3D Transformations

Lecture 6

CISC 440/640

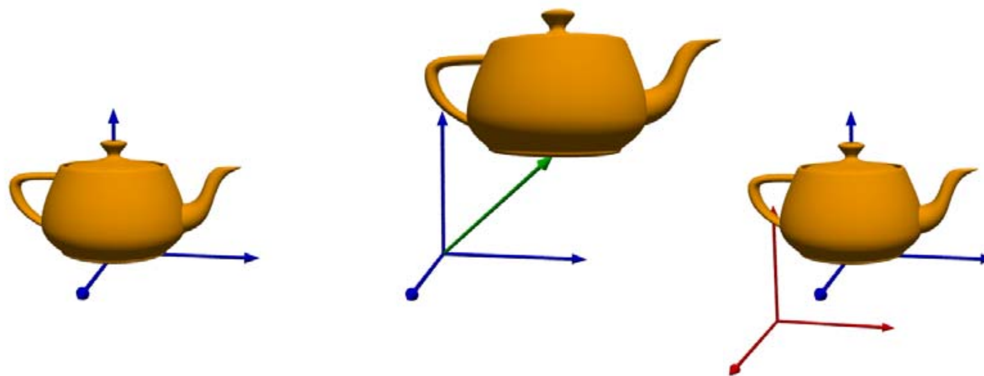
Spring 2015

Translations

- We can *translate* points in given coordinate frame by adding offsets to their coordinates. This can be interpreted as moving the point or changing coordinate frames.

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

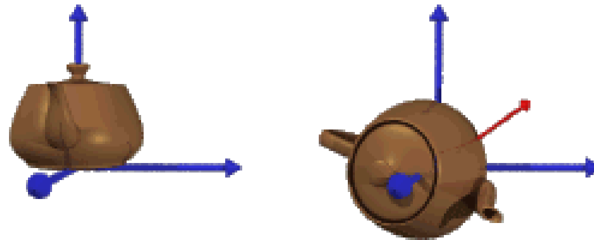
The effect of this translation is something like



COMPUTER GRAPHICS UD

3D Rotations

- Rotations in 3D are considerably more complicated than 2D rotations. In general, rotations are specified by a rotation axis and an angle. In 2D there is only one choice of a rotation axis that leaves points in the plane.



- There are several different ways to present rotations. I will use a different approach than that used in most books. Typically, all possible rotations are treated as the composition of three canonical rotations, one about the x-axis, one about the y-axis and one about the z-axis. In order to use this model you need to do the following. Memorize the three canonical rotations, which aside from the signs of the sines, isn't too hard. Next you have to go through a series of rotations which move the desired rotation axis onto one of your canonical rotations, and then you have to rotate it back without introducing any extraneous twists. **This is a difficult and error-prone process. But worst of all it is ambiguous. There exist several different combinations canonical rotations that result in the same overall result.**

Decomposing a Rotation

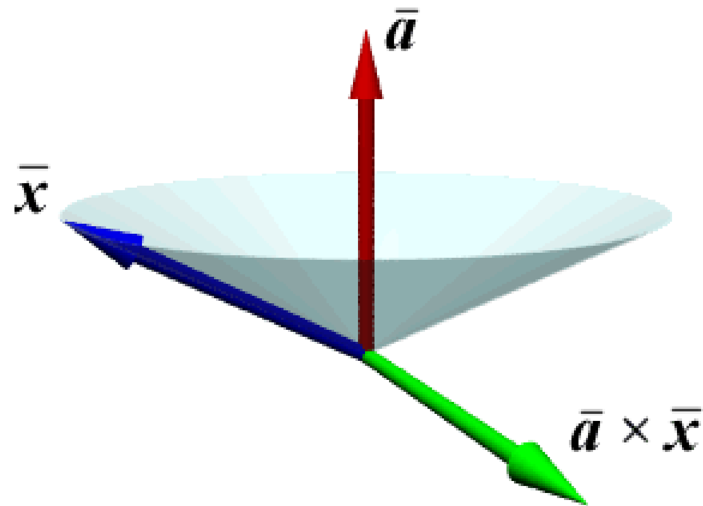
- Here is another way that is both easier to use and provides you with more insights into what rotation is really about. Instead of specifying a rotation by a series of canonical angles, instead we will specify an arbitrary axis of rotation and an angle.

