SANDIA REPORT
SAND2000-2240
Unlimited Release
Printed December 2000

Guidelines for Computer Haptics Protein Simulations

Derek T. Mehlhorn

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

Sandia National Laboratories

SAND2000-2240
Unlimited Release
Printed December 2000

Guidelines for Computer Haptics Protein Simulations

Derek T. Mehlhorn
Computational Biology and Materials Technology Department
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-0316

Abstract

Computer haptics is a field of science that studies how one's sense of touch can be incorporated into a virtual environment. The uses of computer haptics are unlimited, ranging from graphics design, where an artist can feel the clay deform as they work, to protein simulations, where the user can feel the surface and attractive forces that a protein would actually exert. In order to create a haptics simulation, one must first understand the basic elements and the work -load involved in networking complex 3-dimensional computer graphics with a force feedback device. This report outlines the basic elements, as well as advanced and experimental ones, related to computer haptics protein simulations. Simulations and experimentation, that discussion within this text is based on, were done using the haptics programming interface eTouch and a desktop PHANToM 1.0 force feedback device. This report is intended as both an introduction to computer haptics, as well as a tutorial for advanced protein simulation design.

INDEX:

Introduction:

What is Computer Haptics?

Haptics is the field of science that studies the properties of human touch. Subsequently, computer haptics is a field of science that studies how one's sense of touch can be incorporated into a virtual environment. Research in computer haptics is based around allowing the user to interact with the computer in the same way that one would interact with the real world. What this means is that two-dimensional input devices such as a mouse and keyboard simply will not suffice. The user needs to be able to interact in at least three-dimensions, and the computer needs to be able to interact with the user on a physical level.

The uses of computer haptics are unlimited, ranging from graphics design, where an artist can feel the clay deform as they work, to protein simulations, where the user can feel the surface and attractive forces that a protein would actually exert. Computer haptics is the study of how a computer can more accurately and convincingly simulate reality starting with ones sense of touch.

The PHANToM[1], a Computer Haptics Device

Image 1.0 A Desktop Phantom

Within the field of computer haptics, there a number of commercially available devices which allow a broad range of motion and force feedback capabilities. A particularly good haptics device which provides the user with 6 degrees of freedom (x, y, z, yaw, pitch, and roll) as well as 3 degrees of freedom of force feedback (x, y, z) is the PHANToM (Image 1.0). Although there are several different models of the PHANToM, all discussion in this paper is based on the use of a *Model A PHANToM 1.0.*

etouch[2], a Haptics Simulation Environment

The two basic elements of a computer haptics simulation are graphics and forces. In order to generate both of these elements, software that allows your code to network with the force feedback device is required. In the case of the PHANToM, a set of C++ libraries called GHOST[1] (Graphical Haptic Open Software Toolkit) can be used. One thing to keep in mind when undertaking a haptics simulation is that you cannot simply render a series of graphics and be done with it. You actually have to create a full three-dimensional world. Unless your project is to actually create a 3-D interface for computer haptics, it is necessary to acquire an existing haptics environment to create your unique simulation within. A powerful and user-friendly environment that was originally developed within Sandia National Laboratories called eTouch[2], was used in creating and testing the simulations and concepts outlined in this report.

eTouch is essentially a haptics operating system, within which one can produce and run simulations. The eTouch environment is extremely powerful and useful, not only because it controls the multi-processing of the graphics and forces, but also because it provides the user with a number of ready to use tools and perspective devices. Interfaces, such as eTouch, allow the simulation designer to focus on their project specifically, without them needing to be overly concerned with a large number of interfacing issues.

Basic Simulation Elements:

The two basic elements of a computer haptics simulation are graphics and forces. One of the interesting things about a haptics simulation is that both of these elements work independently of each other. This divides the simulation into two distinct parts, each of which present unique problems that can be addressed in independent and different ways.

In a computer haptics simulation, certain process speeds must be maintained. The graphics routines must maintain a 30 Hz refresh rate, i.e. 30 times per second, in order to remain synchronized with the simulation's forces. The force routine must maintain a 1000 Hz refresh rate to prevent "feelable" lapses in the force feedback. Although a number of factors go into making a successful simulation, the success of ones application will ultimately rely on good refresh rates.

