Extending the Friction Cone Algorithm for Arbitrary Polygon Based Haptic Objects

N. Melder W. S. Harwin

Department of Cybernetics,
School of Systems Engineering,
University of Reading,
Reading, RG6 6AY
United Kingdom

Email: N.Melder@rdg.ac.uk W.S.Harwin@rdg.ac.uk

Abstract

Most haptic environments are based on single point interactions whereas in practice, object manipulation requires multiple contact points between the object, fingers, thumb and palm. The Friction Cone Algorithm was developed specifically to work well in a multi-finger haptic environment where object manipulation would occur. However, the Friction Cone Algorithm has two shortcomings when applied to polygon meshes: there is no means of transitioning polygon boundaries or feeling non-convex edges. In order to overcome these deficiencies, Face Directed Connection Graphs have been developed as well as a robust method for applying friction to non-convex edges. Both these extensions are described herein, as well as the implementation issues associated with them.

1. Introduction

Physics simulation in haptic environments requires that manipulated objects have a number of physical properties as in the real world. These include mass and inertia for direct object manipulation as well as properties such as surface texture and frictional characteristics. Work done by the authors has led to techniques to allow for the translation and rotation of multi-contacted objects [1] where mass and inertia are modelled, as well as means for modelling friction [2]. The Friction Cone Algorithm that was developed is a computationally simple means of modelling friction in haptic rendering applications, however, it suffers from two drawbacks, namely that it had no means of transitioning polygon boundaries or feeling non-convex edges. A solution to this problem is to use Face Directed Connection Graphs to store information about which polygons are directly attached to the currently touched polygon.

Successful implementation of a physics simulation requires:

- the identification of collisions.
- an estimation of external forces resulting from the collisions (including frictional forces).
- an appropriate response to the residual forces.

There are many collision detection algorithms and each tends to have certain advantages in different situations. In haptic program design the deciding factor in choosing the best collision algorithm is always the speed of calculation to determine whether a collision has occurred as this information is computed within the inner, haptic (high speed) loop. Two popular collision algorithms are OBBTree[3], and H-Collide[4] although the information returned from a polygon/point collision algorithm is all that is required for modelling friction via the friction cone algorithm.

Concerning collisions involving a haptic device, once a collision has been identified then the appropriate forces must be calculated. A popular method that avoids object push through is described by Zilles and Salisbury [5] where two points are used to track the position and response of a haptic device when in contact with a surface. The haptic interface point is used to describe the endpoint location of the physical haptic interface as sensed by the encoders. A second conceptual point, the god-object, is used to track the history of the contact by locating the position on a surface polygon where forces should be directed. Conceptually this god-object slides along the surface polygons such that the distance to the haptic interaction point is always minimised. A notional spring is then used to compute the force that is to be applied to the haptic interface point such that the person's

finger is pushed towards the god-object. While the haptic interface is in virtual free space the haptic interface point and the god-object are collocated. This approach will give the normal force to the most appropriate surface on the object that has been touched. Lateral friction forces are then usually superimposed onto this normal force based on the detected velocity of slip.

Classical friction models typically have the response as shown in figure 1. Additional characteristics are also evident that are not shown in the figure. These include stick-slip motions where a limit cycle occurs at low velocities, presliding displacement where, before breakaway, the friction appears as a stiff spring, and frictional lag which is responsible for a delay between the velocity and friction variables.

The friction cone method for determining friction [2], can be used to model the classical friction model.

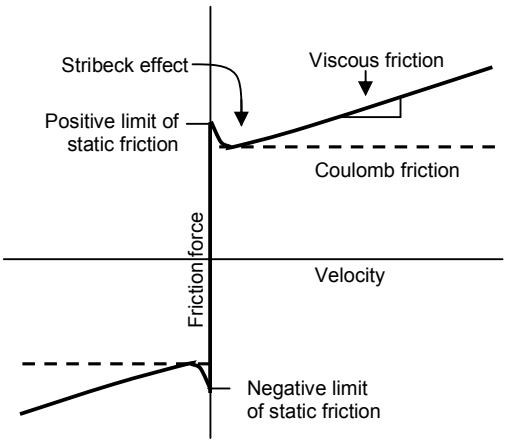

Figure 1: Friction Force vs Velocity

2. The Friction Cone Algorithm

A simple adjustment to the god-object algorithm allows us a better technique to manage friction. A friction cone can be arranged at the haptic interaction point, oriented in the direction of the normal of the contacted surface (see figures 2 and 3). The intersection of this cone with a planar surface on an object (a polygon) will define a friction circle since the surface is normal to the cone [2].

