadavers or laboratory animals. A computer simulation option could reduce costs and allay ethical concerns, while possibly decreasing training time and providing better feedback to the

This paper is based on earlier work appearing in [1] and [2].

trainees. However, for surgical simulation to be useful it must be realistic with respect to tissue deformation, tool interactions, visual rendering, and real-time response. This paper describes a microsurgery training system based on novel computer simulation techniques. The system allows a user to interact with models of deformable tissues using real surgical instruments mounted on trackers. It generates a graphic rendering of the tissue deformations in real-time.

Real-time simulation of deformable objects is needed in many areas of graphic animation, for example to generate cloth motions in animated movies or video games, to provide realistic facial animation for digital actors, and to deform soft tissues in surgical simulations. Deformable objects raise a complex combination of issues ranging from estimating mechanical parameters, to solving large systems of differential equations, to detecting collisions, to modeling responses to collisions. See [3] for problems and techniques in cloth modeling and [4] for issues arising in surgical simulation. Many issues still lack adequate solutions, especially when simulation must be real-time.

Here we focus on fast and realistic simulation of tissue and suture, and detecting and processing collisions among rigid and deformable objects. Our main goal is to develop efficient data structures and algorithms that can process large models at a rate compatible with real-time graphic animation (30 Hz). To achieve this goal, we exploit the fact that many deformations are *local*. By propagating forces in a carefully ordered fashion through an elastic mass-spring mesh, we effectively limit the computations to the portions of objects that undergo significant deformations. To accelerate collision detection, we pre-compute hierarchical representations for all objects in the scene; when objects are being deformed, we only update those parts of the hierarchies that need to be modified.

We have used these algorithms, along with other techniques, to build a system aimed toward microsurgical training. Microsurgery is a well-established surgical field which involves the repair of approximately 1mm vessels and nerves under an operating microscope. It is a necessity in many reconstructive procedures, including the successful reattachment of severed digits. Using a forceps, the surgeon maneuvers a suture (needle and thread) through the two ends of a severed vessel and ties several knots to stitch the two ends together. The two parts of the vessel undergo deformations caused by their interactions with the suture and the forceps. The surgeon receives only visual feedback, as the vessel is too small to produce any perceptible reaction force. Microsurgeons typically acquire their initial skills through months of practice in an animal lab, at which point they still require months of supervision in the operating room. Without practice, these skills can quickly degrade. The need for such a suturing simulation has been previously addressed in [4], and a performance study in [5] discusses the validity of using such a simulator to develop surgical skill, although it does not provide technical details of the actual simulation. Other aspects of suture simulation are discussed in [6].

The main contributions of this paper are the algorithmic tools that we propose for simulating deformable tissue in real-time. We provide extensive quantitative analysis of the performance of these tools. On the other hand, the experimental microsurgery system has only been validated through subjective visual analysis by several users.

Section 2 describes an overview of our simulation system. Sections 3, 4, and 5 present our simulation and collision-detection algorithms, with specific references to the microsurgery simulator. Section 6 discusses current and future work.

2 System Overview

Our software system includes a deformable object simulator, a tool simulator, and a collision detection module. A graphics display allows multiple objects to be rendered from a 3D virtual world onto the screen at 60 Hz or higher. The user has complete control of the view, and may use stereo glasses for true binocular depth perception (rendered at 30 Hz or higher per eye). The positions of the objects are read from the deformable object and tool simulators before each screen refresh.

Deformable object simulation is described in detail in the following two sections. Tool simulation synchronizes virtual surgical tools with real tools that are connected to external tracking devices. Virtual tools consist of one or more rigid parts modeled as triangulated surfaces. Their positions and orientations are controlled by the external devices at high update rates (typically 100 Hz), and other information from the devices may control relative rotations or translations of the parts that make up one tool. Interactions between tools and deformable objects, such as grabbing, poking, and cutting, are dependent on the collision detection module described in Section 5.

The setup for microsurgery includes two real surgical forceps instrumented to detect closure and attached to electromagnetic trackers (miniBIRD of Ascension Technology Corporation). The user's translation, rotation, opening, and closing of these forceps directly controls forceps models in the simulation. Using the forceps, models of blood vessels can be grabbed and deformed. A suture can be manipulated to pierce through and realistically interact with the vessels and the forceps. Stereo glasses allow the necessary depth perception to complete the task. Figure 1 shows a user of the simulator.

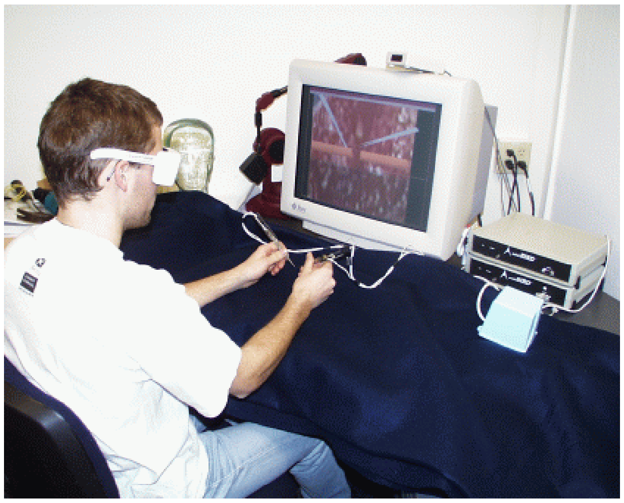


Fig. 1. Setup for microsurgery

The system also supports parallel processing using multithreading. Our implementation on a dual-processor machine (Sun Ultra 60, two 450 MHz processors) can use two threads of execution to separate the simulation and collision detection from the graphical rendering. In this way, visual updates occur at a guaranteed rate while the simulation continues uninterrupted.

3 Computation of Object Deformations

3.1 Relation to Previous Work

Research on modeling deformable objects has increased dramatically in the past few years. Most proposed 3D models/techniques fall into two broad categories, *mass-spring meshes* and *finite elements.*

A mass-spring mesh is a set of point masses connected by elastic links. It represents the tissue geometry and is used to discretize the equations of motion. At each node N an equation defines the force exerted on N by the nodes to which N is connected. Mass-spring models have been used in facial animation [7], cloth motion [8], and surgical simulation [9], to cite only a few works. They are relatively fast and easy to implement, and allow realistic simulation for a wide range of objects, including viscoelastic tissues encountered in surgery.

