Haptic and Graphic Rendering of Deformable Objects based on GPUs

M. de Pascale, G. de Pascale, and D. Prattichizzo
Dipartimento di Ingegneria dell'Informazione
via Roma 56 — Siena 53100, Italy
Email: [mdepascale, gdepacale, prattichizzo]@dii.unisi.it

Abstract— **In this paper we present a new method for real-time interactive haptic and graphic rendering of complex objects locally deformed by multiple contacts. Core algorithms have been designed to be executable also on videoboard's GPU, thus taking advantage of parallel matrix and vector computational power. Although complex physical simulation has been simplified to run on GPUs, results are characterized by high visio-tactile realism perceived by users. Graphical rendering algorithms can be easily added to pre-existing vertex shaders/programs. The proposed method makes use of common triangular meshes, thus making the method a good choice when adding haptic feedback to existing graphical applications.**

I. Introduction

ONE of the most valuable benefits offered by the increasing computational power of PCs is the possibility to improve virtual simulations of real environments through the haptic rendering, that use force feedback-enabled devices to simulate touch [1], [2]. Realistic real-time haptic rendering is a heavy-computation task, and the required computational power is usually taken away from graphics. The haptic rendering load is mainly due to the use of complex mathematical models, needed to accurately simulate physical behaviour, instead of simple approximations that can result satisfactory for 3D rendering. In this paper we present a new method able to realistically simulate visio haptic interaction of locally deformable objects, even on mainstream PCs. Two are the main ideas at the base of the proposed method:

- trading off between exact physical simulation and realistic user perception
- using modern GPUs parallel and vector computational power [3], [4]

The proposed model simulates deformable objects as a set of uncoupled elements, thus avoiding any possible stability issues due to element interaction. Moreover it uses a highly customizable non-linear dynamic local-model able to simulate also objects with different compliance and dynamic behaviours. Such model is perceived by the user as more realistic than the simple elastic model. The absence of links between the dynamic elements modeling the object is compensated by the use of *force fields* to propagate deformations from contact points to the object surface. Deformations are fully customizable, both geometrically than dynamically. The proposed approach is also well suited to handle more than one contact point. A relevant attention is devoted to the computational aspects. At each cycle, only a small stream of data is sent down to the video board instead of the whole deformed mesh. This saves bandwidth and allows for very complex meshes to be used [5]. Moreover any existing graphic shader[1] code can be used, by simply adding few instructions. All these features allow running of real-time visio/haptic simulations with a high level of realism also on commonly diffused PCs.

II. Related Works

At the present the most diffused methods for visio-haptic simulation of deformable objects are springs-masses systems and the many finite element based variations such as FEM [6], BEM [7], LEM [8] and REM [9]. The two main approaches used for such simulations are dynamic and quasi-static. In the dynamic approach the simulation is performed by real-time integration of the motion differential equations. In the quasi-static approach, instead, the equilibrium points of such equations are computed. In the springs-masses systems objects are modelled as a set of point like masses, linked together with springs and dampers. In the finite-elements based methods the entities are modeled as a finite set of elements interacting each other while preserving physical invariants such as masses, energy, volume and others. Differences between various implementations of these methods consist in the shape of elements and relations between them. Springs-masses systems are approached dynamically, since the corresponding equations are easily integrable. Unfortunately they may present stability problems, mainly due to the choice of visco-elastic parameters and of step used for time discretization. On the other side, finite elements systems are approached in a quasi-static setting, due to the computational complexity of the resulting differential equations. Luckily enough finite elements method can be partially precomputed or used for off-line computation of elementary deformations which can than be used at runtime [10].

Another issue in visio-haptic rendering is the mathematical representation of deformable objects surfaces. Meshes and implicit surfaces are the main methods. Mesh-based representations present the great advantage of one-to-one correspondence with the elements of the physical model, thus the triangular mesh needed by 3D graphical adapters is easily built from elements data. On the other hand, thanks to collision detection and haptic feedback easy computation, implicit surfaces

[1]We use the convention of calling the code executable by the GPU Vertex Shaders and Pixel Shaders. It's quite diffused also another nomenclature calling them Vertex Programs and Fragment Programs (as in OpenGL).

0-7803-8578-0/04/$20.00 ©2004 IEEE

are one of the preferred methods when dealing with haptic rendering only. Unfortunately recomputing the triangular mesh for visualization from an implicit surface can be a heavy task.

