Proceedings of the 3rd Annual
IEEE Conference on Automation Science and Engineering
Scottsdale, AZ, USA, Sept 22-25, 2007

GPU-based elastic-object deformation for enhancement of existing haptic applications

Cristian J. Luciano, P. Pat Banerjee, and Silvio H. R. Rizzi

Abstract— Most haptic libraries allow user to feel the resistance of a flexible virtual object by the implementation of a point-based collision detection algorithm and a spring-damper model. Even though the user can feel the deformation at the contact point, the graphics library renders a rigid geometry, causing a conflict of senses in the user's mind. In most cases, the CPU utilization is maximized to achieve the required 1-kHz haptic frame rate without leaving any additional resource to also deform the geometry, while on the other hand, the Graphics Processing Unit (GPU) is underutilized. This paper proposes a computationally inexpensive and efficient GPU-based methodology to significantly enhance user perception of large existing haptic applications without compromising the original haptic feedback. To the best of our knowledge, this is the first implemented algorithm that is able to maintain a graphics frame rate of approximately 60 Hz as well as a haptics frame rate of 1 Khz when deforming complex geometry of approximately 160K vertices. The implementation of the algorithm in a virtual reality neurosurgical simulator has been successful to handle, in real time, complex 3D isosurfaces created from medical MRI and CT images.

I. Introduction

Haptic interaction along with elastic object deformation has been useful for learning many surgical procedure simulations. Virtual human organs are deformed by the contact forces generated from medical instruments. It is well known that realistic simulation of physically-based deformation of elastic objects is very challenging because of the complexity of the computation to be performed in real time. However, a simple spring-damper model can be adequate for achieving a relatively realistic deformation for real-time interactions.

Current haptics libraries such as GHOST and OpenHaptics [19] implement a spring-damper model to allow a point-based force-feedback interaction with 3-DOF haptic devices such as the PHANToM Desktop, Omni, and Premium 1.0. For point-based haptic applications, only the tip of the haptic stylus interacts with the virtual objects. In each frame, a collision detection algorithm checks if the Haptic Interaction Point (HIP) is inside the virtual object. If so, penetration depth is computed as the distance between the current HIP and the corresponding Surface Contact Point (SCP). Based on a spring-damper model at the contact point, the haptics library computes the reaction forces that are proportional to the penetration depth. The more the user penetrates the object, the higher the reaction forces applied by the haptic device.

Even though the spring-damper model implemented by the haptic library allows the user to feel the object resistance at the contact point, there is no explicit deformation of the object geometry. Therefore, the graphics rendering does not show any object deformation at all. The goal of this work is to enhance the user's experience by rendering a coherent graphics deformation of the virtual objects colliding with the haptic probe without compromising the current haptic frame rate.

II. Related work

CPU-based deformation with haptics: In terms of elastic deformation, [10] implemented a physically-based simulation algorithm using pre-computed Green's functions and Capacitance Matrix Algorithms to render object deformation. Haptics rendering is done by GHOST with the undeformed model. Reference [22] developed a suturing simulator based on a mass-springs grid of dynamic vertices to model a flat 3D mesh whose vertices can be displaced by a virtual pair of scissors. Reference [1] proposed a deformable model based on a similar mass-spring grid and a force propagation process to perturb the vertices around the contact point. Reference [5] implemented an implicit multi-contact deformation algorithm that solves Signorini's and Coulomb's laws with a fast Gauss-Seidel-like method obtaining haptic frame rates of 250Hz. Even though these previous works were successful to render interactive deformation with 3D meshes of fewer than 700 vertices, they cannot be extended to complex geometry without affecting the haptics performance, because the CPU power must be shared

Manuscript received April 30, 2007. This work was supported in part by the U.S. Department of Commerce NIST ATP Cooperative Agreement 70NANB1H3014 .

C. J. Luciano is with the Department of Mechanical and Industrial Engineering, University of Illinois, Chicago, IL 60607 USA (phone: 312-996-0579; fax 312-413-0447; email: clucia1@uic.edu).

P. Pat Banerjee is with the Department of Mechanical and Industrial Engineering, University of Illinois, Chicago, IL 60607 USA (email: banerjee@uic.edu).

S. H. R. Rizzi is with the Department of Mechanical and Industrial Engineering, University of Illinois, Chicago, IL 60607 USA (email: srizzi2@uic.edu).

between the haptics rendering and the graphics deformation.

