Object Mouse Trackball

Contents

Introduction

The motivation behind the trackball (aka arcball) is to provide an intuitive user interface for complex 3D object rotation via a simple, virtual sphere - the screen space analogy to the familiar input device bearing the same name.

The sphere is a good choice for a virtual trackball because it makes a good enclosure for most any object; and its surface is smooth and continuous, which is important in the generation of smooth rotations in response to smooth mouse movements. Any smooth, continuous shape, however, could be used, so long as points on its surface can be generated in a consistent way.

Having a ball

The first step in generating a trackball is defining the ball itself, which is simply a sphere at the origin,

$$x^2 + y^2 + z^2 = r^2 \ ,$$

where r is the radius of the sphere.

Next we would like to center our sphere on the object we would like to rotate. To do this, we can project the object's center into screen space. Let O be the projected object center. The equation for our sphere then becomes

$$(x - O_x)^2 + (y - O_y)^2 + z^2 = r^2 \ .$$

This will place our sphere at the screen space origin, which, in OpenGL, is the lower left corner of the screen by default. We can put the origin amid screen with an additional translation of half the screen dimensions in either direction, C, giving the final form of our sphere as

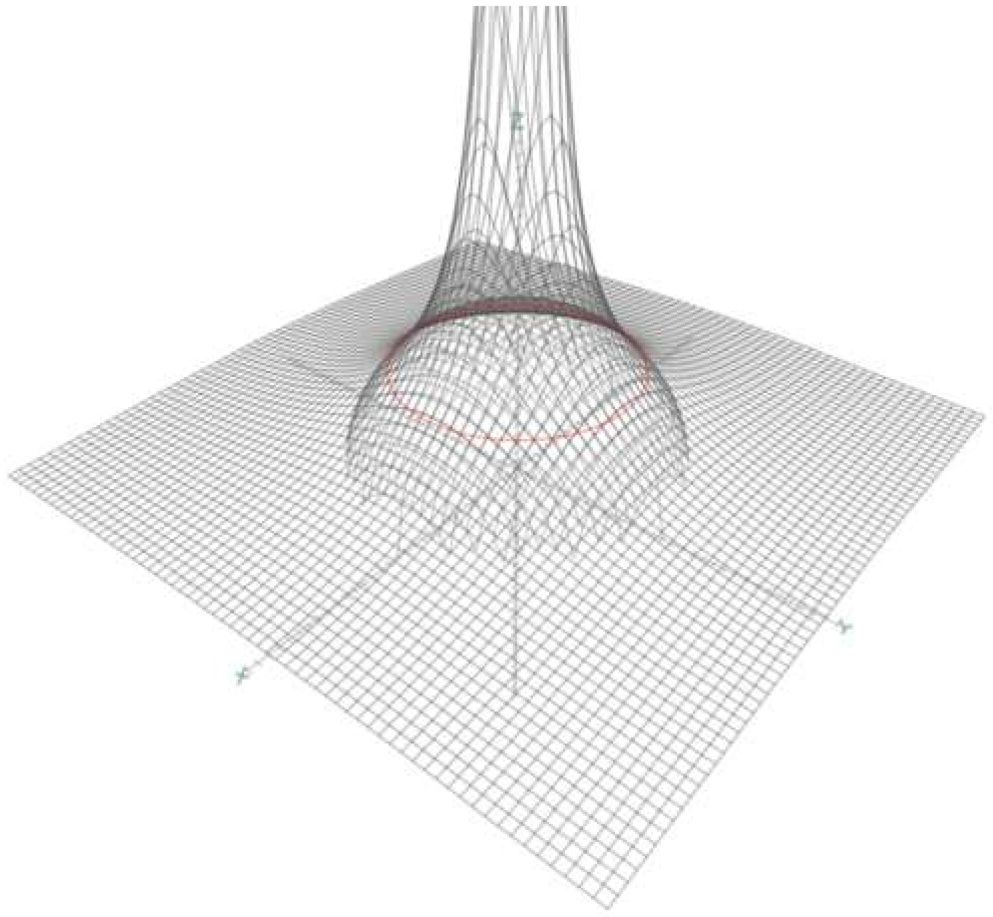
$$(x - C_x - O_x)^2 + (y - C_y - O_y)^2 + z^2 = r^2 \ .$$

Objects outside the view frustum could be culled via a sphere-frustum collision test, for example.

