Simulation of Surgical Cutting in Deformable Bodies using a Game Engine

Martin Kibsgaard[1], Kasper K. Thomsen[1] and Martin Kraus[2]
[1]*School of Information and Communication Technology, Aalborg University, Aalborg, Denmark*
[2]*Department of Architecture, Design, and Media Technology, Aalborg University, Aalborg, Denmark*

Keywords: Robotic Simulation, Medical Training, Virtual Reality, Physics Simulation.

Abstract: Simulators as a training tool for surgeons are becoming more important with the increase of minimally invasive surgery and a wish to limit training on animals, especially in the field of robotic surgery. Accessibility to surgery simulators is currently limited and the ability to cut is restricted. This paper presents a feasibility study for implementing academic methods in a low-cost game engine. Expanding on previous work, a low-cost surgery simulator is implemented to adress these issues. We focus on the implementation of cutting in deformable objects in a game engine. The deformable objects are implemented using a spring mass model combined with a volumetric tetrahedral mesh. The cutting algorithm is semi-progressive and allows for arbitrary cuts in the deformable objects. The prototype was evaluated by a chief surgeon with expertise in robot surgery and experience with commercial simulators. The low-cost prototype presents a step towards robotic surgery simulators that are able to simulate complete surgical procedures.

1 INTRODUCTION

Surgery simulators can reduce the amount of surgery training on animals and patients (Colaco et al., 2012; Liss et al., 2012). However, accessibility to surgery simulators is currently limited either because they are too expensive (Lallas et al., 2005) or because they are only academic prototypes (Mosegaard, 2006; 2003; Mor, 2001; Ganovelli & O'Sullivan, 2001). An implementation of a surgery simulator in a low-cost game engine, as suggested by Grande et al., can be part of a solution to both problems (2013).

This paper investigates the feasibility of implementing a surgery simulator in a low-cost game engine. Some of these game engines have functionality that is needed to create a real-time simulator, which helps lowering development costs by reusing existing components such as graphics and physics engines. We assume that the use of a game engine can ease the workload for other developers for further research or commercial products.

To simulate complete surgical procedures, a deformable model is needed to simulate tissue. In addition the deformable model should support cutting, as dissection of tissue is part of almost every surgical procedure (Poulsen, 2013).

Only some commercial surgery simulators support cutting by removing elements from the deformable object and robotic surgery simulators are limited to cutting of one-dimensional objects, e.g. dissection of a thin vein. On the other hand, some academic prototypes (Mosegaard, 2006; 2003; Mor, 2001; Ganovelli & O'Sullivan, 2001) support more advanced cutting of deformable objects.

Our approach tries to reduce the gap between commercial simulators and academic prototypes by using a low-cost game engine and thereby reduce development costs to make it easier to implement a complete surgical simulator.

Previous work is reviewed in Section 2. The implemented deformable model and cutting algorithm is described in Section 3. In Section 4 the evaluation of our solution is described. The paper ends with conclusions in Section 5 and considerations regarding future work in Section 6.

2 PREVIOUS WORK

This paper builds on research by Grande et al. (2013) and represents the next step in developing low-cost surgery simulators for robotic surgery. Grande et al. (2013) presented a simulation of the controls of

Kibsgaard M., K. Thomsen K. and Kraus M..
Simulation of Surgical Cutting in Deformable Bodies using a Game Engine.
DOI: 10.5220/0004670403420347
In *Proceedings of the 9th International Conference on Computer Graphics Theory and Applications* (GRAPP-2014), pages 342-347
ISBN: 978-989-758-002-4

the *da Vinci* surgery robot using off-the-shelf hardware, specifically a Razer Hydra game controller. This paper continues the work, but focuses on the software needed for cutting in a low-cost surgery simulator. Like Grande et al. (2013), this project is implemented using the low-cost game engine Unity 4.0 and focuses on training simulation for the *da Vinci* surgery robot.

The *da Vinci* Skills Simulator from the same company focuses on teaching surgeons the robot controls and not actual surgery procedures. This and other similar training simulators are fairly expensive and are perceived as unreasonably priced (Grande et al., 2013). Current surgery simulators have only limited support of cutting procedures. Two of the most advanced surgery simulators on the market are LapSim and SEP Robot but these are mainly focused on traditional surgery (Surgical Science, 2013) (SimSurgery, 2013). These simulators simulate cutting by removing entire mesh elements. This approach is simple and stable but can have too low resolution for precision cutting. For robotic surgery we have not been able to find any commercial simulators that support cutting of more than one-dimensional objects.