GPU-based deformation without haptics: In the last two years the use of GPU has proven to be an effective solution to general purpose problems. In the field of interactive elastic object deformation, [16] developed two methods to model a spring-mass system for complex geometry taking advantage of the parallel processing of the GPU. Since the simulation is done exclusively on the GPU, the CPU is not fully utilized, causing an unbalanced load between CPU and GPU. Reference [17] proposed an alternative implementation of the mass-spring system on the GPU. Reference [6] implemented a physics-based deformation that combines a CPU simulation engine with a GPU render engine, achieving a more balanced load. Reference [11] developed another physically-based deformation algorithm with a mesh of 18,000 vertices. However, all these GPU-based deformation methods cannot be seamlessly integrated with existing polygonal-based haptic applications.

GPU-based deformation with haptics: [21] modified their GPU-based implementation to incorporate force feedback, achieving a haptic frame rate of 450 Hz deforming 3D meshes of up to 91,000 vertices. Unfortunately, that frame rate is not high enough to minimize haptic discontinuities and instabilities [13]. References [3] and [4] presented a GPU-based deformation method to achieve both haptics and graphics rendering, implementing a single DOF mass-spring-damper system at the contact point and using shape-functions to distribute the deformation of the closest vertices. Previously implemented algorithms are very useful to incorporate certain haptic feedback to existing graphics applications with deformation. However, since the primary focus of the applications is the graphics performance, they resulted in realistic graphic deformation at the expense of poor haptic performance.

III. Approach Overview

The goal of this work is to take an existing haptic application and enhance the user's perception by the incorporation of graphical deformation, while maintaining a 1-kHz haptic frame rate, and keeping the modifications of the haptic application at a minimum. The existing haptic application consists of a virtual reality ventriculostomy simulator developed by [15] (Fig. 1).

In most CPU-based haptic applications, a common polygonal geometry is processed by both the CPU (for the haptics rendering) and the GPU (for the graphics rendering). Haptics rendering includes collision detection, computation of forces, and bidirectional communication with the haptic device. Due to the complexity and sequential nature of the haptics rendering, as well as the GPU parallel and vector processing power, the haptics rendering takes longer than the graphics rendering. Therefore, the GPU is underutilized most of the time. If deformation is to be incorporated to the haptic application, it is more efficient to let the CPU deal with the haptics rendering while the GPU computes deformation for the graphics rendering in parallel.

Fig. 1. Virtual human brain being deformed by the insertion of a neurosurgical catheter

This paper focuses on point-based local deformation around the contact point, and thus, global deformation and volume preservation are beyond the scope of this research. The implemented algorithm can be thought of as a tradeoff between real-time interaction and sophisticated physics-based realism.

IV. Algorithm

A. Vertex displacement

GHOST implements a 1-DOF spring-damper model, along the normals of the object surface, to let the user feel the resistance of the object as the probe makes contact with the object and then pushes against its surface. Similarly, it is possible to render the object deformation by displacing the vertices along its normals. In the case of point-based haptics rendering, the force feedback calculation is done only at the contact point. However, for graphics rendering we need to compute not only the deformation at the contact point, but also at its neighborhood. In order to do that, we can take advantage of the vertex shader to displace each vertex in parallel while the CPU computes the force feedback.

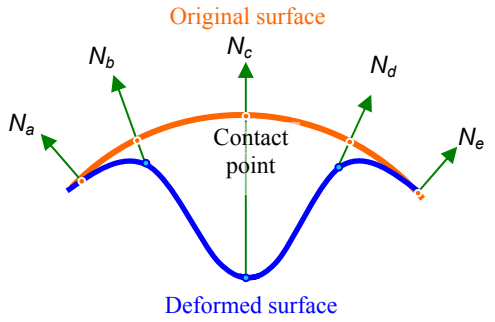

Fig. 2. Vertex displacements along its normals N_i done by the vertex shader

Fig. 2 shows how the vertices are displaced along its normals to deform the object surface. The maximum displacement is found at the contact point. Then the displacement decreases non-linearly as the vertices are located farther away from the contact point. The position of the displaced vertex V_{i+1} can be simply computed by the vertex shader as:

$$V_{i+1} = V_i + N_i * G(d_i)$$

where:
V_i = Position of original vertex
N_i = Normal vector of vertex V_i
$G(d_i)$ = Distribution function