$$\mathbf{c}' = \left(\mathbf{Symmetric}(\vec{a})(1-\cos\theta) + \mathbf{Skew}(\vec{a})\sin\theta + \mathbf{I}\cos\theta\right)\mathbf{c}$$

- The vector *a* specifies the axis of rotation. *This axis vector must be normalized.* The rotation angle is given by θ.
- You might ask "How am I going to remember **this** equation?". However, once you understand the geometry of rotation, the equation will seem obvious.
- The first basic idea is that *any rotation can be decomposed into weighted contributions from three different vectors*.

Geometry of a Rotation

- We can actually define a *natural* basis for rotation in terms of three defining vectors. These vectors are the rotation axis, a vector perpendicular to both the rotation axis and the vector being rotated, and the vector itself. These vectors correspond to the three terms in the rotation expression.



- Let's examine each term

The Symmetric Matrix

- The symmetric matrix of a vector generates the vector component in the direction of the axis.

$$\textbf{Symmetric}(\vec{a}) = \begin{bmatrix} a_x \\ a_y \\ a_z \\ 0 \end{bmatrix} \begin{bmatrix} a_x & a_y & a_z & 0 \end{bmatrix} = \begin{bmatrix} a_x^2 & a_x a_y & a_x a_z & 0 \\ a_x a_y & a_y^2 & a_y a_z & 0 \\ a_x a_z & a_y a_z & a_z^2 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

(Annotation: the 0 in the last row and column is crossed out with a red X and a red "1" is written next to it.)

We also need to fix up this matrix so that it preserves affine spaces

- When applied to a vector it can be considered as scaling by the length of the projection of $\vec{a}$ in the direction of $\dot{x}$.

$$\textbf{Symmetric}(\vec{a})\dot{x} = \begin{bmatrix} a_x \\ a_y \\ a_z \\ 0 \end{bmatrix} \begin{bmatrix} a_x & a_y & a_z & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \vec{a}(\vec{a} \cdot \dot{x})$$

The Skew Symmetric Matrix

- We know how the skew symmetric matrix of a vector generates a vector that is perpendicular to both the axis and it's input vector. It is just a cross product.

$$\mathbf{Skew}(\vec{a}) = \begin{bmatrix} 0 & -a_z & a_y & 0 \\ a_z & 0 & -a_x & 0 \\ -a_y & a_x & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{Skew}(\vec{a})\dot{x} = \vec{a} \times (\dot{x} - \dot{0})$$

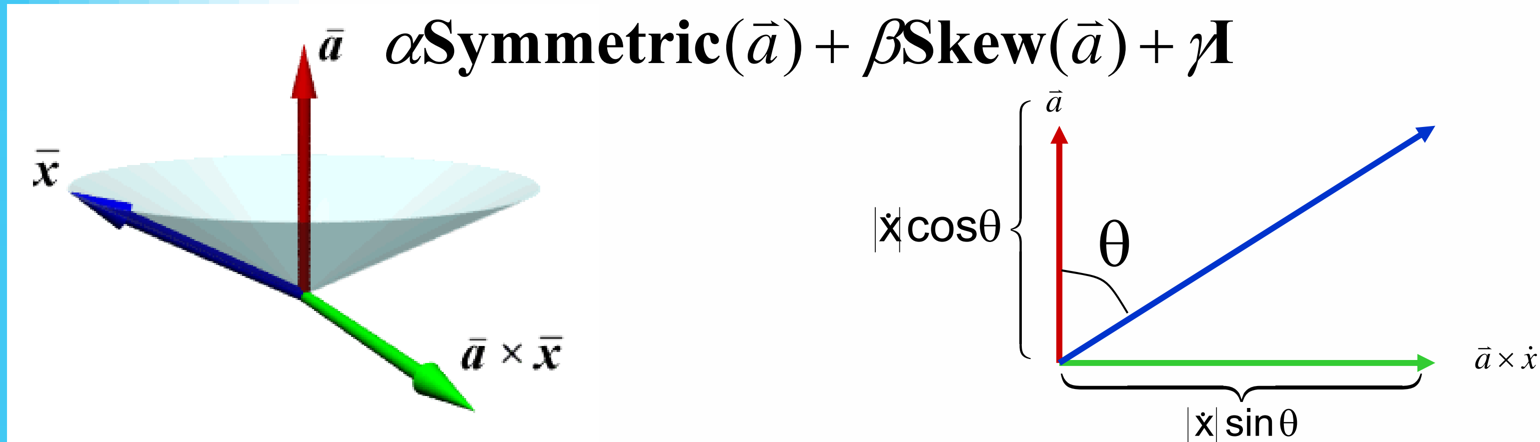
- The identity matrix of the third term generates a vector in the same direction as the input vector.