Graphics Generation:

Simple Shapes

There are a number of different approaches that can be taken when trying to generate graphics for one's simulation. Through my work with eTouch and haptics in general, I concluded that there are two general options for rendering molecular graphics. The first is to use OpenGL's built in simple shapes drawing routines to draw each atom as a single sphere. This is the fastest solution to the problem and also has the potential to provide a high degree of resolution that may be aesthetically superior to other methods. This method is the most quickly compatible with protein data files since to generate a sphere you only need its location and radius. If the number of atoms in one's simulation is relatively low it is advisable to use this method because it is the easiest and least time consuming solution.

Due to its simplicity and the ease with which it networks with data files, the simple shape approach is very tempting, however, its performance and desirability quickly decreases as the size of one's proteins increase. This is because, although easy to implement, the simple shape algorithms are not very fast. As one begins to increase the number of spheres that must be draw, the graphics refresh speed decreases substantially. There are a number of things that can be done in order to counteract the slow down caused by an increased number of spheres. If for example, if your protein has multiple atom types, that you want to render in different colors but that do not appear consecutively within the data file. You can preprocess the data, and sort the atoms by type. By sorting the data and storing it in a data structure within your application, the graphics rendering loop can be optimized by minimizing the data retrieval time as well as a number of other time consuming operations, such as changing material properties. Keep in mind that data preprocessing in pretty much a must for all haptics simulations regardless of ones approach. Due to the relentless refresh rates that such an application demands, the ability to have immediate access to information and the ability to minimize costly graphics definitions is required.

Data sorting can be a powerful method for increasing graphics refresh, however, when the number of atoms gets too great, your simulation will have to begin sacrificing sphere resolution in order to maintain graphics speeds. This is where the simple shapes method begins to loose its appeal, as your spheres turn into low-resolution polygons. There are, however, other tricks that can be used to minimize the ramifications involved in decreasing resolution. For example, decreasing the overall resolution of the protein while increasing the resolution of the spheres in the immediate vicinity of the cursor will maintain resolution in the most important places while maintaining refresh speeds. Although, this method can maintain the required refresh rates, it is a fairly ugly and undesirable solution.

The major problem with using sphere functions to generate graphics is that there are far too many wasted surfaces. OpenGL will create a full three-dimensional sphere for each atom, however, when working with a fully folded protein, a substantial number of sphere surfaces will overlap and a large percentage of the atoms probably will not be visible at all. Thus, OpenGL is either forced to draw these obstructed surfaces, which takes extra computing power, or calculate which surfaces are hidden, and not draw them. Although removing hidden surfaces is substantially faster than drawing them, when dealing with a large number of atoms, it can still take too much time.

Another problem with creating graphics using the sphere functions is that each time you draw a sphere the perspective matrix of your graphics has to be translated over. One cannot simply render a lot of spheres at the positions x, y, and z. Instead one must move the perspective matrix to x, y, z and then draw the sphere. Although data sorting can minimize the number of translations that must be made, this process is still much slower than being able to initially define a large number of absolute object positions.

Triangular Surface Mesh

Due to the large number of wasted surfaces when using spheres, it may be preferable to render your protein using a triangular mesh surface model. The idea behind this method is to determine the shape and contour of the protein's surface, within a given resolution, and create a graphics "shell" of the protein. This way no surfaces are lost because you are specifically creating your graphics to render only what is visible to the user. This is a substantially faster method than using simple shapes, not only because no surfaces are wasted, but also because basic polygons are easier to render.

One possible draw back from this solution is the level of detail that may or may not be possible. In order to create the surface model of your protein, you need to define a vector field of a certain shape. Knowing the center location of each atom, the furthest sphere intercept along these vectors can be calculated (Image 1.1). Each point of intersection will then be used as a vertex in creating a surface mesh

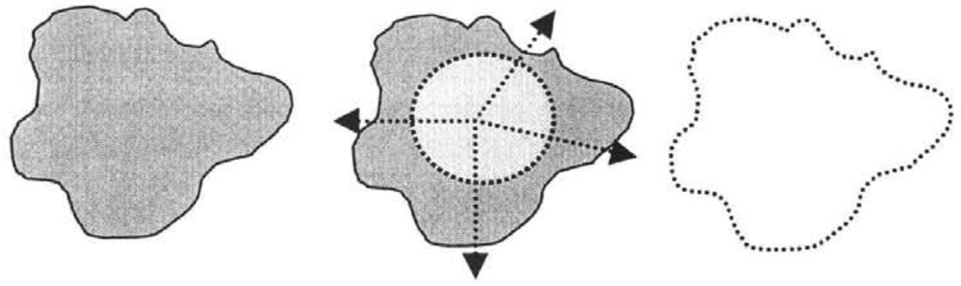

Image 1.1
Shows a how a circular vector field can be used to plot the surface of a semi-spherical protein