The distribution function $G(d_i)$ defines the amount of displacement of the neighbor vertices. It consists of a Gaussian function with mean of zero and variance σ^2 (Fig. 4), which is computed as follows:

$$G(d_i) = p * e^{\left(-\frac{d_i^2}{2\sigma^2}\right)}$$

where:
d_i = Euclidean distance from the contact point to the vertex V_i
p = penetration depth $= |HIP - SCP|$

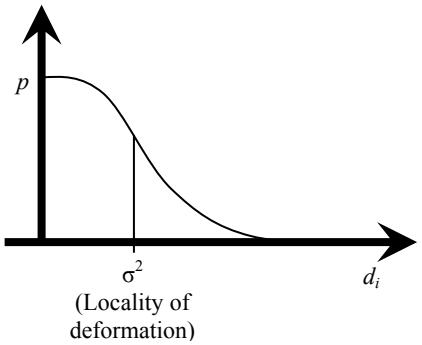

Fig. 4. Gaussian distribution defines the deformation profile

Each vertex shader computes the Euclidean distance from its vertex to the contact point. When that distance is zero, the Gaussian function returns the penetration depth previously computed by GHOST. As we move away from the contact point, the displacement tends to zero, producing a realistic "inverted-bell shape" at the vicinity of the contact point.

The variance σ^2 of the Gaussian function controls the distribution of the deformation. A small variance creates a very local deformation affecting only the closest vertices of the contact point, while a large variance produces a more global deformation around a larger area, allowing us to simulate the behavior of different materials. For example compare Fig. 1 and Fig. 3 to see the difference between brain and skin deformations.

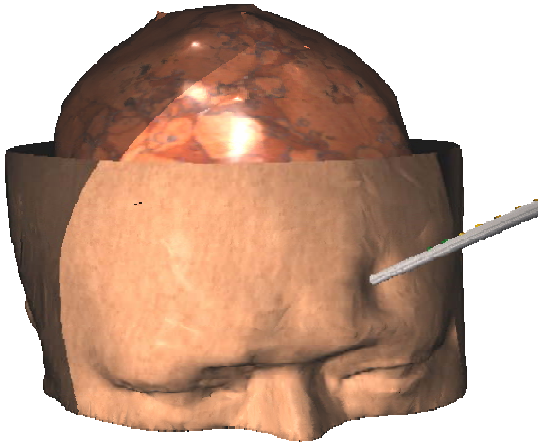

Fig. 3. Skin deformation with different variance

B. *Normal calculation*

In order to achieve a realistic rendering of the deformation, it is necessary not only to deform the surface displacing the vertices, but also to re-compute the normals of the deformed surface. This is a crucial step since lighting depends on the surface normals. A naïve approach would be to send all new vertices back to the CPU, let it compute the new normals, and then send all new normals back to the GPU. However, this would lead to a performance bottleneck due to the slow communication from the GPU to the CPU. Hence this computation needs to be done entirely on the GPU.

The new normals could be computed per vertex on the vertex shader. The fragment shader would then take the new computed normals at the vertices of each triangle and it would interpolate them to obtain the normals at each fragment inside the polygon. However, since we are dealing with a non-structured polygonal mesh, previously decimated to increase the haptics performance, the isosurface has triangles of different sizes. There are flat areas in which their vertex density is too low. Therefore, we get smoother shading performing the normal calculation

per fragment on the fragment shader.

On the CPU, the normals at each vertex are computed considering the vertices of the neighboring polygons. Unfortunately, the GPU cannot follow the same approach because the fragment shader lacks that information. Therefore, a different approach needs to be implemented. Instead of re-computing the normals, the GPU can slightly perturb the old normals to reflect the changes of the deformed surface. The idea is to rotate the original normal N_0 at the vertex v towards the contact point c by a certain angle θ about a rotation axis W (Fig. 5).

