Master's Thesis Report

Stanford-NASA
National Biocomputation Center
Computer Science Department
Stanford University, CA 94305

The Forcegrid: A Buffer Structure for Haptic Interaction with Virtual Elastic Objects

Frederic Mazzella
Supervisors :
Kevin Montgomery & Jean-Claude Latombe

September 15, 2001

Abstract

Today there is considerable interest in surgical simulators for training, operation planning, and outcome prediction. Surgical simulation often involves real-time computation of the deformations of visco-elastic human-body tissues and providing users with both graphic and haptic rendering. As humans are more sensitive to small haptic discrepancies than visual ones, good force rendering requires a refresh rate of 500Hz or more, which is far beyond the capabilities of current simulation techniques. If the simulator runs on a remote server, limited bandwidth, network latency and variance often compromise satisfactory haptic rendering. To address these problems, a new interpolation/extrapolation data structure - the *Forcegrid* - is proposed, which makes it possible to approximate forces at rates greater than 500Hz, regardless of the complexity of the model of the simulated object. The forcegrid is designed for haptic interaction during continuous contact. It is a regular grid defined over the virtual workspace, in which every vertex contains a force calculated from the reaction forces computed at previous simulation cycles. The controller of the haptic device exploits the forcegrid to render forces at the appropriate rate. The two key algorithms associated with the forcegrid are an extrapolation algorithm that updates the forces contained in the grid vertices at every simulation cycle ($\approx$ 30Hz) and an interpolation algorithm that computes the force to be rendered on the haptic device ($\approx$ 500Hz). The forcegrid has been implemented and successfully experimented with on numerous virtual deformable objects.

Contents

Introduction

Over the past few years there has been considerable research in modeling deformable objects. See [1] and [2] for reviews of human tissue modeling and cloth simulation, respectively. An important domain of application is surgical simulation, which can be used for training, planning operations, and predicting surgical outcomes. In many cases, it is necessary or highly desirable to perform the computer simulation in real-time.

Most human-body soft tissues are visco-elastic. Simulating their deformations consists of computing their successive shapes over time. The key feedback is subjective realism. In surgical simulation, this often means that graphic animation alone is not enough and it is critical to also provide force feedback on a haptic device (a mechanical linkage with actuated joints). This device is used to position a model of a surgical instrument in a virtual environment containing objects and to render the reaction forces resulting from contacts between an object and the instrument. Several experiments have shown that the combination of visual and haptic feedbacks provides users with a more realistic experience than graphic feedback alone. [3] [4] [5].

Real-time graphic animation requires the shape of the objects to be updated at 30 Hz or more. This is already a challenging problem. A number of computational and modeling techniques - e.g. mass-spring meshes [17] [6] [7] and finite-element methods [8] [9] - have recently been proposed that provide reasonably satisfactory solutions for elastic objects. A typical technique computes internal forces throughout the model of an object E and derives the deformation of E from these forces. A natural by-product of this computation is the reaction force exerted by E on the surgical instrument. However, as humans are more sensitive to small haptic discrepancies than to visual ones, good force rendering requires a refresh rate of 500Hz, or more, which is far beyond the capabilities of current simulators. In the near future, the expected increase in computational power is more likely to serve more accurate and visually pleasing graphic animations than to feed haptic devices at 500Hz. Moreover, a fair prospect in many surgical applications is that the simulator will run on a remote computer, connected to the controller of the haptic device by a LAN or the Internet and even small network latency, delays, or inconsistencies can compromise satisfactory haptic interaction.

To address this difficulty, we propose a new interpolation/extrapolation data structure - the *forcegrid* - which makes it possible to approximate the reaction force to be rendered on the haptic device at rates greater than

500Hz, regardless of the complexity of the model of the simulated object. The forcegrid is a structure in which we cache previous results generated by the simulator and that the haptic device controller exploits to render forces at the appropriate rate. It is a regular grid placed over the virtual workspace, in which every vertex contains a force. The two key algorithms are the force-filling algorithm, which extrapolates the results returned by the simulator to update the vertex forces, and the interpolation algorithm, which computes the force to be rendered. The forcegrid is independent of both the modeling and computational techniques used by the simulator and the simulation rate. It is also independent of the specific values of the mechanical properties of the elastic object (e.g. distribution of mass, damping and stiffness).

Chapter 1 reviews prior work and concepts used in our research and experiments. Chapter 2 describes the forcegrid and its associated methods. Chapter 3 describes our implementation and our experiments with the forcegrid. Finally, Chapter 4 discusses limitations of the current techniques and proposes remedies to eliminate them.

Chapter 1

Project presentation

1.1 Mass-spring meshes deformation

The algorithms developed for real time (30 Hz+ refresh frequency) tissue deformation can be classified in three groups : based in *mass-spring systems* (Delingette's method [11]), based in *deformable splines* (Terzopoulos' method [12]) and based in *finite elements methods (FEM)* [13] [14]. [10].

We develop the mass-spring system method, in which soft tissues are represented by meshes of nodes, edges and faces. Each node has a position, a velocity, an applied force and a mass. Each edge links two nodes and can be considered as a spring with a spring constant that is either fixed or function of the elongation of the edge (non linear approach). Edge-elongation based forces are applied throughout the entire mesh. Each face is triangular and is just used to graphically display the outer surface of the soft tissue. A face does not possess any physical property (see Figure 1.1).

However, using the same mesh structure, we also experiment with new simulation methods that apply edge-configuration based forces instead of edge-elongation based forces.

All of our simulation methods use step by step computation of dynamics equations with high order approximations when equation integration is needed.

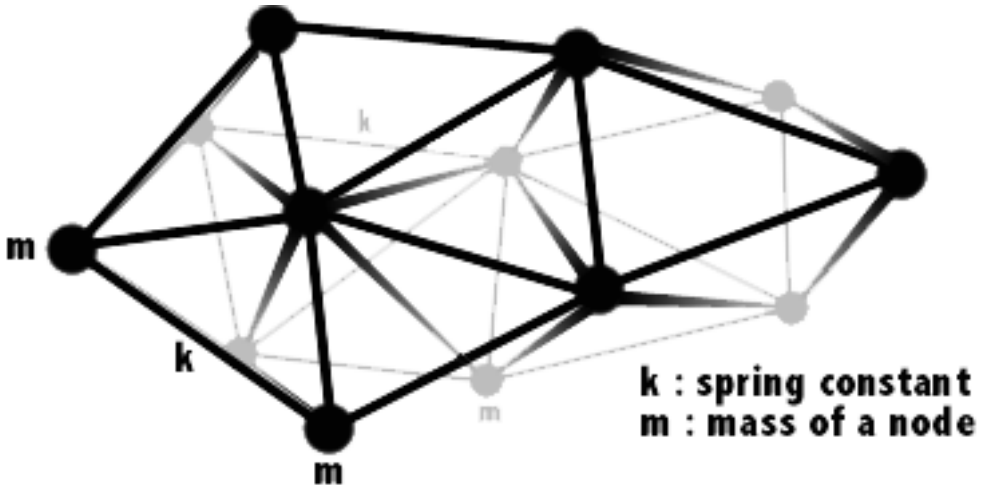

Figure 1.1: *3D shape of a soft tissue represented by a mass-spring system.*

1.2 Force feedback issues

The simplest data exchange protocol between the simulator and the haptic device is described in figure 1.2. The haptic device sends the position of the tip of the tool through the internet (through a controller), and the simulator sends back the force vector to be rendered for that position. This protocol is very efficient for low latency networks and simple soft tissues for which the computation of the forces doesn't take too much time (see figure 1.2).

A number of different techniques have been proposed to improve the quality of haptic feedback during the interaction with an elastic object.

The technique proposed in [15] computes a reaction force that is proportional to the penetration depth of the surgical instrument (relative to the surface of the deformable object at rest). The rendered forces are thus only loosely related to the physical model of the object.

In [16] the haptic simulator achieves greater speed by using a much smaller mesh (to model the deformable object) than the one used by the graphic simulator. One difficult issue is to make sure that the two different meshes provide consistent results.

A "buffer model" is proposed in [22], which is defined as a local physical model of the deformable object. The haptic device interacts with this simpli-

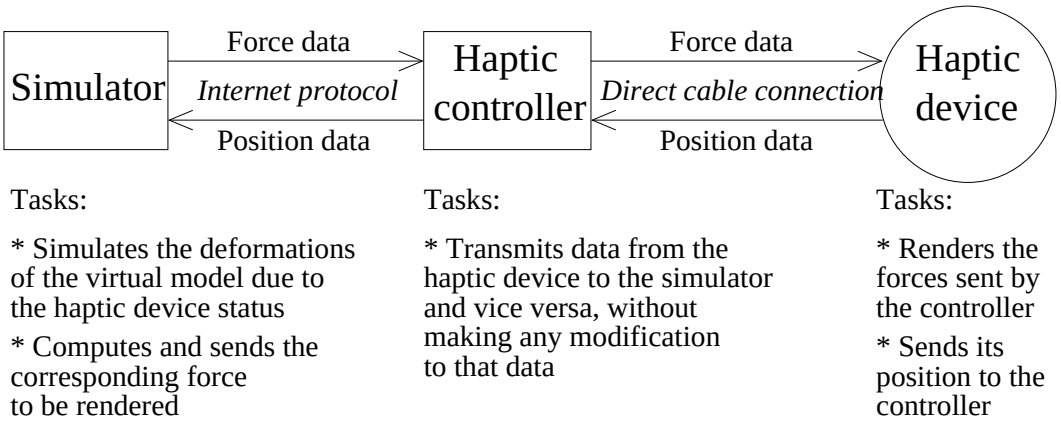

Figure 1.2: *The simplest data exchange protocol used to render forces computed by the simulator.*

fied model, which is a numerical function, whose parameters are periodically adjusted to optimize the fit with the data produced by the simulator. Our forcegrid bears some similarities with this buffer model. However, the forcegrid is not a local model, but a global data-caching structure. While every parameter adjustment in the buffer model of [22] leads to forgetting past results, our forcegrid exploits all previous simulation results. We believe that it is also easier to use, as it does not require selecting a numerical function adapted to the mechanical properties of the object. On the other hand, the buffer model of [22] may give better results during a cutting operation or if the mechanical properties of the object have hysteresis (see Chapter 4).

In [19] fast online computations are achieved by pre-computing forces for a number of deformations. This is an appealing approach, but it suffers from two drawbacks. First, the contact point of the instrument with the object is often not known in advance. Second, forces are computed as linear combinations of relatively sparse pre-computed forces, while human-body tissues are often non-linear.