of triangles. Rendering the surface of the protein as a number of points can also be useful as it can give the user a good perspective of the molecule's overall shape and size. In general, the resolution of ones surface map will increase with the number of vectors (i.e. points) which you define it with. However, it may be that the number of vectors, and thus triangles, required to obtain the desired resolution will be so great as to also slow down the graphics loop. My experience with this method thus far has been promising, and I have been very please with the level of resolution which it is possible to obtain with a minimal amount of processor power. However, this method may not work for all proteins since certain protein shapes may not be well suited for a vector field to map. This method of surface mapping is best suited for spherical proteins and may not adapt well to other shapes.

Other drawbacks of this method include, for one, its time consuming nature. Where simple shapes require a center, a size, a resolution, and a light source, a surface mesh requires an entire program to simply plot it. Once the coordinates of the surface points have been calculated, there is still a substantial amount of work that is required to create a convincing mesh surface. For example, one needs to create texture maps for the protein and define the shadowing of the polygons otherwise it just looks like a 2-dimensional blob. This can be a time consuming process, however, it also allows one to have a lot more control over what one's protein will ultimately look like, which can be beneficial. Also, your simulation cannot work properly without both the force and graphics loops maintaining their given refresh rates. As such, complex and time-consuming solutions to both graphics and force generation may be one's only options.

Force Calculations:

In order to understand many of the difficulties that are inevitably encountered when creating a protein simulation, one must first understand how forces in a computer haptics simulation work. Basically all surface forces in a computer haptics simulation act like springs, where the force they exert is equal to a constant times the depth of penetration, $F = kx$. Other forces such as magnetic or attractive force are simply functions of ones distance from the source. Objects in haptics act like springs because the PHANToM motors are not infinitely strong; therefore it is impossible for them to exert a realistic equal and opposite force. As a result, all objects in haptics simulations are inherently "squishy" as well as "springy." When dealing with single objects or large numbers of non-overlapping objects, surface squishiness and springiness are not a major concern. However, these characteristics cause a large number of problems that can be very difficult to correct.

The first force problems when working with overlapping objects is cause by force springiness. Because force magnitude is dependent on the depth of penetration, when one touches two spheres at the same time but does not penetrate their surfaces an equal distance, one surface will exert a greater force than the other (Image 1.2). This will induce a greater penetration into the other atom, which produces the same result. All of this takes place at one thousand times per second, which causes a high frequency oscillation of the PHANToM that can be felt or even heard. In many circumstances, these unbalanced forces are negligible, however, given the correct placement and number of objects what is called force buzzing can occur.

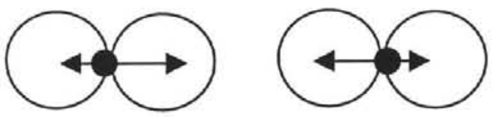
Image 1.2
Shows a force diagram of how force buzzing occurs when interacting with two objects

Force buzzing is simply where the PHANToM vibrates because an inconsistent or unstable stream of forces is being fed to it. Object springiness can cause this because when two identical spheres push on the cursor, and the first sphere pushes harder than the second, the cursor will penetrate the second sphere more deeply, inducing a strong force in that sphere and so on.

There are a number of solutions to this problem, many of which are not simple and may have undesirable side effects. The easiest way to eliminate buzzing, at least to the extent of audible recognition, is to vary the spring constant of the surface forces proportionally to the number of objects being touched. In order to make this work a number of different tricks can be used to ensure that the user does not know that object surfaces will have variable forces. One method that was successful for me was to choose a spring constant range (.05 to .1), and than I would ramp this constant depending upon the change in the number of atoms the cursor is touching. For example, if you begin by touching one atom, you will feel the maximum surface force. Then if you move the cursor to touch more atoms, the force will decrease and will increase again as you touch less. If one's range of forces and the speed at which one ramps them is reasonable, this method can alleviate solid surface buzzing produced by a number of overlapping "solid" objects. The major draw back from this approach is that your proteins will feel softer and squishier which may be undesirable.

