Medicine Meets Virtual Reality
J D. Westwood et al. (Eds.)
IOS Press, 1999

Simulation of Tissue Cutting and Bleeding for Laparoscopic Surgery Using Auxiliary Surfaces

Cagatay Basdogan, Chih-Hao Ho, Mandayam A. Srinivasan

Laboratory for Human and Machine Haptics, Department of Mechanical Engineering and Research Laboratory of Electronics,
Massachusetts Institute of Technology,
Cambridge, MA, 02139

Abstract. Realistic simulation of tissue cutting and bleeding is important components of a surgical simulator that are addressed in this study. Surgeons use a number of instruments to perform incision and dissection of tissues during minimally invasive surgery. For example, a coagulating hook is used to tear and spread the tissue that surrounds organs and scissors are used to dissect the cystic duct during laparoscopic cholecystectomy. During the execution of these procedures, bleeding may occur and blood flows over the tissue surfaces. We have developed computationally fast algorithms to display (1) tissue cutting and (2) bleeding in virtual environments with applications to laparoscopic surgery. Cutting through soft tissue generates an infinitesimally thin slit until the sides of the surface are separated from each other. Simulation of an incision through tissue surface is modeled in three steps: first, the collisions between the instrument and the tissue surface are detected as the simulated cutting tool passes through. Then, the vertices along the cutting path are duplicated. Finally, a simple elastic tissue model is used to separate the vertices from each other to reveal the cut. Accurate simulation of bleeding is a challenging problem because of the complexities of the circulatory system and the physics of viscous fluid flow. There are several fluid flow models described in the literature, but most of them are computationally slow and do not specifically address the problem of blood flowing over soft tissues. We have reviewed the existing models, and have adapted them to our specific task. The key characteristics of our blood flow model are a visually realistic display and real-time computational performance. To display bleeding in virtual environments, we developed a surface flow algorithm. This method is based on a simplified form of the Navier-Stokes equations governing viscous fluid flow. The simplification of these partial differential equations results in a wave equation that can be solved efficiently, in real-time, with finite difference techniques. The solution describes the flow of blood over the polyhedral surfaces representing the anatomical structures and is displayed as a continuous polyhedral surface drawn over the anatomy.

1. Introduction

As the interest in minimally invasive surgical procedures grows, the gap between the old and new procedures as well as instruments also increases. Surgical simulators are currently being developed at many research centers and companies to train doctors and residents with new surgical devices and techniques and to close this gap. However, realistic simulation of tissue cutting and bleeding, which are important components of a typical surgical simulator, still remain unsolved. Although there has been some attempts to solve these two challenging problems in the past, real-time update requirements with limited computational power and the complexity of tissue separation and bleeding models have been the bottlenecks.

include a personal computer with high-end graphics card and a force-feedback device (PHANToM from SensAble Technologies Inc.). During the simulations, the user manipulates the generic stylus of the force-feedback device to simulate the movements of a surgical instrument and to feel its interactions with computer generated anatomical organs [2].

Integration of Vision and Touch: Software integration of visual and haptic modalities was achieved in an efficient manner by creating a hierarchical database for geometrical properties of objects and by programming with multi-threading techniques. In our simulations, visual and haptic servo loops were separated to achieve faster rendering rates [4]. When displaying visual images, it is known that the update rate should be around 30 Hz to appear continuous. On the other hand, to create a satisfying haptic display, the update rates for sending the force commands to the haptic interface needs to be about 1000 Hz. In order to create a VE that satisfies both requirements and optimally use the CPU power of a computer, the visual and the haptic servo loops need to be separated. That is, we run two loops at the same time, with the graphic loop updated at 30 Hz and the haptic loop updated at 1000 Hz. Since there are two loops running at the same time, there is always a chance a conflict occurs in accessing the shared memory. For example, in the case of simulating tissue cutting and deformation, changes in geometry require frequent updates of visual and haptic databases in real-time. This will cause a problem if one loop is writing data to the memory and the other loop is reading from there. In order to avoid this situation, we need to synchronize the two loops. The easiest way to synchronize two loops is to create a boolean flag. When one loop wants to access the shared data, it should check the flag first to see if the data is being accessed by the other loop. If the flag indicates that the shared memory is not being used, the loop can access the data and the flag is set to indicate that the shared memory is currently being used. If the flag indicates that the data is being used by the other loop, the loop waits until the other loop is done. When one loop finishes its operations with the shared memory, it sets the flag back to normal to let other loop access the data.