In [21] extrapolation techniques over time and position are used to estimate reaction forces between two time steps of the graphic simulation. The forcegrid also uses some form of extrapolation technique to fill the forcegrid, but it makes better use of the data flow produced by the simulator. By caching the results successively produced by the simulator, it takes advantage of several of them to compute a new reaction force.

1.3 Improving the update rate during continuous contact

There are different types of interactions between a surgical tool and an object. A surgeon can poke, grab, cut or sutter an object. We focused on the improvement of the haptic simulation of grabbing interations.

In virtual simulation, a grabbing interaction can be decomposed in two phases. First, the user grabbs a part of the virtual model. Then, holding on to this part, he/she moves the virtual tool around. This causes the virtual model to deform and the user to feel corresponding reaction forces. The forcegrid method makes it possible to overcome the low update rate in that second phase, during which we consider that the tool and the grabbed part are in continuous contact. The forcegrid is elaborated with the forces it gets from the simulator, at 30Hz. Then, using the forcegrid, the haptic controller is able to interpolate a new force vector for any position of the sensor, at the haptic rate (500Hz).

For this method to work, the deformable objects must present no hysteresis effect, so that the values of the forces used to build the forcegrid remain valid for future interpolations.

As soon as the user releases the grabbed part and grabbs another part of the object, a brand new forcegrid has to be built from scratch. Indeed, the force field in the haptic world is completely changed by such an action.

Chapter 2

The Forcegrid

2.1 The Concept of a Forcegrid

We consider a haptic device with three degrees of freedom, which produces a desired 3-D force at a point, called the *operational point*, selected in the last link of the haptic device. We map the operational point to a point P in a virtual workspace W. By extension, we also call P the operational point. For simplification, but with no loss of generality, we see W as a rectangular parallelepiped. If this is not the case, we enclose the actual workspace by a parallelepiped W and some regions of W are simply not reachable by P. Each axis of the workspace coordinate system is set parallel to edges of W. Figure 2.1 illustrates.

W contains a virtual elastic object E. A mechanical model of E is given to a simulator that can compute the object's deformations when external forces are applied to points in its surface. The simulator tends to bring the object to equilibrium at each computation step, according to the deformations that the object undergoes. Consider the situation where P is in contact with the surface of E. We consider this contact to be sticky, so that any further motion of P yields a deformation of E. The simulator computes this deformation and renders it on a graphic display in real-time. Our goal is to also render on the haptic device the reaction force exerted by E on P. In practice, we attach a real surgical instrument to the tip of the haptic device. The user moves this instrument, which causes a model of the instrument to move accordingly in the virtual workspace. An efficient collision-detection technique (see Chapter 3) detects the occurrence of a contact between the

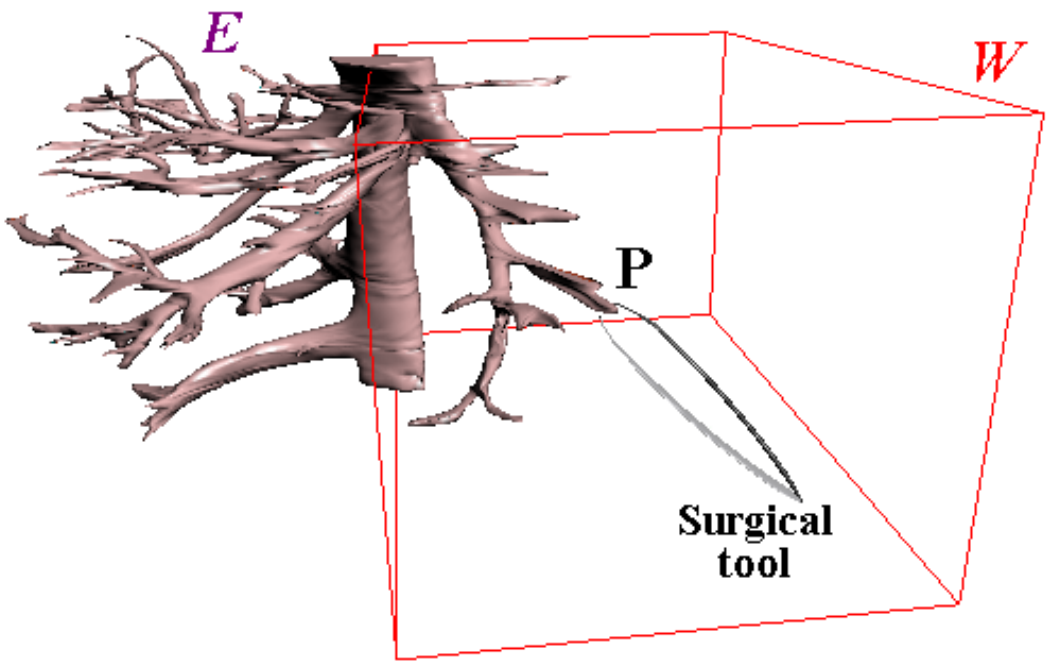

Figure 2.1: *Virtual deformable vena cava in workspace W. W is the space that the tip of the surgical tool can reach.*

simulated instrument and E. The point of contact on the instrument is then chosen to be the operational point P.

We assume that the simulator computes the deformation of E at roughly 30Hz, and provides the value of the reaction force $\vec{F}$ exerted by E at P. The forcegrid allows an approximation of the reaction force to be rendered on the operational point at rates on the order of 500Hz, or greater, independent of the complexity of the model of E. It is essentially a data-caching structure which memorizes previous force computation results. The haptic device controller exploits this data structure concurrently to render forces at the desired rate. More precisely, the forcegrid is defined as a regular grid G of vertices placed over W, in which every vertex V contains a force $\vec{F_V}$. See Figure 2.2.

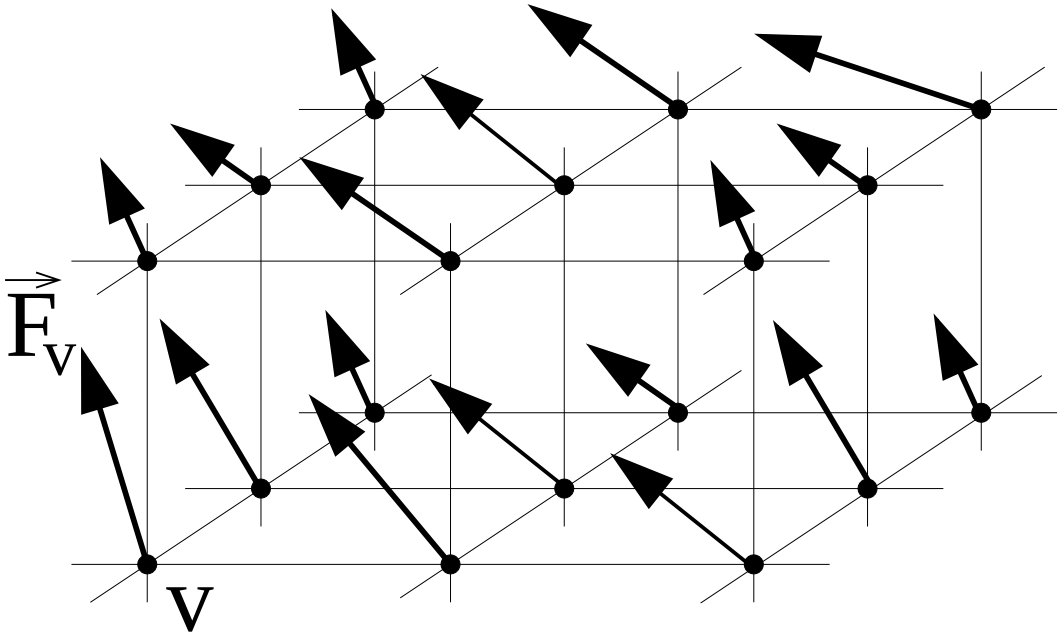

Figure 2.2: *A portion of a forcegrid.*

In the ideal case, $\vec{F_V}$ is the reaction force that would be exerted by E at P if P were located at V. It is not practical to fill G with forces prior to interacting with E. Indeed, the contact point is not known in advance, and filling G when the contact occurs introduces delay. Therefore, we propose in Sections 2.2 and 2.3 a method that computes and updates approximations

of the ideal forces while the haptic device is interacting with E. Whenever the controller of the haptic device needs to render the reaction force at P ($\approx$ 500Hz), it localizes P in the grid and interpolates among the forces labeling neighboring vertices. We now describe the interpolation function.

2.2 Interpolation in a forcegrid

Let $W = [0, X]$x$[0, Y]$x$[0, Z]$ and (p_x+1)x(p_y+1)x(p_z+1) be the size (number of vertices) of G. We define: $\delta_x = \frac{X}{p_x}$, $\delta_y = \frac{Y}{p_y}$, $\delta_z = \frac{Z}{p_z}$. Each cell of G is a parallelepiped whose edges have lengths δ_x, δ_y and δ_z.

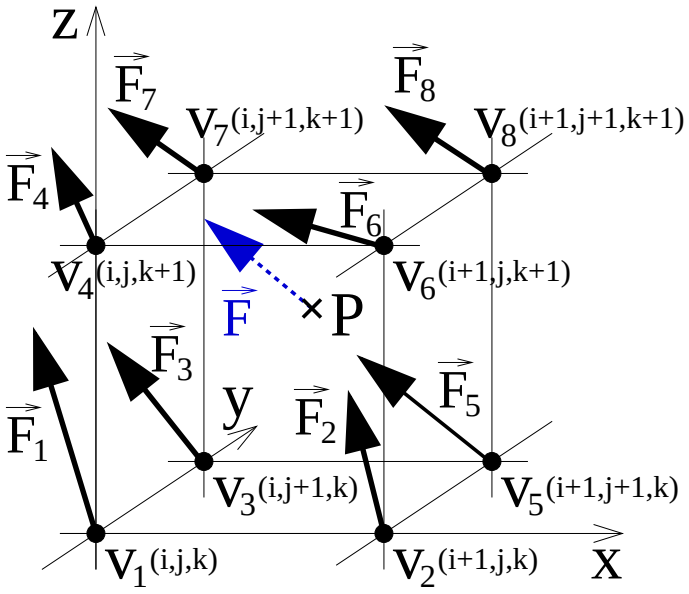

Figure 2.3: *A cell of the forcegrid and its vertex forces.*

The real coordinates of a vertex V of G are of the form $(i.\delta_x, j.\delta_y, k.\delta_z)$, where $i = 0, 1, ..., p_x$, $j = 0, 1, ..., p_y$, and $k = 0, 1, ..., p_z$. We identify each vertex V by its integer coordinates (i, j, k). Similarly, we identify each cell c of G by its vertex (i, j, k) with the smallest indices ($i < p_x - 1$, $j < p_y - 1$, and $z < p_z - 1$). Hence, the 8 vertices of c are obtained by forming all combinations of indices in $\{i, i + 1\}$, $\{j, j + 1\}$, and $\{k, k + 1\}$ (See Figure 2.3). Given the position (x, y, z) of the operational point P in W, the cell c that

contains P is obtained by computing:

$i = \lfloor \frac{x}{\delta_x} \rfloor$, $j = \lfloor \frac{y}{\delta_y} \rfloor$, and $k = \lfloor \frac{z}{\delta_z} \rfloor$

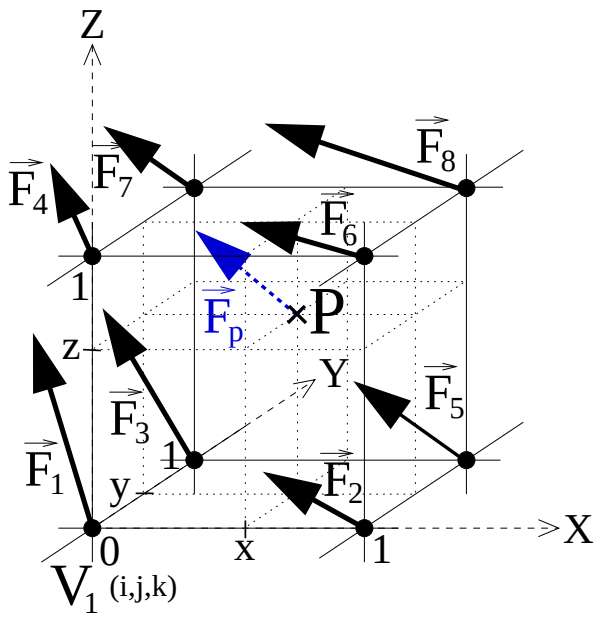

Figure 2.4: *The interpolated force is based on the local x, y and z coordinates.*

To simplify the calculations, we normalize c to be the parallelepiped $[0,1]$x$[0,1]$x$[0,1]$ and we redefine (x, y, z) to stand relative to a coordinate system whose origin is vertex (i, j, k) (see Figure 2.4). Hence, (x, y, z) ϵ $[0,1]$ x$[0,1]$x$[0,1]$. Let $\vec{F_1}, \vec{F_2}, ..., \vec{F_8}$ denote the forces in the 8 vertices of c, as depicted in Figure 2.4. We express the reaction force $\vec{F}$ at P as the following combination:

$$\vec{F_P} = \vec{a} + x.\vec{b} + y.\vec{c} + z.\vec{d} + x.y.\vec{e} + x.z.\vec{f} + y.z.\vec{g} + x.y.z.\vec{h}$$

where $\vec{a}$, $\vec{b}$, $\vec{c}$, $\vec{d}$, $\vec{e}$, $\vec{f}$, $\vec{g}$ and $\vec{h}$ are forces computed from the forces at the 8 vertices:

At $(0,0,0)$, $\vec{F} = \vec{F_1} = \vec{a}$

$$
\begin{aligned}
&\text{At } (1,0,0), \quad \vec{F} = \vec{F_2} = \vec{a} + \vec{b} \\
&\text{At } (0,1,0), \quad \vec{F} = \vec{F_3} = \vec{a} + \vec{c} \\
&\text{At } (0,0,1), \quad \vec{F} = \vec{F_4} = \vec{a} + \vec{d} \\
&\text{At } (1,1,0), \quad \vec{F} = \vec{F_5} = \vec{a} + \vec{b} + \vec{c} + \vec{e} \\
&\text{At } (1,0,1), \quad \vec{F} = \vec{F_6} = \vec{a} + \vec{b} + \vec{d} + \vec{f} \\
&\text{At } (0,1,1), \quad \vec{F} = \vec{F_7} = \vec{a} + \vec{c} + \vec{d} + \vec{g} \\
&\text{At } (1,1,1), \quad \vec{F} = \vec{F_8} = \vec{a} + \vec{b} + \vec{c} + \vec{d} + \vec{e} + \vec{f} + \vec{g} + \vec{h}
\end{aligned}
$$

The above 8 equations can be solved for $\vec{a}$, $\vec{b}, ...,$ $\vec{h}$, which yields the forcegrid interpolation function (see Appendix B):

$$
\begin{aligned}
\vec{F} = \vec{F_1} &+ x.(\vec{F_2} - \vec{F_1}) + y.(\vec{F_3} - \vec{F_1}) + z.(\vec{F_4} - \vec{F_1}) \\
&+ x.y.(\vec{F_5} - \vec{F_3} - \vec{F_2} + \vec{F_1}) \\
&+ x.z.(\vec{F_6} - \vec{F_4} - \vec{F_2} + \vec{F_1}) \\
&+ y.z.(\vec{F_7} - \vec{F_4} - \vec{F_3} + \vec{F_1}) \\
&+ x.y.z.(\vec{F_8} - \vec{F_7} - \vec{F_6} - \vec{F_5} + \vec{F_4} + \vec{F_3} + \vec{F_2} - \vec{F_1})
\end{aligned}
$$

Even when the forces $\vec{F_1}$ through $\vec{F_8}$ are exactly the reaction forces exerted by E when the operational point is respectively located at each of the 8 vertices of c, the above function is still a linear approximation of the actual value of $\vec{F}$ at other locations of P in c. However, in most situations (e.g. no new contact between E and other objects occurs), the actual variation of $\vec{F}$ when P moves is smooth, and the above approximation is satisfactory. For instance, in a mass-spring model, forces at the mesh vertices are smooth (differentiable) functions of the positions of the vertices. The accuracy can be improved by increasing the resolution of G. The forcegrid interpolation function defines a continuous force field over W. Indeed, one can verify that, if P is located in a face bounding two cells or in an edge shared by four cells, then the above formula gives the same value of $\vec{F}$, independent of the cell used. Although the first derivative (gradient) of $\vec{F}$ is not continuous at the cell boundaries, we were unable to feel these discontinuities on the haptic device. If needed, one could eliminate this discontinuity by using a higher-degree interpolation function.

2.3 Filling a Forcegrid

In this section we propose two techniques to fill the vertices of a forcegrid G with forces. Both techniques work online, while the haptic device is interacting with an object E. The first technique - called the *silent method* - is the simplest, and only uses the data computed (at $\approx$ 30Hz) by the graphic simulator. The second technique - called the *request method* - requires that a second copy of the simulator be available on a separate processor. Only the silent method has been implemented and tested so far.

2.3.1 Silent method

One can see the silent method as one making the best possible use of the limited flow of information provided by the graphic simulator. The simulator computes at 30Hz the new shape of E along with the reaction force $\vec{F}$ at the current location of P. At any one time during haptic interaction, each vertex V of G contains a force $\vec{F_V}$ with a certain weight W_V. Initially (i.e., when P makes contact with E), every vertex of G contains the null force with weight 0. Next, every new force $\vec{F}$ computed by the simulator is used to update the forces and weights stored in G. We use a "candle light" technique in which the current location of P is seen as a punctual source of light illuminating the forcegrid with an intensity that decreases with distance. In other words, the influence of $\vec{F}$ is stronger on the vertices of G that are close to P than on those that are further away. More precisely, let V be a vertex of G at Euclidean distance d of P. The force and weight contained in V are updated according to the forcegrid update formulae:

$$\vec{F_V} \leftarrow \frac{W_V.\vec{F_V}+w(d).\vec{F}}{W_V+w(d)}$$

$$W_V \leftarrow W_V + w(d),$$

where $w(d)$ is a decreasing function that goes from 1 to 0 when d grows from 0 to a given constant r, called the *distance of influence* of a reaction force. There are two motivations for bounding the influence of a reaction force: first, the value of $\vec{F}$ is likely to become irrelevant at positions that are too far from the current P; second, if G is dense, setting r small enough significantly reduces the cost of updating the forces in G.Many different weighting functions could be used. The one that we use in our implementation is:

if $d < r$, $w(d) = 1/(1 + d^2 - \frac{d}{r(1+r^2)})$

if $d > r$, $w(d) = 0$

Our experiments showed that r should be set to approximately $2-3$ times the distance that the operational point may traverse between two iterations of the graphic simulator and that the resolution of G should be such that the distances between two adjacent vertices are on the order of $r/10$ to $r/5$.

When r is big, each new reaction force sent by the simulator causes a lot of forces and weights to be updated in the forcegrid. In order to keep a high haptic rate, it is then possible to distribute the updates over several consecutive haptic cycles, as described in Section 3.3.

Note that if P undergoes small motions relative to the resolution of G during some extended period of time, the graphic simulator will compute reaction forces at many points forming a dense cluster in W. Despite the weighting mechanism described above, these forces could have, altogether, an abnormally high impact on the vertex forces within radius r of the cluster's center. Therefore, we set a threshold s on the number of reaction forces computed in a certain cell that can be used to update the forcegrid.

2.3.2 Request method

The request method explicitly asks the simulator to compute forces at specific locations of the operational point, called *query positions*. In most implementations, since the processor running the graphic simulator is already busy, a separate processor running a copy of the simulator (without graphic visualization) is needed to process the queries of the request method.

For each query position, the reaction force $\vec{F}$ must be computed. $\vec{F}$ is then used to update the vertex and their weights in the forcegrid, using the forcegrid update formulae as described for the silent method. The request method should select the successive query positions in order to best fill the forcegrid around the current P, and thus improve the accuracy of the forces rendered on the haptic device. One way of doing this is to consider all the cells whose centers are within some distance ρ of the current P. Typically, ρ should be chosen roughly equal to the distance of influence r. The request method selects the center of the cell that has received the fewest reaction forces so far as the next query position. If there are several such cells, the

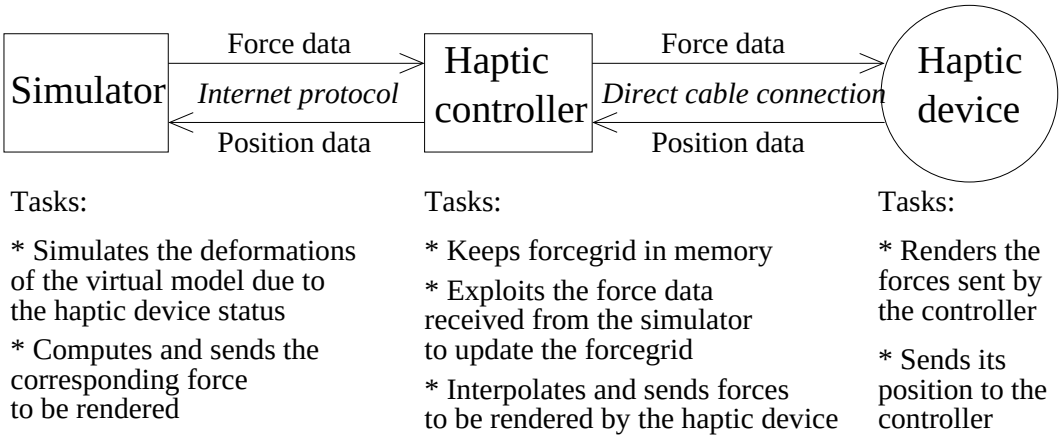

Figure 2.5: *The Silent method protocol: the haptic controller updates the forcegrid with the data it gets from the simulator, without making any special request. A new force can then be interpolated for any configuration of the haptic device, using the forcegrid.*

closest center to the current P is chosen. Obviously, several variant schemes would achieve the same purpose. If implemented, the request method would typically be run in complement to the silent method. Then its queries need not be answered at any particular rate, or even at a constant one.

