A Polygonal Method for Haptic Force Generation

Tom Anderson
Dept. of Electrical and Computer Engineering
University of New Mexico
Albuquerque, NM 87131
Sandia National Labs
Albuquerque, NM 87185-0318
email: tganders@unm.edu

Abstract

Algorithms for computing forces and associated surface deformations (graphical and physical) are given, which, together with a force feedback device can be used to haptically display virtual objects. The 'Bendable Polygon' algorithm, created at Sandia National Labs and the University of New Mexico, for visual rendering of computer generated surfaces is also presented. An implementation using the EIGEN virtual reality environment, and the PHANToM (Trademark) haptic interface, is reported together with suggestions for future research.

1. Introduction

A great deal of effort and investment is currently being directed toward the creation of realistic virtual reality environments. For the most part, these computer generated realities are only visual renderings of the modeled objects. Until haptics is addressed, many VR worlds will be deficient. The need for haptic feedback is not merely aesthetic, as its inclusion has been found to significantly improve performance in certain tasks [12][13]. It has also been shown that people are remarkably good at recognizing objects by the sense of touch alone [10]. In addition, the sense of touch is unique, in that it is the only sense that is bi-directional -- one not only receives input, but affects the world as well. Therefore, haptics can be essential for many applications, without which objects will always seem more "virtual" than "real".

While there has been significant progress in graphical rendering, haptic rendering offers a new and exciting research area, some of which is only now possible because of current advances in force feedback equipment and new software environments. The haptic rendering techniques that are used below focus on the fact that there is a large database of objects which are graphically surface rendered with polygons, and therefore a surface technique is used to compliment these well established graphical methods.

In our laboratory, we use a PHANToM (Trademark), as shown in Figure 1, to provide the force feedback. To render the underlying models visually, we use Silicon Graphics equipment and virtual reality software [5], developed under the EIGEN (Exponential Improvement in Graphics, Engineering, and Networking) project. The PHANToM is a ground based, serial device that provides 3 degrees of freedom for tracking and force feedback on a single point. The EIGEN virtual reality software provides a convenient framework for drawing conventional graphical objects and for interacting with the user in more traditional ways. As in most graphical systems, objects are rendered with polygons, which are individually drawn and shaded to produce the visual scene.

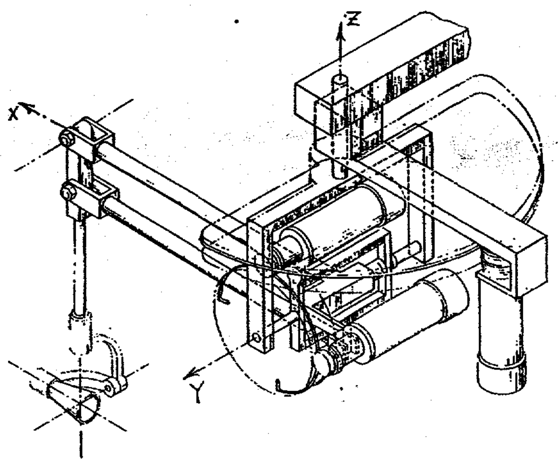

Figure 1: The PHANToM haptic interface [4] (printed with permission).

This work was supported by the United States Department of Energy under Contract DE-AC04-94AL85000

The algorithm first focuses on determining if the user point, or cursor, has touched an object, which requires a collision detection algorithm. Then forces must be created based on penetration depth and surface position, including interpolation for smooth corners.

Graphically, a Bendable Polygon technique is used along with springs and dampers for deformations, which are slave to the haptic process. This makes the graphical deformations happen naturally and easily as opposed to a method in which the graphical and haptic data sets are separate, yet overlap. In addition, this method allows for relatively easy dynamics modeling such as motion or permanent deformations. These algorithms are presented below, and an implementation of them is described together with results and suggestions for further research.

2. Background Methods

Much of the previous haptic rendering research has used methods that work well with either continuous vector fields or simple geometric objects.