Another solution to force buzzing produced by overlapping objects is to not have them. Much in the same way that a graphics shell of a protein molecule can be generated, a force shell for a protein can be created. By creating a force shell made up of a number of predefined *normals*, one no longer has to worry about inter-object forces. The main draw back from this approach is that you can no longer push into a protein. A force shell will have the affect of making your protein completely impenetrable and may greatly reduce the value of one's simulation.

The optimal solution to force buzzing requires a great deal of work and a certain level of expertise in digital filtering. I have done no research on this topic so I cannot go into specifics. However, I do know that it is possible to using digital filtering to remove high frequency forces before they are sent to the PHANToM. This approach should enable one to create smooth protein force simulations without sacrificing any aspects of the simulation.

Maintaining Refresh Rates in Large Simulations

When running almost any size protein or molecular simulation, one will find that it simply isn't possible to maintain the required refresh rates if one is trying to keep track of all the atoms in a simulation simultaneously. In my research, I found that the most effective way to maintain simulation performance is to divide the protein into grids by space. Each turn the grid number of the cursor is calculated and then collisions and ambient forces are checked and calculated for only the atoms in the same grid. Although there is a certain amount of work that is required to maintain force continuity when moving from one grid to another, it is a versatile and useful way to increase one's simulation performance.

Spatial decomposition of one's simulation is required, especially when more advanced simulation elements are added. When one's simulation becomes too complicated for a single processor to handle, each grid can be assigned to a processor of its own. This will enable much more complicated simulations to be performed in real time, and will enable one's simulation to run on a variety of different machines with a number of different processors. Ultimately I believe that the kind of equipment required to run complex molecular simulations involving most of the *Advanced Simulation Elements* described below, would require a cluster of at least ten Pentium processors. A small cluster of PC's would probably be sufficient to run these simulations, but a larger system such as access to C-Plant would be optimal.

Advanced Simulation Elements:

When thinking about designing a protein simulation, it is important to always keep in mind the computational constraints that exist. You must remember that any solutions or features that you are interested in implementing will need to be graphically rendered at 30hz and will physically need to be calculated and corrected at 1000hz. Due to these speed requirements, it is important to consider and implement the best and most efficient solutions possible, and do as many calculations and as much sorting as possible before the simulation begins.

Force Equation Implementation and Integration

One of major problems which pervades all protein simulations, regardless of size, is the issue of calculating interactive forces between each of the atoms in the molecule and the cursor. One of the first problems that one will encounter is scaling the force equations to values that the PHANToM can except. This can be difficult since the PHANToM takes numbers roughly between 0 and 1.5, whereas force equations will yield values in the tens of thousands. Also, scaling forces isn't quite as simple as reducing the force value to be within the zero to one-point-five. Instead there is a difference in the scaling requirements depending on how many atoms are present in the simulation. This is because the forces exerted by each atom are summed and may produce a total force magnitude greater than is desired, or physically impossible for the PHANToM to maintain.

This brings up another problem with force equation scaling, that some forces, inparticular repulsive ones, may go to infinity so quickly that there is no way to scale them to a reasonable level. In my experience the repulsive forces, as produced by real world force equations, are useless. Every haptic device has a limitation of the maximum (8N) and minimum forces that it can produce as well as a maximum maintainable force (1.4N) and the speed at which it can engage these forces. With repulsive forces that instantaneously become infinite, two of the computer haptics physical limitations are reached. First, the PHANToM is unable to instantaneously create a force of the magnitude that the force equation produces. Thus there is a small delay in the force feedback that will be initially manifested by a click (which is the servos disengaging and then re-engaging). This not only creates a disruption in the simulation's forces, but during the several thousandths of a second that the force feedback is not working, the cursor can travel into an object. Since the magnitude of the force that an object will exert in a computer haptics simulation is dependent upon the depth of penetration, when the forces re-engage the PHANToM is told that it is inside a, mathematically, infinitely hard surface. What happens in this situation is the PHANToM will try to produce this infinite force and will fail, which will again cause the servos to momentarily disengage. Since the PHANToM cannot possibly instantaneously exert its maximum force, it will produce a very strong force which will ramp up in about one one-thousandth of a second and then disengage. This will have the effect of create very bad and inconsistent forces since you will touch the surface of the molecule and get, violently, kicked back a short distance before the motors disengage.