III. Method Fundamentals

When talking of "pointlike local deformations" we refer to deformations of the external surface of an object which came in contact with pointlike haptic-driven probes. The width of deformation and the depth of penetration are supposed "small" with respect to the dimension and curvature of the undeformed object. Think, for example, at the deformation produced by pushing with a finger a baby's belly. The method presented here is well suited for small and medium dimension deformations of objects with a smooth surface (such as human organs) in which very specific tactile behaviours are required (total or partial volume conservations, compliance, frictional reactions, heterogeneous and time-dependant stiffness). In this section the core of this method will be described in enough detail to be implemented.

A. Modeling and Simulating

A deformable object is modeled, starting form its triangular mesh representation, as a set of pointlike entities distributed along the surface in correspondence with vertices (thus from now on we'll simply refer to them as "vertex").

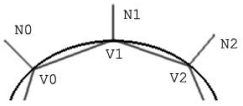

Fig. 1. Object modeling.

Each vertex has a dynamic model with 1 DoF and could move along the correspondent normal to the undeformed surface. The normal vector is usually present in the original mesh representation, as it is used for lighting computations, or can be easily computed. The dynamic system for each vertex should attract it to its initial undeformed position. Even a simple mass-spring-damper triplet suits well. Other non-linear or higher-order systems can add further realism. Equation (1) describes, as an example, the simple mass-spring-damper case.

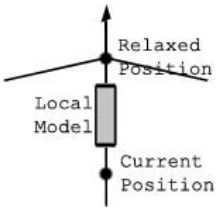

Fig. 2. Local model.

$$m\ddot{h} = -\beta\dot{h} - kh + f \tag{1}$$

Where h represents the distance of the current vertex position from the undeformed position along its normal, and the f term refers to the force acting on the vertex. Every vertex represents a standalone system that can be dynamically simulated integrating the motion equation.

Because each vertex has no interaction with other vertices, simulation stability can be easily guaranteed. The one-to-one correspondence with their graphical representation and the independence from each other allows for vertices integration to be performed on the GPU during rendering phase.

B. Deformations

Deformations are achieved as shaped displacement of vertices heights along the normals to surface, through the use of *force fields*. When a collision is detected the depth of the penetration and the material property of the underlying vertex are used to compute a force field which is propagated through the neighbourhood. This could be performed in two ways:

- On the CPU, by keeping at each vertex topological info about neighbours and performing a BFS visit
- On the GPU, comparing info such as position and normal of each vertex with the center of deformation, in order to recognize involved ones

Using a planar/spatial simplification, the intensity of the force field applied to each involved vertex is function of the distance from the center of the deformation (i.e. the contact point on the surface). We call these functions *shape-functions*. In fact each vertex subject to external forces moves away from its default position toward a well defined equilibrium position. Shape functions represent the shape that the surface will assume at equilibrium in the neighbourhood of the contact point due to deformation. Two deformation related issues that typically cannot be resolved locally are volume conservation and normal vector computation. The former requires global notions about the whole object, the latter information from the neighbours. Using shape-functions these issues can be locally resolved with good results. Approximated volume conservation can be obtained using shape-functions that push inside vertices near the contact point and pull away those farther. Exact volume conservation can be easily computed on a flat surface. Errors increas with the surface curvature, but are small enough not to be visually perceived. When the surface of the object is deformed, it becomes necessary to recompute the normal vector associated with each vertex[2]. The normal is in fact essential for lighting computations. Brute force recomputing of normals requires many computations such as updating neighbour's positions, finding face normals and blending them together. Good approximations can be obtained using a normal correction vector computed from the shape functions. Using derivative of such functions we compute a small vector that added to original vertex normal yields to the new normal. More details on normal correction can be found in [15]. Effects from multiple interactions can be superimposed. As for volume conservation, normals errors are too small to be perceived.

[2]We assume to deal with smooth object, with a single normal for vertex.

Normal correction is faster than computing the new normals geometrically and allows for execution on the GPU, where each vertex is processed in parallel and separately from each others.

