

Chapter 4

Extensions to the God-object method

The god-object method in itself only handles shape, but it provides a solid framework for surface effects. Rather than just computing a feedback force, the god-object method computes a location on the virtual object’s surface where the user can thought to be touching. Surface effects are dependent on the surface location, so having the god-object’s location greatly simplifies their implementation.

This chapter explores some common surface effects and how they can be incorporated into the god-object architecture.

4.1 Friction

The most important haptic effect beyond geometry is friction. Without friction a user is mostly limited to probing, and exploring a virtual environment; with friction one can manipulate virtual objects and change the virtual environment in a straight-forward manner. Friction can also be used to give the feel of doing work; this friction algorithm has been used to simulate the sensation of driving screws with a screwdriver haptic interface (Figure 4-1).

The friction that we require is a static friction; viscous drag is not sufficient. In a virtual environment, a user should be able to support an object using only the friction force without the object slipping. Since viscous friction can only provide a force when there is a relative velocity, it does not fit our needs.

Friction models with stiction, in general, have two states of contact: stiction and kinetic friction. In the stiction state the user has made contact with an object but has not provided enough tangential force to “break away”; when enough force is applied a transition is made to the kinetic friction state. In the kinetic friction state a force is applied in a direction to impose the direction of motion. All friction forces are applied in directions tangential to the normal force.

For a general implementation of the static state:

$$\Delta x = (x_{stiction-point} - x_{haptic-interface}) \quad (4.1)$$

$$\Delta x_{tangential} = (\Delta x - (\Delta x \cdot \hat{N}) \cdot \hat{N}) \quad (4.2)$$

$$F_{friction} = k\Delta x_{tangential} + c\Delta \dot{x}_{tangential} \quad (4.3)$$

where k is the stiffness, c is the viscosity, and \hat{N} is a unit vector normal (outward pointing) to the surface. Equations 4.3 and 3.3 can be used to find the the total feedback force (Equation 4.4).



Figure 4-1: A 1 degree-of-freedom active screwdriver for driving virtual screws

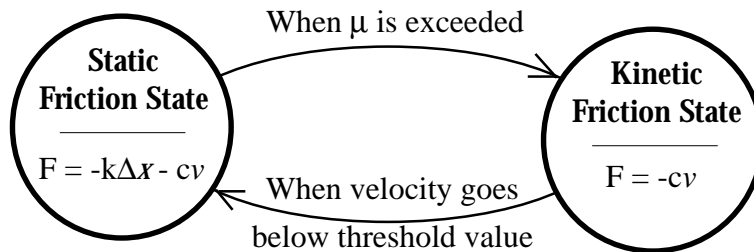


Figure 4-2: State transition diagram for viscous friction model.

$$F_{total} = F_{normal} + F_{friction} \quad (4.4)$$

The transition from static to kinetic friction should be made when:

$$F_{friction} \geq \mu F_{normal} \quad (4.5)$$

where μ is the coefficient of friction.

One published friction model uses viscous friction in the kinetic state. [17] Their two state friction model makes the transition back to the stiction state at a certain velocity threshold (Figure 4-2). Since position is discretized it is difficult to accurately determine the direction of the velocity vector when the velocity is small. By using viscosity for kinetic friction they are assured that only small forces will come from small velocities so any errors in calculation of velocity direction should go unnoticed.

We've chosen to model friction using a Coulomb (velocity independent) friction model. The

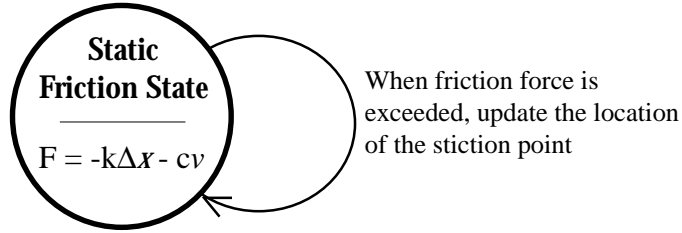


Figure 4-3: State transition diagram for coulomb friction model

magnitude of the kinetic friction force is only a function of normal force and coefficient of friction, μ . The friction force is applied to oppose the direction of motion. At small velocities we will have poor discrimination of velocity direction but the force magnitude can still be large; this will cause vibration if we use a traditional two-state model. Since only the direction of the velocity vector is needed, and not the magnitude, our current position with respect to the stiction point will give us a more stable and accurate direction for the friction force.