Finite element methods (FEMs) use a mesh to decompose the domain over which the differential equations of motion are solved. The mesh represents the domain initially occupied by the object and the FEM technique computes a vector field representing the displacement of each point in this domain. For example, FEMs have been used to model facial tissue and predict surgical

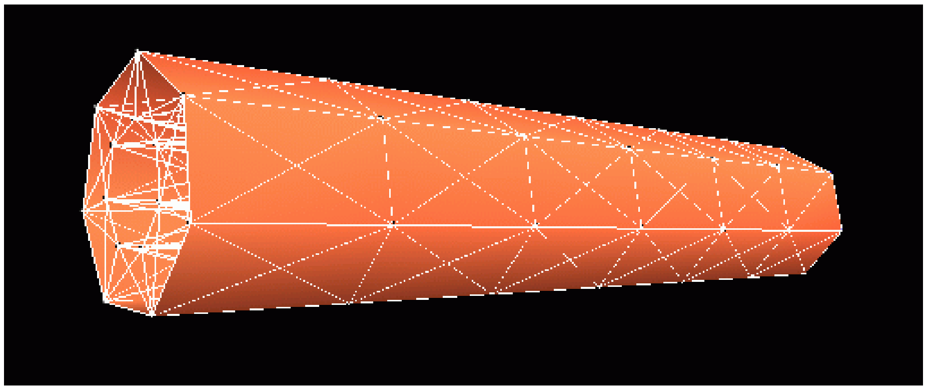


Fig. 2. Mesh example

outcomes [10–12]. They may be more accurate than mass-spring models, but they are also more computationally intensive, especially for complex geometries and large deformations. Some systems use either mass-spring or FEM techniques depending on the situation [6]. Others use preprocessing steps to reduce FEM computation [13,14], and [15] extends the “tensor-mass” model of [14] to non-linear elasticicty.

Other examples of mass-spring models, FEMs, and alternate models are too numerous to cite here, but are cited in [4] and [16].

Mass-spring meshes seem better suited for surgical training – the application domain considered in this paper – which relies more on visual realism than exact, patient-specific deformation, but requires that simulations be performed in real-time. In contrast, FEMs may address better the needs of other applications (e.g., pre-operative surgical planning and predicting the long-term outcome of a surgery), where computations can be done off-line, but must provide accurate, patient-specific results.

3.2 Mass-spring elastic mesh

We represent the geometry of a deformable object by a 3D mesh M of n nodes N_i ($i = 1,...,n$) connected by links L_{ij}, $i, j \in [1, n], i \neq j$. Each node maps to a specific point of the object, so that the displacements of the nodes describe the deformation of the object. The nodes and links on the object's surface are triangulated, whereas the other nodes and links are unrestricted, though it is often convenient to arrange them in a tetrahedral lattice. Figure 2 shows the surface and underlying links of a mesh representing a severed blood vessel; this mesh contains 98 nodes and 581 links. The more complex mesh in Figure 3a consists of 40,889 nodes and 212,206 links forming a tetrahedral lattice.

The mechanical properties (viscoelastic, in most surgical simulation applications) of the object are described by data stored in the nodes and links of M. A mass m_i and a damping coefficient c_i are associated with each node N_i, and a stiffness k_{ij} is associated with each link L_{ij}. The internal force between two nodes N_i and N_j is $\boldsymbol{F}_{ij} = -k_{ij}\Delta_{ij}\boldsymbol{u}_{ij}$, where $\Delta_{ij} = l_{ij} - rl_{ij}$ is the current length of the link minus its resting length, and $\boldsymbol{u}_{ij}$ is the unit vector pointing from N_i toward N_j. The stiffness k_{ij} may be constant or function of Δ_{ij}. In either case, $\boldsymbol{F}_{ij}$ is a function of the coordinate vectors $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ of N_i and N_j. This representation can describe objects that are nonlinear, non-homogeneous, and anisotropic.

At any time t, the motion/deformation of M is described by a system of n differential equations, each expressing the motion of a node N_i:

$$m_i\boldsymbol{a}_i + c_i\boldsymbol{v}_i + \sum_{j\in\sigma(i)} \boldsymbol{F}_{ij}(\boldsymbol{x}_i, \boldsymbol{x}_j) = m_i\boldsymbol{g} + \boldsymbol{F}_i^{ext} \tag{1}$$

where $\boldsymbol{x}_i$ is the coordinate vector of N_i, $\boldsymbol{v}_i$ and $\boldsymbol{a}_i$ are its velocity and acceleration vectors, respectively, $m_i\boldsymbol{g}$ is the gravitational force, and $\boldsymbol{F}_i^{ext}$ is the total external force applied to N_i. $\sigma(i)$ denotes the set of indices of the nodes adjacent (connected by a link) to N_i in M.

3.3 Simulation algorithm

We have developed a "dynamic" and a "quasi-static" simulator. The dynamic simulator uses classical numerical integration techniques such as fourth order Runge-Kutta to solve Eq. 1. However, in many situations encountered in surgical simulation, a simpler algorithm based on quasi-static assumptions gives realistic results at a much faster rate. We describe this quasi-static simulator below.

Assumptions. We refer to the nodes of M that are subject to external forces as the *control* nodes. We assume that the position of each such node is given at any time. In our surgical simulation system, the control nodes correspond to the portions of tissue that are pulled or pushed by surgical instruments or held fixed by bone structures or clamping tools. The positions of the displaced control nodes are obtained online by reading the positions/orientations of tracking devices. We also assume that the velocity of the control nodes is small enough so that the mesh achieves static equilibrium at each instant. This is a reasonable assumption for soft objects with relatively high damping parameters, which is the case for most human-body tissues. When these assumptions do not hold, the dynamic simulator must be used.

Quasi-static algorithm. Under the above assumptions, we neglect dynamic inertial and damping forces. The shape of M is defined by a system of equations expressing that each non-control node N_i is in static equilibrium:

$$\sum_{j \in \sigma(i)} \boldsymbol{F}_{ij}(\boldsymbol{x}_i, \boldsymbol{x}_j) - m_i \boldsymbol{g} = \boldsymbol{0}. \tag{2}$$

Let I be the set of indices of all the non-control nodes of M, and let δ be a constant time step (in our implementation δ is set to 1/30 s). At each time $t = k\delta$, $k = 1, 2, ...$, the quasi-static simulator solves Eq. 2 for the positions of all the non-control nodes. To achieve real-time animation, it returns these positions within time δ. The algorithm is the following:

```
Algorithm QSS:
1. Acquire the positions of all the control nodes
2. Repeat until time $\delta$ has elapsed
      For every $i \in I$
         (a) $\boldsymbol{f}_i \leftarrow \sum_{j \in \sigma(i)} \boldsymbol{F}_{ij} - m_i \boldsymbol{g}$
         (b) $\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i + \alpha \boldsymbol{f}_i$
```

Step 2 computes the residual force applied to each node and displaces the node along this force. A conjugate-gradient-style method can also be used by moving the node along a combination of the old and the new forces. Ideally, the value of the scaling factor α should be chosen as large as possible such that the iteration converges. This choice typically requires experimental trials.

