CSC 4356
Interactive Computer Graphics
Lecture 11: 3D Interaction

Jinwei Ye

http://www.csc.lsu.edu/~jye/CSC4356/

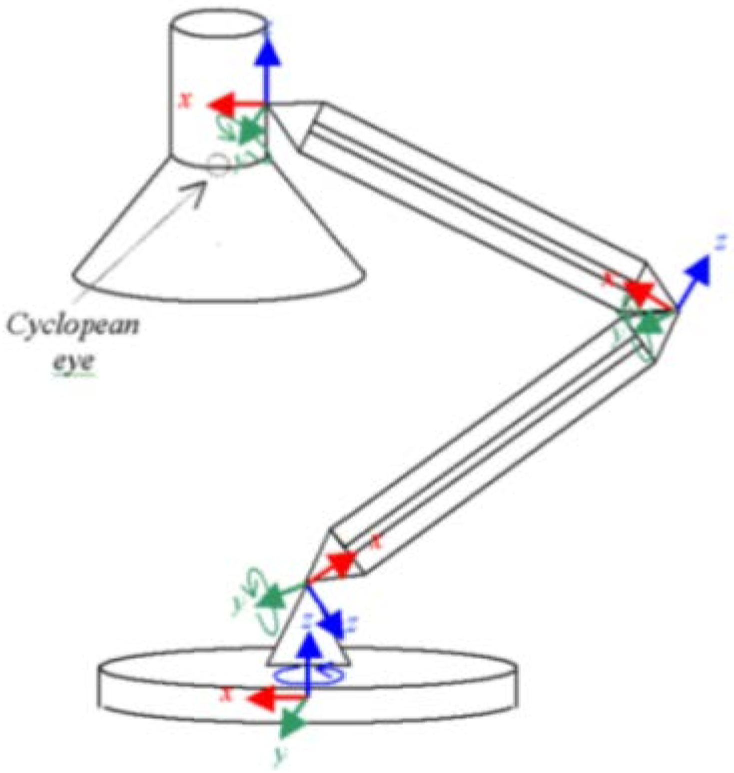
Tue & Thu: 10:30 - 11:50am

218 Tureaud Hall

3D Interaction

Transformation Hierarchies

- Many models are composed of independent moving parts
- Each part defined in its own coordinate system
- Compose transformations to position and orient the model parts



Transformation Hierarchies: Graph Model

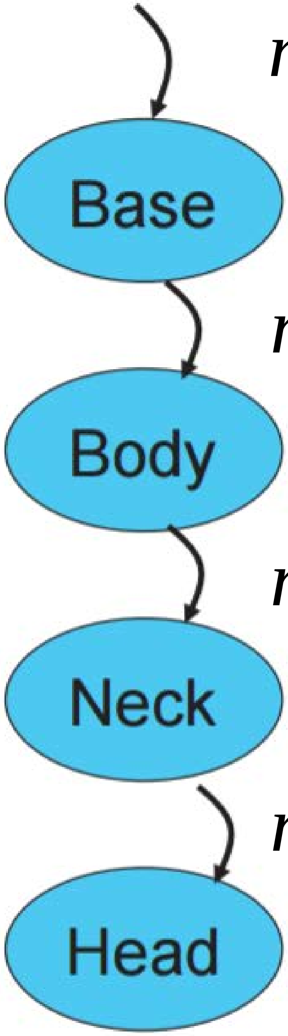
- Model parts are nodes
- Transforms are edges
- What transform is applied to the Head part to get it into world coordinates?

$$\dot{m}_4^t = \dot{w}^t \mathbf{T}_{world}^{base} \mathbf{T}_{base}^{body} \mathbf{T}_{body}^{neck} \mathbf{T}_{neck}^{head}$$

- Suppose that you'd like to rotate the Neck joint at the point where it meets the Body. Then what is the Head's transform to world space?