This algorithm can be visualized as a convict with a ball and chain around his ankle. As long as convict stays within the length of the chain the ball is motionless¹. When the convict tries to go beyond the length of the chain, the ball dragged behind.

The coulomb friction implementation really only has one state (Figure 4-3). The new stiction point is placed on the line between the current position and the old stiction point. It is placed at such a distance from the current position so that the force on the user is equal to maximum friction force (Equation 4.6). This method really eliminates the dependence on the velocity signal; we are no longer susceptible to the noise in the velocity signal which could cause vibration.

$$\Delta x_{\text{tangential}} = \frac{\mu}{k} F_{\text{normal}} - \frac{c}{k} \Delta \dot{x}_{\text{tangential}} \quad (4.6)$$

4.1.1 Friction with Slip

This implementation is very stable, but in this incarnation it is impossible for a user to tell whether he/she is slipping relative to an object or not. If an object is being supported from the sides, the friction forces are constant and equal to the gravitational force on the object. When slipping is taking place, we've specified that a constant force is applied to the user as a criterion for placement of the stiction point. These two sensations are tactilely the same.

In the real world there is usually some vibration associated with slipping. To mimic this in our virtual model we use two different coefficients of friction. One μ is used for checking to see if we should update the location of the stiction point, and another, slightly lower μ is used for calculating the new location of the stiction point. This means that each time a new stiction point is placed the friction force is lower, and a bit of distance will have to be traveled to break away again. When a constant relative velocity is specified we get the saw-tooth wave seen in Figure 4-4. When the stiction point is moved, the new position is calculated using a coefficient of friction smaller than the one used to check if the maximum friction force has been exceeded. When a constant velocity is specified a vibration is felt due to the saw-tooth like variation in friction force.

¹in this case the chain is elastic and is constantly pulling the convict towards the ball.

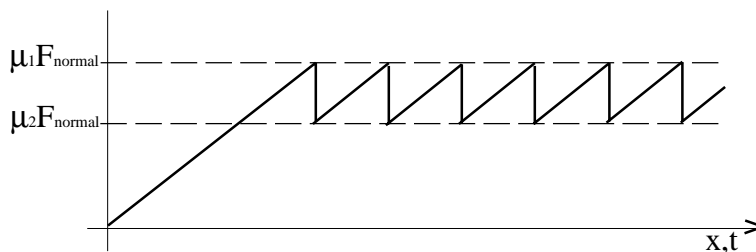


Figure 4-4: Sawtooth vibration from friction model with two friction coefficients

The god-object method makes implementing this algorithm trivial. Given the location computed by the god-object method we can determine what the normal force should be. Since the god-object is always on the virtual object’s surface, there is no non-tangential motion of the god-object. Equation 4.2 can be simplified to Equation 4.7.

$$\Delta x_{\text{tangential}} = (x_{\text{stiction-point}} - x_{\text{god-object}}) \quad (4.7)$$

If the normal force is 0 then the stiction point should be collocated with the god-object. Otherwise the position of the stiction point should only be updated (moved toward the god-object) when the distance from the god-object to the stiction point exceeds the distance from the haptic interface point to the god-object multiplied by μ . The stiction point can then be used (instead of the god-object) for graphical display and force computation (Equation 4.8).

$$\Delta F_{\text{total}} = k(x_{\text{stiction-point}} - x_{\text{haptic-interface}}) \quad (4.8)$$

The stiction point acts as a god-object for the god-object; there are distinct similarities in the two algorithms. They both are necessary because of the inherent compliance in haptic interfaces. The device can’t know which direction it should push on the user, until after he has already tried to move. Both algorithms try to decode the position of the user in the virtual environment (and from that what force should be applied) from the position of the of the haptic interface point.

4.2 Interpolation Algorithms

It is not uncommon for the properties of an object to vary across its surface. There are many mechanisms for continuous variation of a parameter, but interpolation between nodal values is an obvious choice for slowly varying properties. Interpolation has low storage requirements. Surface smoothing can be implemented by interpolating surface normals, and more sophisticated objects can be rendered by varying object impedance across an object.

