Improved Haptic Rendering for Multi-Finger Manipulation Using Friction Cone based God-Objects

W. S. Harwin N. Melder
Department of Cybernetics, University of Reading

Keywords: god-object, multi-contact haptics, friction, friction cones, dynamic haptic environment

Abstract

A novel method of characterising friction in a virtual haptic environment is described and validated in a small experimental two-finger grasp task. The algorithm adapts a common technique called the god-object method, such that the god-object is updated only when it strays outside the friction cone defined by the coefficient of friction of the object surface. The technique not only allows efficient robust estimation of friction but also can give residual forces and torques thereby allowing the object to be manipulated in a natural fashion.

Introduction

Simulation of multi-finger grasps and contact in a haptic device presents a number of problems over the use of single finger interactions. These problems include how to model friction and how to resolve the contribution of forces from multiple finger contact points. This paper presents a method for modelling friction in a multi-finger haptic interface based upon the use of god-objects[1] and friction cones. The paper also discusses the estimation of the residual force in a virtual object so that it can be lifted and rotated with a two or more fingered grasp.

Friction is a vital aspect of modelling in multi-point haptic interactions. A friction model is needed to allow two and three finger grasps without the fingers slipping from the virtual object. Friction is also needed when lifting an object from a surface to stabilise the object in the grasp before it is lifted away. Once clear of the surface it would be reasonable to expect a simulation of gravity such that the weight of the object can be perceived. Furthermore, a mechanism needs to be in place to allow the object to be rotated in all three rotational degrees of freedom so that it can be placed correctly or so that other surfaces can be viewed.

Background

Successful implementation of a friction algorithm in a multi-finger haptic environment requires

1. the identification of collisions,
2. an estimation of external forces resulting from the collisions (including frictional forces), and
3. an appropriate response to the residual forces.

There are many collision detection algorithms and each tend to have certain advantages in different situations. In haptic interface design the deciding factor in choosing the best collision algorithm is always the speed of calculation to determine whether a collision has occurred. Two popular collision algorithms are OBBTree[2], and H-Collide[3] although the information returned from a polygon/point collision algorithm is all that is required.

Once a collision between one or more phantoms has been identified then the appropriate forces must be calculated. A popular method that avoids object push through is that used by Zilles and Salisbury [1] 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.

Perhaps the most comprehensive friction model is the LeGre friction model [4] which in turn is a development of

the Dahl model and includes an internal state to allow for microslip. Richards and Cutosky [5] discuss both the Dahl and the computationally simpler Karnopp model, and then use the latter to characterise the friction characteristics of a number of materials including aluminium-Teflon, aluminium-brass and aluminium-rubber. They note a difficulty in haptic rendering of textures as being the difficulty in getting an accurate estimate of interface velocity at near zero velocities where quantisation noise from the encoders is high. The friction cone method now presented is similar to the Karnopp model and lends itself to multi-point haptic interface devices.

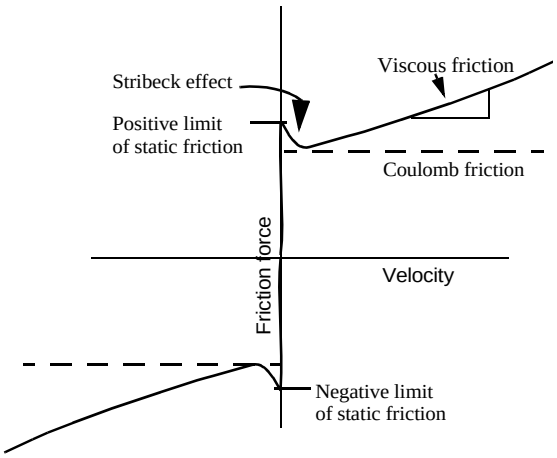

Figure 1: Friction force vs velocity

The Friction Cone Algorithm

A simple adjustment to the god-object algorithm allows us to place a friction cone at the haptic interaction point, oriented in the direction of the normal of the contacted surface. A friction cone is simply defined by the friction angle:

$$\tan\theta = \frac{\text{Max Friction Force}}{\text{Normal/Reaction Force}} = \mu$$

where μ can apply to static or dynamic (coulomb) friction

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.

Whereas in Zilles' paper, as the haptic interface point moves so does the god-object, the approach used here will only move the god-object if the god-object lies outside the circle of friction (see figures 2 and 3). To calculate the size of the friction circle we use the depth of penetration of the haptic interface point in relation to the surface 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.[1] If a surface has a low frictional coefficient then, for a given penetration depth, it will have a smaller friction circle.