Whereas in Zilles' paper, as the haptic interface point moves so does the god-object, the approach used here only moves the god-object if the god-object lies outside the circle of friction. To calculate the size of the friction circle we use the depth of penetration of the haptic interface point in relation to the surface as an indication of force and the coefficient of friction (μ) that has been previously assigned to the penetrated polygon. i.e. radius of friction circle = μ * depth. Since the coefficient of friction remains constant the size of the friction circle is proportional to the depth of the penetration. It is possible to have different frictional properties for different objects simply by having a different coefficient of friction assigned to it so, for example, a surface has a low frictional coefficient then, for a given penetration depth, it will have a smaller friction circle.

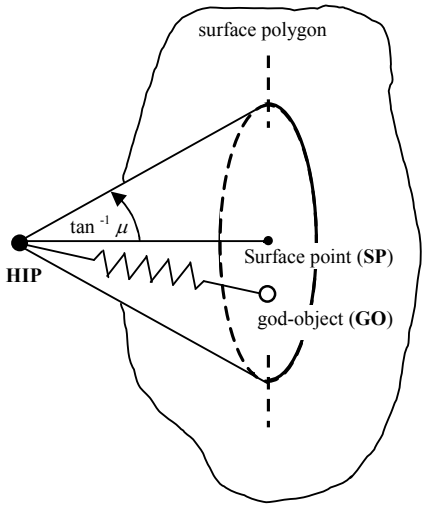

Figure 2: God-object inside friction cone remains unchanged

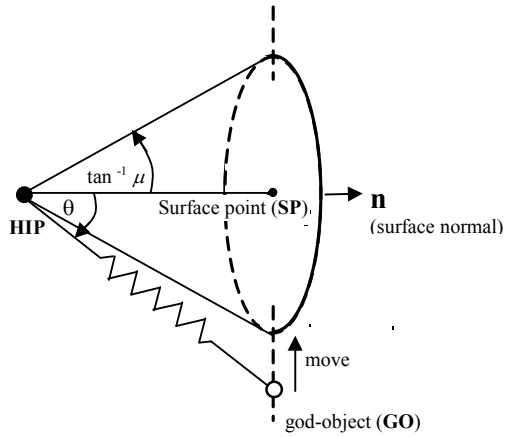

Figure 3: God-object moves to edge of friction cone – note the surface polygon is omitted for clarity

The Friction Cone Algorithm is composed of the following steps and is active whilst the haptic interface point is inside the surface. It assumes that the god-object (GO) has already been placed and now needs updating.

1. First, calculate depth of penetration *d* of the haptic interface point (HIP) below the surface of the polygon. ie. $d = (\mathbf{HIP} - \mathbf{GO}) \bullet \mathbf{n}$.

2. Hence, calculate the location of the surface point (SP). The surface point is the defined as the minimised distance between the HIP and the contacted surface. ie. $\mathbf{SP} = \mathbf{HIP} + d\mathbf{n}$.
3. A circle can then be considered as the intersection of the friction cone with the surface polygon. The radius of this friction cone circle is given by $R = d\mu$ (where μ is the appropriate friction coefficient for the surface).
4. The distance between the surface point (SP) and the current god-object (GO) is then given by $r = |\ \mathbf{GO} - \mathbf{SP}\ |$. The next stage is to compare this distance to the radius of the friction circle and update the god-object if necessary to the edge of the circle.
5. If the god-object is outside the friction circle then update the god-object position to be on the perimeter of the friction circle. ie.

$$\mathbf{GO_{new}} = \mathbf{SP} + R.\frac{(\mathbf{GO} - \mathbf{SP})}{r}$$

   Otherwise, leave the god-object in place.
6. The response force can now be calculated based upon the vector from the HIP to the god-object and will be proportional to the surface stiffness.