C. Collision Detection

Collision detection with object surface is performed in two steps: contact point interpolation and height check. In the former an interpolated point on the surface is found. This point has the characteristic that its interpolated normal goes through the probe position. This can be done with good results using collision detection algorithms on the undeformed surface to find the closest triangle. Note that the probe projection in general does not coincide with an actual vertex. Thus interpolation between the three vertices of the triangle is used to compute a virtual vertex on which collision detection (and haptic rendering) is performed. The monodimensional height check test is performed by projecting probe position on the interpolated vertex normal and checking against current height.

D. Graphical Rendering

The same triangular mesh used to describe the object surface is also used for graphical rendering. Only small corrections, to take in account changes of positions and normals, due to deformation, must be performed. At each cycle the deformed position of each mesh vertex is given by the original position plus the displacement along its normal.

$$\vec{P}_f = \vec{P}_0 + h\vec{N}_0 \tag{2}$$

Where P_f is the final vertex position, P_0 is the undeformed position, h is the current height displacement (i.e. the position of the local dynamic system of that vertex), and N_0 is the undeformed normal. Similarly the actual normal, needed for lighting computations, is obtained by adding to the original normal the normal-correction vectors from deformations perturbing that vertex.

$$\vec{N}_f = \vec{N}_0 + \sum_{i \in D} \vec{N}_i \tag{3}$$

Where N_f is the final normal, and N_i is the normal correction vector produced by each deformation. These computations are performed in parallel on the GPU with only few native instructions and almost with no additional cost.

E. Haptic Rendering

Haptic rendering is easily performed on the interpolated local-model. The projection along the normal is used as input to the local model update function to compute the module of the normal reaction force. This value is used, together with the velocity vector of the probe, to compute the frictional force. Finally normal reaction vector is computed using the actual normal. The sum of the two reactions is sent back to the haptic device

$$\begin{aligned} F_n &= localmodel(P_x, \dot{P}_x) \\ \vec{T} &= \left(\dot{P}_x - \frac{\left\langle \dot{P}_x, \vec{N}_f \right\rangle}{\left\| \dot{P}_x \right\|} \vec{N}_f \right) \\ \vec{F}_n &= F_n \cdot \vec{N}_f \\ \vec{F}_t &= F_n \cdot K \cdot \vec{T} / \left\| \vec{T} \right\| \\ \vec{F}_f &= \vec{F}_n + \vec{F}_t \end{aligned} \tag{4}$$

Where F_n is the normal component of force feedback, F_t the tangential component and F_f the force feedback. P_x is the probe position, T is the tangential component of probe velocity, and K is the friction coefficient.

IV. GPU Related

As previously stated graphical rendering of deformed objects requires new positions and normals to be computed. In the naive solution the new values are built on the CPU and then the whole new mesh is sent to video board. Using simple height displacements we can keep undeformed vertex positions and normals in a static buffer on the video memory and can move only a lighter four-values stream at each cycle. Static video memory buffers and lower bandwidth usage are two key aspects for achieving optimal video performances. In more detail, for each vertex a vector (n_x,n_y,n_z,h), describing normal correction vector and height displacement, is enough to compute the new triangular mesh configuration. Even the "displacement/correction" stream is largely static, and only a small number of parameters must be updated. Using triangular meshes with high locality of vertices[3] and a simple zone subdivisions scheme, required data bandwidth can be further decreased. Moreover vertex locality helps GPU vertex-cache hits and thus increase throughput. The following two simple lines can be added to almost any Cg/HLSL shader to support deformations.

```
currentPosition = defaultPosition +
                  heightDisplacement * defaultNormal;
currentNormal   = defaultNormal + normalCorrection;
```

As previously sketched out, other computational parts of the proposed method can be performed on the GPU. Dynamics integration can be easily computed in parallel with high precision on 2nd generation programmable videoboards supporting 128bit floating-point color format of rendering targets, and even with single passes on 3rd generation adapters with support for multiple rendering targets. On these last generation videoboards, thanks to the infinite length allowed for shader code, even shape functions could be implemented

[3] in which vertices belonging to the same triangle are typically very near in the data stream. This properties implies a certain locality of vertices interested by a deformation.

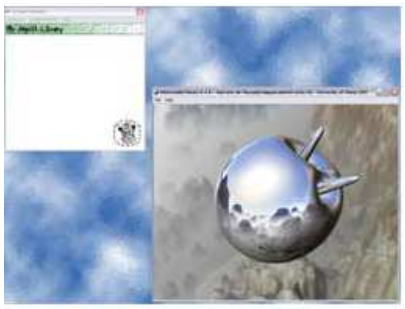

Fig. 3. The deformable demo.