The timeout condition of Step 2 guarantees that QSS operates in real-time even as the size of the mesh M increases. Hence, Step 2 is not guaranteed to reach exact equilibrium at every step, that is, some N_i's will have a non-zero force $\boldsymbol{f}_i$ acting on them after δ amount of time. As mesh size increases, each iteration of Step 2 will take longer, and thus fewer loops will be possible in the allowed time. By comparing the positions computed by QSS to the actual equilibrium positions (computed without timeout), we can measure how the accuracy of the simulation degrades as the mesh complexity increases.

Step 2(b) updates the position of each non-control node using the most recently computed positions of the adjacent nodes, rather than those computed at the previous iteration of Step 2. This scheme is most advantageous when the nodes are processed in a wave-propagation order starting at the displaced control nodes and expanding towards the nodes farthest away from any displaced node. This ordering is computed by a breadth-first scan of the mesh M:

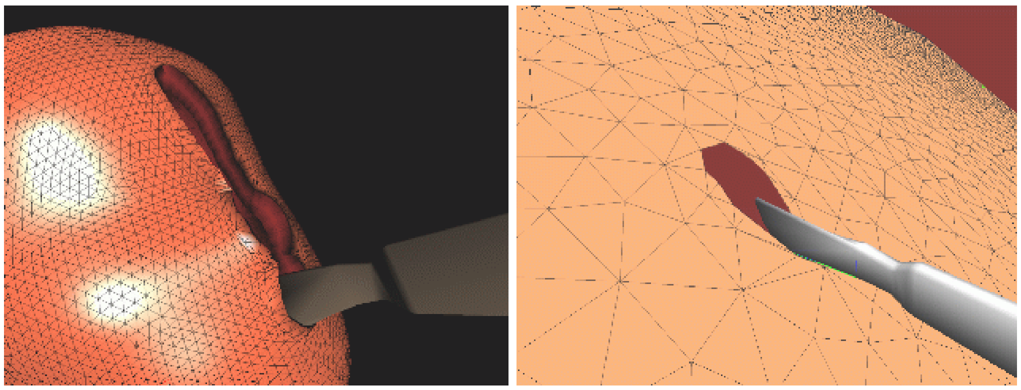

(a) (b)

Fig. 3. Simulation of cutting operations (Courtesy C. Bruyns)

Algorithm NODE-ORDERING:

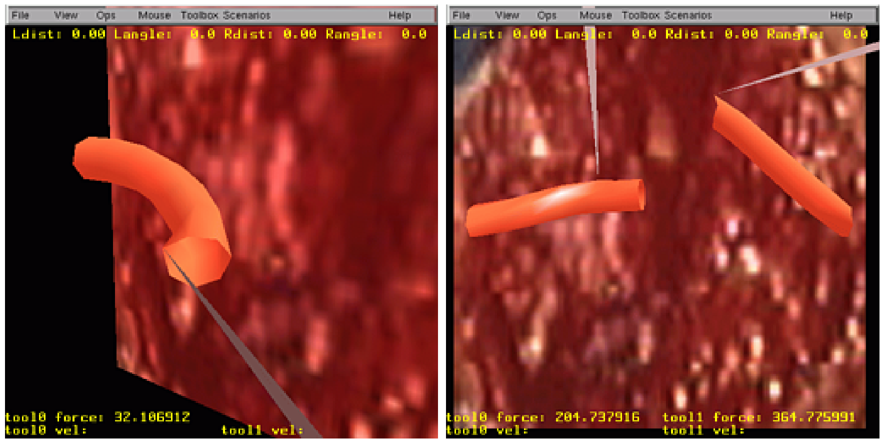
```
1. Initialize $I$ to the empty list
2. Mark the displaced control nodes in $M$ to be at level 0
3. For $k = 1, 2, ...$, mark the unmarked nodes adjacent to a
   node at level $k - 1$ to be at level $k$, and store them in $I$,
   until all non-control nodes have been marked.
```

The displaced nodes may be arbitrarily distributed over the mesh. The outcome of NODE-ORDERING is a list I of nodes such that if index i appears before index j in I then the level of N_i is less than or equal to that of N_j. QSS processes the nodes as they have been ordered in I.

Node ordering enables another major computational savings. During an iteration of Step 2, if the positions of all the nodes at some level k are modified by less than a small pre-specified amount, then the algorithm stops propagating the deformation further. In this way, the number of levels treated at each iteration adjusts automatically. This computation *cutout* is especially useful when object deformations are local.

While QSS is being executed, any change in the set of displaced nodes only requires re-invoking NODE-ORDERING to compute a new ordered set I. This set normally changes rarely (when nodes are grabbed or released), and since NODE-ORDERING is a simple breadth-first scan of the nodes, calling it adds a negligible computational cost to the simulation. Similarly, the mesh's topology may change at any time. For example, in Figure 3 cutting operations are being performed. Links are removed from the mesh as they are crossed by the scalpel, so that the mesh used by QSS changes frequently. (In principle, such a cutting operation violates assumptions made above, and a dynamic simulator should be used. Nevertheless, QSS, which was used to produce Figure 3, gives realistic results.)

Models. The vessels in our simulation are modeled as double-hulled cylinders, with the inner and outer cylinders representing the thickness of the ves-


File View Ops Mouse Toolbox Scenarios Help
File View Ops Mouse Toolbox Scenarios Help

```
Ldist: 0.00 Langle:  0.0 Rdist: 0.00 Rangle:  0.0
tool0 force: 32.106912
tool0 vel:                tool1 vel:

Ldist: 0.00 Langle:  0.0 Rdist: 0.00 Rangle:  0.0
tool0 force: 204.737916   tool1 force: 364.775991
tool0 vel:                tool1 vel:
```

Fig. 4. Forceps deforming severed vessel

sel. Each cylinder is modeled by several layers of nodes, with the layers evenly spaced, and each layer consisting of several nodes evenly spaced around a circle. Each node is connected by deformable links to its neighbors within a layer and in neighboring layers. There are also connections between the inner and outer cylinders, which provide torsional stability, preventing the vessel from twisting excessively around its long axis. One end layer of each vessel is fixed in space, representing the fact that the vessels are clamped down during surgery, and only a portion of their length can be manipulated. We set all masses $m_i = 1$, and we used trial and error with subjective feedback about the quality of deformations to set $k = 50$ for the stiffnesses of springs within a layer and $k = 100$ for springs connecting adjacent layers. This trial and error process was done under the direct supervision of a microsurgeon. Figure 2a shows a smooth-shaded vessel and its underlying links. As the user displaces individual nodes of the vessels, the quasi-static algorithm described above is used to calculate the deformation. Figure 4 shows some examples of deforming the vessels with forceps. Gravity is implemented as an external force that can be either off or on, and in these figures it is off, because the most interesting deformations are caused by the user displacements.

