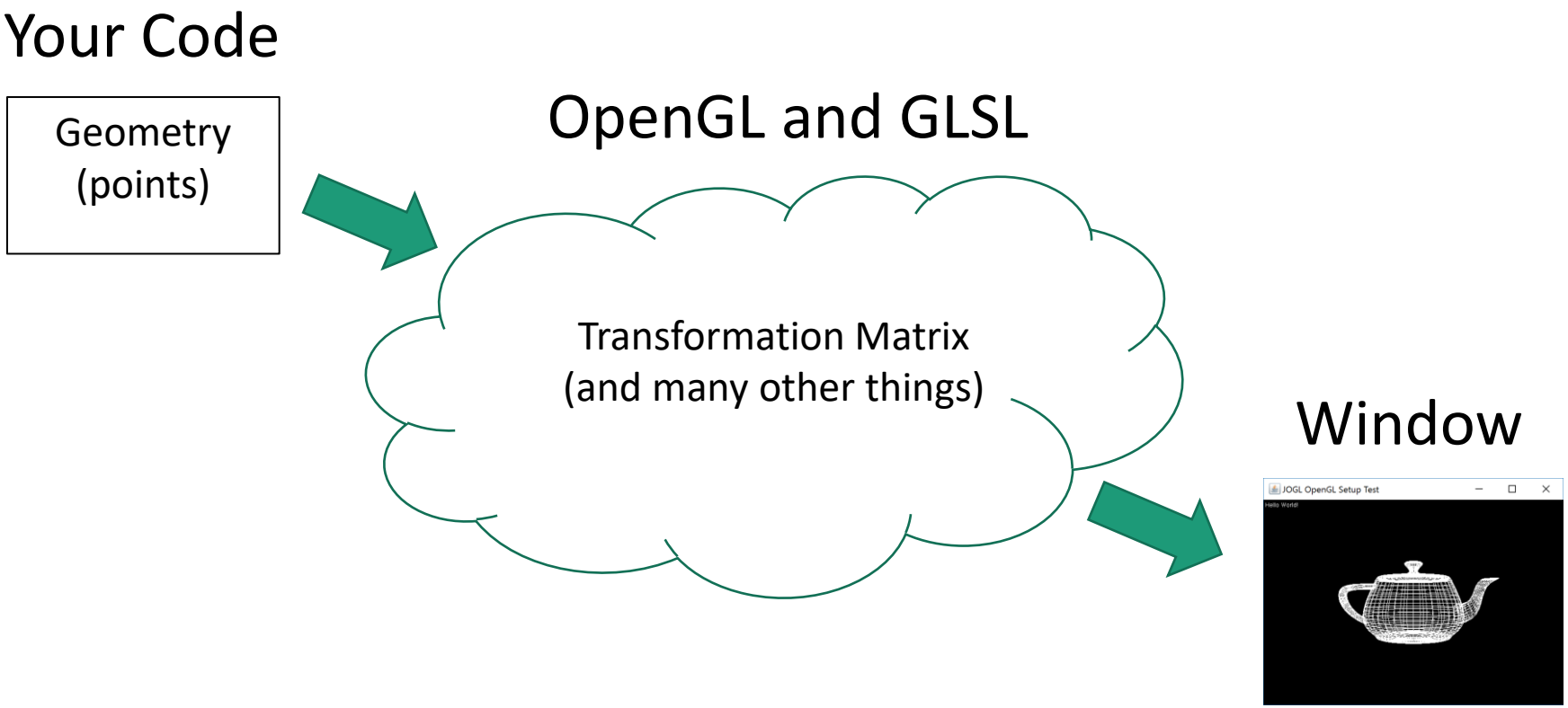
3D Transformations

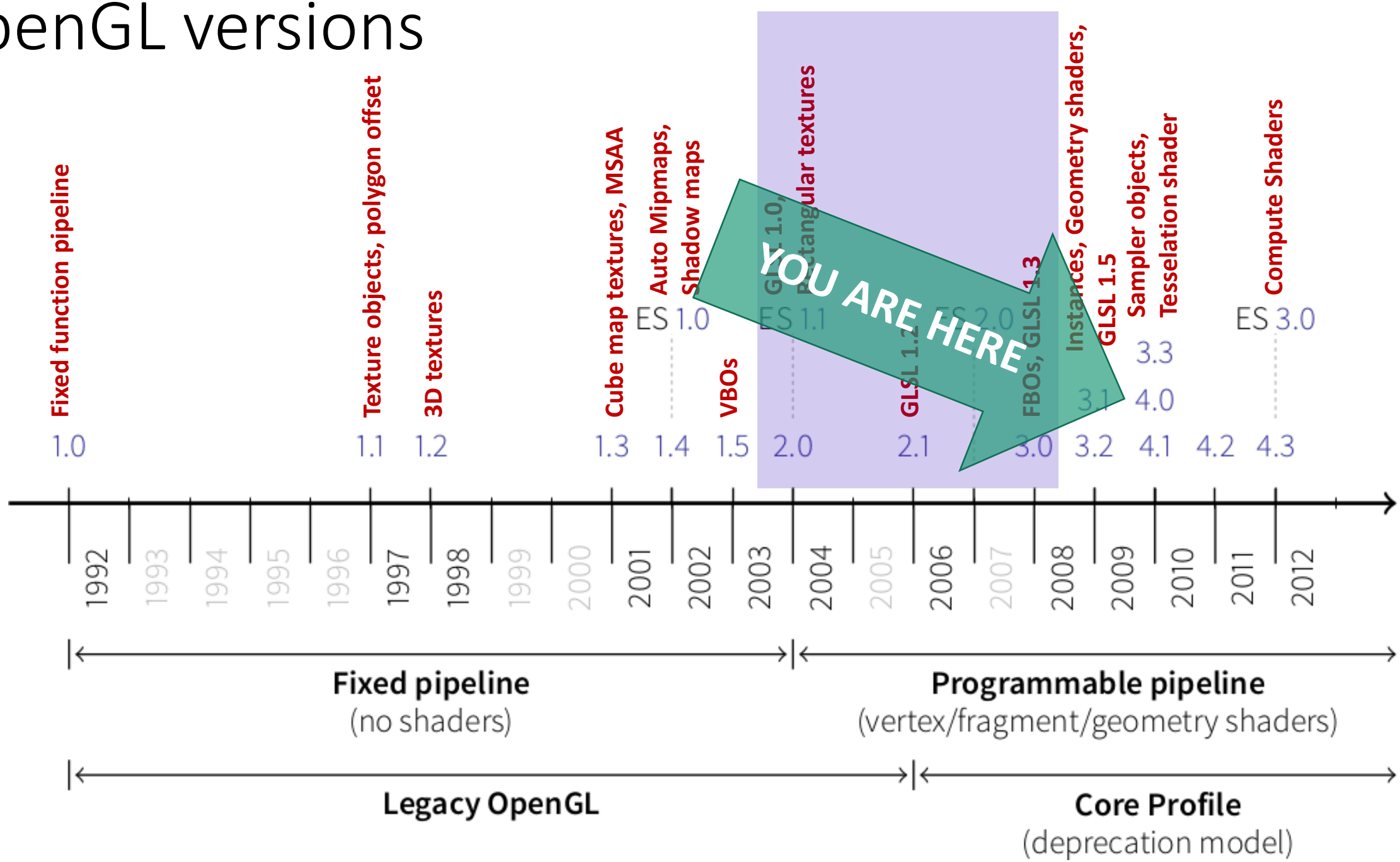
COMP557

Paul Kry

Transformations in OpenGL



OpenGL versions



Let's look at the basic Java + JOGL GL4 code…

```
public class A0App implements GLEventListener {

    public static void main(String[] args) {
        new A0App();
    }

    public A0App() {
        GLProfile glprofile = GLProfile.get( GLProfile.GL4 );
        GLCapabilities glcapabilities = new GLCapabilities( glprofile );
        GLCanvas glcanvas = new GLCanvas( glcapabilities );
        glcanvas.addGLEventListener(this);
        FPSAnimator animator;
        animator = new FPSAnimator(glcanvas, 60);
        animator.start();
        final JFrame jframe = new JFrame( "JOGL OpenGL Setup Test" );
        jframe.addWindowListener( new WindowAdapter() {
            public void windowClosing( WindowEvent windowevent ) {
                jframe.dispose();
                System.exit( 0 );
            }
        });
        jframe.getContentPane().add( glcanvas, BorderLayout.CENTER );
        jframe.setSize( 500, 500 );
        jframe.setVisible( true );
    }
```

Interface for GL init and display callbakcs

Use default GL4 capabilities (rare need for anything but the defaults)