$$\dot{m}_3^t = \dot{m}_2^t \mathbf{T}_{body}^{neck} \mathbf{R}$$

$$\dot{m}_4^t = \dot{w}^t \mathbf{T}_{world}^{base} \mathbf{T}_{base}^{body} \mathbf{T}_{body}^{neck} \mathbf{R} \mathbf{T}_{neck}^{head}$$



$$\dot{m}_1^t = \dot{w}^t \mathbf{T}_{world}^{base}$$

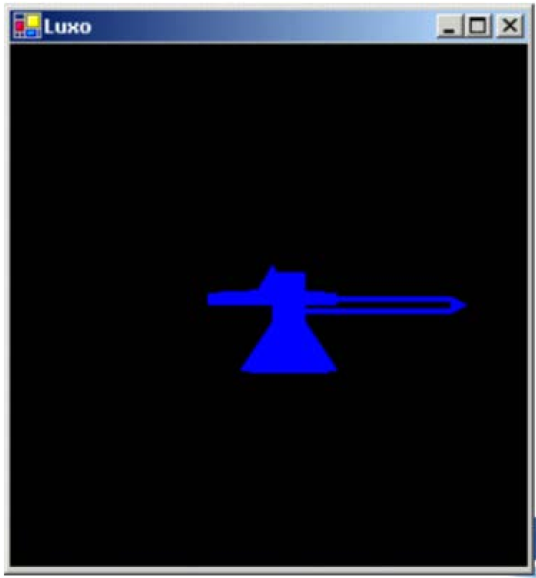
$$\dot{m}_2^t = \dot{m}_1^t \mathbf{T}_{base}^{body}$$

$$\dot{m}_3^t = \dot{m}_2^t \mathbf{T}_{body}^{neck}$$

$$\dot{m}_4^t = \dot{m}_3^t \mathbf{T}_{neck}^{head}$$

Code Example (1st Try)

```
public void Draw() {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glLoadIdentity();
        gluLookat(0, 0,-60, 0,0,0, 0,1,0); // world-to-camera transform
        glColor3d(0,0,1);
        glRotated(-90, 1, 0, 0); // base-to-world transform
        Draw(Lamp.BASE);
        Draw(Lamp.BODY);
        Draw(Lamp.NECK);
        Draw(Lamp.HEAD);
        glFlush();
}
```

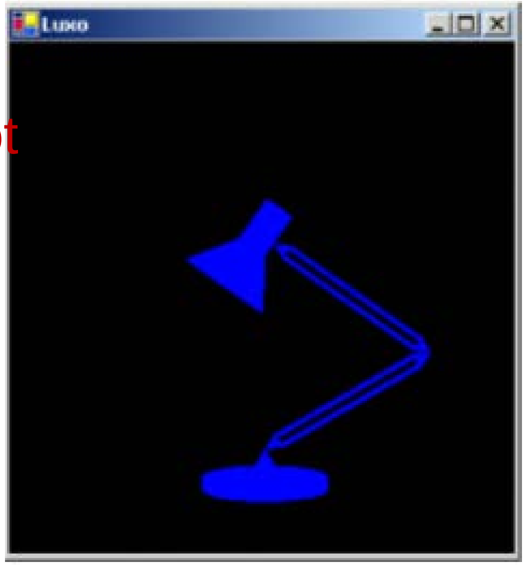


Code Example (2nd Try)

```
public void Draw() {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glLoadIdentity();
        gluLookat(0, 0,-60, 0,0,0, 0,1,0); // world-to-camera transform
        glColor3d(0,0,1);
        glRotated(-90, 1, 0, 0); // base-to-world transform
        Draw(Lamp.BASE);
        glTranslated(0,0,2.5); // body-to-base transform
        Draw(Lamp.BODY);
        glTranslated(12,0,0); // neck-to-body transform
        Draw(Lamp.NECK);
        glTranslated(12,0,0); // head-to-neck transform
        Draw(Lamp.HEAD);
        glFlush();
}
```

Code Example (3rd Try)

```
public void Draw() {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glLoadIdentity();
        gluLookat(0, 12, -60, 0,0,0, 0,1,0); // world-to-camera transform
        glColor3d(0,0,1);
        glRotated(-90, 1, 0, 0); // base-to-world transform
        Draw(Lamp.BASE);
        glTranslated(0,0,2.5); // body-to-base transform
        glRotated(-30, 0, 1, 0); // rotate body at base pivot
        Draw(Lamp.BODY);
        glTranslated(12,0,0); // neck-to-body transform
        glRotated(-115, 0, 1, 0); // rotate neck at body pivot
        Draw(Lamp.NECK);
        glTranslated(12,0,0); // head-to-neck transform
        glRotated(180, 1, 0, 0);// rotate head at neck pivot
        Draw(Lamp.HEAD);
        glFlush();
}
```