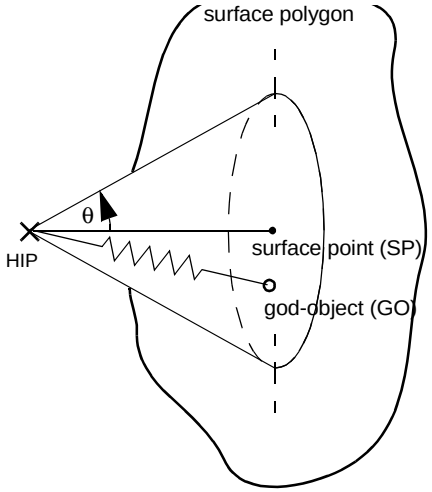

Figure 2: god-object inside friction cone remains unchanged

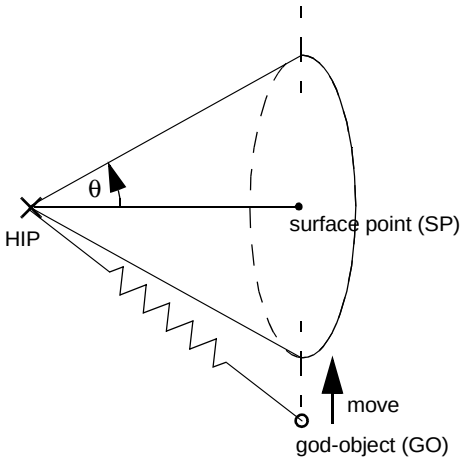

Figure 3: god-object moves to edge of friction cone

1. It is also possible to assign the friction coefficient to a 'friction map'. This would be analogous to a graphical texture where the ‘colour’ variable represents the frictional coefficient. When a collision is detected, the point on the friction map would be checked to find the appropriate frictional coefficient.

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.

1. Calculate the surface point (SP) of the haptic interface point (HIP). The surface point is defined as the minimised distance between the haptic interface point and the contacted surface.This is the same location used for the GO as defined in [1].
2. Create friction cone based upon friction coefficient of the surface and the penetration depth of the HIP. This can be defined as a simple multiplication, radius of friction circle = depth * μ (where μ is the friction coefficient).
3. Calculate distance between SP and god-object (GO).
4. If the distance between the GO and SP > Radius of friction circle move GO onto circumference of friction circle otherwise do nothing.

The following pseudo code explains the algorithm. It should be noted that all the positional variables (SP, GO and HIP) are co-ordinates in the polygon space (i.e. the polygon being tested passes through the origin.) Note also that the GO, and the SP, are located on the surface of the polygon of interest and that the surface normal is a unit vector.

```
WhileHIP is inside surface
do
    float DepthOfHIP =
            dotproduct(surface.normal, HIP)
    SP = HIP +  DepthOfHIP
    RadiusOfFrictionCircle = DepthOfHip *
            SurfaceFrictionConstant

    DirSPGO= GO -SP
    DistSPGO = modulus(DirSPGO)

    if (DistSPGO > RadiusOfFrictionCircle)
        GO =  SP + (DirSPGO.normalize() *
            RadiusOfFrictionCircle)

end while
```

From the algorithm, it can be seen that when the GO is outside the friction cone it will 'jump' to the closest point on the circumference of the friction circle. This provides the equivalent to static friction.

Residual Force Resolution

To determine the required forces that should be applied to the haptic interface the position of the GO and HIP are used. Because the GO and HIP are not in the same place once a collision has occurred the direction vector from the HIP to the GO are used as the parameters to control the direction and force that is required by the haptic interface. Similarly, the force that is being applied to the object is the direction vector from the GO to the HIP. Since the friction cone algorithm is determined per HIP, in a multi-finger haptic interface, it is possible for an object to be influenced by more than one HIP i.e. the object may be affected by more than one force at any one time. Because all the forces are stored as vectors, by summing all these force vectors, the resultant vector is the residual force in the object. Likewise, the force moments around the object's centre of gravity will give the residual torque. This residual force can then be input to a suitable movement algorithm to determine how the object reacts to this applied force. The addition of a centre of mass and moments of inertia to the object allows the modelling of gravity and rotational forces. Since gravity can be represented as a vector force it is trivial to add to the simulation. Modelling rotational torques is similar and is done by calculating moments about the object's centre of mass.