Thus when the GO is outside the friction cone it will 'jump' to the closest point on the circumference of the friction circle and provides the equivalent to static friction. It is relatively straightforward to modify this algorithm to model both dynamic (coulomb) and static friction. It is necessary to note which of the two states (static or dynamic) an individual contact point is occupying, it will either be slipping on the surface, in which case the state is 'slipping' and the coefficient of dynamic friction is used in calculating the friction circle radius (μ_d), or it is 'not slipping' in which case the coefficient of static friction (μ_s) is used in calculations. Figure 4 gives a state transition diagram for these two states. Transition between these two states is controlled by a comparison between the angle at the HIP between the god object and the surface point (θ).

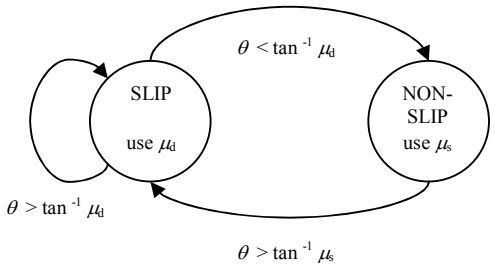

Figure 4: State transitions for dynamic (coloumb) and static friction conditions

3. Object Manipulation

Where an object needs to be handled with a multi-finger grasp, it is now relatively simple to use the friction cone algorithm for each point of contact between a finger and the object. The strength of this algorithm is that now a sum of forces/torques on the object can be easily calculated and this residual force/torque used to calculate the object acceleration and hence, by integration, its velocity and position/angular velocity and orientation. This then allows arbitrary grasps to be made on an object and the stability of the grasp determined by the friction cone algorithm. Details of these calculations are given in [1].

4. Arbitrary Shaped Objects using Face Directed Connection Graphs

Previous haptic surface rendering algorithms transition polygon boundaries by determining where the HIP is in relation to the surface polygons, edges and vertices [5][6]. However, in these implementations it is always possible to determine the location of the HIP from the location of the god-object and vice-versa. Since the friction cone algorithm does not have this property it is necessary to use a novel approach to boundary crossing.

Face Directed Connection Graphs store information about how faces are attached to each other ie. for any given face, it is possible to quickly determine all the connecting faces. Using a Face Directed Connection Graph and by examining the position of the god-object as it moves along a surface, it is possible to determine when the god-object crosses a plane defined by a connected polygon's plane equation. This is done by pre-computing and storing the Voronoi like regions [7] associated with the mesh in the Face Directed Connection Graph and determining when the god-object has traversed from one region to another.

Figure 5 shows the Face Directed Connection Graph (FDCG) for a simple polygonal cube made up of six faces. By storing the D value (perpendicular distance from the plane containing the polygon to the object frame origin) for each face in the face structure (effectively turning it into a plane since each face also stores its normal vector) and by incorporating the vertices that are shared between the faces it is possible to use FDCGs in a god-object based system [3] to simplify the detection of edge crossing. Since the vertex data of the connected faces is stored in the graph, finding the equation of any polygon edge is a simple matter. It can also be seen that each loop of the graph can be made to contain the same vertex number. Conversely, given three faces that are connected together (ie. a corner) it is easy to see which vertex they all share. FDCGs can be pre-computed directly from the mesh data where the normal and D value for each face can be easily calculated.

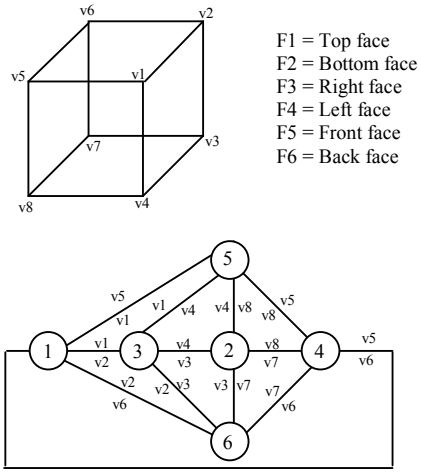

Figure 5: A polygon cube and its associated face directed connection graph

The FDCG is comprised of nodes, connections and corners where a node is equivalent to a face, a connection is equivalent to the edge between two connected faces and a corner is equivalent to a vertex.