The second limitation of the PHANToM that is reached is its ability to maintain a large force consistently. One can actually scale the forces low enough that when you artificially penetrate the atoms surface, while the motors are still engaging, the PHANToM doesn't exert a great enough force to knock you out again. It does, however, exceed its maximum exertable force thresh-hold, which means that the motors will have to shut down momentarily. The result is that you can push through your repulsive fields but the forces will jerkily engage and disengage the entire way.

The solution to the problem produced by repulsive forces is to simply remove them. Basically all that the repulsive field should "feel" like, is a hard surface so each atom should be defined as such. When the cursor gets too close to an object, and the attractive forces become repulsive, one can substitute a reasonable, constant, surface force for the infinite repulsive ones. This may not produce a mathematically realistic result, but will produce a realistic haptic result for the user.

Whereas repulsive forces are realistically unusable in a haptics simulation, attractive forces are not, nor would a protein simulation be worth much if they were. Attractive forces are far more reasonable in their rate of change, and are therefore relatively easy to scale. However, the magnitude of the attractive force is not the only problem that one will encounter. As was described in the *Force Calculations* section, there is a large problem with computer haptics and force buzzing. Overlapping forces can cause the buzzing that is produced by a hard surface of overlapping objects, to be enhanced by the attractive forces that are constantly pulling the cursor against the protein surface. Another problem that results from grafting attractive forces onto a hard surface is that if you were previously varying the stiffness of your protein surface to eliminate buzzing, you need to find a different solution. This is because as you are moving across the protein the cursor is being pulled against its surface and the user can feel the change in resistance as the forces are scaled to eliminate buzzing. Unfortunately, the solution to this problem, short of digital filtering, is to decrease the overall surface force of the protein, i.e. make everything equally hard or rather equally squishy. The side effect of this solution is that your protein will be pretty soft.

The real problem with overlaying forces is that they simply aren't compatible. Obviously one is repulsive and one is attractive, but they are not instantaneously so, and thus they get into a similar situation as the previously outlined inter-object problem. What happens is that in the time it takes the forces to engage, the cursor has moved too far into the solid or attractive forces. This in turn pushes the cursor back into the opposite forces and so on and so forth, causing precisely the same kind of force buzzing as before. Basically, any time you have directly opposing forces your going to have force problems. Thus, one of the two opposing forces needs to be removed or reduced. Remember, however, that you only want to modify the forces that are "normal" (perpendicular) to the surface of the object you are in contact with. Modifying all the forces will affect the overall value and realism that your simulation provides.

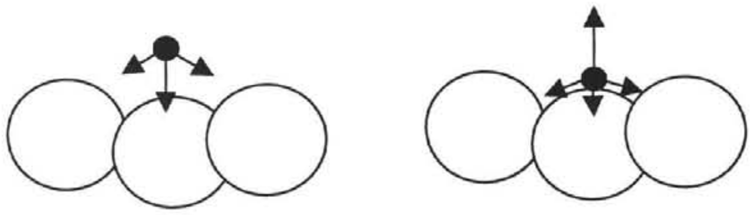

Image 1.3
This image shows how the component of the attractive force *normal* to the surface of an atom is reduced to eliminate force buzzing

I found that to eliminate force buzzing, when working with multiple force types, it is best to use a combination of scaling techniques. First of all it is necessary that the surface of the protein does not produce any buzzing of its own. Thus, as was previously mentioned, one needs to reduce its overall stiffness proportionally to eliminate surface buzzing. Once you have made the surface *squishy,* you must filter the attractive forces so that they do not strongly conflict with the surface ones. This entails isolating and eliminating a portion of the force vector that is *normal* to the surface of each atom that the cursor is in contact with (Image 1.3). This will have the affect of smoothing out the forces as long as you are perfectly in contact with the surface. However, this will also produce an amount of buzzing immediately above the surface cause by the engaging and disengaging of the *normal* scaling. To correct this the bound at which *normal* scaling is engaged and the bound at which it is disengage need to be different, i.e. you need to get further away from an atom to re-engage *normal* forces than you do to disengage. It may also be necessary to "ramp" the *normal* forces back up to full strength in order to prevent jumpy and inconsistent forces.

Solute Molecules

The two major issues involved in creating a simulation in which there are a number of solute molecules surrounding the protein that the user can interact with are, graphics rendering and speed of movement calculations. The main problem would be related to calculating how the molecules should behave depending on what the user does. The computing difficulty of this problem can be further increased if one wants the solute molecules to "flow" around the simulation. The computing power that ambient molecule motion will require is substantial, varying depending upon the number of molecules and the kinds of interactions being calculated.