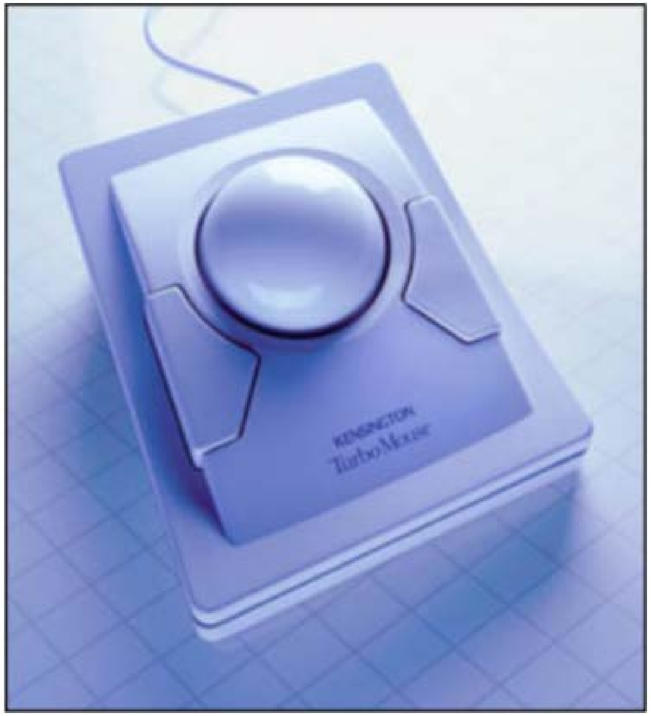


Interaction Paradigm

- Can move objects or camera
  - Object moving is more intuitive if the object "sticks" to the mouse when dragging
- Move w.r.t. to camera frame
  - Pan: move in plane perpendicular to view direction
  - Dolly: move along the view direction
  - Zoom/Scale: look like dolly (objects gets bigger or smaller) but position remain fixed
  - Rotate & Roll: object spinning about an axis

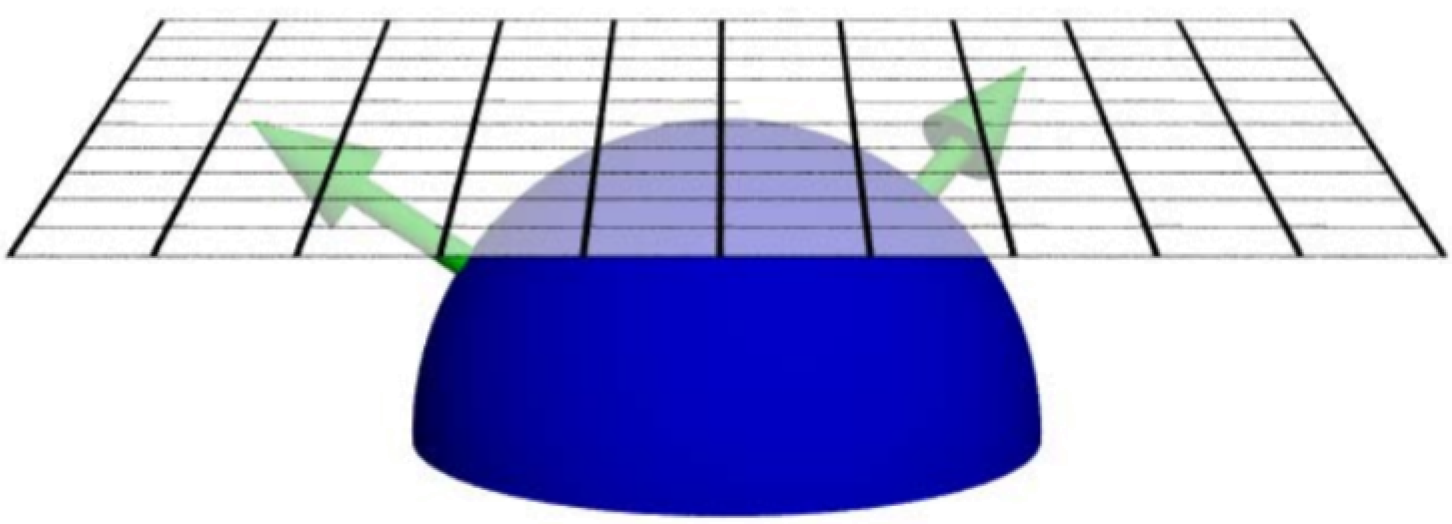
Example: Trackball

- A common UI for manipulating objects
- Two degree of freedom device
- Differential behavior provides a intuitive rotation specification