The node contains the following information:

FacePlane	- the normal of this face
Connections	- array of connections to this node

The connection contains the following:

SharedVertex[2]	- the two shared vertices
Nodes[2]	- the two connected node
EdgeVector	- the normalised vector from SharedVertex[0] to SharedVertex[1]
ConType	- connection type (is either convex, concave or coplanar)
VoronoiPlane[2]	- see below for explanation
EndPlane[2]	- see below for explanation

The corner contains the following:

Vertex	- the 3D point in space
Connections	- array of connections that share this corner

The Voronoi planes in the connection are associated with the two connected nodes such that VoronoiPlane[0] is the Voronoi plane for Node[0] and VoronoiPlane[1] is the Voronoi plane for Node[1]. The Voronoi plane is simply a plane that passes through the two, shared vertices with a normal perpendicular to the normal of the associated node.

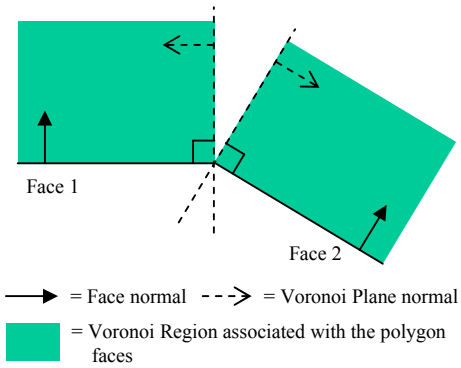

Figure 6: Voronoi planes around an Edge

The Voronoi planes are calculated as follows:

```
vertex1 = connection->SharedVertex[0];
vertex2 = connection->SharedVertex[1];
vertex3 = Node[0]->faceNormal + vertex2;

vector1 = vertex1 - vertex2;
vector2 = vertex1 - vertex3;

normal  = cross(vector1, vector2);

voronoiFaceNormal[0] = Normalize(vNormal);
voronoiD[0] = -dot(vertex1, vNormal);
```

Similarly, the end planes are associated with the two shared vertices. Each of the shared vertices lie on an end plane and the normal is either the EdgeVector or the negative EdgeVector of the connection.

When computing the Voronoi and end planes, it is important to ensure that a consistent winding is used in the mesh data structure and that this is translated into the connections in the FDCG otherwise the direction of the computed normals for the Voronoi planes may be incorrect leading to unpredictable transitions between the afflicted polygons.

If data duplication is not desired and both the visual and haptic mesh are identical, it is possible to easily modify the data structures required for graphical rendering to include all the necessary components of the FDCG.

5. Crossing polygon boundaries

The following state diagram shows the manner in which the god-object can transition over the surface.

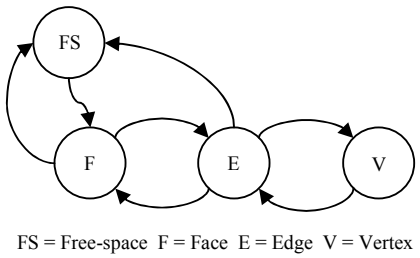

Figure 7: State transition diagram of how the god-object moves across the surface of an object

In order to determine whether a plane has been crossed, it is necessary to store the previous location of the traversing point. The distance to the plane is then calculated with the current location and the previous location and if there is a change in sign then a plane has been crossed. ie.

```
CurrDist = (CurrPos dot PlaneNormal) + PlaneD
OldDist  = (OldPos  dot PlaneNormal) + PlaneD

If (CurrDist * OldDist < 0) then plane has
                                been crossed
```

where dot performs a dot product on the two vectors. Sections 5a to 5g give the transitions required for figure 7. Figure 7 excludes some transitions (such as vertex to free space and vertex to face) because these state transitions should never occur. Instead, an intermediate transition should occur ie. in order to move from a vertex to free space, a vertex to edge transition followed by an edge to face transition, followed by a face to free space transition should occur. Section 7 Implementation Issues details this further.