Set up regular callbacks

Set the frame size and make it visible

```
    drawable.setGL(new DebugGL4(drawable.getGL().getGL4()));
    GL4 gl = drawable.getGL().getGL4();
    gl.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    gl.glClearDepth(1.0f);
    gl.glEnable(GL.GL_DEPTH_TEST);
    gl.glDepthFunc(GL.GL_LEQUAL);
    gl.glEnable( GL4.GL_BLEND );
    gl.glBlendFunc( GL4.GL_SRC_ALPHA, GL4.GL_ONE_MINUS_SRC_ALPHA );
    gl.glEnable( GL4.GL_LINE_SMOOTH );

    // Create the GLSL program
    glslProgramID = createProgram( drawable, "minimal" );
    // Get the IDs of the parameters (i.e., uniforms and attributes)
    gl.glUseProgram( glslProgramID );
    MMatrixID = gl.glGetUniformLocation( glslProgramID, "M" );
    VMatrixID = gl.glGetUniformLocation( glslProgramID, "V" );
    PMatrixID = gl.glGetUniformLocation( glslProgramID, "P" );
    positionAttributeID = gl.glGetAttribLocation( glslProgramID, "position" );

    // Initialize the vertex and index buffers
    FloatBuffer vertexBuffer = GLBuffers.newDirectFloatBuffer(positionData);
    ShortBuffer elementBuffer = GLBuffers.newDirectShortBuffer(elementData);
    int[] bufferIDs = new int[2];
    gl.glGenBuffers( 2, bufferIDs, 0 );
    positionBufferID = bufferIDs[0];
    elementBufferID = bufferIDs[1];
    gl.glBindBuffer( GL4.GL_ARRAY_BUFFER, positionBufferID );
    gl.glBufferData( GL4.GL_ARRAY_BUFFER, vertexBuffer.capacity() * Float.BYTES, vertexBuffer, GL4.GL_STATIC_DRAW );
    gl.glBindBuffer( GL4.GL_ELEMENT_ARRAY_BUFFER, elementBufferID );
    gl.glBufferData( GL4.GL_ELEMENT_ARRAY_BUFFER, elementBuffer.capacity() * Short.BYTES, elementBuffer, GL4.GL_STATIC_DRAW );
}
```

Force error checking on every OpenGL call, very useful for debugging

Clear with RGB 0 (black), 1 is the alpha (we cover transparency later)

Set up how the depth buffer works (more later!)

Prettier lines with GL_BLEND and GL_LINE_SMOOTH

Create the GLSL program (next slide) and get ID of things that we want to set

Create buffers for our vertex data and triangles (elements) and put that data into OpenGL buffers.

Minimal GLSL Program

GL Shading Language (GLSL) runs on the graphics card and has data types and methods useful for doing computer graphics. Vertex programs work with vertex data, and can pass information to the fragment program (not in this case though). Fragment programs do the computations for individual pixels. We will see more soon!

Vertex Program minimal.vp

```
#version 400

uniform mat4 P;
uniform mat4 V;
uniform mat4 M;

in vec4 position;

void main() {
   gl_Position = P * V * M * position;
}
```

Fragment Program minimal.fp

```
#version 400

out vec4 fragColor;

void main(void) {
   fragColor = vec4(1,0,0,1);
}
```


Based on slides by Steve Marschner

```
/** Model matrix */
private Matrix4f MMatrix = new Matrix4f();
/** View matrix */
private Matrix4f VMatrix = new Matrix4f();
/** Projection matrix */
private Matrix4f PMatrix = new Matrix4f();

@Override
public void display( GLAutoDrawable drawable ) {
    GL4 gl = drawable.getGL().getGL4();
    gl.glClear( GL4.GL_COLOR_BUFFER_BIT | GL4.GL_DEPTH_BUFFER_BIT );

    MMatrix.rotX( System.nanoTime()/1e9f );

    glUniformMatrix( gl, MMatrixID, MMatrix );
    glUniformMatrix( gl, VMatrixID, VMatrix );
    glUniformMatrix( gl, PMatrixID, PMatrix );

    gl.glUseProgram( glslProgramID );

    gl.glEnableVertexAttribArray( positionAttributeID );
    gl.glBindBuffer( GL4.GL_ARRAY_BUFFER, positionBufferID );
    gl.glVertexAttribPointer( positionAttributeID, 3, GL4.GL_FLOAT, false, 3*Float.BYTES, 0 );
    gl.glBindBuffer( GL4.GL_ELEMENT_ARRAY_BUFFER, elementBufferID );
    gl.glPolygonMode( GL4.GL_FRONT_AND_BACK, GL4.GL_LINE ); // otherwise GL_FILL is the default
    gl.glDrawElements( GL4.GL_TRIANGLES, elementData.length, GL4.GL_UNSIGNED_SHORT, 0 );
    gl.glDisableVertexAttribArray( positionAttributeID );
}
```

