Haptic Feedback for the GPU-Based Surgical Simulator

Thomas Sangild SØRENSEN [a,1] and Jesper MOSEGAARD [b]

[a] Centre for Advanced Visualisation and Interaction, [b] Department of Computer Science, University of Aarhus, Denmark

Abstract. The GPU has proven to be a powerful processor to compute spring-mass based surgical simulations. It has not previously been shown however, how to effectively implement haptic interaction with a simulation running entirely on the GPU. This paper describes a method to calculate haptic feedback with limited performance cost. It allows easy balancing of the GPU workload between calculations of simulation, visualisation, and the haptic feedback.

Keywords. Haptic feedback, GPU, surgical simulation, congenital heart disease.

Introduction

Surgical simulators have traditionally been implemented on the CPU and many algorithms have been proposed in order to calculate realistically looking soft-tissue deformations in real-time. An overview of the field is provided in [1]. Haptic feedback is often used to increase the realism of user interactions. This provides an additional challenge as force feedback must be provided at least at 500 Hz to feel smooth. To achieve such an update rate from a simulation running at a much lower frequency, extrapolation schemes have been developed, e.g. [2,3]. Overall,

Figure 1. Surgery simulation on a congenitally malformed heart. The simulator runs entirely on the GPU. The surgical instruments provide haptic feedback.

speed has been a major concern as many time-consuming tasks all had to be handled on the CPU. Motivated by this issue, it was shown in [4] that a twenty to thirty fold acceleration of a spring-mass based surgical simulation could be achieved when moving computations from the CPU to the GPU (i.e. the graphics card). This allowed real-time
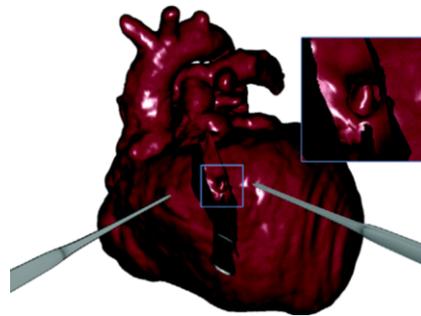
---

[1] Corresponding author: Thomas Sangild Sørensen, CAVI, University of Aarhus, Aabogade 34, 8200 Aarhus N, Denmark; E-mail: sangild@cavi.dk

surgical simulation on very complex organs, such as the heart, for the first time. Calculating tissue deformations on the GPU does however expose some previously unaddressed problems on how to resolve haptic interaction. Since communication with the haptic devices must be handled on the CPU, synchronization and data transfer between the GPU and the CPU is necessary. Unfortunately this is a relatively slow operation. Hence, we must carefully design the communication scheme to avoid new performance bottlenecks.

In this paper we describe and evaluate an efficient method of haptic interaction with the GPU based surgical simulator [4]. Haptic feedback is provided in response to collisions between instruments and tissue. The overall design criterion is to allow efficient, smooth, two-handed haptic interaction and easy balancing of the workload between simulation, visualisation, and delivery of force feedback.

A driving force behind the described simulator and proposed tools is an effort to develop a surgical simulator dedicated to procedures in congenital heart disease, hence the illustration at the front page (Figure 1).

1. Materials and Methods

1.1. Hardware and Software Platforms

The hardware platform used to evaluate the simulation was a personal computer running Windows XP on an AMD FX-55 CPU, and 2 GB of memory. The graphics bus was PCI Express x 16. The proposed algorithms were tested on three different graphics cards, a Geforce 6800 Ultra, a Quadro FX 4400, and a Geforce 7800 GTX, all from Nvidia. Two Phantom Omnis (Sensable Technologies) were used to achieve haptic interaction by low-level access through the accompanying OpenHaptics Toolkit. All programming was done in C++ using OpenGL and Cg with some manual modifications of the compiled vertex and fragment programs. The cardiac model used throughout this paper were obtained from three-dimensional MRI [5] using the segmentation algorithm described in [6]. The marching cubes algorithm was used for all surface reconstructions. Throughout this paper we will consider a spring mass simulation of 20.270 particles visualised by a surface of 137.490 faces.