5a. Free space to Face

This is determined by the collision detection algorithm and will define the initial active face.

5b. Face to Free space

A face to free space transition can be determined by the distance of the haptic interaction point (HIP) to the face plane; if the distance is positive, then the HIP has transitioned out of the plane and is thus in free space.

5c. Face to Edge

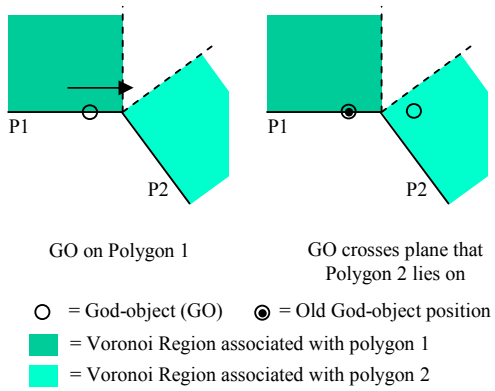

Figure 8: God-object crossing from a face to an edge

Once the god-object has been set, it is necessary to determine whether the god-object has moved out of the Voronoi region associated with the face. This can be achieved by comparing the distance of the old god-object to the Voronoi planes stored in the connections that are attached to the currently active node. If there is a change in sign, then that connection has been transitioned and the corresponding edge is made active (see section 6 for using the friction cone algorithm on edges). In essence, the god-object has moved out of the face Voronoi region into an edge Voronoi region.

The god-object is then repositioned on the edge where the transition occurred.

5d. Edge to Face

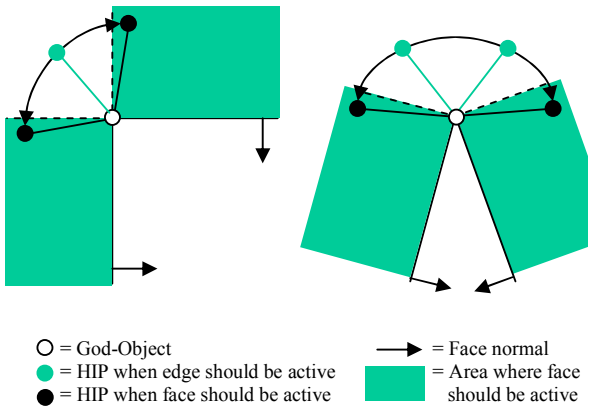

Figure 9: HIP entering a shaded region in this non-convex example causes an edge to face transition to occur

Edge to face transitions need to be determined every haptic update since it is based upon the position of the haptic interaction point (HIP) and not the god-object.

An edge to face transition will occur when the HIP moves into a face Voronoi region, at which point the corresponding node is made active. However, if the current edge is convex, it may be possible for the HIP to be behind both Voronoi planes stored in the connection. In this case, the face plane that the HIP is closest to is made active.

The god-object does not need to be repositioned.

5e. Edge to Free space

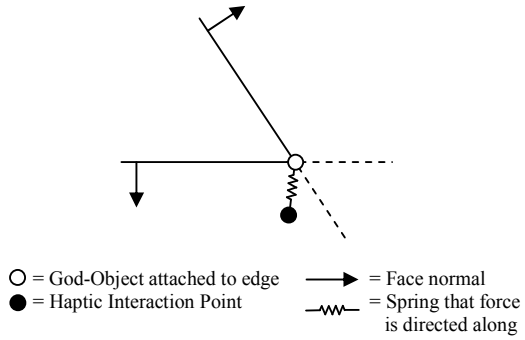

Figure 10: Transitioning from a convex edge into free space

Figure 10 shows the position of the haptic interaction point and the god-object when the edge is first made active (ie. the edge has been made active but no transitions have yet been determined). Edge to free space transitions occur only if the edge is acute and convex and can be tested by checking whether the edge is convex and that the HIP is in front of one connected face plane and behind the other face plane. Although it appears possible that the HIP could be in front of both face planes at the same time, this will never be possible since the HIP must have been behind one of the face planes before it made a face to edge transition (otherwise a face to free space transition would have occurred).