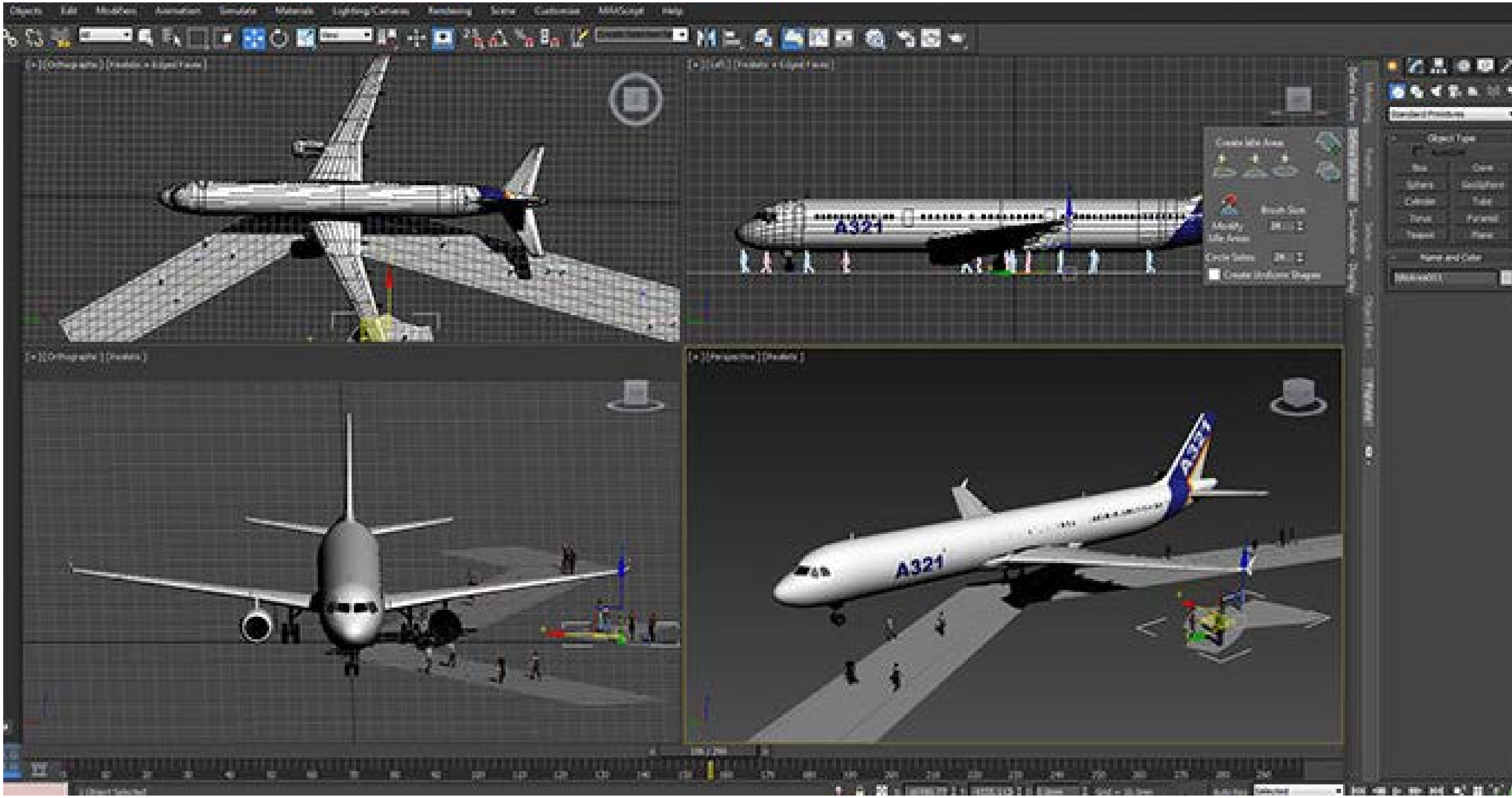


A Virtual Trackball

- Imagine the viewport as floating above, and just touching an actual trackball
- You receive the coordinates in screen space of the MouseDown() and MouseMove() events
- What is the axis of rotation?
- What is the angle of rotation?