In order to compute collisions between the simulated instruments and 3D objects in an efficient manner, we create and utilize two types of hierarchical database [4]. The first one is a hierarchical bounding box tree for collision detection purposes. This hierarchical tree reduces the number of collision checks and enables us to find the collision point in a fast manner. During the pre-processing stage, we create this bounding box tree for each polyhedral object. At the highest level of the tree is a bounding box that covers the whole polyhedron. At lower levels, the bounding box of each parent has two children. Each child of the parent has a bounding box that covers approximately half the number of polygons of its parent. The hierarchical tree branches down in this manner until each child covers only one polygon. During the simulations, we first check the collision between the surgical tool and the highest level bounding box. If there is a collision, we then check the collision between the tool and the bounding boxes at branches. The process is repeated till the lowest level is reached. Finally, the collision between the polygon (that is inside the lowest level bounding box) and the line segment model of the surgical instrument is checked and the collision point is computed.

The second database is for primitive hierarchy. We separate each polyhedron into three types of primitives: vertices, lines (i.e. edges), and polygons. In our database, each primitive has a pointer to its neighboring primitives. The primitive hierarchy helps us to quickly access the

The two major challenges in simulating 3D tissue cutting are (a) developing physically-based tissue models that can estimate the interaction forces and the new positions of the nodes that are separated in real-time, and (b) accessing and updating the haptic and graphics databases in real-time to handle topological changes in geometry as a result of the cut. Song and Reddy [8] suggest an interactive 2D finite-element template for cutting elastic virtual surfaces. A template of nodes moves as the cutting tool moves over a flat 2D tissue surface. This technique is novel, but its extension to 3D virtual objects was not discussed in the paper. Reinig et al. [6] suggest a tissue cutting technique implemented on 3D surfaces that require incremental update of vertices, connectivity arrays, and texture maps. Their approach of mapping textures onto the regions where virtual tissue surface spreads apart to reveal the cut provide a visually realistic display. However, the paper does not include any details on haptic rendering, synchronization of vision and haptics, or a force model.

Simulation of bleeding is also a challenging problem because of the complexities of the circulatory system and the physics of viscous fluid flow. There are several fluid flow models described in the fluid mechanics literature, but most of them are computationally slow. Many attempts have been made to speed up the calculations by computer scientists to display the flow at interactive rates. A stable and computationally less expensive technique was developed by Kass and Miller [5]. This method is relatively easy to implement and is based on the solution of a simplified form of the Navier-Stokes equations, known as the wave equation. The fluid surface was represented with a 2D height field and the wave equation was solved to update the fluid columns. O'Brien and Hodgins [7] extend this approach to simulate a wider range of behaviors including splashing fluids. Their model runs at interactive speeds for small size meshes and the update rate slows down, as the mesh becomes finer. Chen et al. [3] describe a method for real-time fluid simulation in networked virtual environments. They solved the Navier-Stokes equations for 2D space and mapped the surface into 3D space using the corresponding pressures in the fluid flow field. The model can simulate objects that are moving or floating in fluids at interactive rates. To our knowledge, not much attention has been paid to the simulation of blood flow over tissue surfaces. Basdogan et al. [1] suggested two separate algorithms for simulating blood flow. To represent blood that flows over a surface and pools at the bottom of a depression, they implemented a *surface flow* algorithm. To represent pulsating bleeding (blood moving through air), they designed and implemented a *particle* flow model.