2.3.3 Comparison of the two methods

The request method is more accurate

In the request method, the haptic controller receives forces from the simulator that are located at vertices of the forcegrid. This ensures that a force stored at the vertex V in the forcegrid was mostly influenced by a force that was computed by the simulator for this exact location. On the contrary, the silent method never gets forces computed especially for the vertices of the forcegrid. With the silen method, a force stored at a vertex V was built from force values computed for surrounding locations, but not especially for V. Thus, in addition to the fact that forces generated by the haptic controller are linear approximations of forces stored at the vertices of the forcegrid, the silent method builds a forcegrid of approximated forces.

This is a theoretical drawback though as in practice, we were unable to

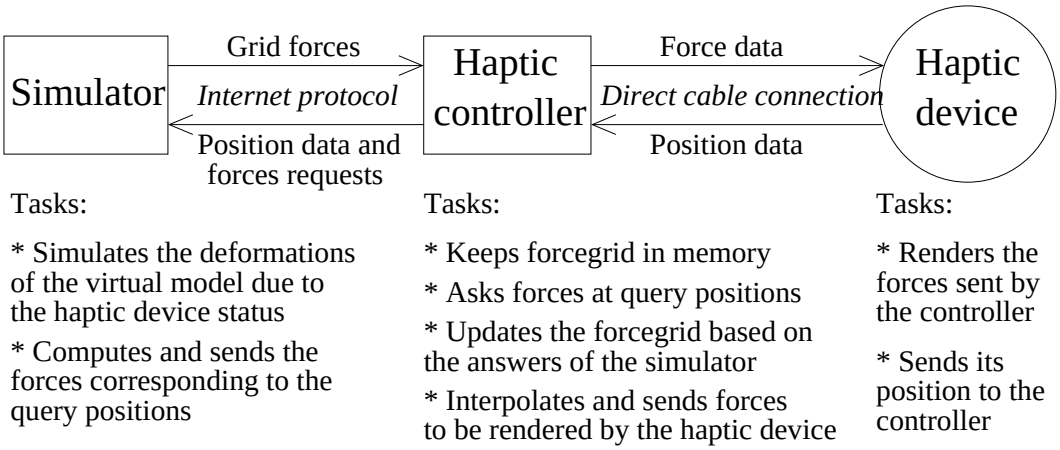

Figure 2.6: *The Request method protocol: the haptic controller asks the forces it needs at the query positions.*

feel a loss of accuracy with the silent method. Moreover, the comparison of off-line results with results obtained with the silent method do not show any inaccuracy (see Section 3.3).

The silent method is much simpler to implement

The silent method is transparent to the simulator. No requests are made and the haptic controller does not need anything else than forces corresponding to the tool's positions, which the simulator computes anyway. The request method is much more demanding for the simulator, as two models must run at the same time. The Forcegrid method was implemented to improve situations where the update rate of the simulator is low. If we ask one simulator to run the two models, the update rate will likely be divided by two, and the quality of the graphic display may become unacceptable. Thus, we must have two computers running two separate copies of the simulator, one for each model. This brings lots of implementation issues, as any modification performed on one model must also be performed on the other (elastic properties modifications, extrusion, cutting...). This requires an implementation in which all commands of the simulator can be sent through the internet from one computer to the other, and thus a modification of the simulator program in almost every part.

Chapter 3

Implementation and experiments

3.1 System

We have developed a software system integrating a forcegrid and conducted experiments. The current system only uses the silent method, as described in Section 2.3 and Figure 2.5.

The forcegrid software (implemented in C++) runs on the controller of the haptic device, whose processor is a 500MHz Intel Celeron. The compiled code is rather small (approximately 40Kbytes for the code, and 1Mbytes for the grid). In our system, we use the PHANTOM Desktop of SensAble Technologies. It is important that the haptic device has extremely low friction and inertia to provide satisfactory interaction with an elastic object. Not all available devices satisfy this requirement. The architecture used to drive the PHANTOM set the update rate to the constant rate of 500Hz. In order to keep such a high update rate, the forcegrid calculations should not take more than 2ms per haptic cycle to complete.

The graphic simulator runs on a Sun Microsystems Ultra 60 workstation, and uses the mass-spring technique described in [17] and [18], with either linear or non-linear springs. However, the forcegrid software is independent of the modeling and computational techniques used by the simulator.

The interaction protocol between the simulator and the forcegrid is Internet protocol UDP, which does not require the forcegrid to acknowledge received data, nor the simulator to wait until acknowledgement has been re-

ceived. The haptic device controller and the simulator can be connected by a simple serial line, a LAN, or through the Internet.

Initially, the forcegrid only contains null forces and the user moves the surgical instrument near deformable objects. A collision checker detects the occurrence of the contact between the instrument and any object [17] and returns the contact point. At this instant, the reaction force is null or very small. It takes 1/30 second for the simulator to return its first reaction force. During that short period of time the forces rendered on the haptic device is the null force. As long as the haptic device is moved at reasonable velocity, this small delay is imperceptible to the user, since the deformation of the object and the actual reaction forces are small anyway. Once the simulator has returned a force, the forcegrid is filled with non-zero forces.

We experimented with various resolutions of the forcegrid and several values of the distance of influence r. With the PHANTOM Desktop, a grid of size 50 x 50 x 50 with $r = 1/6$ of the size of W gave quasi-optimal results.

3.2 Examples

As usual, it is difficult to report on interactive haptic experiments in a paper. Since our mass-spring software allows us to easily describe new models of deformable objects, we have experimented with numerous models. Subjective haptic fidelity, as assessed by people internal and external to our group, was reported to be excellent. Forces were rendered smoothly and realistically, with no perceptible discontinuities or tremors. We experimented with a deformable model of a stomach. Figure 3.1(a) shows the stomach at rest, and figure 3.1(b) represents a deformation generated with our soft tissue deformation method. The underlying structure of that stomach is a mesh of $4,000$ vertices (point masses) and $11,000$ links (springs). Figure 3.2 shows the stomach in a surgical training environment with other human organs and surgical tools.

In one experiment, we input three different models having the same geometry at rest, but distinct mechanical parameters. In one model spring stiffness was uniformly set higher than in another model, while in the third model, stiffness varied greatly across the mass-spring mesh. Users were consistently able to correctly distinguish among these three models with only haptic interaction, and no visual feedback. In contrast, they were unable to do so with only visual feedback.

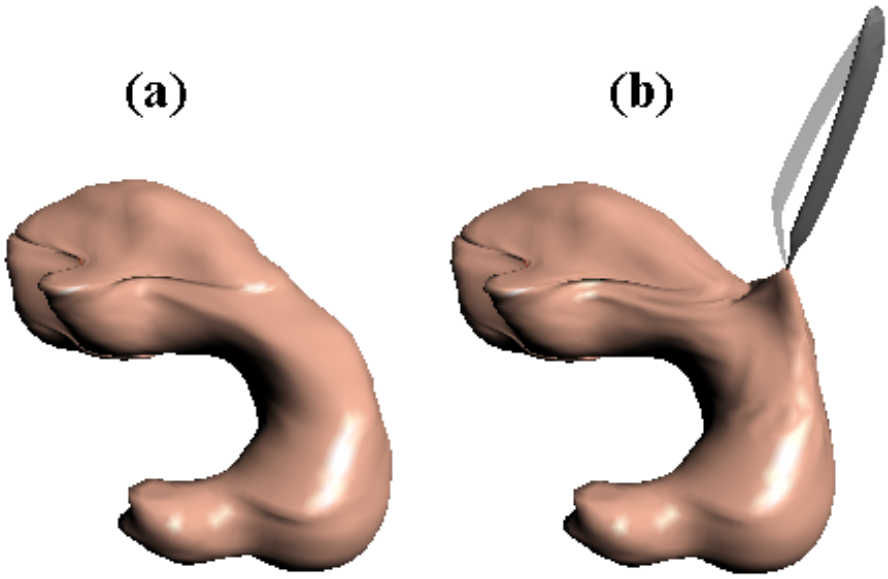

Figure 3.1: *Deformable stomach. (a): at rest. (b): deformed with our soft tissue deformation method.*

3.3 Quantitative evaluation

We measured the running times of the modules implementing forcegrid interpolation and filling. The forcegrid interpolation, including the localization of P in the grid and the evaluation of the interpolation function, takes $5\mu s$. The forcegrid filling takes time linear in the number of grid vertices within the influence distance r from P. Updating the force for one vertex takes $1\mu s$. In a 50 x 50 x 50 grid with r set to 1/6 of the size of the workspace, forces are updated for roughly $3,000$ vertices. Thus, a complete forcegrid filling operation takes $3,000 * 1\mu s = 3$ms. To keep a high haptic rate, the $3,000$ update operations were split over consecutive update cycles of the haptic device. One haptic cycle takes 2ms at the update rate of 500Hz. This allows $2,000$ vertices to be updated per haptic cycle ($2,000 * 1\mu s = 2$ms). In fact, with a simulation update rate of 30Hz, the simulator only sends a new reaction force to the controller every $33ms$. Therefore, splitting the update operations allows up to $33,000$ vertices to be updated per forcegrid filling operation, by groups of $2,000$. Even with dense forcegrids, updating $33,000$ vertices per simulation cycle is more than enough.