Performance evaluation. The above algorithms are written in C++ and were tested on one 450 MHz processor of a Sun Ultra 60 workstation with 1 GB RAM. To address visual realism, we asked surgeons to verify that the deformations were similar in shape and velocity to those encountered in clinical operations. Figures 4, 7, and 8 show such deformations.

We did other experiments to quantitatively evaluate the performance of QSS. In particular, we created a regular mesh whose nodes form a rectangular lattice. Each node is linked to its neighbors in the x, y, and z directions. It is also diagonally linked to neighbors in the xy, xz, and yz planes. All links are given the same constant stiffness. Our meshes ranged from a 3x3x3 box (27 nodes and 64 links) to a 20x20x20 box (8000 nodes and 66120 links).

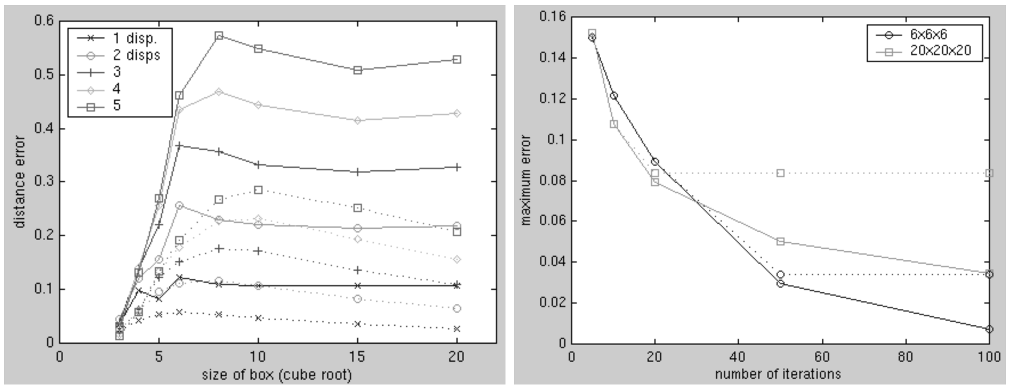


Fig. 5. (a) Maximum (solid lines) and average (dashed lines) error vs box size, for 5 successive displacements of the control node (b) Maximum error without (solid lines) and with (dashed lines) computation cutout vs number of iterations

Given one such lattice mesh, we fix the nodes of an arbitrarily chosen "bottom" face, and displace the middle node of the top face upwards by one unit. We then run QSS, but instead of running Step 2 for a fixed amount of time, we do a fixed number k of iterations (k = 100, 50, 20, 10, or 5), where an iteration involves updating $\boldsymbol{f}_i$ and $\boldsymbol{x}_i$ for all the non-control nodes. We repeat several times this cycle of displacing the control node (by an additional unit) and doing k iterations, and after each cycle we record the position of each node. Errors are computed as the distances between these positions and the actual equilibrium positions. The equilibrium positions are found by running QSS until the force on each non-control node is zero.

Figure 5*a* shows maximum and average errors for the different mesh sizes for 5 consecutive cycles of unit displacement followed by $k = 10$ iterations. We see that errors increase slightly as the box grows from 27 to 216 nodes, but then minimally as size increases to 8,000 nodes. The larger boxes have many nodes which may be moving hardly at all, which contribute to lowering the average error. However, the maximum error follows the same pattern at about twice the average error, and remains at most about 10% of the magnitude of the displacement of the control node for all mesh sizes.

Figure 5*b* shows maximum error after one unit displacement for different numbers k of iterations. It compares QSS with and without the computation cutout described previously. The plots for two different boxes are shown together, and we can see that errors are quite low in all cases, even for the largest box and the lowest number of iterations. When using the cutout, errors do not strictly drop off to 0 as the number of iterations increases, but the more important difference is the sharp decrease in the time required per iteration. Table 1 displays the number of iterations that QSS performs while maintaining a 30 Hz update rate, without and with cutout.

Table 1
Effect of computation cutout on simulation rate

Mesh	Nodes	Edges	Iterations at 30 Hz without cutout	Iterations at 30 Hz with cutout
6x6x6	216	1440	24	24
8x8x8	512	3696	9.5	13
10x10x10	1000	7560	4.6	8
15x15x15	3375	27090	1.2	7
20x20x20	8000	66120	0.4	6

These experiments demonstrate that for the objects given, we can reasonably maintain a small relative error given only 10 or fewer iterations of QSS Step 2 per time interval. Furthermore, the cutout method can complete these iterations in the appropriate 1/30 second, and scale to meshes containing thousands of nodes with no significant performance penalty.

4 Simulation of the Suture

The suture is deformable but not elastic, so the above deformation techniques based on mass-spring models are not applicable. Instead, the suture behaves as a needle and thread, which can stretch minimally if at all, and has a free-form shape which is affected by gravity and direct contacts. To achieve realistic deformation, we model the suture as an articulated object: 200 short straight links are sequentially connected at nodes which act as spherical joints. The joints allow two degrees of rotational freedom, while the links are rigid and short enough that the suture shape appears smooth. By keeping the angles between the first few links fixed, we can model a rigid needle at one end of the suture.

To model the motion of the suture, constraint-based techniques are used. Any node of the suture may be constrained by another object in the system. For example, one node might be grasped by a forceps, and thus its position is constrained by the forceps. If the suture has pierced through a vessel, a node will be constrained by the position of the vessel. Finally, if the suture is draped over another object, nodes will be constrained by that object.