To simulate deformable objects, different methods have been presented. A common and simple method is the spring mass model. In this model, the object is represented as a three-dimensional mesh, with vertices representing fractions of the object mass. Edges in the mesh represent structural springs, that give the object its deformable properties. Bending springs and shearing springs can be introduced to increase model stability and rigidity (Erleben et al., 2005). These springs can be abstracted to second neighbor springs for non-grid meshes (Erleben et al., 2005). The spring mass model creates a system of differential equations that need an integration method. Mosegaard (2003) suggests to use a Verlet solver for real time purposes as it has the best combination of speed and stability. A different approach to simulate deformable objects is the finite element method, which is more physically correct and precise (Georgii & Dick, 2012). However, it is more difficult to change mesh topology in real time with this method (Mosegaard, 2006).

Previous research on real-time cutting in deformable meshes can be coarsley grouped into three categories: element removal, vertex moving and progressive remeshing (Ganovelli & O'Sullivan, 2001). The simplest method presented is element removal. This method removes a geometric element, like a tetrahedron, when it comes into contact with the cutting tool. This method is fast, but does not provide realistic looking results and requires high resolution meshes for precise cutting (Ganovelli & O'Sullivan, 2001). A mainly visual improvement is to move the vertices from the removed element to the surface of the cutting instrument, to make the cut the correct size. This model looks more natural when cut, but can cause simulation problems with the deformable model. The last method of cutting is to remesh the cut element, and have the newly created geometry reflect the cut. Using this method a much higher spatial resolution can be achieved in the cut. However, the method is fairly complex and can have problems with the increase of model complexity with larger and numerous cuts (Mor, 2001). A framework for medical simulation (SOFA, 2011) exists, however it is not used in this paper as development appears to have stopped and the paper wishes to expand on (Grande et al., 2013) by using a commercial game engine.

3 METHOD

3.1 Deformable Model

The spring mass model is chosen as the deformable model for this project, because of its simplicity and flexibility in regard to mesh topology. It is applied to tetrahedral meshes using second neighbor springs to improve volume preservation and force propagation.

As most objects, especially in game engines, are only surface representations of solid objects, a surface mesh to volumetric mesh conversion is necessary. For this procedure the program TetGen is integrated with Unity. This program performs Delaunay tetrahedralization to create a tetrahedral volume mesh (Si, 2011). A user interface has been implemented for Unity to enable developers to make this conversion without any user interaction with TetGen (see Figure 1).

After this conversion, all necessary information is stored in a comprehensive data structure as seen in Figure 2. Most of this data structure specifies associations between the different elements in the tetrahedral mesh.

Our spring mass model uses Hookes law for springs to calculate the forces on each vertex. In addition damping is added to each spring. The force from a spring on a vertex i is then Equation 1 and 2 combined (Erleben et al., 2005).

$$\vec{F}_i = -c_s \left(l_R - \left| \vec{x}_i - \vec{x}_j \right| \right) \frac{\vec{x}_i - \vec{x}_j}{\left| \vec{x}_i - \vec{x}_j \right|} \tag{1}$$

$$\vec{F}_i = -c_d \frac{(\vec{x}_i - \vec{x}_j) \cdot (\vec{v}_i - \vec{v}_j)}{\left| \vec{x}_i - \vec{x}_j \right|} \frac{\vec{x}_i - \vec{x}_j}{\left| \vec{x}_i - \vec{x}_j \right|} \tag{2}$$

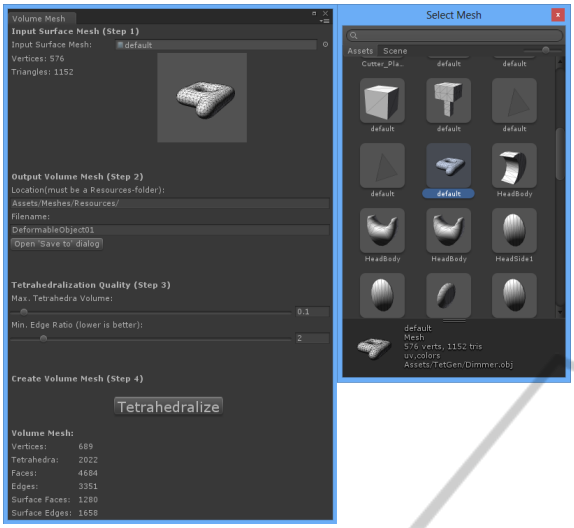

Figure 1: User Interface for converting any surface mesh in Unity to a tetrahedral volumetric mesh.