Continuous vector fields, for example electric potential, can be modeled by mapping the electric force vector directly to the mechanical force felt by the user. In this way, a user can 'feel around' through the space permeated by the electrical (or other vector) field.

Massie [3] has modeled the forces associated with touching simple geometric surfaces (a wall, for example) by using a force function, **F** = k**X**, where **F** is the force vector, k is a constant, and **X** is a vector normal to the surface, with a magnitude proportional to the depth of penetration into a virtual surface. The surface can be made to feel stiffer by adding a viscous damping term to the force equation so that **F** = k**X** + b**V** [2], where **V** is a velocity vector associated with the user point, and b is another empirical constant.

A box felt from outside its boundaries (Figure 2) presents a slightly more difficult problem. The direction of the force is dependent on the entry path as shown in Figure 2a. One solution for the problem [3] is shown in figure 2b, where the rectangle is split into four different areas, each with an associated direction.

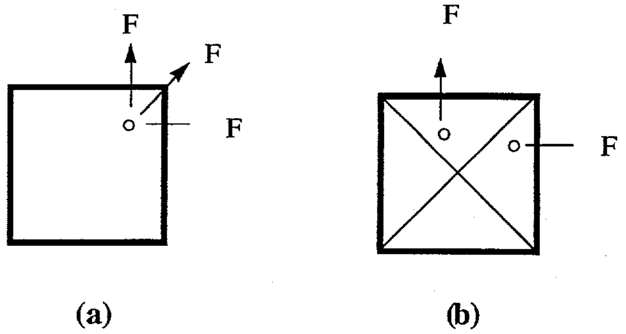

Figure 2: Vector field approach to modeling forces on a square. (a) It is not clear which of the forces is appropriate. The force is path dependent. (b) A vector field solution. By splitting the square up into fourths (in 3D there are eight pyramid shaped pieces) an appropriate force is always presented.

Spheres can be modeled so that the force is always in a direction away from the center, with a magnitude determined by the distance from the center.

All of the above vector field approaches have limitations in that they are specific to particular situations or apply to relatively simple geometric shapes. Also, thin objects can be unintentionally penetrated through with such methods. It is conceivable that one might take geometric primitives, and add them to make more complex shapes, but in general this does not always lead to the correct force [6].

Thus, there is a great need for a more coherent approach to haptic rendering and modeling interactions with complex objects [2], in which a larger base of haptically renderable objects can be obtained. Although one might take a volumetric approach, surface approaches take advantage of a large database of objects that already exist.

Zilles and Salisbury [6] use such an approach. They have modeled rigid polygonal surfaces using a technique known as the "god-object" method after Dworkin and Zeltzer. The constraint-based god object method of Zilles allows a user to intuitively control a point probing a virtual object while preventing the point from penetrating the object. However, the forces and surface deformations associated with just such penetrations in non-rigid objects are the subject of this research and are described below.

In general, the haptic forces will be the result of a combination of heuristics coupled with some collision detection algorithm. Fortunately, the complicated collision detection schemes needed for the general intersection of two arbitrary shapes is not required because of the point interaction paradigm -- collisions need only be found with a single user point.

3. Rendering Virtual Objects

The algorithms described are meant to be used with any arbitrary data set consisting simply of vertex points. However, data sets must be comprised of only triangles, must be enclosed, and each triangle must be touching three others. In general, these characteristics are common, and are often easily achievable in any case by simply separating polygons into smaller, triangular polygons. Each polygon and each vertex is assigned a number for keeping track of interactions.

3.1 Force Generation

The first issue encountered with a surface based approach is collision detection, but is simplified by the fact that collisions are only found for a single point, the cursor. The simple collision detection algorithm used increases computations linearly with the number of polygons, and although there is a vast literature on collision detection, the algorithm used here is effective for these purposes and can be expanded for large data sets as described in section 4.1.

After a collision is detected the forces must be presented. The forces are in the normal direction and are proportional to the penetration depth into an object. The directions are interpolated at edges as described below. When the penetration depth of the active polygon becomes negative, the cursor has left the object and the forces are discontinued.