5f. Edge to Vertex

An edge to vertex transition is defined as when the god-object moves off the end of the edge (ie. past one of the shared vertices), but not onto a face. Determining if the god-object is in front of either of the associated end planes tests this transition type. If it is, then the corresponding vertex is made active and the god-object is repositioned at the active vertex.

5g. Vertex to Edge

This transition will occur if the HIP moves into the edge Voronoi region, ie. it is behind both face planes that are stored in a connection connected to this corner.

6. The Friction Cone Algorithm when on an Edge

A slight modification to the friction cone algorithm is required in order to allow it to work on an edge. Whereas in the face case, the surface point is defined as **SP** = **HIP** + d**n** (see above), in the case of an edge there is no normal to use! The equations shown below give a point on the edge that is perpendicular to the HIP so this is used to define the surface point instead.

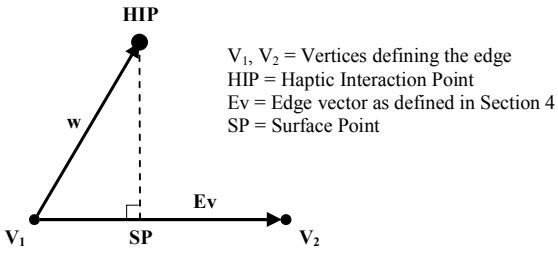

Figure 11: Calculating the surface point when on an edge

$$\mathbf{Ev} = \mathbf{V}_2 - \mathbf{V}_1$$

$$\mathbf{w} = \mathbf{HIP} - \mathbf{V}_1$$

$$b = \left(\frac{\mathbf{w} \cdot \mathbf{Ev}}{\mathbf{Ev} \cdot \mathbf{Ev}} \right)$$

$$\mathbf{SP} = \mathbf{V}_1 + b\mathbf{Ev}$$

$$\mathbf{HipToEdge} = \mathbf{SP} - \mathbf{HIP}$$

7. Implementation Issues

Implementing the algorithms as described may cause undesirable effects while transitioning coplanar and convex polygon boundaries since the god-object is always placed on the edge during polygon-edge-polygon movement. This can be easily remedied by recursively changing the state (ie. polygon, edge, vertex active) until the god-object does not need to be moved. This requires that the original god-object is not updated until the proposed god-object comes to rest. By implementing the algorithms in this way, coplanar polygons will always feel smooth, otherwise they feel as though they have a 'sticky' ridge where the polygons are joined. Similarly, convex polygon crossings feel more realistic instead of having a 'sticky' edge. The disadvantage of this recursion is that the

code is not deterministic, however, in practice, the computational burden of this recursion is low.

The advantages of using Voronoi regions to determine transitions are two fold: precision errors are no longer problematic and the small plane problem is removed. Precision errors occur when the god-object isn't placed exactly on the surface of a polygon / edge (ie. it is either above or below the surface). Previous edge transitioning methods that were developed exhibited problems when this was the case which would result in the god-object being incorrectly placed on edges or surfaces. This caused the god-object to become fixed on an edge or be attached onto the wrong side of the plane of a polygon (ie. on the portion of the plane that is in free space). The described methods don't suffer from these problems since the god-object is explicitly placed every time a transition occurs.

The small plane problem describes the situation where the god-object moves across two edges. This could potentially result in the god-object being attached to the portion of the plane that is in free space. Using Voronoi regions, this problem does not exist since the transition occurs when the god-object moves outside of the Voronoi region of the active polygon, edge or vertex.

Haptic resolution/update errors manifest themselves whenever a transition occurs that has not been defined in the transition state diagram (figure 7). They occur because the resolution/update of the device allows for a jump between two normally impossible states (eg. god-object transitions from a vertex to free space, instead of from a vertex to free space via an edge and a plane as would happen in a continuous system) and are easily remedied by the addition of these extra state transitions. The updated state transition diagram (figure 12) is shown below where the shaded lines are the additional/modified state transitions.