Applications: Design



3DS Max

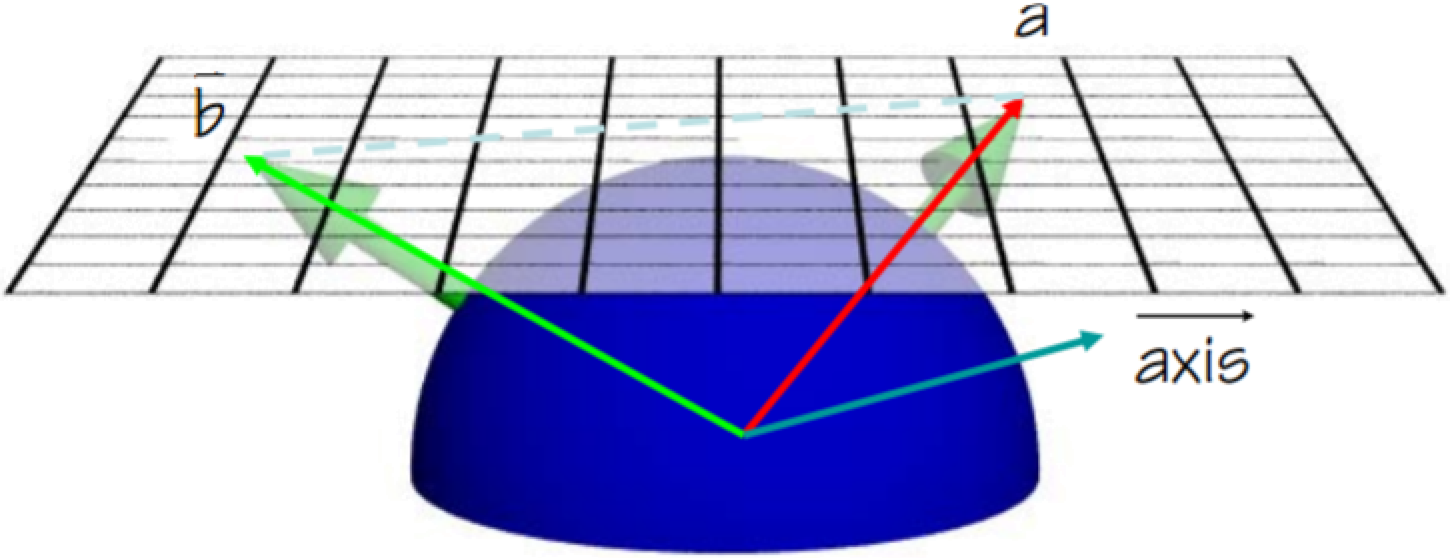
Applications: Games

Pokemon Go

Application: 360˚ photo/video

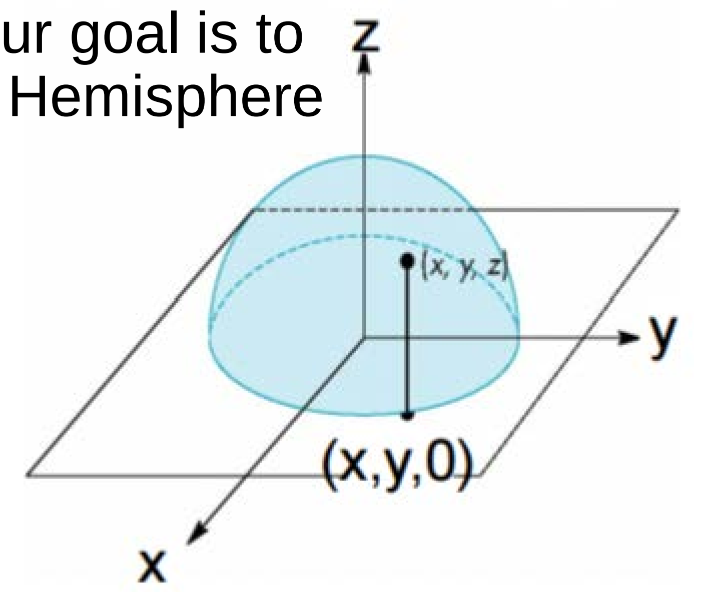
Computing the Rotation

- Construct a vector $\vec{a}$ from the center of rotation of the virtual trackball to the point of the MouseDown() event.
- Construct a 2nd vector $\vec{b}$ from the center of rotation for a given MouseMove() event.
- Normalize $\hat{a} = \frac{\vec{a}}{|\vec{a}|}$, and $\hat{b} = \frac{\vec{b}}{|\vec{b}|}$, and then compute $\overrightarrow{axis} = \hat{a} \times \hat{b}$
- Then find the $angle = \cos^{-1}(\hat{a} \cdot \hat{b})$, and construct $\mathbf{R} = Rotate(\text{angle}, \frac{\overrightarrow{axis}}{|\text{axis}|})$