Fig. 4. The deformable demo with three probes.

directly as code. Also collision detection can take greatly benefits by using the brute force power of any programmable graphical adapter. Details of such computations are available in the literature [11] and are beyond the scope of this paper. Work is in progress to completely exploit all the usable features of GPUs.

V. Deformable Demo

The Deformable Demo has been written in C++ on Win32 API for WindowsXP. Low level access to haptic devices has been obtained through the Haptik Library [12]. For the graphical related aspects it has been used DirectX 9.0 [13] to access the graphical hardware and the Cg runtime of nVidia [14] to manage the graphical shaders/programs. Deformable demo supports realtime deformation of loadable tri-mesh objects with multiple probes, while rendering with some advanced graphical features such environment cube mapping, reflection and refraction, bump mapping, self-shadowing. Some screenshot of the deformable demo are reported in (Fig 3/4).

A. Implementation Details

As previously stated collision detection is performed monodimensionally by projecting probe position on the interpolated contact vertex normal. In the actual demo implementation core of collision detection is thus performed using a modified AABB tree, in which the boxes contain vertices instead of triangles. This structure is used to perform nearest vertex detection. Dealing with vertex trees instead of triangle trees yield to two useful properties:

- child boxes are strictly contained in father box
- only connected boxes are overlapping (the tree behaves as an oriented graph)

The first property makes the volume decreasing quickly when descending the tree. The second one allows a local search strategy when performing successive search. Instead of restarting from the root, the search continues from the last matching leaf moving up till a full containing box is found. At this point search can continue as usual, surely finding the nearest vertex. When moving slowly on the surface, this assures on average a faster search then complete traversal especially when dealing with highly complex meshes. Nearest vertex search is performed linearly moving through neighbourhoods when probe is in contact with surface.

Conclusions and Future Work

Current demonstrative implementation of the model runs with very realistic visio haptic rendering on common PCs, even when GPU is used only for graphics and not for model-related computations. Deformable Demo could be downloaded at http://sirslab.dii.unisi.it/haptic/deformable.htm

Work is in progress to optimize the software implementation. Moreover enhancements are currently under investigation to add forms of global deformations, non-pointlike probes, and to deal with surface singularity using normals drift.

References

[1] SensAble Technologies, *The PHANToM Device*, www.sensable.com.
[2] ForceDimension, *The Delta Device*, www.forcedimension.com
[3] nVidia Corporation, *The GeForce Family GPUs*, www.nvidia.com
[4] ATI Technologies, *The Radeon Family VPUs*, www.ati.com
[5] nVidia Corporation, *GDC2003 Presentation*, http://developer.nvidia.com/object/presentations.html
[6] J. N. Reddy, *An Introduction to the Finite Element Method,* McGraw Hill, 2nd edition, 1993.
[7] F. Hartmann, *Introduction to Boundary Elements: Theory and Applications*, Springer-Verlag, Berlin, 1989.
[8] R. Balaniuk, I.F. Costa, LEM "An approach for physically based soft tissue simulation suitable for haptic interaction," in Proc. of the 5th PHANTOM Users Group Workshop - PUG00, Aspen, USA, October 2000.
[9] R. Balaniuk and K. Salisbury, "Soft-Tissue Simulation Using the Radial Elements Method," in Proc. of Surgery Simulation and Soft Tissue Modeling (IS4TM), 2003.
[10] D. James and D. K. Pai, "A Unified Treatment of Elastostatic and Rigid Contact Simulation for Real Time Haptics," in *Haptics-e, the Electronic Journal of Haptics Research,* Vol. 2, No. 1, September 2001.
[11] *General-Purpose Computation Using Graphics Hardware,* www.gpgpu.com
[12] M. de Pascale, G. de Pascale, D. Prattichizzo, F. Barbagli, "The Haptik Library: a Component based Architecture for Haptic Devices Access," in Proc. Eurohaptics 2004, Munich, June 2004.
[13] Microsoft Corporation, *Direct X*, http://msdn.microsoft.com
[14] nVidia Corporation, *The Cg Language*, http://developer.nvidia.com
[15] M. de Pacale, G. de Pascale, and D. Prattichizzo "Haptic interaction design based on GPUs" Internal Report N. 717-03, Siena Robotics and System Lab, University of Siena, 2003.