An initial problem is encountered because of the arbitrary nature of the data set. Because it is desired that only vertices, in any order, need be specified to touch and see the data set, the outward directions of the polygons are also arbitrary. This is particularly a problem when trying to interpolate between adjoining polygons for smooth edges.

To solve this ambiguity an array is created which contains a 1 or a -1 for each polygon, depending on whether the normal points outwards or inwards, respectively. To get a value in the array, a point, point 'A', is projected from within a specific polygon in the normal direction of that polygon to a sphere that encloses all of the polygonal objects, to give point 'B' (Figure 3).

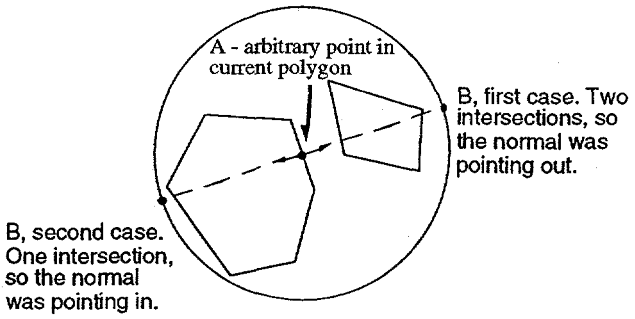
A - arbitrary point in current polygon
B, first case. Two intersections, so the normal was pointing out.
B, second case. One intersection, so the normal was pointing in.

Figure 3: The creation of the *OUT* array. A cross section of two objects and the method for finding a value in the *OUT* array are shown.

Then the number of polygons that are intersected by the segment from point A to point B are counted in the same way that collisions are detected for the cursor (current and previous positions). Because objects are all enclosed, an even number of intersections implies that the original normal was pointing outward and an odd number of intersections means that the normal was pointing inward. By multiplying each normal when it is used by the corresponding value in the array, the normals always point in the outward direction. The information contained in the array could be found within the cycle as it runs, but would require more cycle time. This represents a trade-off between cycle time and memory. In addition, the array (as with any arrays described) can be windowed so that only the infomation concerning polygons in the immediate area is kept in memory.

One would then like to be able to slide the cursor from one polygon to another to touch the entire object. Several problems arise while trying to do this. First, the active or current polygon must always be known so that the direction of the force can be determined. Second, while sliding across polygons, if there is a sudden change in depth or direction of the normal force, then corners feel sharp and distinct even when there is only a slight angle between the adjoining polygons.

Originally, the transition from one polygon to another was accomplished by finding the distance from the cursor to the three edges of the triangle's normal projection. If the cursor crossed the projection then there would be a new active polygon. The problem with this approach was that the distance from the cursor to the current polygon would change when changing polygons, and a small jerk would be felt even when the normal direction was interpolated correctly. This is a problem not found in graphics interpolation because a second variable, depth, is included in the overall interpolation. Also, the distance to the edge of the plane would change, which was used in interpolating the direction of the force. And finally, there can be places within an object that are not in any triangle's projection.

Therefore, a different method was determined in which the distances to 'edge' planes rather than the normal planes were found (Figure 4b). The edge planes are determined from the two triangle vertices in common and a vector which is the average of the normals of the two planes.

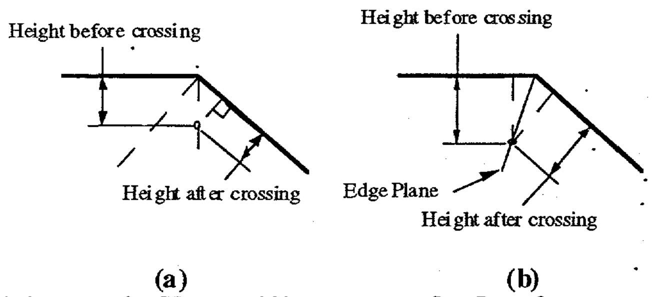

Figure 4: Two different methods of determining when the cursor crosses over to another polygon.