Of mice and manipulation

As the mouse moves over the surface of our sphere it traces out a curve. Each position on this curve from the sphere's center is a vector, and any two consecutive vectors spans a plane through the center of the sphere. The normal of this plane is our rotation axis, and the distance traveled by the mouse between any two points implicitly determines the amount of rotation.

In detail, as the mouse moves over the screen we first need to generate points on the sphere. That's a simple matter of evaluating our equation,



Sphere and hyperbolic sheet. The circle of intersection is in red.

$$(x - C_x - O_x)^2 + (y - C_y - O_y)^2 + z^2 = r^2 \ ,$$

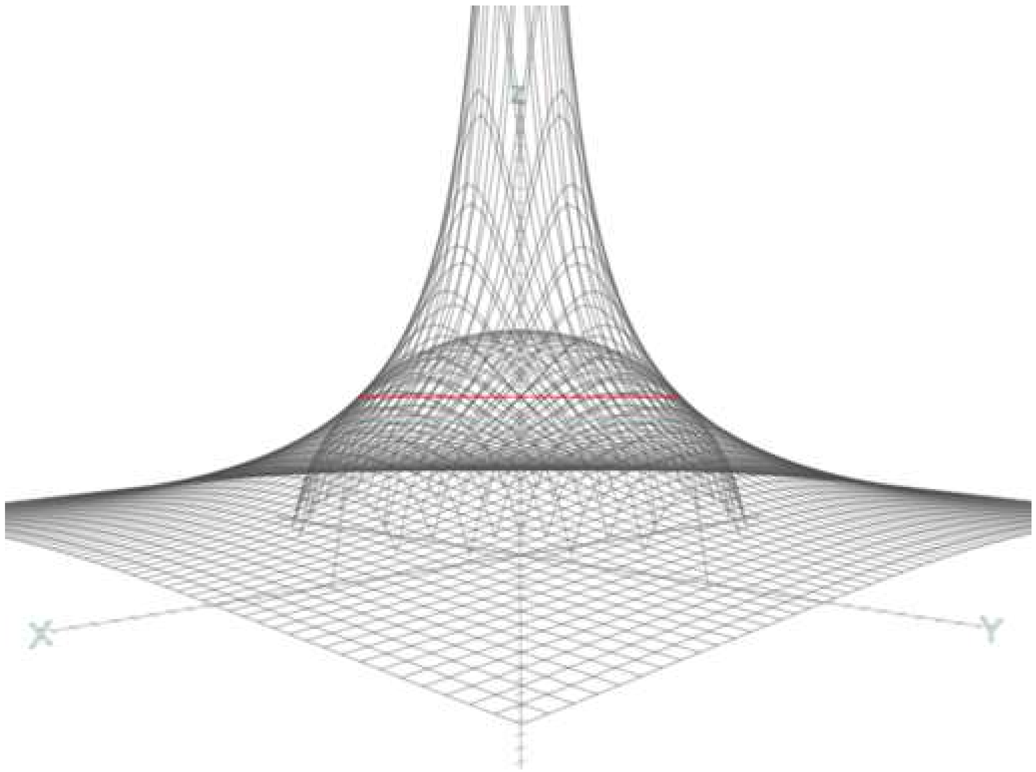
for a given mouse x and y.

But what of points outside the sphere? Frustum culling restricts us to spheres at least partially visible, but we can still generate points outside of the sphere but within the screen's extent, which would generate complex results (square roots of negative numbers). This can be seen in the explicit form of the sphere,

$$z(x', y') = \sqrt{r^2 - (x'^2 + y'^2)},$$

where x' and y' are the centered mouse x and y shorthands from above. Wherever $x'^2 + y'^2 > r^2$, we're in trouble.