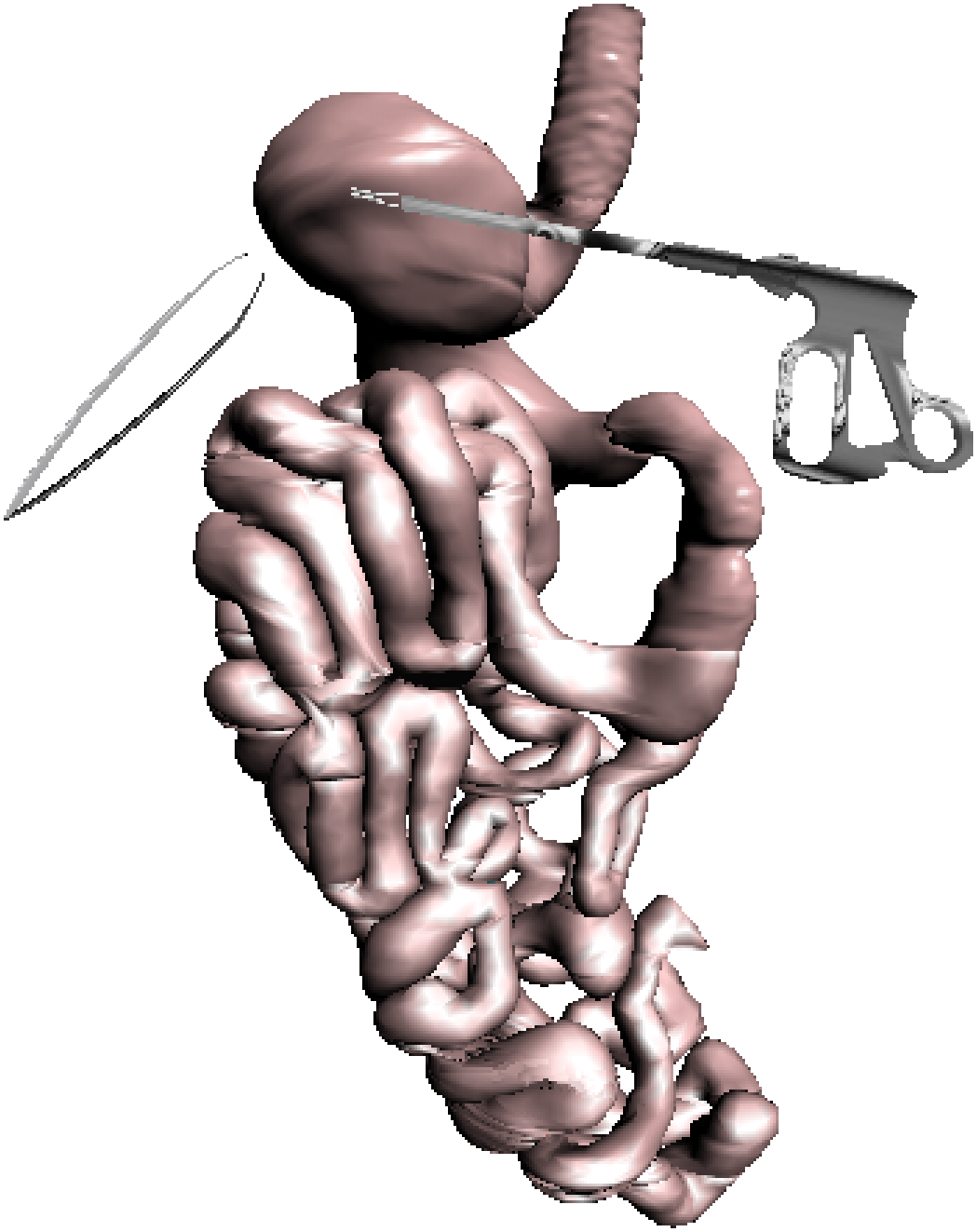

Figure 3.2: *The deformable stomach in a surgical training environment with other human organs and surgical tools.*

We also attempted to estimate the errors in the rendered forces by comparing the forces rendered through different techniques. We generated a sinusoidal movement of the instrument grabbing the surface of an object in a direction Δ orthogonal to the surface of the object, with a changing of speed. We recorded simultaneously the position of the tool along the axis Δ and the projection of the reaction forces along that axis. We imposed realistically heavy testing conditions: the object used was sophisticated ($10,000$ vertices and $35,000$ links), and the simulator was running at 30Hz.

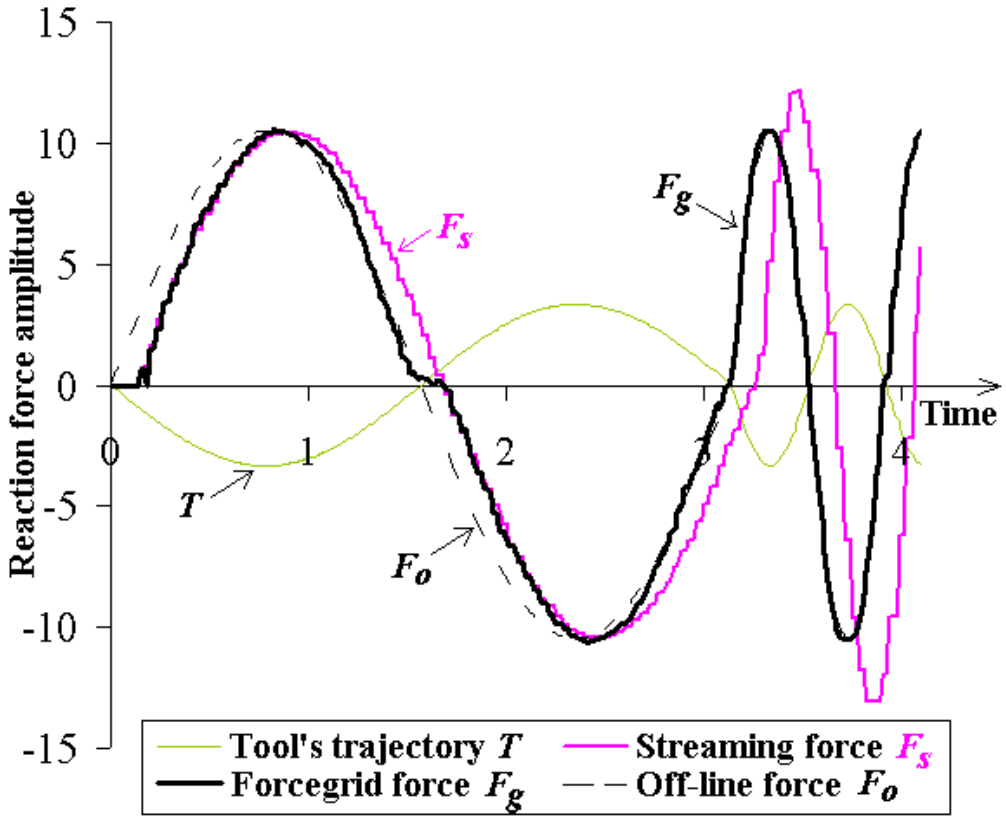

Figure 3.3: *Trajectory of the operational point and forces rendered with different techniques.*

Figure 3.3 shows the trajectory of the tip of the tool and 3 other curves that represent different reaction forces produced by 3 techniques. $t = 0$ is the instant when the contact is detected with the deformable object. The first

curve plots streaming forces F_s, i.e. forces that the simulator computes and sends according to the position of the operational point it receives from the controller. The second curve represents forcegrid forces F_g, forces rendered through our forcegrid implementation, using the silent method to fill the forcegrid. Finally, the last curve represents off-line forces F_o, actual forces that would be rendered if the simulator had an infinite time to compute the reaction force between each haptic cycle. F_s is actually the force that the controller receives in the forcegrid implementation, and with which it produces F_g. The curve representing F_s presents delayand jerkiness due to the 30Hz simulation rate and undergoes the inaccuracy produced by the simulation method when the movements are too fast. After a short initial adaptation period, F_g eliminates all these problems. The curves representing F_g and F_o become very similar whenever the operational point comes close to previously visited positions, regardless of the speed of the tool. Figure 3.4 presents the instantaneous errors of F_g and F_s with respect to F_o, for the same movement as in Figure 3.3. Note that the error for F_g tends to decrease over time, but the error for F_s increases when the tool moves faster. The computational delay imposed by the simulator is an important factor for the accuracy of F_s.

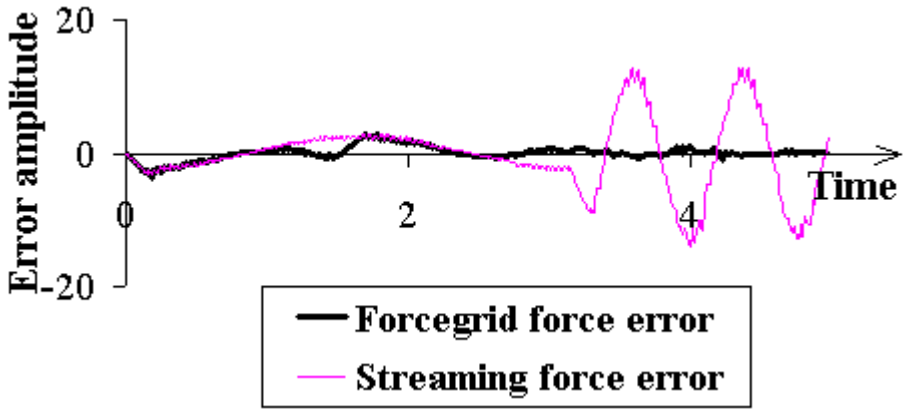

Figure 3.4: *Instantaneous errors of F_g and F_s.*

Chapter 4

Future work

4.1 More than 3 degrees of freedom

In the current form of the forcegrid, the haptic device must have three degrees of freedom and there can be only one instrument (hence, a single operational point) interacting with the deformable object. If the haptic device had six degrees of freedom (and was able to render torques as well), the forcegrid would have to be a 6-D grid and its size could become prohibitive. Similarly, if there were several haptic devices simultaneously interacting with the same object (each carrying a different surgical instrument), we could encode the positions of the various operational points in a single forcegrid - hence, if there are n haptic devices with 3 degrees of freedom each, the forcegrid would be a $3n$-D grid. But, again, its size would likely be prohibitive. A 6-dof haptic device would be particularly useful to render contacts other than point contacts, such as contacts over small surface areas (since the deformable object tends to locally conform to the rigid instrument), as well as multiple contacts (an instrument may have several disconnected contacts with a deformable object). Multiple haptic devices are needed to simulate two-hand or multi-surgeon surgery.