The method shown in Figure 4b makes the penetration depth and distance to the edge planes consistent while sliding to another polygon.

When a change in the active polygon is detected, the new active polygon can be found by finding the other polygon (there are only two triangles that share both vertices) that contains the two vertices in the edge plane that was crossed. Currently this information is stored in an array rather than finding it as the cycle runs, which saves on cycle time.

In addition to consistent distances, the directions of the forces need to be consistent to keep the edges smooth. This was accomplished by interpolating between adjoining polygons in a way similar to Phong shading. When the projection of the cursor into the active polygon comes within a fixed distance from the edge planes, the normal is interpolated and then normalized. Although the distance is currently fixed, it would be more appropriate in the future to make the distance depend on the characteristics of the polygon (i.e. area, side lengths, etc.). When the cursor's projection is near two edges (i.e. near a vertex), the normal is interpolated as shown in Figure 5, point B.

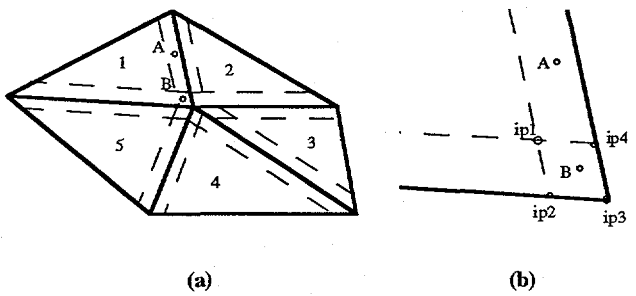

Figure 5: Interpolation of normal direction.

The direction at point 'A' is interpolated (not necessarily linearly) between triangles 1 and 2. At point 'B', the normal is interpolated from 4 points, ip1 through ip4, all of which are normalized. Ip1 is the normal direction of triangle 1. Ip2 is the average of the normals of triangles 1 and 5. Ip3 is the average of the normals of triangles 1 and 2. Finally, ip4 is the average of the normals of all five triangles.Care must be taken to make sure the direction is continuous while interpolating over boundaries.

3.2 Bendable Polygon Algorithm

As the forces are presented to the user, the visual aspects of the virtual object must change also. A compliant object should deform as the cursor moves into it. If the object does not deform, then the cursor can be lost visually within it.

To solve this, the Bendable Polygon technique is used. When the cursor first touches a polygon, the polygon is split into 6 different polygons as shown in Figure 6. The cursor is projected normally to the plane of the active polygon, and then that point is projected to the edges of the triangle, making base points that are the framework for the approach. The 'edge base points' move as the cursor moves, always normally projected to the sides of the current polygon. When the polygons are Gouraud or Phong shaded, the edges look smooth and the polygon seems to bend. The effects of the Bendable Polygon technique decrease with increased graphical detail, but for relatively large polygons, the effect works well and allows for lower levels of detail.

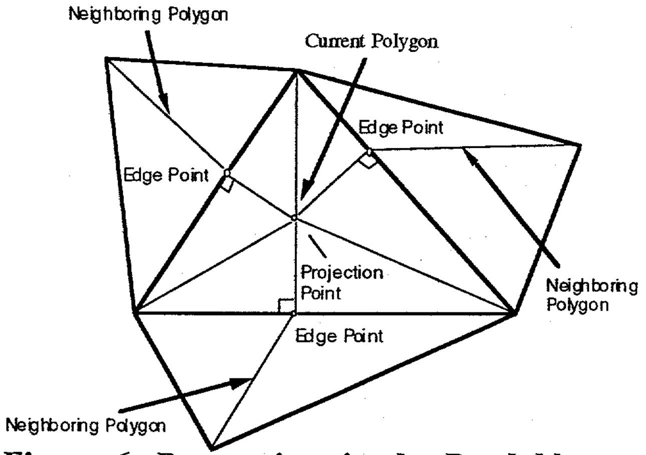

Figure 6: Base points in the Bendable Polygon approach for graphical deformation.