The motion is then calculated in a "follow-the-leader" manner as follows: a constrained node N_i is moved by the constraining object from $\vec{x}_{i.old}$ to $\vec{x}_{i.new}$. We then compute the new position $\vec{x}_{i+1.new}$ of its neighbor N_{i+1} as the point a distance d along the line from $\vec{x}_{i.new}$ to $\vec{x}_{i+1.old}$, where d is the (fixed) length of the link connecting N_i and N_{i+1}. The same is done for node N_{i-1}. This motion

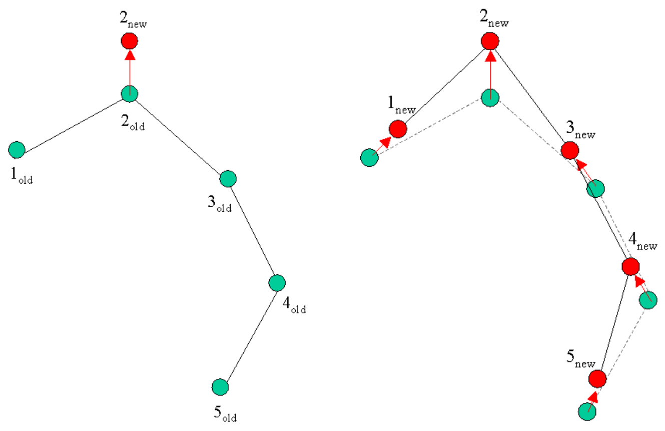
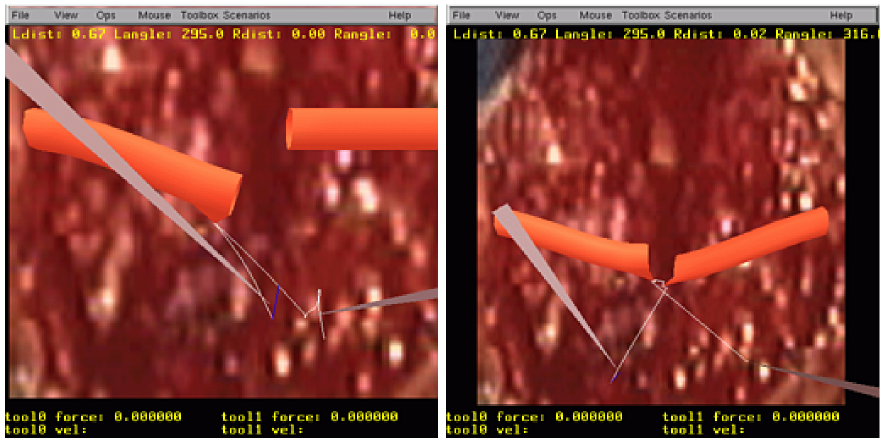


Fig. 6. (a) Constrained node 2 is moved (b) Nodes 1 and 3 follow, then 4, then 5



File View Ops Mouse Toolbox Scenarios Help
File View Ops Mouse Toolbox Scenarios Help

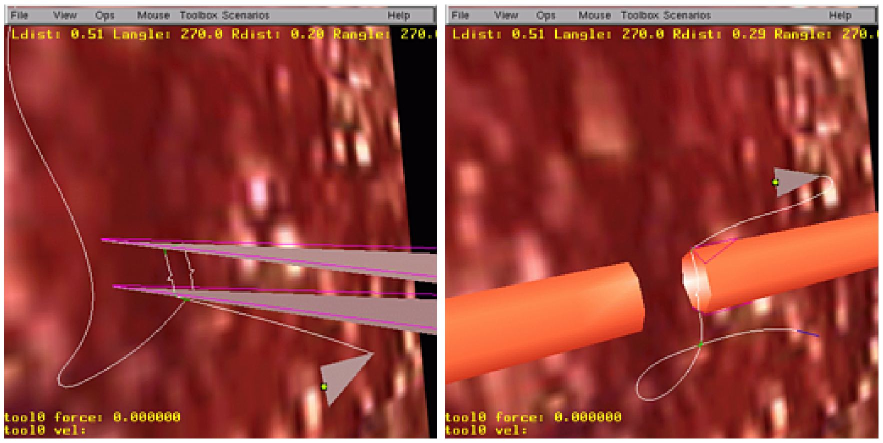
```
Ldist: 0.67 Langle: 295.0 Rdist: 0.00 Rangle: 0.0
tool0 force: 0.000000        tool1 force: 0.000000
tool0 vel:                   tool1 vel:
```

```
Ldist: 0.67 Langle: 295.0 Rdist: 0.02 Rangle: 316.
tool0 force: 0.000000        tool1 force: 0.000000
tool0 vel:                   tool1 vel:
```

Fig. 7. (a) Suture pulling vessel down, (b) Suture pulling two vessels together

is propagated up and down the suture to N_{i+1}, N_{i+2}, ... and N_{i-1}, N_{i-2}, ... until the next constrained node or one end of the suture is reached. Figure 6 demonstrates this technique for four links of a suture. For nodes between two constrained nodes N_i and N_j, the preceding algorithm will compute two preliminary results, propagating from N_i to N_j, and from N_j to N_i. These results are averaged to give the final position.

Certain constraints are designated as *soft*, such as where the suture is piercing a vessel or draped over another object, whereas the forceps grabbing the suture is a *hard* constraint. The distinction is that the suture can slide over and/or through a soft constraint, changing which node of the suture is constrained. It may be the case that two constraints move in opposing directions and cause the suture to stretch between them. In that case, the suture will slide through a soft constraint to decrease the stretch, but will break if between two hard constraints (in real surgery, it is not difficult to break the suture by pulling with both forceps in opposite directions). Additionally, if the suture is pierced through a vessel and is pulled on both ends, the suture will pull the vessel, causing it to deform as in Fig. 7*a*. Figure 7*b* shows the suture pulling together


File View Ops Mouse Toolbox Scenarios Help
File View Ops Mouse Toolbox Scenarios Help

```
Ldist: 0.51 Langle: 270.0 Rdist: 0.20 Rangle: 270.
tool0 force: 0.000000
tool0 vel:
```

```
Ldist: 0.51 Langle: 270.0 Rdist: 0.29 Rangle: 270.
tool0 force: 0.000000
tool0 vel:
```

Fig. 8. (a) Suture wrapped around forceps, (b) Suture colliding with self and vessel end

the two vessels.

5 Collisions and Interactions

Interaction with virtual tools is a necessary component of any realistic simulator, and has attracted recent attention [17]. Almost all object interactions depend at some level on collision detection [18]. Grabbing is achieved by finding nodes colliding with the tip of the grabbing object (e.g. forceps). Piercing the vessel requires finding a collision between the needle edges and a vessel face. Draping the suture around another object also involves edge to face collisions (Figures 8*a* and 8*b* show the suture around forceps and vessel), and draping it around itself requires edge to edge self-collisions. Other interactions modeled by the system (although not specifically in the microsurgery simulation) include prodding one object with another (face to face collisions), and cutting one object with another (edge to face collisions). In the remainder of this section, we present background on collision detection and a new algorithm for collision detection among deformable objects.