1.2. Spring Mass Simulation on the GPU

This section provides a short description of our GPU based simulator, since some terminology from its implementation is necessary to explain the extension with haptic interaction. Further details of the GPU implementation can be found in [4]. First we discretise the volume of the concerned organ, e.g. the heart muscle mass, into a set of particles arranged in a regular three-dimensional grid (Figure 2). Each particle is connected in a fixed pattern to its 18 nearest neighbours. The grid is mapped to a 2D-texture such that each particle is represented by a single
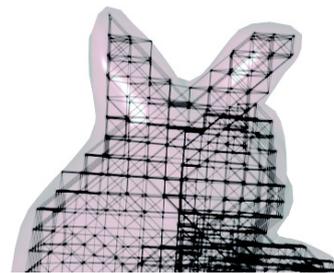

Figure 2. Particles in the spring mass system are connected in a regular, three-dimensional grid (black). Each particle is allowed to move, but constrained by the springs to neighbouring particles.

texel (Figure 3). We name this texture the *position-texture*. Conceptually, parallel computation of the spring mass system is invoked by rendering a single quad covering the entire position-texture. Processing of the white (void) particles is avoided by a depth test. A fragment program computes the forces that influence each particle due to its spring connections and the spring mass differential equation is solved by numerical integration to obtain updated particle positions. We refer to one pass of these calculations as a *simulation-step* in the remaining paper. A surface is

Figure 3. Excerpt of the position-texture. Each particle in the three-dimensional grid (Figure 2) is mapped to a unique texel (non-white). White texels correspond to void simulation particles outside the tissue.

constructed from the boundary nodes in the spring mass system and used for visualisation. During visualisation, a vertex program performs texture lookups in the position-texture to obtain the most recent particle positions. This allows us to use a static display list of the initial surface to also render it deformed. Rendering this surface is subsequently referred to as a *visualisation-step*. In [7] a mapping that decouples the visualised mesh from the physical simulation was presented. This allows for arbitrarily detailed surface meshes to be deformed by an underlying physical simulation (Figure 1 and Figure 2).
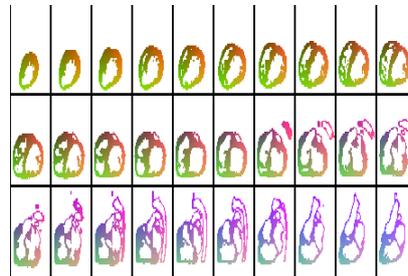
1.2.1. Probing and Grabbing

The probing gesture (i.e. touching the tissue with an instrument) was realised entirely on the GPU by extending the fragment program responsible for the simulation-step. Prior to writing the final position to the position-texture, each fragment determines if the corresponding particle has moved inside the bounding ellipsoid of an instrument. In that case the particle is projected to the boundary of the ellipsoid before updating the position-texture.

The grabbing gesture was designed to make use of both the CPU and the GPU. At the beginning of the gesture we read back the position-texture to the CPU *once* to identify particles inside the instrument's bounding volume and store pointers to these. For the duration of the grabbing gesture a dedicated fragment program writes the absolute positions of the grabbed particles into the position-texture based on the current position of the instrument.

1.2.2. Haptic Feedback

Communication with the haptic device is handled by the haptic thread on the CPU. We are consequently required to continuously read back data from the GPU to the CPU in order to provide haptic feedback. We could transfer the entire position texture and let the CPU compute the relevant forces from the current simulation state. It is much cheaper however to compute these forces on the GPU and read back only the result. In the following we describe how to achieve this for the grabbing and probing gestures:

During a grabbing gesture the CPU holds a list of the grabbed particles and corresponding coordinates in the position-texture as described in section 1.2.1. We create an off-screen *force-buffer* at the size of this list and for each particle a dedicated

fragment program is run. This fragment program looks up the position of all neighbours to the current particle in the position-texture and calculates the force stored in each spring connection. The individual spring forces are summed to find the overall force vector affecting the particle. This vector is stored in the force-buffer at the position corresponding to the processed particle. Finally the force-buffer is read back to the CPU and passed to the haptic thread. If two hands are active both sets of corresponding forces are returned in a single readback. This is an optimisation as each readback implies a relatively costly synchronisation between the GPU and the CPU.