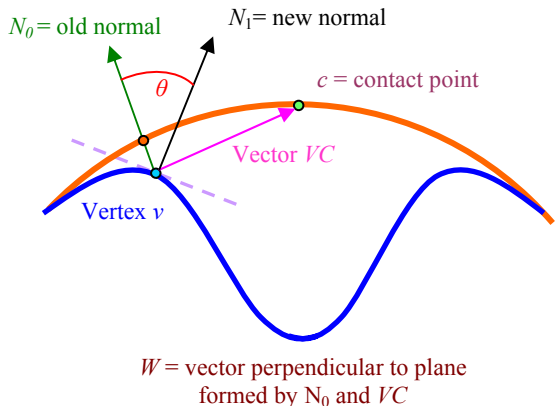

Fig. 5. Normal calculation done by the fragment shader

Vector VC is computed from the new vertex v (displaced by the vertex shader) to the contact point c. The rotation axis W (coming out of the page) is defined by the vector perpendicular to the plane formed by the vectors N_0 and VC and is computed with the cross product function as:

$$W = N_0 \times VC$$

The rotation angle θ depends on the slope of the deformed surface at each vertex v, or even more precisely at each fragment since the normal calculation is performed by the fragment shader. Since the deformed geometry follows a Gaussian function around the contact point, the surface slope at each fragment can be computed as the first derivative of the Gaussian function:

$$G'(d_i) = p * \left(-\frac{d_i}{\sigma^2} * e^{\left(-\frac{d_i^2}{2\sigma^2} \right)} \right)$$

Now the rotation angle θ is simply defined as:

$$\theta = \arctan\left(-G'(d_i)\right)$$

Knowing the rotation axis W and the angle θ, it is very efficient to compute the perturbed normal N_1 using the cross and dot product functions as follows:

$$N_1 = T + (N_0 - T) * \cos(\theta) + \\ + (N_0 \times W) * \sin(\theta)$$

where:

$$T = W * (N_0 \bullet W)$$

It is important to mention that since the geometry deformation and the normal computation need to be consistent, both the vertex and the fragment shaders must compute the Euclidean distance from the vertex to the contact point in eye-coordinate system.

The resulting normals are then used by the fragment shader to render the spot light as well as the directional light achieving a realistic rendering of the elastic deformation.

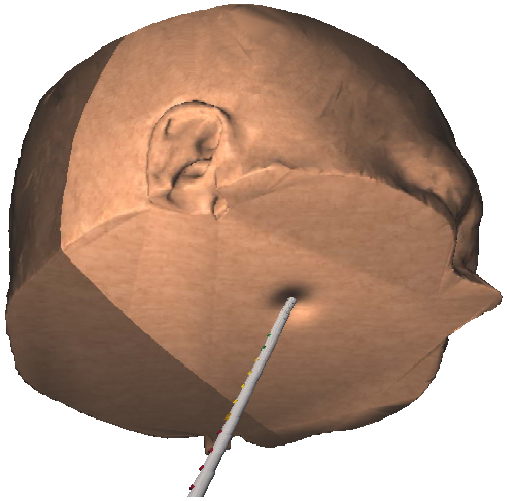

Fig. 6. Details of the normal perturbation at a low-vertex-density area

Fig. 6 shows the worst case scenario applying the deformation algorithm on an area of the geometry in which the vertex density is extremely low. The flat surface consists of few large triangles produced by the isosurface decimation process. Here the vertex shader does not displace any vertex at all because all of them are far away from the contact point. However, since the fragment shader computes the normals, it gives the illusion that the surface is properly deformed.

Previous Jacobian-based GPU-shader algorithms had to compute either the inverse of the 3x3 Jacobian matrix, or the tangents and binormals [17]. This step becomes very time consuming if the normal perturbation is done by the fragment shader. The algorithm presented here consists of a more efficient alternative.

V. IMPLEMENTATION

Two equivalent scene-graphs are stored in the main memory and handled by two libraries running on the CPU: GHOST and an open-source Open Inventor implementation [2]. A haptic scene-graph is created by GHOST to perform the haptics rendering, and a graphics scene-graph is created by Coin to perform the graphics rendering. Coin also serves as the communication link between the CPU and the GPU. Fig. 7 shows the interaction between the CPU and GPU, as well as the communication with the haptic device and the stereoscopic display of the *ImmersiveTouch*™ augmented reality system [14].

In the initialization phase, both haptic and graphic scene-graphs are created by reading a common set of 3D models stored in VRML files. The 3D models are complex polygonal isosurfaces (that represent skin and brain) previously extracted from a 3D volume created from CT scan data of a real patient.

In the interaction phase, since haptics rendering needs to run at 1 kHz, and 60 Hz is fast enough for the graphics rendering, GHOST and Coin run in two separate threads at different speeds. GHOST traverses the haptic scene-graph, detects the collision between the end-effector of the haptic stylus and the 3D models, and computes the reaction forces to be sent to the haptic device. Concurrently, Coin traverses the graphic scene-graph and sends the position of the contact point, the penetration depth, and the distribution of the deformation to the GPU.