One way of dealing with many degrees of freedom is to drop the regular grid. Instead, we may consider the configuration space of all the haptic devices interacting with the deformable object. We can store the forces/torques computed by the simulator at the configurations where these computations took place. Over time, we will get forces over an irregular distribution of points, instead of a grid. To render forces/torques at the current configura-

tion of the haptic devices, we then identify the p closest configurations where the forces/torques have been previously computed by the simulator, and we interpolate among those forces/torques. Range techniques exist to efficiently identify closest neighbors in a metric space of any dimension [26].

4.2 Treating discontinuities

The forcegrid assumes that the behavior of the deformable object is smooth. In practice, however, discontinuities may occur. The occurrence of the contact between the surgical instrument and the deformable object is already one such discontinuity. Other discontinuities occur if, while deforming, the object makes contact with other objects (deformable or not) and with itself (self-collisions). As long as discontinuity events are sparse over time, it is possible to switch to a new forcegrid at each event. Some discontinuities, like the occurrence of new contacts, can be quickly detected by using hierarchical [17] or feature-tracking techniques [24], or a combination of both [23]. When discontinuities occur over extended period of time (e.g., while an instrument slides on the surface of the deformable object or cuts through it [25]), the forcegrid is no longer an appropriate tool. Indeed, previously computed forces are no longer valid, hence caching them is useless. A similar situation occurs if the elastic behavior of the deformable object has some hysteresis (but most existing simulators are unable to cope with hysteresis anyway). To deal with this issue, we could "forget" old forces and update the forcegrid accordingly. However, the forcegrid would lose some of its appeal and its updating would be more costly. In those cases, the buffer model of [22] or the simpler interpolation method of [21] might be better choices.

4.3 Anticipation with the request method

In the request method, the tool does not have to be in a certain area for the controller to request information about that area. We can imagine that the requests made to the simulator anticipate the position where the tool will be in a near future. This anticipation can be based on the direction and amplitude of the velocity of the tool at any instant. This quantitative data is directly accessible to the haptic controller, so there would be no extra delay introduced.

4.4 Extrapolation

In the forcegrid implementation we described, we use the values of the 8 forces $\vec{F_i}$ at the 8 vertices of the cell c that contains P to interpolate a force vector $\vec{F}$ in P. If P were outside of c, we could still use the values of the $\vec{F_i}$ to compute a force vector in P. This force vector would then be an extrapolation of the $\vec{F_i}$. We would still use the formula we derived in Section 2.2 for $\vec{F}$, except that this time, the coordinates x, y and z of P would be outside of the interval $[0, ...1]$. See Appendix A for a graph of the 2D interpolation of a scalar value outside of the delimited area (figure 3).

The extrapolation would be done during the grid building process: the $\vec{F_i}$ themselves would be extrapolated. In the filling implementation we described in Section 2.3, the algorithm weights the force $\vec{F}$ received for P and updates a nearby force $\vec{F_i}$ for the vertex V_i based on the distance from P to V_i. However, the actual value that contributes to $\vec{F_i}$ in that process is $\vec{F}$ and not an extrapolation of $\vec{F}$. Using an extrapolation of $\vec{F}$ seems more relevant as it would spread forces throughout the forcegrid using space orientation information, and not only distance information.

Conclusion

The forcegrid structure presented in this paper offers a number of advantages over previous haptic rendering techniques. It is independent of the modeling and computational techniques used to simulate the deformations of the virtual elastic object. It is also independent of the cycle time of the simulator and of the specific parameters of the physical model of the elastic object (mass, stiffness, damping, incompressibility, etc.). It is particularly easy to implement and the total compiled code is quite small, so that it can easily run on the local processor of any haptic device. It provides stable haptic behavior for a wide range of values of the forcegrid parameters. Experiments with many modeled objects have shown that it yields satisfactory haptic interaction.

Acknoledgements

The research reported in this paper was performed in the Stanford-NASA National Biocomputation Center at Stanford University. It was funded in part by grants from the National Science Foundation (IIS-9907060), the NIH National Libraries of Medicine (NLM-3506), and the National Aeronautics and Space Administration (NAS-NCC2-1010). This work benefited from discussions with Arnaud Tellier, Joel Brown, Remis Balaniuk, Cynthia Bruyns and Benjamin Lerman.

Appendix A

Interpolating a scalar value in a 2D world

If we have four scalar values S_1, S_2, S_3, and S_4 associated with four points P_1, P_2, P_3 and P_4 at the four corners of a square in a 2-D plane, we can interpolate a scalar value S at any position P inside the square. We place two coordinate axes X and Y (see figure 1) and use a formula of the form:

$$S = a + b.x + c.y + d.x.y$$

where $a,\ b,\ c$ and d are variables which depend on the four values S_1, S_2, S_3, and S_4.

We solve the equation with the following border conditions:

At $(0,0)$, $S = S_1 = a$
At $(1,0)$, $S = S_2 = a + b$
At $(0,1)$, $S_P = S_3 = a + c$
At $(1,1)$, $S_P = S_4 = a + b + c + d$

Thus we get:

$a = S_1$
$b = S_2 - S_1$
$c = S_3 - S_1$
$d = S_4 - S_3 - S_2 + S_1$

and the final formula for S is:

$$S = S_1 + x.(S_2 - S_1) + y.(S_3 - S_1) + x.y.(S_4 - S_3 - S_2 + S_1)$$

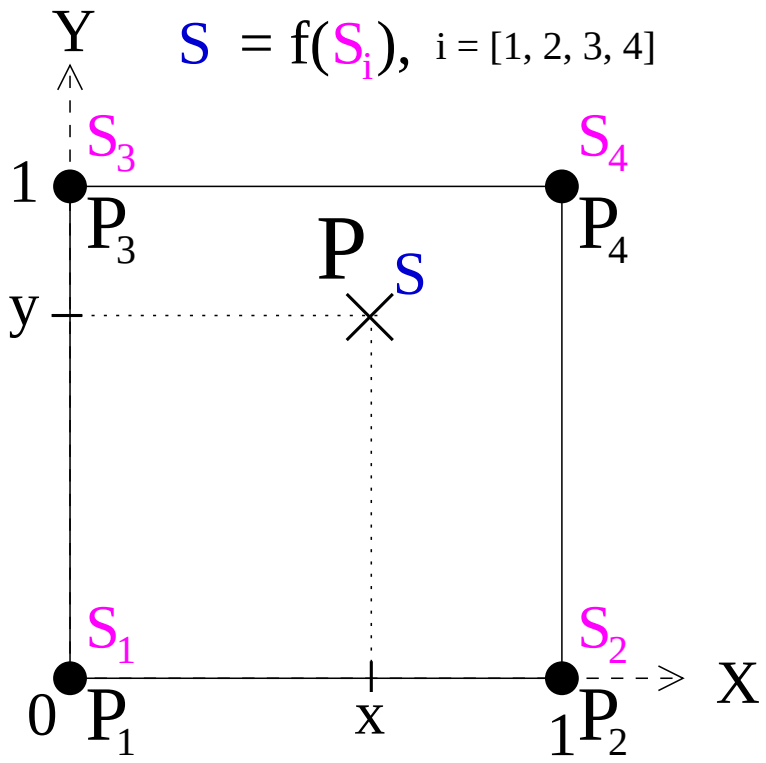

Figure 1: *A scalar value S can be interpolated in P knowing the four corner values S_1, S_2, S_3, and S_4.*

We can draw S as a function of the 2-D position of P, for P at every location inside or outside of the square delimited by P_1, P_2, P_3 and P_4 (see figures 2 and 3).