Here, we propose *auxiliary* surfaces, to simulate tissue cutting and bleeding. Auxiliary surfaces provide a convenient means for mapping 3D surface geometry onto a local 2D rectangular mesh to reduce the dimensionality of the physics-based problem in consideration (e.g. cutting, bleeding, deformation, etc.) by one so that it can be solved efficiently in real-time. The solution then can be mapped back to the original 3D surface to display the results. This paper is an extension of our earlier study, which suggests techniques for displaying tissue cutting and bleeding using the auxiliary surface approach [1].

2. Methods

Set-up: We have developed a proof-of-concept simulation system for simulating tissue deformations and feeling interaction forces. The hardware components of our simulation set-up neighbors of the collision points when it is necessary. For example, when a cutting instrument contacts a primitive of an object in the current loop, we know that, in the next loop, it can only contact with primitives that are in the close neighborhood of the contacted primitive. Therefore, we need to check collisions only between the instrument and these neighboring primitives, instead of all the primitives of the object. This approach reduces the computational time drastically and makes the collision detection almost independent of the number of polygons.

Auxiliary Surfaces: The concept of auxiliary surfaces is easy to explain. The equations that govern the physics of surgical interactions, particularly the tissue cutting and bleeding are too difficult to solve for arbitrary shaped 3D surfaces. Even if the numerical solutions are available, real-time update rates cannot be achieved due to the limited computational power of today's computers. However, these equations can be solved with relative ease in 2D space with some approximations. For example, tissue-cutting problem can be solved for a flat 2D grid using a FEM approach as suggested in [8]. Similarly, a 2D height map can be used to represent the blood surface, and the simplified Navier-Stokes equations can be solved in real-time for a 2D grid to update the height value at each node.

In order to generate an auxiliary surface, one can start with a 2D rectangular grid of points placed over the region of interest. Then, a ray is sent from each node of this grid to the 3D geometrical model of the tissue and the first intersection point of each ray with the 3D polygonal surface is computed. Figure 1 depicts this process of ray tracing and describes how we project the grid surface onto the 3D surface.

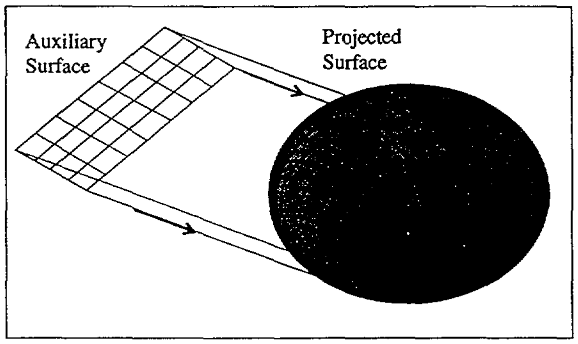

Figure 1. The schematic that describes the auxiliary surface: Equations that govern the physics of cutting or bleeding can be locally solved for 2D rectangular mesh (i.e. an auxiliary surface) and the results of computation are mapped back onto the surface of 3D object.

Simulation of Cutting: The cutting instrument was modeled as a line segment (see the grouping of laparascopic instruments described in [2]) in our simulations for fast detection of collisions. As the generic stylus of the force feedback device (and hence, the 3D model of the selected virtual instrument) is manipulated by the user, the tip and tail coordinates of the simulated instrument are updated. The tip and tail coordinates are then used to detect if the instrument collided with the 3D object (edge-polygon collision). Then, the collisions between the line segment model of the instrument and the edges of the 3D object (edge-edge collisions) are detected to determine the collision points as the instrument passes through the surface. We then determine and duplicate the vertices of polyhedron that are close to the collision points along the line of cut. Finally, a simple polynomial model is used to separate the duplicated vertices from each other to visually expose the cut. To simulate the physics involved in cutting and to compute the new positions of the separated vertices, one can use an auxiliary grid, made of a 2D network of point masses connected to each other with initially stretched springs and dampers. To simulate force interactions between a simulated cutting tool and tissue surfaces, a spring force proportional to the indentation depth along the surface normal (F = kx) and a damping force proportional to the tangential velocity (F = dv) have been implemented. A simple, but an efficient trick in here is to update the visual database only to expose the cut and not to update the haptic database. For simulating force interactions following the cut, we can still reflect the interaction forces to the user as if there is no change in the haptic database, except for returning zero force when the tip of the simulated instrument enters to the incised region and reducing the stiffness of the surrounding tissue.