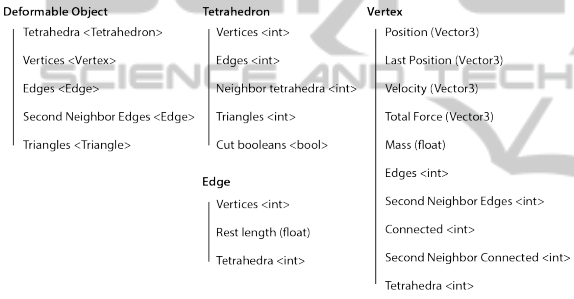

Figure 2: Data structures for each deformable object. <*type*>indicates that it is a list of *type* and (*type*) indicates that it is an instance of *type*.

where i and j are the vertices at each end of a spring, $\vec{F}_i$ is the force on the i^{th} vertex, c_s and c_d are the spring and damper constants, l_R is the spring's rest length, $\vec{x}$ is position and $\vec{v}$ is velocity.

For added volume preservation in the model, and to avoid unstable geometric deformations, the concept of second neighbor springs is implemented. These springs are calculated the same way as the structural springs, but have separate spring and damping constants. The forces from all connected springs are used in Equation 3 to calculate the new position for each vertex.

$$\vec{x}_{i+1} = 2\vec{x}_i - \vec{x}_{i-1} + \vec{a}_i \Delta t^2 \quad (3)$$

where $\vec{x}_i$ a vertex's current position, $\vec{a}$ is acceleration and t is time.

The Verlet solver is used as it has a good balance between speed and stability (Mosegaard, 2003). For each vertex a small linear drag is added to remove energy from the model and gravity is added to make the model fall down. To prevent the object from disappearing through the floor, a collision test with the ground plane is implemented. If a vertex is below the ground plane it is moved to the height of the ground plane, and its velocity is set to zero to approximate high friction. Lower friction can be achieved by keeping some or all of the velocity.

A similar approach is taken with the interaction object. Each vertex is tested against a sphere around the interaction object and moved outside the sphere and the velocity is set to zero. To simulate grabbing of tissue each vertex is tested for being inside the sphere of influence of the grabbing object, this sphere is slightly larger than the collision sphere. If a vertex is inside the sphere of influence when grabbing is enabled, forces are applied to the vertex to follow the grabbing object. This approach is chosen instead of directly manipulating the vertex position as it allows a cap on the force applied to vertices, which can guarantee model stability in case of unnaturally fast interactions.

3.2 Cutting

For this paper a simplified approach based on (Mor, 2001) is presented. The method presented by Mor (2001) has 60 unique cases of intersections. The method in this paper seeks to reduce this to only two cases. This model rests on one simplification; only tetrahedra that are cut completely through are cut in the model. This eliminates the per element progressive part of the method seen in (Mor, 2001), but it allows for much simplification. The only cases to be considered in the model are a cut intersecting three or four edges of a tetrahedron.

The algorithm for the model follows three steps: detecting intersections, subdividing the mesh and snapping vertices to neighboring elements.

To detect the intersection, the cutting instrument is first simplified to one cutting edge. A ray-triangle algorithm is used to test for tetrahedra intersections between the cutting edge and triangles in the mesh. For intersections with the edges, the cutting edge expands a plane from its position from the previous physics step and its current position. This plane consists of two triangles which use the same ray-triangle algorithm to detect edge intersections.The edge intersections are used to determine what type of cut is being created. The triangle intersections are used to determine if a tetrahedron is still being cut.

The subdivision of cut elements is implemented using the optimal tetrahedralization found by Mor (2001) and is illustrated in Figure 3. These subdivisions represent the two cases for tetrahedron in-

tersections.

In order to maintain mesh stability, vertices shared between subdivided elements must be detected and snapped to the same position. We base the procedure on the mesh topology in order to avoid snapping errors in case the object deforms during the cut.

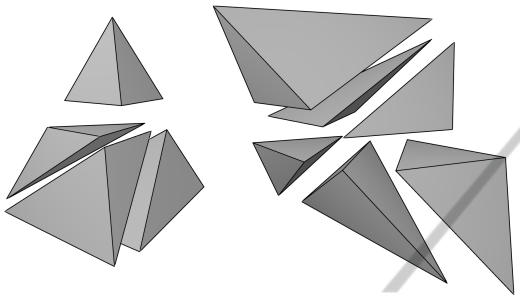

Figure 3: Tetrahedron with three edges cut (left) and with four edges cut (right). Both cut by a horizontal motion.