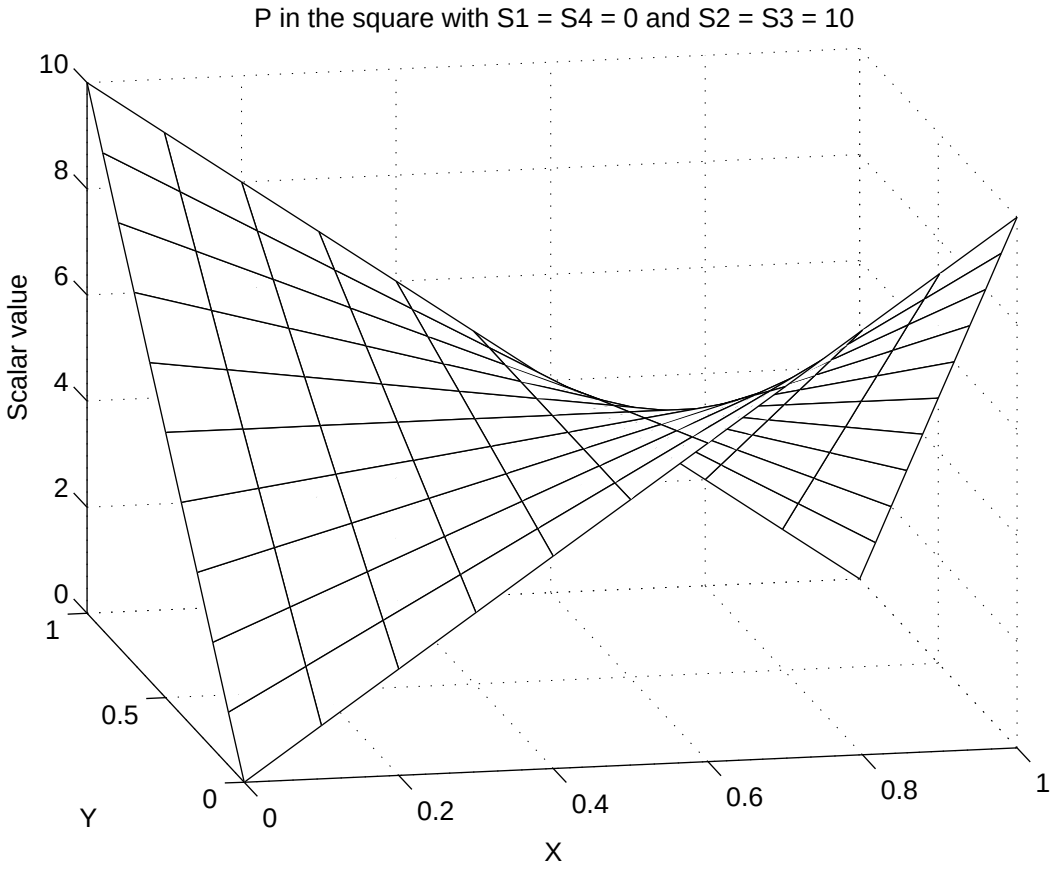

Figure 2: *Graph of S for P in the square. x and y vary between* 0 *and* 1

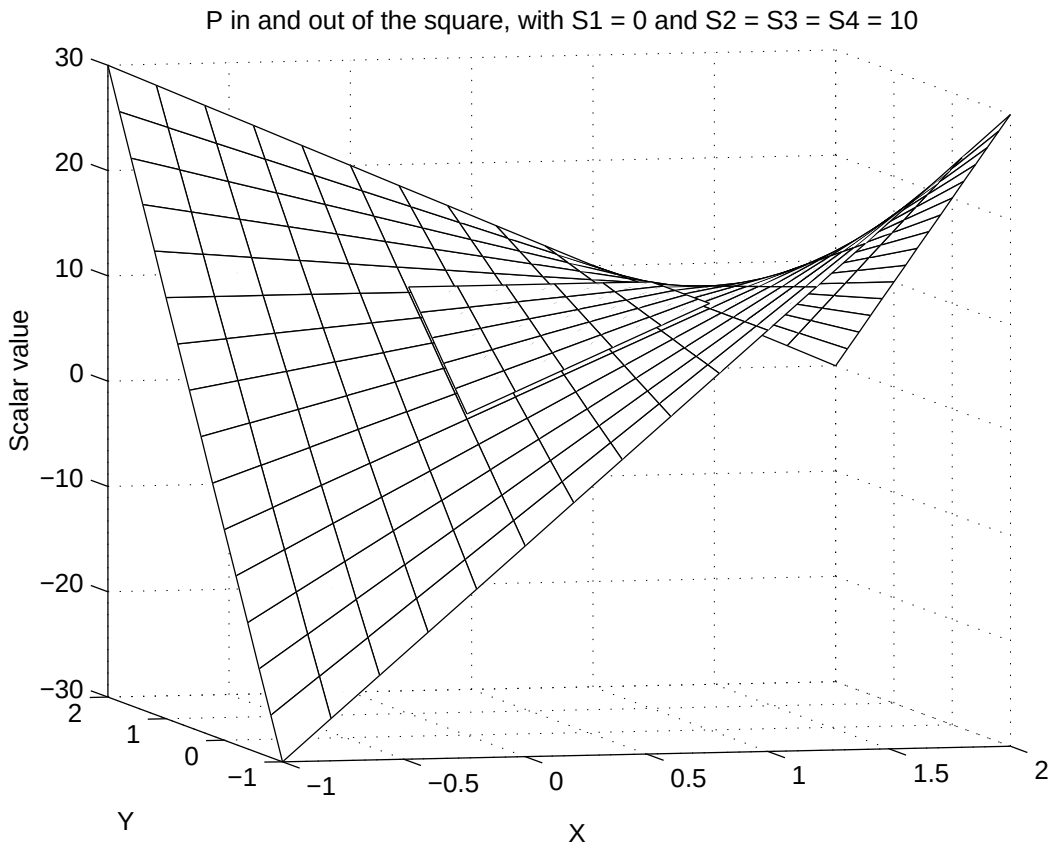

Figure 3: *Graph of S for P inside (surelevated) and outside of the square. x and y vary between -1 and 2*

Appendix B

Interpolation function for the forcegrid

The interpolation method used in the Forcegrid implementation is an extension of the method described in Appendix A. This time, we have vector values in a 3-D domain instead of scalar values in a 2-D domain. As stated in the report, we compute a new force $\vec{F}$ at tool's position P based on the normalized coordinates x, y and z of P in the local cell formed by the 8 vertices surrounding P (See figures 4 and 5).

Global approach

The dependence between $\vec{F}$ and the 8 forces $\vec{F_i}$ of the 8 surrounding vertices is established by solving the following equation $\vec{F}$:

$$\vec{F_P} = \vec{a} + x.\vec{b} + y.\vec{c} + z.\vec{d} + x.y.\vec{e} + x.z.\vec{f} + y.z.\vec{g} + x.y.z.\vec{h}$$

where $\vec{a},\ \vec{b},\ \vec{c},\ \vec{d},\ \vec{e},\ \vec{f},\ \vec{g}$ and $\vec{h}$ are forces computed from the forces at the 8 vertices:

At $(0,0,0)$, $\vec{F} = \vec{F_1} = \vec{a}$
At $(1,0,0)$, $\vec{F} = \vec{F_2} = \vec{a} + \vec{b}$
At $(0,1,0)$, $\vec{F} = \vec{F_3} = \vec{a} + \vec{c}$
At $(0,0,1)$, $\vec{F} = \vec{F_4} = \vec{a} + \vec{d}$
At $(1,1,0)$, $\vec{F} = \vec{F_5} = \vec{a} + \vec{b} + \vec{c} + \vec{e}$
At $(1,0,1)$, $\vec{F} = \vec{F_6} = \vec{a} + \vec{b} + \vec{d} + \vec{f}$
At $(0,1,1)$, $\vec{F} = \vec{F_7} = \vec{a} + \vec{c} + \vec{d} + \vec{g}$
At $(1,1,1)$, $\vec{F} = \vec{F_8} = \vec{a} + \vec{b} + \vec{c} + \vec{d} + \vec{e} + \vec{f} + \vec{g} + \vec{h}$

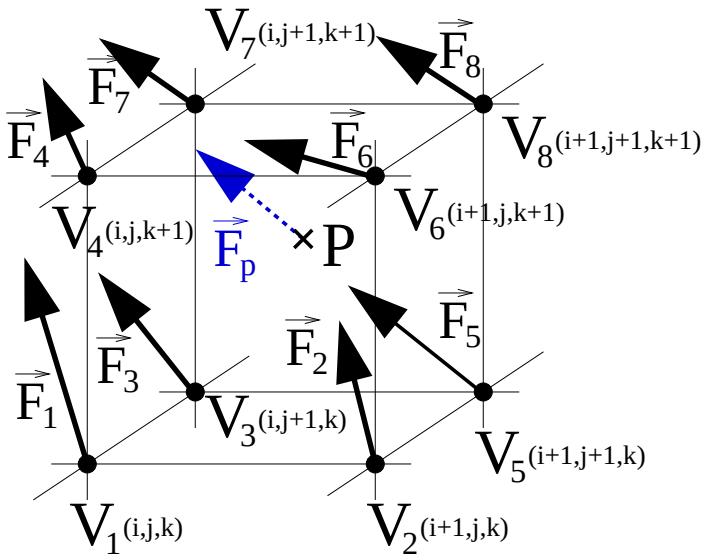

Figure 4: *V_1 is the nearest vertex to P that has its three space coordinates smaller than those of P. The interpolation uses the 8 forces $\vec{F}_i$ located at the 8 vertices V_i of the cell containing P. If the coordinates of V_1 in the 3-D matrix are (i, j, k), those 8 forces are determined by the indices shown on the figure.*

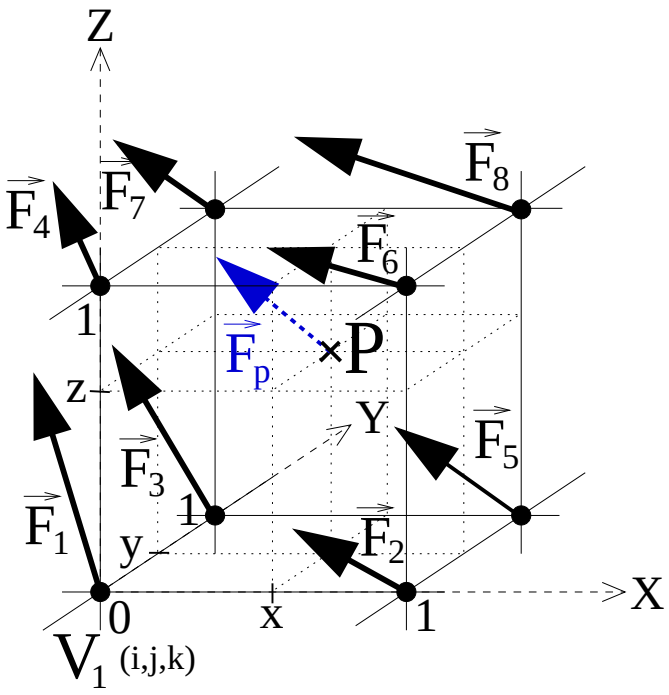

Figure 5: *The interpolated force is based on the local x, y and z coordinates.*

Thus we get:

$$\begin{aligned}
\vec{a} &= \vec{F_1} \\
\vec{b} &= \vec{F_2} - \vec{F_1} \\
\vec{c} &= \vec{F_3} - \vec{F_1} \\
\vec{d} &= \vec{F_4} - \vec{F_1} \\
\vec{e} &= \vec{F_5} - \vec{F_3} - \vec{F_2} + \vec{F_1} \\
\vec{f} &= \vec{F_6} - \vec{F_4} - \vec{F_2} + \vec{F_1} \\
\vec{g} &= \vec{F_7} - \vec{F_4} - \vec{F_3} + \vec{F_1} \\
\vec{h} &= \vec{F_8} - \vec{F_7} - \vec{F_6} - \vec{F_5} + \vec{F_4} + \vec{F_3} + \vec{F_2} - \vec{F_1}
\end{aligned}$$

and the final formula for $\vec{F}$ is:

$$\begin{aligned}
\vec{F} = \vec{F_1} &+ x.(\vec{F_2} - \vec{F_1}) + y.(\vec{F_3} - \vec{F_1}) + z.(\vec{F_4} - \vec{F_1}) \\
&+ x.y.(\vec{F_5} - \vec{F_3} - \vec{F_2} + \vec{F_1}) \\
&+ x.z.(\vec{F_6} - \vec{F_4} - \vec{F_2} + \vec{F_1}) \\
&+ y.z.(\vec{F_7} - \vec{F_4} - \vec{F_3} + \vec{F_1}) \\
&+ x.y.z.(\vec{F_8} - \vec{F_7} - \vec{F_6} - \vec{F_5} + \vec{F_4} + \vec{F_3} + \vec{F_2} - \vec{F_1})
\end{aligned}$$

Step by step approaches

We can consider other approaches to find this final formula. In figure 6, we label the points A, B, C, D, E and F, projections of P on the 6 faces.

We consider 6 force vectors $\vec{F_A}, \vec{F_B}, \vec{F_C}, \vec{F_D}, \vec{F_E}$ and $\vec{F_F}$ associated with those points. We use the interpolation formula described in Appendix A for a each coordinate of those vectors to interpolate their values. In order to do so, we consider the 4 forces at the 4 vertices of the face on which the point lies. Then, we interpolate a final value for a force $\vec{F}$ in P by linearly interpolating between two computed vectors in A and F, or B and E, or C and D. This is a 2-D then 1-D linear approach.