The cursor is then connected by a spring and a damper to each of the vertices in the active polygon and to each of the edge points. The cursor does not have any forces applied to it by the springs, but future work might show interesting results if it did. All six of those points, in turn are connected to the base points shown in Figure 6, also by springs and dampers. When the cursor moves into a polygon, the vertices (open circles) are pulled away from their respective base points (filled circles), making the polygon bend as shown in Figure 7.

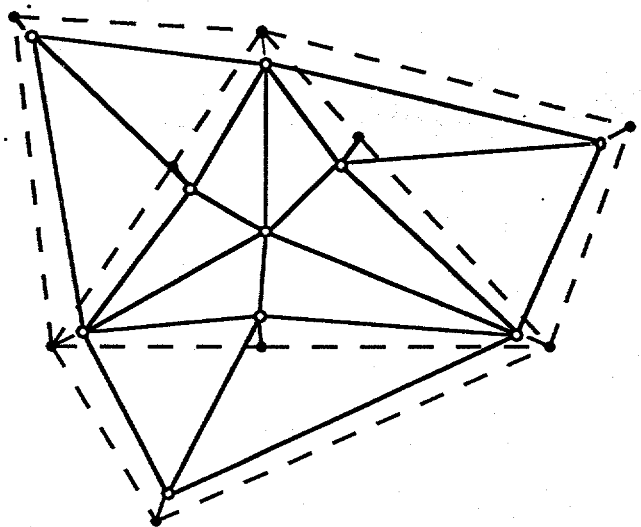

Figure 7: Base Points and Vertex Points in the Bendable Polygon Algorithm.

The dashed lines represent the object while it is not deformed, and the solid lines represent the polygons that the user sees after it is deformed.

As the cursor moves towards an edge, the springs from the edge points to their respective base points lose strength, so the indentation in a polygon remains consistent even as the cursor moves across different polygons. Different spring constants and different levels of strength reduction give different amounts of indentation (i.e. a small indentation on a water balloon as opposed to a larger indentation on a trampoline). The springs from the corner vertices to their base points work similarly, in that they lose strength as the cursor approaches them, so the indentation remains consistent at the corners of the polygons as well.

Then, each of the vertices throughout the object is connected by springs and dampers to each of the vertices touching it, to give an overall ability of large scale deformation. In large data sets, the number of calculations would become extremely large, so springs might, for example, only be connected to vertices in the general surrounding area of the cursor, depending on the application.

Because the edge points split the current triangle, each of the 3 neighboring triangles must be split into two triangles as well so that there is no gap (Figure 6). The EIGEN loop therefore draws 6 triangles in place of the original, 2 triangles in place of each neighboring triangle, and then draws all of the rest of the triangles.

The lighting is accomplished by finding the normals for each vertex, an average of all the normals of the polygons containing that vertex. Then the surface is either Phong or Gouraud shaded according to those normals.

One problem that the Bendable Polygon method presents is that there are extra vertices when the cursor touches a polygon. Because of this, there are more points to interpolate from in the Phong shading, and therefore there can be a slight change in the object's appearance when it is first touched. The problem is worse with relatively large polygons and with graphic materials that are shiny (large amounts of specular reflectance). The problem diminishes with more lighting and more detail (more polygons).

3.3 The Haptic Control Loop

One issue that was encountered while trying to integrate the PHANToM into the EIGEN environment was the differing cycle loops needed for visual and haptic rendering, an often encountered issue with haptic displays. EIGEN runs at a cycle rate of approximately 30 Hz, which is common in graphics, but a much higher rate is necessary for haptic simulation. The sampling rate of the haptic controller must be at least 10-20 times the desired system bandwidth [8], and therefore a cycle rate of at least 1000 Hz is desired.

To solve this problem, a separate process is forked off from the original EIGEN process to control the haptic simulation [7]. This effectively divides the overall visual/haptic renderer into two interacting, but asynchronous tasks, which communicate through EIGEN's shared memory paradigms. The two processes can then run at the differing rates.