4.2.1 Surface Smoothing

The constraint-based god-object method simulates rigid polyhedra, but few objects in real life can be accurately described as rigid polyhedra. A model could be refined enough that it would be

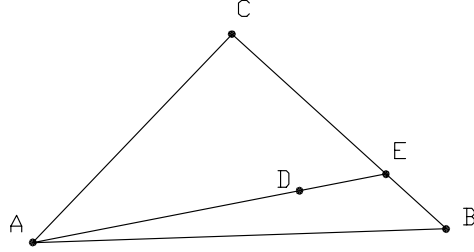


Figure 4-5: Interpolation method for continuous variation of parameters

indistinguishable (to a human) from a continuous object, but such a model would require extensive memory and processing time. An algorithm for smoothing the intersections of the polygons would provide seemingly continuous objects with a coarse mesh. This section will describe a smoothing algorithm that has been used with the constraint-based god-object method.

In the best of worlds, we'd like to be able to model objects using curved surfaces, but these options are computationally intensive. [15] Non-Uniform Rational B-Splines (NURBS) surfaces would be a pretty obvious candidate due to their wide acceptance in the design and geometric modeling industries. Collision detection with such a surface is expensive. Because there is no simple transformation from global coordinates to surface coordinates, we have to iterate to find the point of the surface closest to our point in global coordinates.

Due to all of the difficulties involved with curved surfaces, the constraint based god-object method employs planar surfaces but interpolates surface normals between nodes. Section 2.1.2 explains that humans have a rather poor sense of position but are extremely sensitive to discontinuities of force direction. Our assumption is that if we can smooth the force normals, then the actual shape of the object is secondary.

In the real world the surface normal of the object is the derivative of the surface, necessitating a smooth surface to have a continuous surface normal. With virtual objects there is no such constraint; the surface normal can be decoupled from the shape of the object.

Our smoothing algorithm is very reminiscent of Phong shading used in graphics. [8] A surface normal is associated with each node in each polygon. The appropriate surface normal can then be computed for any location (point D) in the polygon by interpolating between the normals at its three nodes (A, B, and C). The interpolation can be done projecting the vector \overline{AD} to the line \overline{BC} to find the point E (Figure 4-5). The surface normal at point E can be found by adding a contributions from node B and C:

$$\hat{E} = \frac{|\overline{BE}|}{|\overline{BC}|} \hat{B} + \frac{|\overline{CE}|}{|\overline{BC}|} \hat{C} \quad (4.9)$$

The normal at point D can then be found by interpolating between the normals of points A and E:

$$\hat{D} = \frac{|\overline{AD}|}{|\overline{AE}|} \hat{A} + \frac{|\overline{DE}|}{|\overline{AE}|} \hat{E} \quad (4.10)$$

It should be noted that the interpolation scheme is moderately simple because the god-object has provided a location on the surface which we can compare to nodal locations. This algorithm smoothes objects rather well, but only works correctly for a modest range of angles. When the angle

between surfaces gets much less than 150 degrees, there is a noticeable change in displayed surface normal when the boundary is crossed. As was shown in Figure 3-5, there can be a discontinuity in the motion of the god-object if the angle between surfaces is large. The interpolation scheme relies on continuous motion of the god-object to generate a continuous surface normal. This limitation in the shading algorithm is not much of a drawback. Usually the models required for aesthetic graphics are refined enough to meet our haptic needs.

4.2.2 Stiffness and Viscosity/Impedance

Interpolation can be used to vary stiffness and damping in a continuous fashion.² Each node of an object can be assigned its own stiffness and damping. The impedance at any point on the object can be interpolated in exactly the same way the surface normals were.

One subtle thing we will miss with such a simple implementation of impedance blending is energy conservation. [22] It will be possible to press in at a location with a small stiffness, move laterally, and then let the object push you back out at a location where the stiffness is greater. We have allowed “passive” objects to do work on the user. The severity of this effect is directly related to the rate of change of stiffness. This phenomena should not make the device go unstable, so the only problem will be the conceptual understanding of the user. With friction, texture, and surface shading, it will be very difficult to evaluate the work done through a closed loop path. A more pure approach is currently being undertaken to assure that the net work around a loop will be zero. [22]

4.3 Surface Mapping Algorithms

For effects that vary greatly over a small range of motion, interpolation between nodal values is not sufficient. Texture is such an effect. Our haptic representation of texture can take much of its form from the graphics method of texture mapping. A texture map consists of a square data pattern described in texture coordinates (u,v). The process of texture mapping involves translating a position in surface coordinates (s,t) to texture coordinates (u,v) to find the value on the map which applies to the current location. The most difficult part is determining which locations on the texture map correspond to the nodes of the object. Though originally exploited for texture, this method is applicable to many effects.