5.1 Related Work

Research on collision detection between rigid objects has a long history in robotics, graphics, and solid modeling. Two main families of methods have been proposed: *feature-based* (e.g. [19–21]) and *hierarchical* (e.g. [22–26]). A feature-based method exploits temporal and spatial coherence in the geometric model to maintain the pair of closest features. A hierarchical method precomputes a hierarchy of bounding volumes for every object. During a collision test, the hierarchies are used to quickly discard large subsets of the object surfaces that are too far apart to possibly collide. Hierarchies using various primitive volumes have been proposed. While some volumes allow closer-fit approximation, they also yield more costly intersection checks. Spheres give good results over a broad range of objects.

Although each approach has distinct advantages, the hierarchical approach is better suited when objects are highly concave. The main issue in using it with deformable objects is that pre-computed hierarchies may become invalid when objects deform, while re-computing new ones at each collision query would be too time consuming. Below, we propose an algorithm that does not modify the topology of the hierarchy representing an object, but only updates the size and location of the primitive volumes labeling the nodes of this hierarchy. Our algorithm derives from the one proposed by Quinlan [26] for rigid objects.

Fewer works exist for deformable objects (e.g. [27,28,9,29,30]). The algorithm in [28] is the closest to ours. It also builds a tree of fixed topological structure. But this tree is not balanced (which may seriously increase the cost of collision queries) and its nodes are axis-aligned boxes. The main difference, however, is in the tree-maintenance algorithm. Unlike [28], our algorithm exploits the locality of most deformations to minimize the number of node updates. The algorithm in [30] is specifically aimed at detecting self-collisions of cloth-like objects, an issue that we have not carefully studied so far.

5.2 Quinlan's algorithm

Sphere tree of an object. Let A be a (rigid) object represented by its triangulated surface S. Quinlan's algorithm covers every triangle in S with small spheres of predefined radius ε and constructs an approximately balanced binary tree T that has one leaf per sphere of radius ε. Each other node N in T is a sphere that encloses all the leaf spheres of the sub-tree rooted at N. T is constructed by recursively partitioning the set E of leaf spheres contained in a sub-tree (initially the set of all leaf spheres in T) into two subsets E_1 and E_2 of equal cardinality, until each subset contains a single leaf sphere. The partitioning operation tries to minimize the intersection and the radii of the two spheres that respectively enclose the leaf spheres in E_1 and E_2. A technique to partition the set E first computes the box that is aligned to the object's coordinate frame and contains the centers of the leaf spheres in E. It then divides the leaf spheres along the longest side of this box.

Collision detection. Let T_1 and T_2 be the respective sphere trees of two (rigid) objects A_1 and A_2. A collision query is specified by the position and orientation of A_1 relative to A_2. Collision detection is performed by a depth-first traversal of T_1 and T_2 during which pairs of spheres from the two trees are examined. If two intermediate spheres have null intersection, then the leaf spheres they contain cannot possibly intersect, and the traversal is pruned; otherwise the children of one of the two nodes are examined. If two leaf spheres intersect, the two triangles tiled by these spheres are explicitly tested for collision. For N_1 and N_2, the root spheres of T_1 and T_2, respectively, the following recursive algorithm either finds the colliding triangles and returns 1, or returns 0 if there is no collision:

```
Algorithm COLLISION($N_1$, $N_2$):
1. If $N_1$ and $N_2$ have null intersection then return 0
2. Else
   (a) If both $N_1$ and $N_2$ are leaf spheres then test the corresponding
       two triangles for collision; return 1 if they collide and 0 otherwise
   (b) If $N_2$ is smaller than $N_1$ then switch $N_1$ and $N_2$
   (c) If COLLISION($N_1$,left-child($N_2$)) = 1 then return 1
       Else if COLLISION($N_1$,right-child($N_2$)) = 1 then return 1
       Else return 0
```

5.3 Application to deformable objects

To use COLLISION, we must maintain the sphere tree of every deforming object. We propose a new sphere tree whose balanced structure is computed only once. When an object deforms, the structure of its tree remains fixed, i.e., no sphere is ever added or removed; only the radii and positions of some spheres are adjusted. Moreover, the maintenance algorithm performs adjustments only where they are needed.

Construction of a sphere tree. Let S be the triangulated surface of a deformable object A in some initial shape. The pre-computed tree T for A differs from the one in [26] in two ways:

(1) Instead of tiling the triangles of S with small equal-sized spheres, we assign each triangle a single leaf sphere of T – the smallest sphere enclosing the triangle. Hence, when S undergoes a deformation, the number of leaf spheres of T remains constant. Moreover, updating the radius and position of the sphere enclosing a deforming triangle is faster than computing a new tiling.

(2) The approximately balanced structure of T is generated in the same way, by recursively partitioning the leaf spheres into two subsets of equal size. But the radius and position of each non-leaf sphere is computed to enclose the sphere's two children. This yields a slightly bigger sphere than the one computed to contain the descendant leaf spheres, but the computation is much faster.

Collision detection. COLLISION is used unchanged.

Maintenance of a sphere tree. Each deformation of one triangle of S requires adjusting the radius and position of the corresponding leaf sphere and of all its ancestors up to the root of T. Our algorithm performs those changes only prior to processing a query. The operation is done bottom-up, using a priority queue Q of spheres sorted by decreasing depths in T. Q is initialized to contain all the leaf spheres that enclose triangles that have been deformed since the last update of T. It is then used as follows:

```
Algorithm MAINTENANCE:
  While $Q$ is not empty do
    1. $w \leftarrow$ extract($Q$)
    2. Adjust the radius and position of $w$
    3. Insert($Q$,parent($w$))
```

The only spheres that are modified are those that contain at least one deformed triangle. Each such sphere is modified only once, even if it contains several deformed triangles.

Clearly, MAINTENANCE will perform better when deformations are local than when they are scattered throughout S. More specifically, a local deformation of S affecting $k \ll s$ triangles, where s is the total number of triangles in S, results in a total update time of $\mathrm{O}(k + \log s)$. Instead, if the k triangles are spread over the leaves, this cost can be $\mathrm{O}(k \log s)$. However, in the worst case, if all triangles have changed shape, the maintenance operation only takes time $\mathrm{O}(s)$.