Simulation of Bleeding: Fluid flow is governed by a set of partial differential equations known as the Navier-Stokes equations. Integration of these equations over time is computationally very expensive and thus real-time simulations are not possible. Kass and Miller [5] developed a computationally less expensive technique that is based on the solution of wave equations. Using this technique, we represented the fluid surface as a flat 2D surface grid and solved the wave equations using a finite difference scheme to update the fluid depth at each node of the grid. The process starts with the following wave equation:

$$\frac{\partial^2 h}{\partial t^2} = gd\left(\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2}\right) \qquad \text{(Eq. 1)}$$

where, *h(x,y)* represents the height of the tissue surface (base) at location (x,y) in the global reference frame, and *d(x,y)* is the depth of the fluid at location (x,y) relative to the base. Additionally, *g* is the gravitational constant and *t* is the time. For computational simplification, this equation is solved for fluid depth in the X and Y directions separately. Then, the resulting depth values are averaged and are used in the graphical display as well as in subsequent numerical iterations.

The surface flow algorithm uses two timers to control the calculations, each of which fires at an interval of approximately 0.1 second. When the first timer fires, the database of tissues is scanned to find actively bleeding regions. For each one, the exact locations and flow rates of the blood sources are calculated, based on the updated tissue positions and vital signs. These values are then used to adjust the depth values in the blood grid. For each blood source, a new blood depth is calculated from the source's flow rate, and the depth is stored in the appropriate element of the blood grid. When the second timer fires, the current values of the blood depths in the grid are iterated forward (over some time interval, according to equation 1), to calculate the movement of the blood during that time period. The new blood depths are then displayed graphically, as described in the next paragraph.

In order to display the flow of fluid (blood) over the polygonal tissue surfaces, we again followed the auxiliary surface approach. We generated a large rectangular polyhedral surface, colored red, to represent the blood surface, and initially placed it slightly below the tissue surface to hide it from view. Then, after solving the wave equation to find the depth of the fluid at each node on the rectangular grid, we raised the grid point (in the Y direction) so that it appeared at the appropriate depth value above the tissue surface (see figure 2).

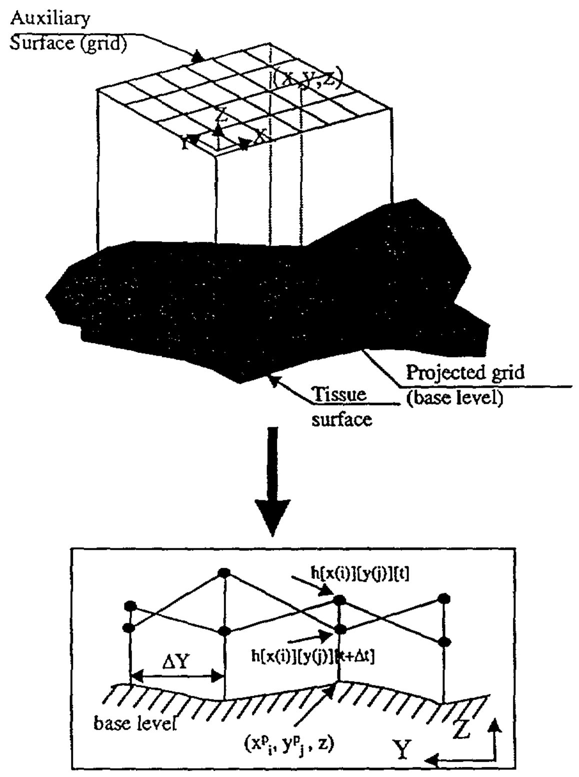

Figure 2. Schematic representation of the surface flow model. The first step, shown at the top of the figure, is to create a grid of vertices that is slightly below the tissue surface in the region of interest. This grid is created by projecting parallel rays down to the model surface, determining the contact points, and then moving the points down a small amount so that they are hidden from view. The second step, shown at the bottom of the figure, is to solve the wave equation over this grid in order to determine the height of blood at each point. The points with non-zero height are displayed by elevating each grid point the appropriate amount (the blood depth) so that it is visible above the tissue model.