The current implementation does not consider degenerate tetrahedra that make the model unstable. Ganovelli & O'Sullivan (2001) suggests to collapse nearly degenerate tetrahedra based on edge-ratio. In its current form, the model assumes equal mass for all vertices, which leads to increased mass in cut areas. Changing the mass to reflect the cut exposes the vertices to larger forces in relation to mass, which in turn lead to model instability. Additionally, the springs created from a cut spring should have higher spring constants than the original spring based on Youngs modulus. However, this also induces larger forces on affected vertices. A solution to both problems is to run the simulation at a smaller physics timestep.

4 EVALUATION

Dr. Johan Poulsen, Chief Surgeon at Aalborg University Hospital, helped us evaluate our implementation of cutting in deformable objects, by comparing it to real surgery and the hospitals newly acquired robotic surgery simulation. He performs his surgeries using the *da Vinci* robot.

Johan Poulsen confirms that the interaction model suggested by Grande et al. (2013) mimics the input of the surgeon's console (controls to the robot) very well. However, a stereoscopic display is necessary to simulate the surgeon's console properly.

After trying the simulation for a few minutes, Johan Poulsen said that it is a good simulation of the surgery robot, and that the ability to cut is useful. With minor additions, it could be a great exercise. Adding an object in which to place the cut off tissue, and having a visible area of the tissue to remove, could be such an exercise and would only require a small amount of extra implementation. The cutting procedure Johan Poulsen tried during the evaluation can be seen in Figure 4.

Because Mosegaard (2006) suggests that gravity is not necessary in his heart surgery simulation, we asked Johan Poulsen about his opinion on the lack of gravity. For general-purpose surgery simulation, he believes that gravity is required. During real surgery, gravity is used to control blood flow and viscera placement; e.g. placing the patient almost upside down to move intestines away from the operating area during surgery in the lower part of the abdomen. During robotic surgery, a third arm is also sometimes used to keep the tissue in place while cutting, or to keep other tissue away. This would be difficult to simulate realistically without gravity.

Currently, the best training scenario for robotic surgery is operating on a pig that is still alive (wet training). The hospital's current simulator has reduced the amount of wet training by some; even without cutting and without any complete surgery exercises. Johan Poulsen believes that with cutting in a deformable object and a few more technologies, a simulation can reduce the dependency on wet training. In addition to cutting, what is required is: procedural material, suturing and bleeding. Procedural material (texture) is needed to create visually realistic cutting. The inside of an organ is different from the outside, and it should be possible for the user to tell unhealthy tissue from healthy tissue. Suturing and bleeding are already implemented in current simulators to some degree.

5 CONCLUSION

This paper presents a working prototype for cutting in deformable objects. The prototype has been evaluated by an expert user, who found it to be a good simulation of robotic surgery and the ability to cut useful. The prototype presents a step towards robotic surgery simulators that can simulate whole procedures, and presents the feasibility of implementing surgery simulators in game engines. Some features still need to be implemented to make the tool commercially applicable.