Doing so, we get the following formulas for the 6 force vectors, as functions of the $\vec{F_i}$, $i = [1, 2...8]$:

$$\begin{aligned}
\vec{F_A} &= \vec{F_1} + x.(\vec{F_2} - \vec{F_1}) + y.(\vec{F_3} - \vec{F_1}) + x.y.(\vec{F_5} - \vec{F_3} - \vec{F_2} + \vec{F_1}) \\
\vec{F_B} &= \vec{F_1} + x.(\vec{F_2} - \vec{F_1}) + z.(\vec{F_4} - \vec{F_1}) + x.z.(\vec{F_6} - \vec{F_4} - \vec{F_2} + \vec{F_1}) \\
\vec{F_C} &= \vec{F_1} + y.(\vec{F_3} - \vec{F_1}) + z.(\vec{F_4} - \vec{F_1}) + y.z.(\vec{F_7} - \vec{F_4} - \vec{F_3} + \vec{F_1})
\end{aligned}$$

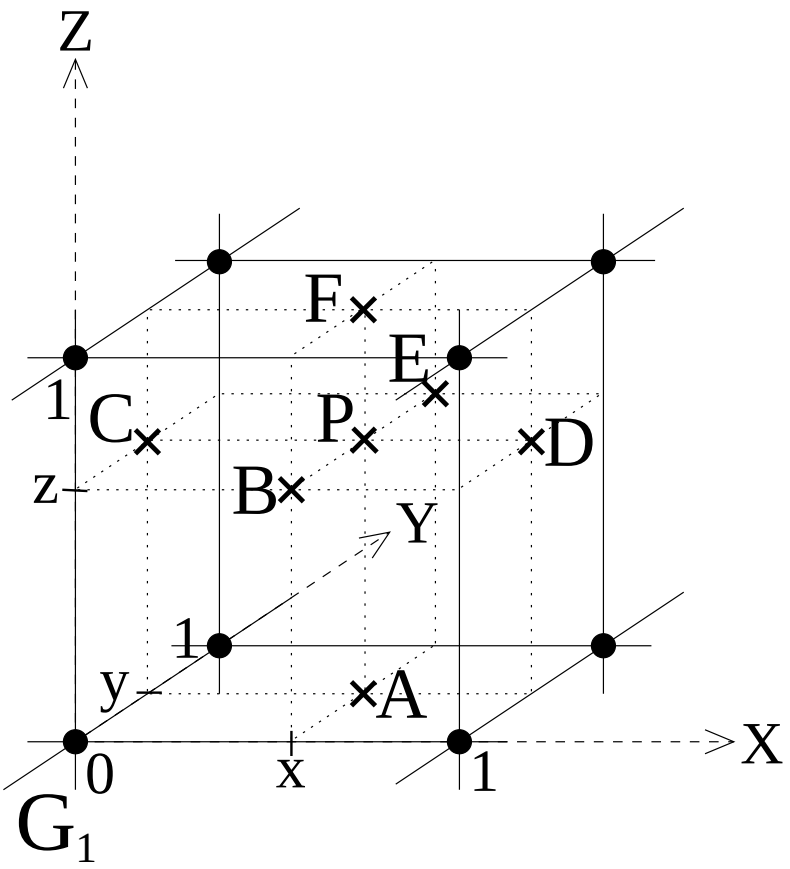

Figure 6: *Considering other points for a* 2*-D then* 1*-D linear approach*

$$\vec{F_D} = \vec{F_2} + y.(\vec{F_3} - \vec{F_2}) + z.(\vec{F_6} - \vec{F_2}) + y.z.(\vec{F_8} - \vec{F_6} - \vec{F_3} + \vec{F_2})$$
$$\vec{F_E} = \vec{F_3} + x.(\vec{F_5} - \vec{F_3}) + z.(\vec{F_7} - \vec{F_3}) + x.z.(\vec{F_8} - \vec{F_7} - \vec{F_5} + \vec{F_3})$$
$$\vec{F_F} = \vec{F_4} + x.(\vec{F_6} - \vec{F_4}) + y.(\vec{F_7} - \vec{F_4}) + x.y.(\vec{F_8} - \vec{F_7} - \vec{F_6} + \vec{F_4})$$

Then we think of $\vec{F}$, the force value in P, as a linear unidimensional interpolation between A and F on the line (AF) or between B and E on the line (BE) or between C and Don the line (CD).

We write:

$$\vec{F} = \vec{F_A} + z.(\vec{F_F} - \vec{F_A})$$
or $\vec{F} = \vec{F_B} + y.(\vec{F_E} - \vec{F_B})$
or $\vec{F} = \vec{F_C} + x.(\vec{F_D} - \vec{F_C})$

Those three ways to interpolate $\vec{F}$ give the same formula for $\vec{F}$, which is the one we also get with the global approach. We are always doing simple linear interpolations. We can also think of the 1-D then 2-D linear approach, or even the 1-D, 1-D and 1-D linear approach, and they will all yield the same formula for $\vec{F}$.

Bibliography

[1] H. Delingette *"Towards Realistic Soft Tissue Modeling in Medical Simulation"*, Rapport de Recherche #3506, INRIA, Sophia-Antipolis, France, Sept. 1998.

[2] D.H. House and D.E. Breen (eds.) *Cloth Modeling and Animation* AK Peters, Natick, MA, 2000.

[3] N. Ayache, S. Cotin, and H. Delingette *Surgery Simulation with Visual and Haptic Feedback*, Robotics Research, Springer, 1998, pp 311–316.

[4] B. Jackson and L. Rosenberg *Force Feedback and Medical Simulation*, Interactive Technology and the New Paradigm for Health-Care, IOS Press, 1995, pp 147–151.

[5] B. Marcus *Hands On: Haptic Feedback in Surgical Simulation*, Proceedings of Medicine Meets Virtual Reality IV (MMVR'96), San Diego, CA, January 1996, pp 134–139.

[6] A. Joukhadar and C. Laugier *Dynamic Simulation: Model, Basic Algorithms, and Optimization*, Algorithms for Robotic Motion and Manipulation, AK Peters, Natick, MA, 1997, pp. 419-434.

[7] M. Teschner, S. Girod, B. Girod *Efficient and Robust Soft Tissue Prediction in Craniofacial Surgery Simulation Using Individual Patient's Data Sets*, Proceedings of the International Conference on Computer Assisted Radiology and Surgery (CARS'99), Paris, France, June 23-26, 1999, pp. 635–639.

[8] J. Berkley, S. Weghorst, H. Gladstone, G. Raugi, D. Berg, and M. Ganter *Fast Finite Element Modeling for Surgical Simulation*

Proceedings of Medicine Meets Virtual Reality (MMVR'99), ISO Press, 1999, pp. 55–61.

[9] M. Bro-Nielsen and S. Cotin *Real-Time Volumetric Deformable Models for Surgery Simulation Using Finite Elements and Condensation* Proceedings of Eurographics'96, Vol. 15, 1996, pp. 57–66.

[10] J. D. Westwood, H. M. Hoffman, R. A. Robb, Don Stredney *"Medicine Meets Virtual Reality"*, IOS Press, Amsterdam, Netherlands, 1999.

[11] H. Delingette., *"Simplex Meshes: a General Representation for 3D Shape Resonstruction"*, Research Report #2214, INRIA Grenoble, France.

[12] D. Terzopoulos et al. *"Physically Based Models with Rigid and Deformable Components"*, IEEE Computer Graphics & Applications, November 1988, pp 41–51.

[13] *"Finite Element Method"*, http://vislab.eecs.uic.edu/∼bzarit/588/face8.html

[14] *"What is the Finite Element Method"*, http://www.dundee.ac.uk/∼rimackie/CE424/Lec2/Lecture2.htm

[15] C. Bosdogan, C. Ho, M. A. Srinivasan, S. D. Small, and S. L. Dawson. *"Force Interaction in Laparoscopic Simulation: Haptic Rendering Soft Tissues"*, Proceedings of the Medicine Meets Virtual Reality (MMVR'6) Conference, January 1998, pp 28–31.

[16] G. Burdea, G. Patounakis, V. Popescu, and R. E. Weiss. *"Virtual Reality Training for the Diagnosis of Prostate Cancer"*, IEE International Symposium on Virtual Reality and Applications (VRAIS'98), Atlanta, Georgia, March 1998, pp 190–197.

[17] J. Brown, K. Montgomery, J.C. Latombe, M. Stephanides. *"A Microsurgery Simulation System"*, International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI'01), Utrecht, The Netherlands, Oct. 14-17, 2001.

[18] J. Brown, S. Sorkin, C. Bruyns , J.C. Latombe, K. Montgomery, and M. Stephanides. *"Real-Time Simulation of Deformable Objects: Tools and Application"*, IEEE Conf. on Computer Animation, Seoul, Korea, Nov. 6-8, 2001.

[19] S. Cotin, H. Delingette, and N. Ayache *"Real-Time Elastic Deformations of Soft Tissues for Surgery Simulation"*, IEEE Transactions On Visualization and Computer Graphics, 5(1):62-73, January-March 1999.

[20] G. Picinbono and J.-C. Lombardo *"Extrapolation: a Solution for Force Feedback?"*, International Scientific Workshop on Virtual Reality and Prototyping, Laval (France), June 3-4 1999.

[21] G. Picinbono, J.-C. Lombardo, H. Delingette and N. Ayache *"Anisotropy, Interaction and Extrapolation for Surgery Simulation"*, J. Visualisation and Computer Animation, 2001 (to appear).

[22] R. Balaniuk *"Using Fast Local Modelling to Buffer Haptic Data"* Proceedings of The Fourth PHANTOM Users Group Workshop (PUG99), Boston, MA October, 1999.

[23] M.C. Lin *"Fast and Accurate Collision Detection for Virtual Environments"* IEEE Scientific Visualization Conf., 1999.

[24] M. Lin and J. Canny *"Efficient Algorithms for Incremental Distance Computation"* Proceedings of the IEEE International Conference on Robotics and Automation, 1991, pp. 1008–1014.

[25] S. Cotin, H. Delingette, and N. Ayache *"A Hybrid Elastic Model Allowing Real-Time Cutting, Deformation and Force Feedback for Surgery Training and Simulation"* The Visual Computer, 2000, 16(8):437–452.

[26] P.K. Agarwal *"Range Searching. Handbook of Discrete and Computational Geometry"* CRC Press, 1997, pp. 575–598.

More documentation on the haptic devices we use

http://www.sensable.com/haptic/products/desktop.html

http://www.immersion.com/products/custom/laproimpulse.shtml