During a probing gesture the situation is more difficult since the set of particles contributing to the force feedback is no longer constant during the gesture, but changes in each frame depending on the position of the instrument. We can re-use the force-buffer approach however, if we extend it with a fast method to pick the particles that collide with the instrument. Note that instruments can only collide with surface particles. As a fast picking algorithm, we propose to render a *simulation-surface* into an offscreen *picking-buffer*. A simulation-surface is a mesh that connects the boundary particles in the spring mass simulation. It is not necessarily identical to the high-resolution mesh used in the visualisation-step due to the mapping presented in [7]. The

Figure 4. Contents of the picking-buffer. The simulation-surface is projected into the picking-buffer as represented by the black mesh. The surface is "shaded" with a colour representing the texture coordinate in the position-texture (Figure 3) of the corresponding particle in the spring-mass system (Figure 2).

simulation-surface is rendered as seen from the base of the associated instrument. It will be "shaded" with colours that provide texture coordinates to the nearest simulation node in the position-texture. The result is a Voronoi-like diagram as illustrated in Figure 4. During force-buffer calculations we use this diagram to identify which particles in the spring mass system that are potentially touching the instrument. Consider a point on the boundary of the instrument. Transforming this point with the modelview and projection matrices that were used when rendering the picking-buffer results in a *picking-coordinate*. Using this picking-coordinate for a texture lookup in the picking-buffer provides the texture coordinate to the corresponding particle in the position-texture due to the shading of the simulation-surface. To determine if the instrument is actually near the picked node, the third colour-component of the picking-buffer stores the distance from the picked point on the simulation-surface to the instrument.

The discussion above describes how to use the picking-buffer to find the position-texture coordinates of the particles colliding with the probing instrument. It boils down to a single texture lookup for each sampling point on the boundary of the instrument. In the previously described case of grabbing, these position-texture coordinates were given directly as input to the fragment program which updated the force-buffer. When probing, we extend this fragment program to use instead a texture lookup in the picking-buffer to obtain the desired position-texture coordinate. The calculation of probing forces is initialised from the CPU, which keeps a list of sampling points on each instrument's boundary. One by one, the sampling points are projected onto the picking buffer and the resulting picking-coordinates passed to the force computing fragment program. Each resulting force is stored in the corresponding entry in the force-buffer, which is finally read back to the haptic thread on the CPU.
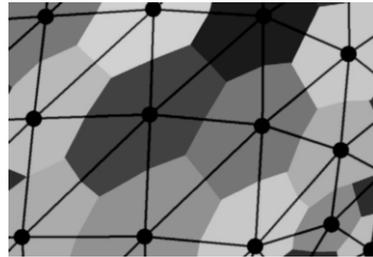
Table 1. GPU rendering times on selected graphics cards.

[1] One simulation-step in a spring mass system of 20.270 nodes (18 neighbours each).
[2] One visualisation-step according to [7] (90.868 vertices / 137.490 faces).
[3] One rendering step of the off-screen picking-buffer (12.031 vertices / 46.928 faces).
[4] One calculation and subsequent readback of force feedback from 50 particles.

	Geforce 6800 Ultra	Quadro FX 4400	Geforce 7800 GTX
Simulation-step [1]	2.5 ms	3.5 ms	0.9 ms
Visualisation-step [2]	19.9 ms	20.8 ms	11.1 ms
Picking-buffer [3,4]	6.3 ms	5.3 ms	2.6 ms
Force-buffer [4]	0.3 ms	0.7 ms	0.2 ms

2. Results

Table 1 summarises the performance measurements obtained from the simulator for each of the tested graphics cards. As expected the newest GPU, the Geforce 7800 GTX, is the fastest for fragment and vertex processing (rows 1-3). It performs significantly better than the other two cards. To calculate and read back the accumulated spring forces the cards perform comparably due to the relatively high synchronisation cost of initiating a data transfer (row 4).