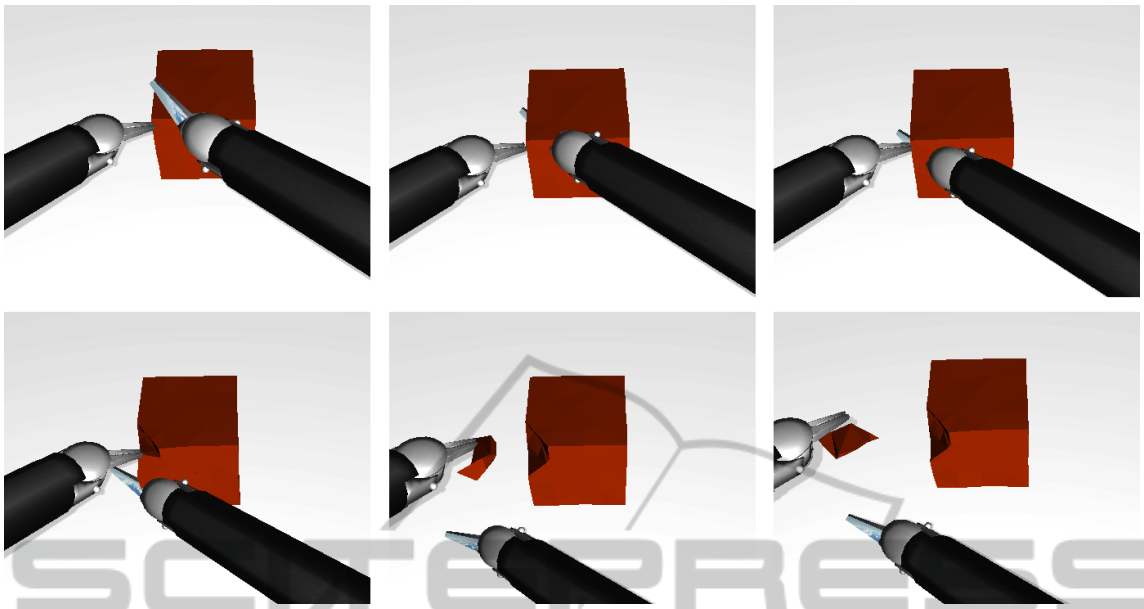

Figure 4: Image sequence of a cutting procedure, where the robot tools are used to remove a corner from a deformable box. The model contains 74 vertices, 196 tetrahedra elements and a total of 1167 structural and next neighbour springs. It is running at 475 frames per second with 150 physics steps per second on a i7-3770k at 3.5GHz using one of eight threads.

6 FUTURE WORK

The current deformable model is too elastic to simulate tissue because the simulation is only stable and in real-time for rather small spring constants. To improve performance, the deformable model was also implemented on the graphics processing unit (GPU). The massively parallel structure of the GPU allows many springs to be calculated simultaneously, which allows for a significantly smaller timestep and thereby larger spring constants. Currently this is not implemented with cutting, as the cutting algorithm requires more advanced data structures that are not easily transferrable to the GPU. However, the many ray-triangle calculations in the cutting algorithm could also be done in parallel on the GPU.

The expert evaluation concluded that several elements are needed before the system can simulate entire procedures. Suturing is an important part of surgery and it should be implemented as a part of the deformable model to have more realistic interaction with the rest of the simulated procedure. Bleeding is also an essential part to implement as it is an aspect that can complicate a surgical procedure by obscuring vision to important areas, as well as it being unhealthy for the patient. Volumetric texture is needed for the surgeon to be able to detect unhealthy tissue and know where to cut. This texture must stay consistent while the object is deformed and must still be correct with the new topology created by cutting.

ACKNOWLEDGEMENTS

Thanks to Johan Poulsen for sharing his knowledge and helping us evaluate our prototype. Thanks to Knud Henriksen for his questions and comments on the project.

REFERENCES

Colaco, M., Balica, A., Su, D., & Barone, J. (2012). Initial experiences with ross surgical simulator in residency training: a validity and model analysis. *Journal of Robotic Surgery*, pages 1–5.

Erleben, K., Sporring, J., Henriksen, K., & Dohlmann, H. (2005). *Physics-based animation*. Charles River Media Hingham.

Ganovelli, F. & O'Sullivan, C. (2001). Animating cuts with on-the-fly re-meshing. *Eurographics 2001 - Short Presentations*.

Georgii, J. & Dick, C. (2012). Efficient finite element methods for deformable bodies in medical applications. *Critical Reviews in Biomedical Engineering*, 40(2).

Grande, K., Kibsgaard, M., & Jensen, R. S. (2013). Low-cost simulation of robotic surgery. *VRIC '13: Proceedings of the 2013 Virtual Reality International Conference.*

Lallas, C. D., Davis, & Members Of The Society Of Urologic Robotic Surgeons, J. W. (2005). Robotic surgery training with commercially available simulation systems in 2011: a current review and practice pattern

survey from the society of urologic robotic surgeons. *Journal of Endourology*, 26(3):283 – 293.

Liss, M., Abdelshehid, C., Quach, S., Lusch, A., Graversen, J., Landman, J., & McDougall, E. (2012). Validation, correlation, and comparison of the dv trainer^{TM}and the dv surgical skills simulator^{TM}using the mimic^{TM}software for urologic robotic surgical education. *Journal of Endourology*, 26(12):1629–1634.

Mor, A. (2001). *Progressive Cutting with Minimal New Element Creation of Soft Tissue Models for Interactive Surgical Simulation*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Mosegaard, J. (2003). Realtime cardiac surgery simulation. Master's thesis, Department of Computer Science, University of Aarhus, Denmark.

Mosegaard, J. (2006). *Cardiac Surgery Simulation - Graphics Hardware meets Congenital Heart Disease*. PhD thesis, Department of Computer Science, University of Aarhus, Denmark.

Poulsen, J. (2013). Personal communication. Chief Surgeon at Aalborg University Hospital.

Si, H. (2011). Tetgen: A quality tetrahedral mesh generator. http://wias-berlin.de/software/tetgen/.

SimSurgery (2013). Sep robot: A learning tool for robotic surgery. http://www.simsurgery.com/robot.html.

SOFA (2011). Simulation open framework architecture. http://www.sofa-framework.org/home.

Surgical Science (2013). Lapsim: The proven training system. http://www.surgical-science.com/lapsim-the-proven-training-system/.