Computer Graphics

Weighting Factors

- Weighting factors combine the 3 component vectors

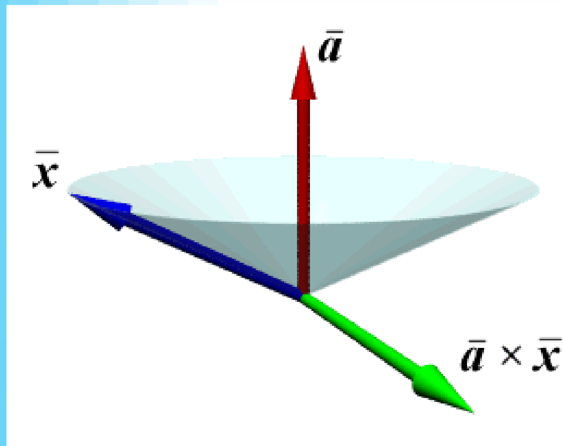
$$\alpha \textbf{Symmetric}(\vec{a}) + \beta \textbf{Skew}(\vec{a}) + \gamma \textbf{I}$$

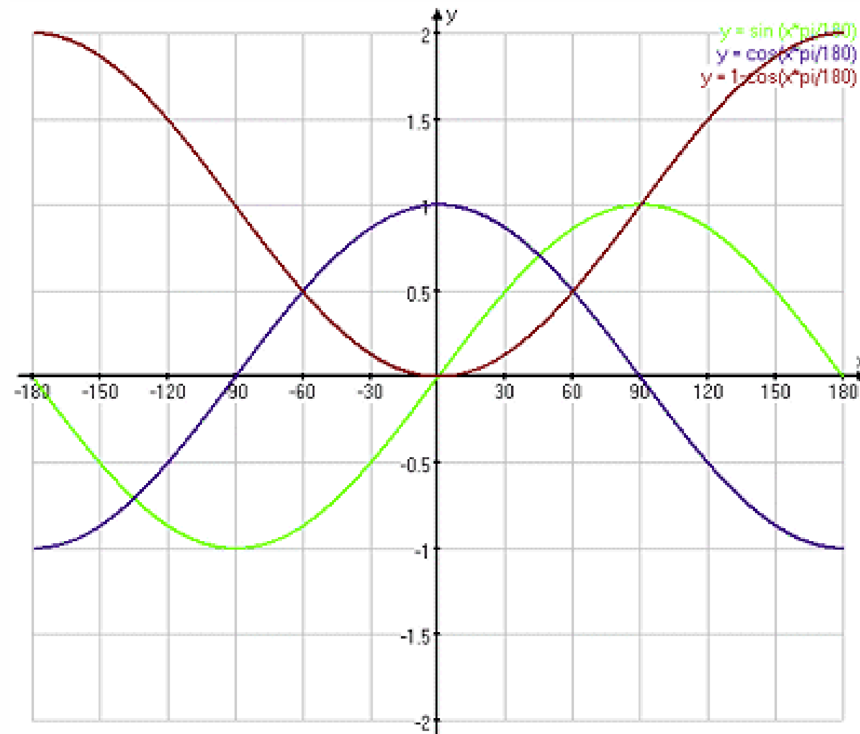


- When θ = 0 then the sum should reduce to identity (α = 0, β = 0, γ = 1)
- When θ = 90 the symmetric and skew parts should be 1, (α = 1, β = 1, γ = 0)
- When θ = -90 the symmetric and skew parts should be (α = 1, β = -1, γ = 0)
- When θ = 180 then the weights are (α = 2, β = 0, γ = -1)
- The result after a rotation will always be in the same hemisphere relative to as the axis as the original point. Thus, $\alpha \geq 0$.