4. Additional Issues

An effective haptic model needs other key concepts to be effective, including dynamics, friction, and texture. These effects are an important supplement to the existing algorithm.

4.1 Collision Detection

If the number of polygons in a data set was extremely large, the calculations for the collision detection could become a limiting factor. To solve this, the virtual space could be separated into volumes, and only those polygons within the current volume would need be checked. This would decrease the haptic servo time dramatically and would make nearly any size data set possible if the volumes were created effectively, but with the cost of more memory to describe which polygons are in which volumes. However, only a 'buffering area' (the immediate surrounding volumes and information about polygons within them) would necessarily be stored in memory, as the buffering area could be loaded into memory as needed. This would produce a windowing effect, which is characteristic of this type of haptic rendering because of the point interaction paradigm.

4.2 Texture and Friction

Texture is currently accomplished by simply changing the effective distance (the distance used to create the magnitude of the force) as a function of position using different amplitude and frequency sine waves. Other methods, such as those used in the sandpaper system [11] are being explored.

The incorporation of friction was a more complex problem. When the cursor moves across a surface, the friction force is in the direction opposite the movement. Because the PHANToM's encoders are not continuous (like any kind of tracking, they have a spatial resolution), when the cursor is nearly stopped the force opposite the movement pushes the cursor which in turn creates a friction force in the opposite direction. Because of the relatively high bandwidth of the PHANToM, high frequency vibrations result. This problem was eliminated by making the friction partly dependent on velocity, and by low-pass filtering of the directional changes of the friction.

The magnitude of the friction force was divided into two different areas. The first was Coulomb friction which was dependent on the depth within the object and a coefficient of dynamic friction. The second part of the friction magnitude, viscous damping, was based on the cursor's velocity. The percentage of each of the two different parts gave different effects for surface properties such as roughness or surface viscosity. A "stiction point" [2] was used to model static friction.

One additional issue with friction is that the friction force should only be in the tangential direction, or else the object will feel 'sticky' when pulling away from it (unless one wants a sticky object). By taking only the tangential component of the friction force, and adding interpolation near edges, the cursor can move freely away from the object but will have resistance to sliding across it.

4.3 Dynamics

Several dynamics effects are easily accomplished using the concept of the base points. The first is three degree of freedom (x,y,z) motion. This is accomplished by making the positions of the base points relative to a single point with a given mass. Then the force applied to the cursor (as developed above) is equal but opposite to the force applied to the mass point. The base points could be positioned relative to 3 points rather than 1, and the 3 rotational degrees of freedom could be added, but with the cost of much more complicated dynamics equations. One should note that elastic deformations still occur when the object is moving.

Acceleration is then equal to the force divided by the mass, and velocity and position can be found by integrating over time or by use of standard motion equations. The time variable in these equations was chosen arbitrarily, rather than finding it exactly from the servo rate. This gives more freedom in the 'feel' of an object and gave several surprising effects. When the time variable was decreased, an object felt more massive and felt as if it were moving through a thick liquid.

The effect of making the time variable small gave rise to another feature. When the time variable was made small enough, an object was difficult and slow to move. This feature that is currently being researched by giving each of the base points a mass and letting them move relative to each other. Then, by making the time variable small enough in their motion equations, and limiting to some degree their movement, a permanently deforming model could be achieved. The base points would move according to the springs connected to the vertex points, and the movement of the object could be made to feel like clay which could, for example, lead to virtual sculpting. Some preliminary applications using this technique were successful to a certain degree, but there were some problems when the model was deformed too much. The results were encouraging nonetheless.

Finally, the base points could be programmed to move in other ways so that the model would deform. For example, one might feel a tin can and its elastic/plastic deformation. The can would deform with force from the cursor. The vertices would move away from the base points, and return to their original positions if the forces were not too large. If, however, the forces on the can increased past a certain threshold, the base points could snap into different positions, which would be the can's new position. Finally, to finish the example, the base points could all be positioned relative to 3 points which would control the can's motion through space, and friction and texture could be added for further realism.