After initializing all of the grid points using the projection technique described above, the blood surface was completely invisible, pending the blood depth calculations. If the tissue surface in the region of interest is altered, due to surgical actions such as incising tissue, the base coordinates of the blood grid must be updated. This is done by a subroutine that is called whenever a surgical instrument interacts with the 3D model. The contact point between the instrument and the tissue is sent to this subroutine, which computes the new tissue base coordinates in the neighborhood of the contact point.

Conclusions

In this paper, we propose *auxiliary* surfaces to simulate both tissue cutting and bleeding. Similar approach can be used to simulate other instrument-tissue interactions including tissue deformation and piercing. Our current approach to simulate cutting does not require us to make significant changes in either visual or haptic data structures, since only the vertices that are along the line of cut are duplicated. However, this may create a "zigzag" cut if the direction of the cutting path is changed frequently. Moreover, we also observed that implementation of topological changes in geometry such as cutting require a fast and frequent access to the database for updating connectivity information (e.g. inserting or subtracting vertices, lines, or polygons to/from visual and haptic databases) in real-time. Changes in geometry also destroy the pre-computations associated with efficient collision detection for the haptic display (e.g. the bounding boxes and hierarchical trees that are pre-constructed for fast collision detection purposes need to be updated after the cut). However, an easy remedy to this problem is to keep the original haptic database and return no collision when the cutting tool is inside the incised area. In the future, we plan to improve our database (e.g. using linked lists instead of arrays so that we can insert/subtract items to/from the database) to handle topological changes in a more efficient manner. We should also note that bleeding model creates some artifacts in the calculations as a result of the modeling simplifications, though they do not appear to be significant. The numerical solutions are quite stable and the computations are fast enough to display the results in real-time.

References

[1] Basdogan, C., Loan, J.P., Rosen, J.M., Delp, S.L., 1996, "An interactive model of the human lower limb for simulation of surgical procedures in virtual environments", the *Winter Annual Meeting of ASME'96*, BED-Vol. 33, pp. 439-440, Nov. 17-22, Atlanta.
[2] Basdogan, C., Ho, C., Srinivasan, M.A., Small, S., Dawson, S., 1998, "Force interactions in laparoscopic simulations: haptic rendering of soft tissues" *Proceedings of MMVR'6 Conference*, pp. 385-391, San Diego, CA, January 19-22.
[3] Chen, J. X., Lobo, N., Hughes, C. E. and Moshell, J. M., 1995, "Simulation and Synchronizatoin of Fluids in a DIS", First Workshop on Simulation and Interaction in Virtual Environments (SIVE), University of Iowa, Iowa City, Iowa, July 13-15.
[4] Ho, C., Basdogan, C., Srinivasan, M.A., 1998, "An Efficient Haptic Rendering Technique for Displaying Polygonal Objects with Surface Details in Virtual Environments" submitted to Presence: Teleoperators and Virtual Environments.
[5] Kass, M., Miller, G., 1990, "Rapid, Stable Fluid Dynamics for Computer Graphics," *Computer Graphics*, Vol. 24, No. 4, 1990, pp. 49-57.
[6] Reinig, K. D., Rush, C.G., Pelster H.L., Spitzer, V.M., Heath, J.A., 1996, "Real-Time Visually and Haptically Accurate Surgical Simulation", *Proceedings of MMVR'4 Conference*, pp. 542-545, San Diego, CA.
[7] O'Brien, J. F., Hodgins, J. K., 1995. "Dynamic Simulation of Splashing Fluids", Proceedings of Computer Animation '95, Geneva Switzerland, April 19-21, pp 198-205.
[8] Song, G., Reddy, N., 1995, "Tissue Cutting in Virtual Environments", *Proceedings of MMVR'3 Conference*, pp. 359-364, San Diego, CA.