The GPU deforms the geometry displacing the vertices around the contact point, and then computes the normals of the deformed geometry.

The deformation algorithm was implemented in C++ using OpenGL Shading Language (version 1.20) [7] and the CVS version of Coin (version 3.0), which supports shaders. The hardware used to run the simulation for the demo video was a dual Xeon 3.6 GHz Windows PC with nVidia Quadro FX 3400 graphics card.

VI. REAL-TIME PERFORMANCE RESULTS

Table I shows the performance of the haptics-based brain surgery simulator running the deformation algorithm. It demonstrates how the algorithm is able to maintain acceptable graphic and haptic frame rates.

Haptics frame rate was obtained by using the gstGetPhantomUpdateRate function available in GHOST. Graphics frame rate was obtained by running the function created by [8].

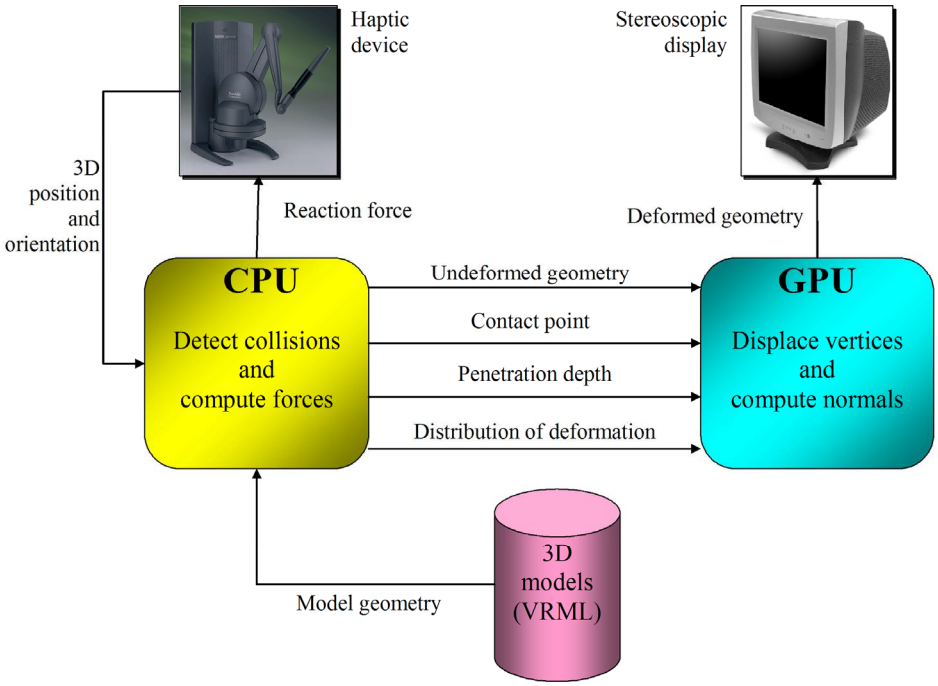

Fig. 7. Implementation architecture

TABLE I
PERFORMANCE RESULTS

3D model	vertices	faces	graphics fps	haptics fps
Brain	68,856	22,952	61	1000
Skin	90,207	30,069	60	1000
Skin + Brain	159,063	53,021	59	999

VII. CONCLUSIONS AND FUTURE RESEARCH

An efficient real-time deformation of elastic objects has been implemented using OpenGL Shading Language. While the CPU performs the haptic rendering allowing the user to feel the haptic deformation at the contact point, visual deformation of the area around the contact point is computed on the GPU. The main contribution of this paper is the fact that existing point-based haptic applications using a spring-damper model can be easily modified by programming an underutilized GPU to enhance user perception without affecting the original haptic performance.

One of the limitations of the algorithm is due to the simple Euclidean distance used to define the amount of deformation at a particular vertex and fragment. The algorithm presents some undesired side effects when deforming areas close to the contact point (in a straight line) but disconnected from the affected area. A better approach would be to compute the geodesic distance. Unfortunately, the geodesic distance cannot be easily computed on the GPU because it lacks global information about the object topology. However, the deformation achieved by the algorithm has shown to be realistic for most parts of the models used in the ventriculostomy simulator.

The algorithm considers a single contact point because the goal is to enhance an existing haptic application that currently works with a point-based collision detection library. However, the same approach can easily be extended to multiple-point haptic applications using an object-to-object collision detection library.