Mapping Mouse Point to Hemisphere

- How to compute $\vec{a}$ and $\vec{b}$?
- Assuming the mouse position is (x,y), our goal is to map the mouse position to a point on a Hemisphere
- Hemisphere point P
  - x = x
  - y = y
  - z = $\sqrt{1 - x^2 - y^2}$ (assume the radius = 1)
- If a point is outside the circle, project it to the nearest point on the circle
- We need to normalize mouse position (x,y) to NDC [-1,1]
  - Origin of your viewport is the top-left corner



Implementation: Key Steps

- Detect the left-button of the mouse being depressed.
- Keep track of the last known mouse position.
- Treat the mouse position as the projection of a point on the hemi-sphere down to the image plane (along the z-axis), and determine that point on the hemi-sphere.
- Detect the mouse movement
- Determine the great circle connecting the old mouse-hemi-sphere point to the current mouse-hemi-sphere point.
- Calculate the normal to this plane. This will be the axis about which to rotate.
- Rotate about the axis
- Force a redraw of the scene.

Some Help with Virtual Trackball

- Treat the mouse position as the projection of a point on the hemi-sphere down to the image plane (along the z-axis), and determine that point on the hemi-sphere.

```
//
// Utility routine to calculate the 3D position of a
// projected unit vector onto the xy-plane. Given any
// point on the xy-plane, we can think of it as the projection
// from a sphere down onto the plane. The inverse is what we
// are after.
//
Vec3f CSierpinskiSolidsView::trackBallMapping(CPoint point)
{

        Vec3f v;
        float d;
        v.x = (2.0*point.x - windowSize.x) / windowSize.x;
        v.y = (windowSize.y - 2.0*point.y) / windowSize.y;
        v.z = 0.0;
        d = v.Length();
        d = (d<1.0) ? d : 1.0;
        v.z = sqrtf(1.001 - d*d);
        v.Normalize(); // Still need to normalize, since we only capped d, not v.
        return v;

}
```

Determine Rotation Axis and Angle

<table>
<tr>
<td>

- Detect the mouse movement

</td>
<td>

```
void CSierpinskiSolidsView::OnMouseMove(UINT nFlags, CPoint point)
{

    //
    // Handle any necessary mouse movements
    //
    Vec3f direction;
    float pixel_diff;
    float rot_angle, zoom_factor;
    Vec3f curPoint;
    switch (Movement)
    {

    case ROTATE : // Left-mouse button is being held down
    {
    curPoint = trackBallMapping( point ); // Map the mouse position to a logical
    // sphere location.
    direction = curPoint - lastPoint;
    float velocity = direction.Length();
    if( velocity > 0.0001 ) // If little movement - do nothing.
    {
```

</td>
</tr>
<tr>
<td>

- Determine the great circle connecting the old mouse-hemi-sphere point to the current mouse-hemi-sphere point.
- Calculate the normal to this plane. This will be the axis about which to rotate.

</td>
<td>

```
    //
    // Rotate about the axis that is perpendicular to the great circle connecting the
    mouse movements.
    //
    Vec3f rotAxis;
    rotAxis.crossProd( lastPoint, curPoint );
    rot_angle = velocity * m_ROTSCALE;
```

