CS417 Program 4
Virtual Trackball notes

April 1, 2003

These notes are to supplement the assignment handout for CS418 Spring 2003 Program 4. The handout specifies that you should implement something called "virtual trackball rotation." This is a popular user interface method that lets the user define arbitrary 3D rotations using mouse clicks in a 2D window.

The user interface metaphor used in the virtual trackball interface is that of a spherical trackball that the user rotates by placing a finger on the ball and moving it. The trackball itself is not drawn in the scene; it is just imagined to be floating right in front of the viewer so that it projects to a circle inscribed in the viewport. As with many other transformation interfaces, the idea is to apply a transformation that keeps the point that was clicked on under the mouse—only in this case the point that was "clicked on" is on an imaginary trackball rather than on the object that's actually being manipulated.

Figure 1 illustrates the specifics of how the trackball works. The previous mouse point (where the user metaphorically put their finger on the sphere) and the current mouse position (where the finger has moved to now) are each projected orthographically onto a sphere inscribed in the viewport, resulting in vectors $\mathbf{u}_0$ and $\mathbf{u}_1$ from the center of the sphere to the two points. This projection can be done very simply. Denote the old and new mouse points as (x_0, y_0) and (x_1, y_1) respectively, and assume that the viewport coordinates run from -1 to 1 on the y axis. Since the trackball is always in the same position in the image there is no need to look at the camera position or projection at all; we simply project along z to the unit sphere. This means that if $\mathbf{u}_i = (x_i, y_i, z_i)$ we have the first two coordinates directly from the mouse point, and we can compute the third using the unit radius of the sphere:

$$z_i = \sqrt{1 - x_i^2 - y_i^2}$$

. If (x_i, y_i) falls outside the unit circle, this will fail. So you need to check for

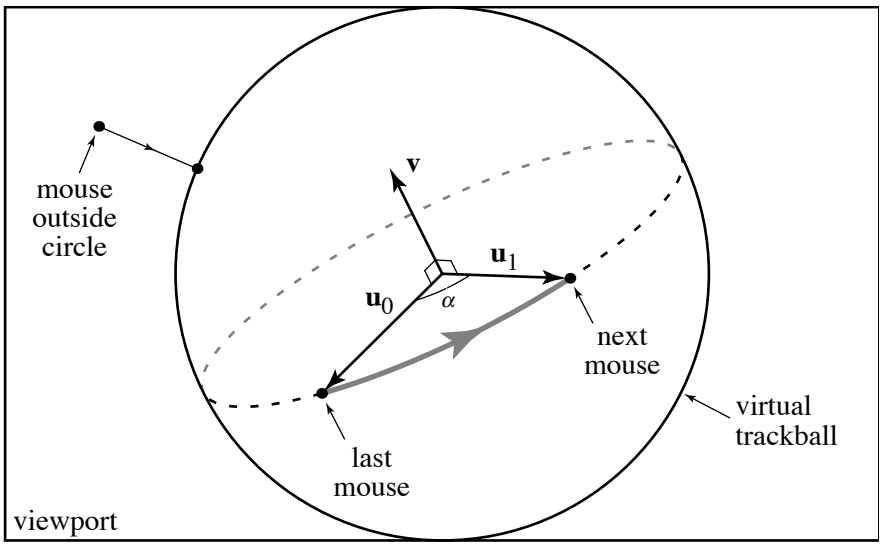

Figure 1: Vectors for computing the virtual trackball rotation. $\mathbf{v} = \mathbf{u}_0 \times \mathbf{u}_1$ and $\alpha = \sin^{-1} \|\mathbf{v}\|$.

this; the correct behavior is to project the clicked point to the nearest point on the sphere's silhouette, which you can do by setting $z_i = 0$ and renormalizing.

Now that we have $\mathbf{u}_0$ and $\mathbf{u}_1$ in hand, we need to compute a rotation that will take $\mathbf{u}_0$ to $\mathbf{u}_1$. There are many such rotations, but the smallest one in some sense is the one that simply moves $\mathbf{u}_0$ along a great circle to $\mathbf{u}_1$—that is, a rotation about the normal to the plane defined by $\mathbf{u}_0$ and $\mathbf{u}_1$. Expressed using an axis and an angle, this is a rotation about the axis $\mathbf{v} = \mathbf{u}_0 \times \mathbf{u}_1$ by the angle $\sin^{-1} \|v\|$. Since we know the angle will always be less than 180°, the right hand rule assures us that the rotation about $\mathbf{v}$ will always be counterclockwise to take $\mathbf{u}_0$ to $\mathbf{u}_1$.

The resulting rotation should be around the camera's target point. That way an object can be rotated about its center by first targeting on it using the "look at" feature from the previous assignment.

To summarize, here are the steps to follow, given the two 2D mouse points:

1. project the two points ont the unit sphere
2. compute the axis perpendicular to the two vectors and the angle between the vectors
3. call an axis-angle rotation routine to set up a rotation matrix.
4. apply that rotation matrix in the right way

With a properly working trackball, the user can generate rotations about the three coordinate axes. To rotate around y, click in the center of the window and drag straight left or right. To rotate around x, click in the center of the window and drag straight up or down. To rotate around z, click in the area outside the trackball and drag