5.4 Performance evaluation

The above algorithms were implemented in C++. We give results of experimental performance tests on an Intel 400-MHz Pentium II processor, with 256-MB memory and running Windows 2000.

Sphere tree construction. The pre-computation of an object's sphere tree need not be particularly efficient, since it is done only once per object, prior to any simulation. Our software runs in time proportional to the number of triangles and takes on the order of 0.1 milliseconds per triangle.

Sphere tree maintenance. To evaluate MAINTENANCE we considered a surface S initialized to a flat horizontal 100x100 grid, with each square split into two triangles. Hence, S consists of 20,000 triangles. To create a local deformation of S, we pick a vertex V, a radius ρ of deformation between 1 and 10, and a direction (upward or downward), all at random. Each vertex within distance ρ of V is translated by a distance inversely proportional to its distance to V. To create scattered deformations of S, we repeat this process

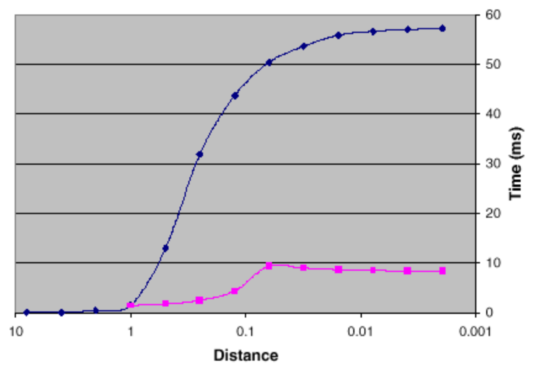


Fig. 9. Dark curve: query time vs. separating distance for non-colliding objects, Light curve: query time vs. penetration distance for objects already in collision

several times. After computing the sphere tree for the flat surface, we ran MAINTENANCE to update this tree after various deformations. We measured a running time for MAINTENANCE of 0.06 milliseconds per deformed triangle.

Collision queries We considered two objects. One is modeled as a flat square mesh of 8 by 8 units tessellated with 8,192 triangles. The other object is a spherical ball, 2 units in diameter and tessellated with 1,024 triangles. We moved the ball along a straight path though the center of the square mesh, from 64 units separation to one unit penetration. The query times for different relative positions of the objects are shown in Figure 9. When the objects are far apart, each query is extremely fast and takes on the order of tenths of a millisecond. When they get closer than 1 unit together, query time grows quickly (dark curve) to an asymptote of just under 60 milliseconds, as more spheres in the trees had to be examined to rule out a possible collision. Once the objects are in collision, the query time drops sharply to under 10 milliseconds (light curve). This sharp drop suggests that a timeout could be imposed on COLLISION, with a relatively minor risk of not detecting a collision once in a while. We did similar experiments after deforming the two objects and updating their sphere trees using MAINTENANCE. We observed no significant degradation of the query times, even for rather large deformations of the objects. The unintuitive explanation of this result is that the topology of the sphere tree is largely shaped by the adjacency relation between triangles, which itself remains invariant under any deformation of the object.

Re-implementation After the above testing, the code was implemented as one module in our overall system. One change allows a sphere to bound features which are not triangles, such as the individual links of the suture. Another change allows the reporting of all pairs of colliding features, rather than just one.

6 Conclusion and Future Work

We have designed new fast algorithms for simulating the deformations of soft objects and detecting collisions among deforming and rigid objects. These algorithms take advantage of several characteristics of surgical training: (1) visual realism is more important than accurate, patient-specific simulation; (2) most deformations are local; (3) human-body tissues are well damped; and (4) surgical instruments have relatively slow motions. Our simulator exploits these characteristics to solve quasi-static equations using a "wave-propagation" technique that has an automatic computation cutout when deformations become insignificant. The collision algorithm exploits deformation locality to minimize the number of updates in the hierarchical representations of the deforming objects.

These algorithms have been integrated into a virtual-reality system for simulating the suturing of small blood vessels. This system has been used by plastic and reconstructive surgeons in our lab and at various exhibits, and subjective feedback in the form of questionnaires distributed at the year 2000 annual meeting of the American Society of Plastic and Reconstructive Surgeons deemed the simulation realistic and potentially very useful. One next step would be experimental and clinical verification, by having surgeons who are learning the procedure use this tool, and assessing the quality of their virtual repairs through measurements such as angle and position of vessel piercing. We could then try to establish quantitatively how practicing with the simulator affects future quality of real vessel repairs. Another step would be to quantitatively study the visual realism, by comparing the simulation with video of real surgeries, using computer vision or other techniques.

We are also investigating other surgical applications. While force feedback is irrelevant in microsurgery, it is critical in many other applications [4]. QSS can compute the force applied to each displaced control node in an elastic mesh. But it does not achieve an update rate compatible with haptic interaction (roughly 1000 Hz). To connect our simulator to haptic devices, we are developing fast techniques to interpolate between forces computed by QSS [31]. The elastic mesh model does not allow the explicit representation of incompressibility constraints often encountered in human-body tissues. A technique proposed in [32] to overcome this limitation is to apply artificial corrective forces to surface nodes to keep the object's volume approximately constant. Extending our collision-detection module to efficiently detect collisions of an object with itself is another short-term goal.

There are many other issues to consider, such as the simulation of knot-tying [33], the detection of mesh degeneracies (e.g., when one link crosses another), and the modeling of collision responses. As more issues are addressed in a simulation system, more algorithms will run concurrently, and their efficiency will become even more critical.

Acknowledgements

This research was conducted in the Stanford-NASA Biocomputation Center. It was supported by grants from the National Aeronautics and Space Administration (NAS-NCC2-1010), the National Science Foundation (IIS-9907060), and the NIH National Libraries of Medicine (NLM-3506). It has also benefited from equipment gifts made by Sun Microsystems and Intel. This paper was improved by discussions with Cynthia Bruyns and Frédéric Mazzella. Figure 3 was generated by Cynthia Bruyns using the deformable object simulator described in this paper and the cutting algorithms described in [34].

References

[1] J. Brown, K. Montgomery, J.-C. Latombe, M. Stephanides, A microsurgery simulation system, in: Medical Image Computing and Computer-Assisted Intervention – MICCAI 2001, 2001, pp. 137–144.

[2] J. Brown, S. Sorkin, C. Bruyns, J.-C. Latombe, K. Montgomery, M. Stephanides, Real-time simulation of deformable objects: Tools and application, in: Computer Animation 2001, 2001, pp. 228–236.