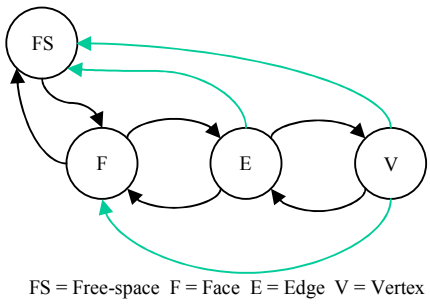

Figure 12: State transition diagram with the erroneous states added

For transitions into free-space it is necessary to examine the position of the HIP against the face planes; if the HIP is behind no faces then it is in free space. This allows for vertex to free space and for concave edge to free space transitions at minimal cost.

Vertex to Face transitions must occur when the HIP moves directly from a corner Voronoi region directly into a face Voronoi region. To achieve this, it is necessary to first determine which nodes share the corner and then test if the HIP lies inside a face Voronoi region. It is possible to store the connected nodes in the FDCG (in the corner structure) for this test, however, since this transition occurs rarely, calculating the connected nodes at run time adds little processing cost to the state. This can be further optimised by caching the connected nodes the first time this state is entered.

The website http://www.cyber.rdg.ac.uk/ISRG/haptics/ contains MPEGs of multi-finger haptic manipulation of a sphere and a cube using the algorithms described herein and in [1][2].

8. Conclusions

The friction cone algorithm and the associated face transition algorithms provide a mechanism to model arbitrary rigid bodies in a haptic environment that allows both the ability to model friction and subsequently to lift and manipulate the body in an appropriate physics based world (for example, gravity can be set to values appropriate to the earth, the moon, or zero). The algorithms are efficient since only the features (faces or edges) connected to the active polygon / edge / vertex are used and so the polygon count of the model being rendered is not a factor in the processing time required.

It is also possible to apply the algorithms described to parametric surfaces with minimal modification since the friction cone algorithm returns a direction and magnitude of the force required for display. The only change that is required is the determination of the surface normal that is used. This has been tested against simple parametric objects (spheres, cylinders, cubes and planes) although complex parametric objects (made up of multiple surfaces) have not currently been investigated.

These algorithms can also be used with deformable meshes provided that the appropriate Voronoi planes are updated when the object deforms.

9. Acknowledgements

The authors are pleased to acknowledge support from the EPSRC project “Haptic cues in multi-point interactions with virtual and remote objects” (GR/R10455/01) for this work.

10. References

[1] N. Melder, W. S. Harwin and P. M. Sharkey, Translation and Rotation of Multi-Point Contacted Virtual Objects, *Proceedings of Eurohaptics Conference*, 2003

[2] W. S. Harwin and N. Melder, Improved Haptic Rendering for Multi-Finger Manipulation Using Friction Cone based God-Objects, *Proceedings of Eurohaptics Conference*, 2002

[3] S. Gottschalk, M. C. Lin and D. Manocha, OBBTree: A Hierarchical Structure for Rapid Interface Detection, *Proceedings of ACM Siggraph,* 1996

[4] A. Gregory, M. Lin, S. Gottschalk and R. Taylor, H. Collide: A Framework for Fast and Accurate Collision Detection for Haptic Interactions, *Proceedings of IEEE Virtual Reality Conference*, 1999

[5] C. B. Zilles and J. K. Salisbury, A Constraint-based God-object Method for Haptic Display, *Proceedings of International Conference on Intelligent Robots and Systems*, 1995.

[6] C. Ho, C. Basdogan, M. A. Srinivasan, Efficient Point-Based Rendering Techniques for Haptic Display of Virtual Objects, *Presence* 8(5) Oct 1999 pp 477-491

[7] A. Schinner, Features and Voronoi regions, Last known web address: http://octopus.th.physik.uni-frankfurt.de/ ~schinner/algorithm/node13.html, May 1995