5. Results and Further Research

The preliminary results from this study have been promising. The methods used were successful for interacting with objects of various elasticities and shapes. Even very elastic objects with few polygons worked well with the combination of the force representation and the Bendable Polygon approach, as the objects seemed to deform naturally both haptically and visually.

Additionally, there are limits in the computing power needed in this algorithm because of the point interaction paradigm. Because only the immediate area around the user point is necessarily rendered, it is expected that windowing will allow even large data sets to be haptically rendered.

Future research will include further study of permanent deformations, more complicated dynamics modeling, and windowing effects. In addition, Sandia intends to apply this algorithm to specific applications in manufacturing, medical research, and other types of visualization.

6. Conclusions

This algorithm, although still developmental, has been used effectively to touch several data sets. Because of the current polygonal nature of virtual reality and graphics, it is expected that this method will continue to support other efforts at Sandia without significant changes in procedure or philosophy.

Each of the aspects of the overall algorithm could be used separately in other algorithms, or likewise, certain aspects could be changed in this algorithm. However, the methods shown have worked well together and hopefully will continue to grow.

7. Acknowledgments

I would like to thank Tom Caudell, George Davidson, Arthurine Breckenridge and John M. Linebarger for their help in writing this paper.

8. References

[1] J. Foley, A. van Dam, S. Feiner, and J. Hughes, *Computer Graphics, Principles and Practice*, Addison-Wesley Publishing Co., Reading, MA, 1990.

[2] K. Salisbury, D. Brock, T. Massie, N. Swarup, and C. Zilles, "Haptic Rendering: Programming Touch Interaction with Virtual Objects", ACM Symposium on Interactive 3D Graphics, Monterey, CA, 1995.

[3] T. Massie and K. Salisbury, "The Phantom Haptic Interface: A Device for Probing Virtual Objects", Proceedings of the ASME Winter Annual Meeting, Symposium on Haptic Interfaces for Virtual Environment and Teleoperator

Systems, Chicago, IL, 1994.

[4] SensAble Devices Inc., “The PHANToM” literature from SensAble Devices Inc. 225 Court St., Vanceburg, KY 41179.

[5] C. Maples and C. Peterson, “MUSE, A functionality-Based, Human-Computer Interface”, The International Journal of Virtual Reality, Vol. 1, No. 1, pp. 2-9, Winter 1995.

[6] C. Zilles, J. Salisbury, “A Constraint-based God-object Method for Haptic Display”, IEEE/RSJ International Conference on Intelligent Robots and Systems, Human Robot Interaction and Cooperative Robots, 1995.

[7] T. Anderson, “Integration of the PHANToM Haptic Interface with the EIGEN Virtual Environment”, senior design lab project, University of New Mexico, Albuquerque, NM, 1996.

[8] P. Buttolo, B. Hannaford, B. McNeely, “Introduction to Haptic Simulation”, Tutorial 2A, IEEE/VRAIS tutorial notes, March, 1996.

[9] M. Mortenson, *Computer Graphics Handbook*, Industrial Press Inc., New York, NY, 1990.

[10] R. Klatzky and S. Lederman, “Toward a Computational Model of Constraint-Driven Exploration and Haptic Object Identification”, Perception, Vol. 22, pp. 597-621, 1993.

[11] M. Minsky, M. Ouh-Young, O. Steele, F. Brooks, and M. Behensky, “Feeling and Seeing: Issues in Force Display”, Computer Graphics, Vol. 24, No. 2, pp. 235-243, 1990.

[12] F. Brooks, M. Ouh-Young, J. Batter, P. Kilpatrick, “Project GROPE - Haptic Displays for Scientific Visualization”, Proceedings of SIGGRAPH ‘90, Dallas, Texas, August 6-10, 1990.

[13] L. Fabiani and G. Burdea, “Human Interface Using the Rutgers Master II Force Feedback Interface”, IEEE/proceedings of VRAIS, 1996.