The texture map mechanism could also be used to vary stiffness and damping. Using a texture map would give the object designer much more control at the cost of increased storage. If a model has a simple geometry (e.g. a cube) but a complex pattern of stiffness or viscosity, texture mapping these variables would save from having to refine the mesh. Most realistic objects will require a sufficiently fine mesh that texture maps for stiffness and viscosity should not be necessary. The impedance only specifies the position the user ends up for a specified force and the position sense of humans is rather poor.

4.3.1 Texture Concepts

Like in graphics, a haptic effect which does require this exactness is texture. As we’ve hinted at before (in Section 2.1.1) texture is really a phenomena which is sensed with the mechano-receptors

²Interpolation for non-linear stiffness equations should be possible assuming it can be made to stay stable.

in your finger tip; the human kinesthetic system can not perceive motions on the scale of texture. Realistic texture would require a tactile array display. Although our hardware doesn't include such a display, we can still give some sense of texture by other means.

To simulate texture without a tactile display, we need to stimulate the human tactile system in a secondary manner, through vibration. We can vary the force applied to user as function of position (or more specifically relative position to an object being stroked). The neurons we are trying to stimulate are perceptible to high frequencies (10-1000Hz); if the "period" of this variation is too large then it will be hard to perceive.

Extensive work on texture has been done with a two degree-of-freedom haptic interface including an attempt to categorize textures. [13] This work is important for understanding the types of texture that can be simulated, but it is not directly applicable to 3-dimensional objects. Using the graphics method bump mapping we take an inherently 3-dimensional approach.

4.3.2 Bump Mapping

In the graphics world, bump maps are used to correctly display the illumination of a bumpy surface. Since the color of a pixel is highly dependent on the orientation of the surface it is important to have the correct surface normal for each pixel. Bump maps are much like texture maps, but instead of associating a color with a location in (u,v) it associates a small displacement to be applied in the surface's normal direction.

These small displacements give rise to a new shape and this new shape means that the surface has new surface normals. A good approximation to the new normal is: [2]

$$\vec{N}_{new} = \vec{N} + \frac{B_u(\vec{N} \times \vec{P}_t) - B_v(\vec{N} \times \vec{P}_s)}{|\vec{N}|} \quad (4.11)$$

where \vec{N} is the surface normal, B_u and B_v are the partial derivatives of the bump map with respect to the u and v directions, and \vec{P}_s and \vec{P}_t are the partial derivatives of the equation $\vec{P} = [x(s, t), y(s, t), z(s, t)]$ in the s and t directions. The new surface color can be computed using the lighting which would fall on a pixel with the newly computed surface normal.

This method works remarkably well for graphics applications, but haptic implementation is somewhat different. The slight variation of height due to the bump map might not be sensible by the user, but this information is directly available. The magnitude of the feedback force vector can be increased or decreased slightly depending on the entry in the bump map. The new normal direction can be used with the magnitude computed by the god-object algorithm as long as the variation in normal direction are small. If much more than ± 5 deg are used the system is likely to be unstable. A bump-map extension to the god object should be able to simulate materials like wood, sand paper, or rusted metal, but anything much more significant would probably have to be encoded into geometry.

This was implemented in a limited form for use with the god-object algorithm. Surfaces were covered by with a periodic texture. A 3x3 transformation matrix was computed based on surface position and applied to the normal force vector. We found this to be capable of reproducing sandpaper-like textures.

4.4 Conclusion/Summary

Many of the ideas explored in graphical rendering, including shading and texture mapping, are applicable to haptic rendering. There are some major differences between graphical methods and haptic methods. Graphical methods need to redraw a whole screen of pixels every 30-60 Hz while our haptic methods need to update the force on a single point at about 1kHz.

All of these effects require a surface location to generate their effect. In graphics computations, rays of light always interact with the surface of the object³ so the surface location is always known. Haptic interfaces can penetrate into the volume of an object so a method is needed to determine its location on the object's surface. The constraint-based god-object approach satisfies this need.

³with the exception of translucent objects.