A simulator was developed to support haptic interaction with cardiac models. One example from the running simulator is shown in Figure 1, where an incision was made to reveal the exact location of a ventricular septal defect in a congenitally malformed heart. A movie of the running simulator is available at
http://www.daimi.au.dk/~sangild/MMVR06.wmv.

3. Discussion

It is clear from Table 1 that surface visualisation is the single most expensive task in the simulator. In fact, several simulation-steps could be performed for each visualisation-step while maintaining an overall frame rate of at least 30 Hz. Here the term *overall frame rate* covers the accumulated cost of a number of simulation-steps, a visualisation-step, rendering of the picking-buffers, and finally calculation and readback of the force buffers. Each simulation-step should be followed by rendering and readback of the force-buffer to ensure the highest update frequency of the haptic devices. When probing, the additional cost of updating the picking-buffers must also be considered. As seen from the 3rd row in Table 1, updating the picking buffer after each simulation-step significantly reduces the number of simulation-steps possible per overall frame. We briefly mention two strategies for high-frequency haptic rendering that do not necessitate picking-buffer updates after each simulation-step. The first strategy is to read back the force feedback from the GPU at e.g. 30 Hz and use the algorithms presented in [2,3] to extrapolate this data to the desired update frequency on the CPU. Another strategy is to allow the use a slightly outdated picking-buffer but proceed with

the update and readback of the force-buffer based on updated instrument positions. We have chosen the latter of these two strategies.

Using the Geforce 7800 GTX as an example, we describe a combination of steps that will lead to an overall frame rate of 30 Hz running the simulator with haptic interaction. An overall frame rate requirement of 30 Hz corresponds to 33 ms available per frame. 11.1 ms are used for surface visualisation. Two picking-buffers (one for each hand) will be updated once in each overall frame, leaving 17 ms for simulation and force readback. During this period we can perform approximately 15 simulation-steps with subsequent rendering and readback of the force-buffers. This corresponds to a simulation-step frequency of 450 Hz.

Comparing the overall system performance to a similar implementation on the CPU, the GPU implementation is much faster. With the work presented in this article we believe to be one step closer to a fully functional cardiac surgery simulator. As a next step we are planning to extend the simulator with support for suturing of patches to close e.g. septal defects.

Acknowledgments

References

[1]   A. Liu, F. Tendick, K. Cleary and C. R. Kaufmann. A survey of surgical simulation: applications, technology, and education. Presence: Teleoperators and Virtual Environments. 2003;12(6):599–614.

[2]   F. Mazzella, K. Montgomery, J. C. Latombe. The Forcegrid: A Buffer Structure for Haptic Interaction with Virtual Elastic Objects. ICRA 2002:939-46.

[3]   G. Picinbono, J-C. Lombardo, H. Delingette, N. Ayache. Improving realism of a surgery simulator: linear anisotropic elasticity, complex interactions and force extrapolation. Journal of Visualisation and Computer Animation, 13(3):147-67.

[4]   J. Mosegaard, P. Herborg, T.S. Sørensen. A GPU accelerated spring-mass system for surgical simulation. Medicine Meets Virtual Reality 13, 2005:342-8.

[5]   T.S. Sørensen, H. Körperich , G.F. Greil, J. Eichhorn, P. Barth, H. Meyer, E.M. Pedersen, P. Beerbaum. Operator-independent isotropic three-dimensional magnetic resonance imaging for morphology in congenital heart disease: a validation study. Circulation. 2004;110(2):163-9.

[6]   T.S. Sørensen, E.M. Pedersen, O.K. Hansen, K. Sørensen. Visualization of morphological details in congenitally malformed hearts: Virtual, three-dimensional reconstruction from magnetic resonance imaging. Cardiology in the Young. 2003;13(5):451-60.

[7]   J. Mosegaard, T.S. Sørensen. Real-time Deformation of Detailed Geometry Based on Mappings to a Less Detailed Physical Simulation on the GPU. IPT & EGVE Workshop 2005:105-10.