One nice way to get around this unpleasantness is to use a piecewise combination of surfaces in place of a single sphere. In this way we can switch to one surface in a region where things are awkward in another. One surface that works well in this case is the hyperbolic sheet given by



View of sphere and hyperbolic sheet in the plane of intersection. The profile curve is C^1 continuous at the crossover.

$$z(x', y') = \frac{r^2/2}{\sqrt{x'^2 + y'^2}}.$$

This surface has the shape of an inverted trumpet, with the mouth piece at infinity along the z axis and the horn opening flaring out to cover the xy-plane. With the sphere nestled snugly at its opening, we can transition very smoothly from a ball to a flaring horn defined everywhere except the origin. The crossover from sphere to sheet occurs at their intersection, which is defined by the circle

$$x'^2 + y'^2 = \frac{r^2}{2}.$$

One might be tempted to simply ignore points outside the sphere altogether; but as the mouse approaches the perimeter of the sphere response can quickly become clumsy and unintuitive due to ∂z/∂x and ∂z/∂y approaching infinity near the silhouette edge. So in addition to avoiding negative numbers in our radical, the primary motivation for the transition is to make rotational falloff as graceful as possible: the farther the mouse ventures from the center of the sphere the less angular displacement is generated between any two mouse positions, which tapers off completely as the sheet flattens off to zero at infinity.

Sit and spin

So now that we can generate points for our surface anywhere on the screen, there's still the matter of generating a rotation from them. The standard way to do this is through the use of quaternions, which provide a very compact, convenient and robust representation for rotations. What makes quaternions particularly nice in this situation is the property that a normalized quaternion always represents a rotation -- as arbitrary, incremental rotations will be applied to our objects and in no particular order, the importance of this property can't be overstated.

With quaternions in hand, object rotations are very straightforward. In fact, with our axis and angle from above, they're nigh automatic. Any two consecutive (mouse) positions on our surface are enough to generate two vectors, the cross product of which is the rotation axis, and the inverse cosine of the dot product of which is the angle. The basic procedure is as follows:

$$z(x', y') = \begin{cases} \sqrt{r^2 - (x'^2 + y'^2)} & x'^2 + y'^2 \leq \frac{r^2}{2} \\ \frac{r^2/2}{\sqrt{x'^2 + y'^2}} & \text{otherwise} \end{cases}$$

$$V_1 = \frac{(x'_1, y'_1, z(x'_1, y'_1))}{|(x'_1, y'_1, z(x'_1, y'_1))|}$$

$$V_2 = \frac{(x'_2, y'_2, z(x'_2, y'_2))}{|(x'_2, y'_2, z(x'_2, y'_2))|}$$

$$N = V_1 \times V_2$$

$$\theta = \arccos V_1 \cdot V_2$$

$$Q = (\cos\frac{\theta}{2}, \sin\frac{\theta}{2} N)$$

With each incremental mouse movement from (x'_1, y'_1) to (x'_2, y'_2) we apply (accumulate) the rotation Q and then unitize the result to ensure we retain a rotation.

Numeric drift and a snapper

While we can be sure our coordinate axes will remain orthogonal under rotation, due to limited floating point precision, we can't fully escape numeric drift in our calculations: After several arbitrary rotations, we can't guarantee that a reversal of the sequence will lead to our original basis exactly. To help stem the numeric tide, it's possible to snap the rotation axes to the standard axes wherever they're very close to one another. Since a quaternion can be readily converted to and from an axis-angle representation, it's possible to first convert the quaternion to an axis-angle, snap the axis (if very close to a standard axis), angle snap, then convert the result back to a quaternion for rotation. The angle snap must be performed in lockstep with the standard axis snap, otherwise a "stair casing" or rotational aliasing effect will ensue with each incremental rotation, the result of which is an ungainly bobble and teeter of the coordinate frame.

Conclusion

The virtual trackball can provide an intuitive user interface for the interactive rotation of models in 3D.

References

- Graphics Gems (https://github.com/erich666/GraphicsGems)
  - Arcball (https://github.com/erich666/GraphicsGems/tree/master/gemsiv/arcball)