</td>
</tr>
</table>

Apply GL Rotation

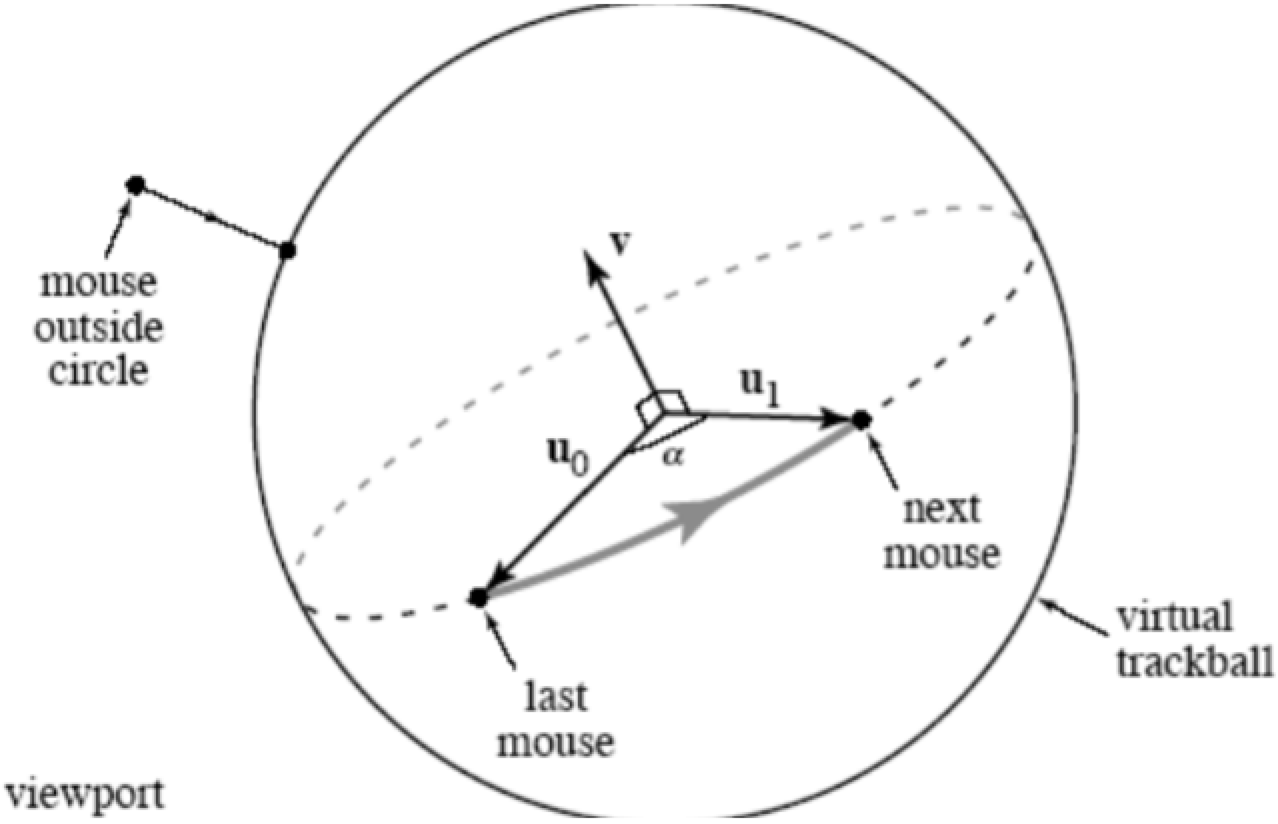
- Very important: the order!

- Read off the current matrix, since we want this operation to be the last transformation, not the first, and OpenGL does things LIFO.
- Reset the model-view matrix to the identity
- Rotate about the axis
- Multiply the resulting matrix by the saved matrix.

```
//
// We need to apply the rotation as the last transformation.
// 1. Get the current matrix and save it.
// 2. Set the matrix to the identity matrix (clear it).
// 3. Apply the trackball rotation.
// 4. Pre-multiply it by the saved matrix.
//
glGetFloatv( GL_MODELVIEW_MATRIX, (GLfloat *) objectXform
);
glLoadIdentity();
glRotatef( rot_angle, rotAxis.x, rotAxis.y, rotAxis.z );
glMultMatrixf( (GLfloat *) objectXform );
```

Virtual Trackball

- Visualization of the algorithm



Other Interactions?

- Translation?
- Scale?
- Order Matters!

GLUT UI Functions

- void **glutMouseFunc** (void (*func)(int button, int state, int x, int y));

  // sets the mouse callback for the *current window*.
- void **glutMotionFunc** (void (*func)(int x, int y));

  // set the motion callbacks respectively for the *current window*.
- void **glutMouseWheelFunc** (void(*func)(int wheel, int direction, int x, int y));

  // Sets the mouse wheel callback for the current window.
- void **glutKeyboardFunc** (void (*func)(unsigned char key, int x, int y));

  // sets the keyboard callback for the *current window*.
- void **glutSpecialFunc** (void (*func)(int key, int x, int y));

  // sets the special keyboard callback for the *current window*.

Programming Assignment 2

- I’ll post on course website this afternoon
- I’ll provide skeleton code and OBJ reader
- You need to implement basic 3D interactions
  - Trackball
  - Translation
  - Scaling
- Due on 10/10 midnight (11:59pm)