Not only will the x, y, z motion of each molecule need to be determined, but the rotational characteristics of each molecule must also be calculated. The rendering of the graphics themselves may prove challenging from a processing standpoint, however, the implementation of this kind of simulation feature would require multiple processors, and as such I would dedicate at lease one to graphics generation alone. I would estimate that the total simulation's graphics would occupy one processor, and that at least two or three processors would be required to calculate the motion, including rotation, of the solute molecules. This number will, however, vary with the number of molecules being simulated.

If a multi-processor machine is not available, it may be possible to run a low level, small number, solute simulation on a single processor, where the solutes are stationary until acted upon. Even for a simulation of limited magnitude it would be desirable to have at least two processors to split the work.

Molecule Interaction and Deformation

Intermolecular interactions are a very operation intensive feature to implement. In order to produce the correct forces for a small multi-atom protein rubbing or pushing against a large one requires that one calculate the forces exerted on each of the atom in the small molecule. This means that if you have a two-atom molecule, your force calculations will take twice as long as if you only had one. The time it takes to calculate the forces will increase proportionally to the size of the interacting molecule until the molecule is large enough to support other techniques of manipulation, such as spatial decomposition.

A feature where one would push the cursor into a protein and actually deform its overall shape would produce the same general problems as creating solute molecules. It would, however, be substantially easier and would probably run on a single processor. Possible problems would include a lag time between when the cursor moves into the protein and when the protein changes shape. This feature should be fairly easy to integrate into a simulation that already simulates solute molecules, especially if it is running on multiple processors using a spatial decomposition scheme.

Transparent Graphics Options

Transparent graphics for things like solute molecules or so one can see the inside of a protein are very tricky because they must be rendered from back to front. If the image were stationary this would be easy. One would only need to sort the data by depth at the beginning of the simulation. However, in a simulation, the viewing perspective can be changed, which would reorganize the graphics relative to the view screen. Also solute molecules could be floating around the screen constantly, moving in front of and behind different objects. Due to these factors transparent object generation becomes extremely impractical because it requires a reorganizing of all the graphics simulation information every turn. An alternative to actually render transparent surfaces is to create objects using a loose scattering of points that can create the illusion of a transparent object.

3D force Vector Diagrams

A feature that could be quite useful and relatively simple computationally as well as graphically is to create a vector diagram of the forces that the cursor would feel when interaction with a protein. This could be done with lines representing force vectors, or regions of space designated by a population of point that would imitate a transparent surface. This would be useful in further relating one's sense of sight and touch together in the simulation without unduly stressing the computational requirements of the application.

Current Simulation Abilities

I believe that all of the features outlined in the *Advanced Simulation Features* section are possible to accomplish given current computer technologies. These kinds of simulations are well suited towards PC clusters since it is practically guaranteed that you will need to break the simulation elements down by space. The concerns I have about completing a simulation with the outlined features are not related to the existence of the required resources, but the accessibility of them and the time required to create the simulation. A possible problem will be having a machine with multiple processors that can be used, real time, to run simulations. It simply isn't possible to submit a batch job and wait for information to come back a week later. Development of C-Plant and the prominence of other smaller clusters, however, should provide all the power and accessibility that is required.

Creation of the simulation alone would be a substantial task, as most of the issues involved have never been done in a real time simulation. There is also the concern that haptics devices such as the PHANToM simply will not suffice for the level of detail which this kind of simulation is aimed at. The acquisition and integration of different, or possibly multiple, devices would require much more work. The force related part of the simulations have so far shown themselves to be the most difficult, and would become even more so as the number of force feedback degrees of freedom begins to increase. I have faith, however, that it is currently possible to, with a substantial time contribution, create a simulation containing the elements and concepts outlined in the report.

Bibliography:

1. SensAble Technologies, Woburn MA. http://www.sensable.com

2. Tom Anderson, Novint Technologies. http://www.novint.com

DISTRIBUTION:

1	MS-9018 Central Technical Files
1	MS-0612 Review & Approval Desk, 9612 For DOE/OSTI
2	MS-0899 Technical Library, 9616
2	MS-0316 Grant S. Heffelfinger, 9235
2	MS-0318 Arthurine Breckenridge, 9227
1	MS-1111 Marcus G. Martin, 9235
1	MS-1674 T. A. Mehlhorn
1	MS-1674 Derek T. Mehlhorn