[3] D. H. House, D. E. Breen (Eds.), Cloth Modeling and Animation, A.K. Peters, Ltd., 2000.

[4] H. Delingette, Towards realistic soft tissue modeling in medical simulation, in: Proceedings of the IEEE : Special Issue on Surgery Simulation, 1998, pp. 512–523.

[5] R. V. O'Toole, R. R. Playter, T. M. Krummel, W. C. Blank, N. H. Cornelius, W. R. Roberts, W. J. Bell, M. Raibert, Measuring and developing suturing technique with a virtual reality surgical simulator, Journal of the American College of Surgeons 189 (1) (1999) 114–127.

[6] U. G. Kühnapfel, H. K. Çakmak, H. Maaß, Endoscopic surgery training using virtual reality and deformable tissue simulation, Computers & Graphics 24 (2000) 671–682.

[7] D. Terzopoulos, K. Waters, Physically-based facial modelling, analysis, and animation, Journal of Visualization and Computer Animation 1 (1990) 73–80.

[8] D. Baraff, A. Witkin, Large steps in cloth simulation, in: ACM SIGGRAPH 98 Conference Proceedings, 1998, pp. 43–52.

[9] A. Joukhadar, C. Laugier, Dynamic simulation: Model, basic algorithms, and optimization, in: J.-P. Laumond, M. Overmars (Eds.), Algorithms For Robotic Motion and Manipulation, A.K. Peters, Ltd., 1997, pp. 419–434.

[10] E. Keeve, S. Girod, P. Pfeifle, B. Girod, Anatomy-based facial tissue modeling using the finite element method, in: Proceedings of IEEE Visualization, 1996.

[11] R. M. Koch, M. H. Gross, F. R. Carls, D. F. von Büren, G. Fankhauser, Y. I. H. Parish, Simulating facial surgery using finite element models, in: ACM SIGGRAPH 96 Conference Proceedings, 1996, pp. 421–428.

[12] S. D. Pieper, D. R. Laub, Jr., J. M. Rosen, A finite-element facial model for simulating plastic surgery, Plastic and Reconstructive Surgery 96 (5) (1995) 1100–1105.

[13] M. Bro-Nielsen, S. Cotin, Real-time volumetric deformable models for surgery simulation using finite elements and condensation, Computer Graphics Forum (Eurographics '96) 15 (3) (1996) 57–66.

[14] S. Cotin, H. Delingette, N. Ayache, A hybrid elastic model allowing real-time cutting, deformations and force-feedback for surgery training and simulation, The Visual Computer 16 (8) (2000) 437–452.

[15] G. Picinbono, H. Delingette, N. Ayache, Non-linear and anisotropic elastic soft tissue models for medical simulation, in: Proceedings of the IEEE International Conference on Robotics and Automation, 2001.

[16] S. Cotin, H. Delingette, N. Ayache, Real-time elastic deformations of soft tissues for surgery simulation, IEEE Transactions On Visualization and Computer Graphics 5 (1) (1999) 62–73.

[17] C. Basdogan, Simulation of instrument-tissue interactions and system integration, in: Medicine Meets Virtual Reality 2001, 2001, http://eis.jpl.nasa.gov/~basdogan/Tutorials/MMVRTuto01.pdf.

[18] M. C. Lin, S. Gottschalk, Collision detection between geometric models: A survey, in: IMA Conference on Mathematics of Surfaces, 1998, pp. 37–56.

[19] D. Baraff, Curved surfaces and coherences for non-penetrating rigid body simulation, Computer Graphics 24 (4) (1990) 19–28.

[20] M. C. Lin, J. F. Canny, A fast algorithm for incremental distance calculation, in: Proceedings of the IEEE International Conference on Robotics and Automation, 1991, pp. 1008–1014.

[21] B. Mirtich, V-clip: Fast and robust polyhedral collision detection, ACM Transactions on Graphics 17 (3) (1998) 177–208.

[22] J. D. Cohen, M. C. Lin, D. Manocha, M. K. Ponamgi, I-COLLIDE: An interactive and exact collision detection system for large-scale environments, in: Proceedings of ACM Interactive 3D Graphics Conference, 1995, pp. 189–196.

[23] S. Gottschalk, M. C. Lin, D. Manocha, OBB-tree: A hierarchical structure for rapid interference detection, in: ACM SIGGRAPH 96 Conference Proceedings, 1996, pp. 171–180.

[24] J. T. Klosowski, M. Held, J. S. Mitchell, H. Sowizral, K. Zikan, Efficient collision detection using bounding volumes hierarchies of k-DOPs, IEEE Transactions On Visualization and Computer Graphics 4 (1) (1998) 21–36.

[25] I. J. Palmer, R. L. Grimsdale, Collision detection for animation using sphere-trees, Computer Graphics Forum 14 (2) (1995) 105–116.

[26] S. Quinlan, Efficient distance computation between non-convex objects, in: Proceedings of the IEEE International Conference On Robotics and Automation, 1994, pp. 3324–3329.

[27] D. Baraff, A. Witkin, Dynamic simulation of non-penetrating flexible bodies, Computer Graphics 26 (2) (1992) 303–308.

[28] G. van den Bergen, Efficient collision detection of complex deformable models using AABB trees, Journal of Graphics Tools 2 (4) (1997) 1–13.

[29] A. Smith, Y. Kitamura, H. Takemura, F. Kishino, A simple and efficient method for accurate collision detection among deformable polyhedral objects in arbitrary motion, in: Proceedings of the IEEE Virtual Reality Annual International Symposium, 1995, pp. 136–145.

[30] P. Volino, N. Magnenat-Thalmann, Collision and self-collision detection: Efficient and robust solutions for highly deformable surfaces, in: Eurographics Workshop on Animation and Simulation, 1995, pp. 55–65.

[31] F. Mazzella, The forcegrid: A buffer structure for haptic interaction with virtual elastic objects, Master's thesis, Computer Science Department, Stanford University (Sep. 2001).

[32] E. Keeve, S. Girod, B. Girod, Craniofacial surgery simulation, in: Proceedings of the 4th International Conference on Visualization in Biomedical Computing (VBC '96), 1996, pp. 541–546.

[33] A. Larsson, Intracorporeal suturing and knot tying in surgical simulation, in: Medicine Meets Virtual Reality 2001, 2001, pp. 266–271.

[34] C. D. Bruyns, S. Senger, Interactive cutting of 3d surface meshes, Computers & Graphics 25 (2001) 635–642.