Results

The algorithm has been tested in a two finger grasp mechanism based on two Phantom 1.5 haptic interfaces running rt-linux version 2.4. The drivers for the two phantoms are modified from those developed by Zdenek Kabelac and Petr Konecny at the HCI-Lab, Masaryk University at Brno, Czech Republic (http://www.fi.muni.cz)

Results for a simple environment using the cone friction algorithm are shown in figure 4. The environment consists of a cube whose sides have a high coefficient of friction μ = 2 (hence cone apex angle is approximately 126°) that is sitting on a simulated floor. In the experiment the cube is grasped between two fingers and thrown upwards. Figure 4 shows two such vertical throws. The first proceeds normally where the grasp aperture closes on the cube and when a stable grasp has been achieved a rapid vertical movement is performed so as to toss the cube. Cube release is characterised by a rapid opening of grasp aperture coincident with a brief downwards movement of the fingers. The cube is recaptured between second 10 and 12. The second throw begins at 12.4 seconds and on this occasion an unsuccessful attempt is made to catch the cube as it falls. Although these results do not detail the actual workings of the friction cone algorithm they illustrate it in operation. Only a static value of friction is used and the transition from contact to release requires greater move-

ment than would be needed in a real throwing operation, however the results demonstrates that the algorithm has a good degree of realism in that there is a realistic opportunity for the cube to be caught in ‘mid flight’

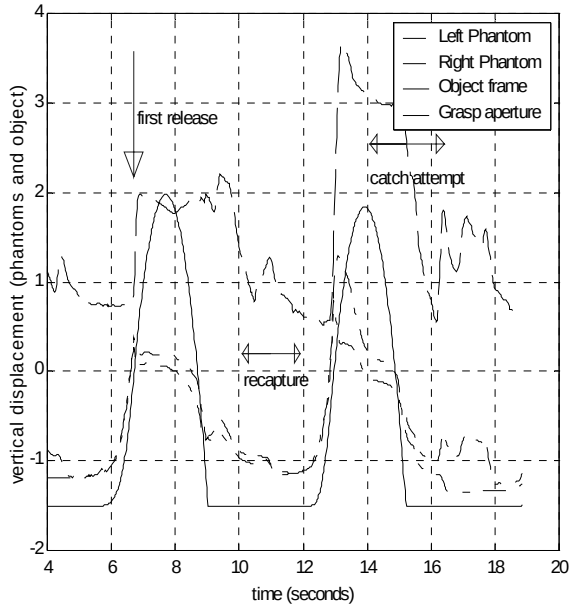

Figure 4: two throws of a virtual cube

Discussion

The friction cone algorithm is a classic friction model that allows for the simulation of slip on an object's surface. Because of its simplicity and exclusive use of vectors, it is computationally efficient yet it also supplies important information regarding the actual slippage (the direction that the slippage occurred and the distance of the slip). With the appropriate hardware (e.g. vibratory cue) it is possible to use this extra information to give the actual feeling of the object 'slipping through the fingers'. The algorithm is also fully scalable to work in any number of dimensions with any number of 'fingers' simply by the implicit nature of vectors.

Although appropriate hardware would allow for high frequency sensory cues to aid in the perception of the slipping, this could also be provided as an audible sound as well as a superimposed force that modulates the direction and magnitude of the force that the PHANToM applys to the user. It is appropriate to generate this ‘blip’ in an annulus cone such that when the GO is in the annulus the force applied to the user has a high frequency component to emulate slip.

It is also possible to modify the friction cone algorithm to model dynamic friction through the use of a second, smaller friction cone. On initial contact the friction cone algorithm would use the static friction cone until the GO needs to be moved. The GO would be moved to the edge of the dynamic friction cone but then the dynamic friction cone would be used to determine if the GO needs to be moved.

Management of the GO as the HIP transitions across a polygon is an ongoing area of work.

Conclusions

The friction cone algorithm provides a useful design mechanism that not only successfully models friction but also allows multifinger interactions to occur naturally. The algorithm has several possible extensions that serve to improve the realism of the haptic interface.

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.

References

[1] C. B. Zilles and J. K. Salisbury, A Constraint-based God-object Method for Haptic Display, *International conference on intelligent robots and systems*, 1995

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

[3] 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

[4] P. Lischinsky, C. Canadas-de-Wit and G. Morel, Friction compensation for an industrial hydraulic robot, *IEEE Control systems magazine* 19(1) Feb 1999 pp25-32

[5] C. Richards and M.R. Cutkosky, Friction modeling and display in haptic applications involving user performance, *Proceedings of the 2002 IEEE International Conference on Robotics and Automation* May 2002 pp605-611