Since the collision detection and the computation of forces are performed with the original (undeformed) geometry, this approach cannot be extended to plastic (i.e. permanent) deformation.

GPU-based plastic as well as object-to-object deformation for haptic applications using a 6-DOF haptic device will be explored in future to find an open-source alternative to commercially-available haptic medical simulators, such as LapSim [9] and Lap Mentor [20].

REFERENCES

[1] Choi, K.S., Sun, H., Heng, P.A. 2003. Interactive deformation of soft tissues with haptic feedback for medical learning. *IEEE Transactions on Information Technology in Biomedicine*, Vol. 7, No. 4, December.

[2] Coin, Open Inventor, www.coin3d.org

[3] De Pascale, M., De Pascale, G., Prattichizzo, D., Barbagli, F. 2004. A GPU-friendly method for haptic and graphic rendering of deformable objects, In *Proceedings of Eurohaptics*, pp. 44-51.

[4] De Pascale, M., Sarcuni, G., Prattichizzo, D. 2005. Real-time soft-finger grasping of physically based quasi-rigid objects, In *Proceedings of World Haptics Conference*, pp. 545-546.

[5] Duriez, C.; Dubois, F.; Kheddar, A.; Andriot, C Realistic haptic rendering of interacting deformable objects in virtual environments, *IEEE Transactions on Visualization and Computer Graphics*, Vol.12, Iss.1, Jan.-Feb. 2006, pp. 36- 47.

[6] Georgii, J., Westermann, R. 2005. Interactive simulation and rendering of heterogeneous deformable bodies. In *Proceedings of VMV*.

[7] GLSL, OpenGL Shading Language, www.opengl.org/documentation/glsl/

[8] Howard, T., Craven, M., http://www.cs.manchester.ac.uk/software/OpenGL/frames_wl.txt

[9] Immersion Medical Corporation, www.immersion.com/medical/products/laparoscopy/index.php

[10] James, D.L., Pai, D.K. 2001. A unified treatment of elastostatic contact simulation for real time haptics. *Haptics-e, The Electronic Journal of Haptic Research*, Vol. 2, No. 1

[11] James, D.L., Pai, D.K. 2002. DyRT: Dynamic Response Textures for Real Time Deformation Simulation With Graphics Hardware. *ACM Transactions on Graphics (SIGGRAPH 2002)*, Vol. 21, No. 3, July, 2002, pp. 582 - 585

[12] Kim, M., Punak, S., Cendan, J., Kurenov, S., Peters, J. 2006. Exploiting graphics hardware for haptic authoring. In *Proceedings of Medicine Meets Virtual Reality 14*, pp 255-260.

[13] Lin, M., Salisbury, K. 2004. Haptic Rendering--Beyond Visual Computing, *IEEE Computer Graphics and Applications*, Volume 24, Issue 2

[14] Luciano, C., Banerjee, P., Florea, L., Dawe, G. 2005 Design of the *ImmersiveTouch*™: a High-Performance Haptic Augmented VR System, *Proceedings of Human-Computer Interaction.*

[15] Luciano, C., Banerjee, P., Lemole, G.M., CHARBEL, F. 2006. Second generation haptic ventriculostomy simulator using the ImmersiveTouch™ system. In *Proceedings of Medicine Meets Virtual Reality 14*, pp 343-348.

[16] Mosegaard, J., Sørensen, T.S. 2005. GPU accelerated surgical simulators for complex morphology. In *Proceedings of IEEE Virtual Reality.* pp. 147-153.

[17] Randima, F. 2004. GPU-Gems: Programming techniques, tips and tricks for real-time graphics, chapter 42.

[18] Ranzuglia, G., Cignoni, P., Ganovelli, F., Scopino, R. 2006. Implementing mesh-based approaches for deformable objects on GPU. In *Proceedings of 4th Conference Eurographics*.

[19] Sensable Technologies, www.sensable.com

[20] SIMBIONIX, www.simbionix.com/LAP_Mentor.html

[21] Sørensen, T.S., Mosegaard, J. 2006. Haptic feedback for the GPU-based surgical simulator. In *Proceedings of Medicine Meets Virtual Reality 14*. pp. 523-528.

[22] Webster, R.W., Zimmerman, D.I., Mohler, B.J., Melkonian, M.G., Haluck, R.S. 2001. A prototype haptic suturing simulator. In *Proceedings of Medicine Meets Virtual Reality 9*. pp 567-56.