Pulling it all together

- The weights vary periodically (sinusoids)
- The rotated point moves along a circle perpendicular to the axis. Thus, $\beta = \sin\theta$





- Similarly, we can reason about the remaining weighting factors

Sanity Check

- Consider a rotation by about the x-axis.

$$\text{Rotate}\left(\begin{bmatrix}1\\0\\0\end{bmatrix},\theta\right)=\begin{bmatrix}1&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&1\end{bmatrix}(1-\cos\theta)+\begin{bmatrix}0&0&0&0\\0&0&-1&0\\0&1&0&0\\0&0&0&0\end{bmatrix}\sin\theta+\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}\cos\theta$$

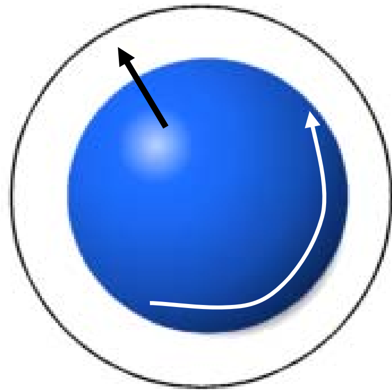
$$\text{Rotate}\left(\begin{bmatrix}1\\0\\0\end{bmatrix},\theta\right)=\begin{bmatrix}1&0&0&0\\0&\cos\theta&-\sin\theta&0\\0&\sin\theta&\cos\theta&0\\0&0&0&1\end{bmatrix}$$

- You can check it in any computer graphics book, but you don’t need to memorize it.

COMPUTER GRAPHICS

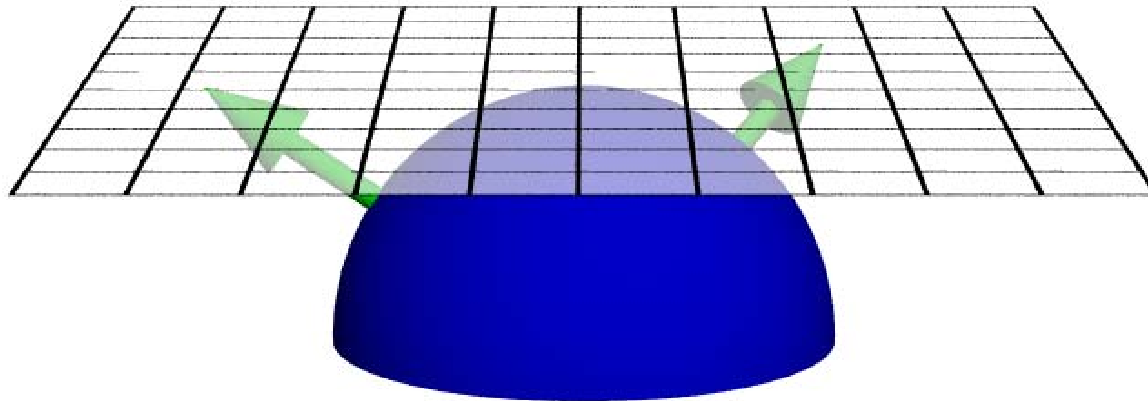
Exercise: A Trackball Interface

- A common UI for manipulating objects
- Virtual trackball
- 2 degree of freedom device
- However, its 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?



Some help with virtual trackball

- http://www.cse.ohio-state.edu/~crawfis/cis781/Slides/VirtualTrackball.html
- Map mouse to the sphere

- 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

• Detect the mouse movement	`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.`
	`{`

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

COMPUTER GRAPHICS

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 );
```

Computer Graphics