We will see how to set up these matrices in the next classes

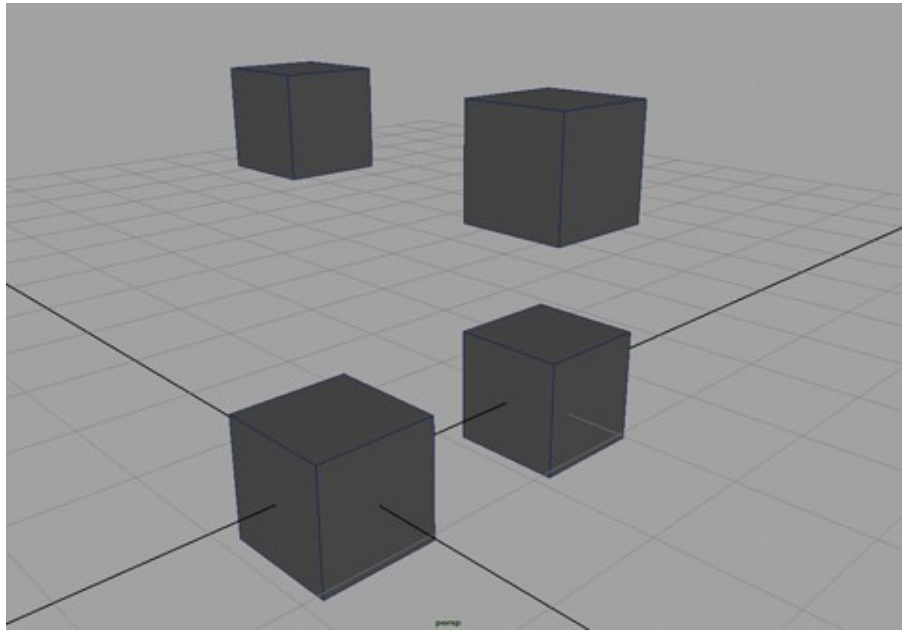
Experiment with the Modelling matrix!

Pass the matrices to OpenGL as “uniform” data, and note that it is stored column major form!

Bind the buffers, and call DrawElements to draw the triangles, using the specified GLSL program!

Translation ~~glTranslated(x,y,z)~~

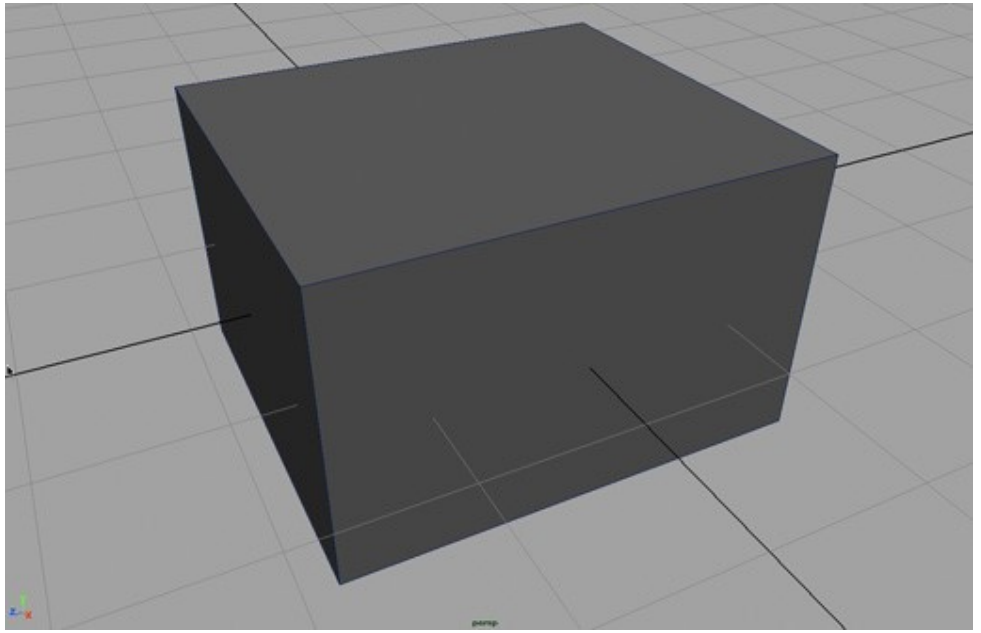
$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$



Based on slides by Steve Marschner

Scaling ~~glScaled(x, y, z)~~

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$



Based on slides by Steve Marschner

Rotation ~~glRotate(deg, x, y, z)~~

- For example, rotation about the z axis

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Based on slides by Steve Marschner

Rotation ~~glRotate(deg, x, y, z)~~

- For example, rotation about the x axis

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

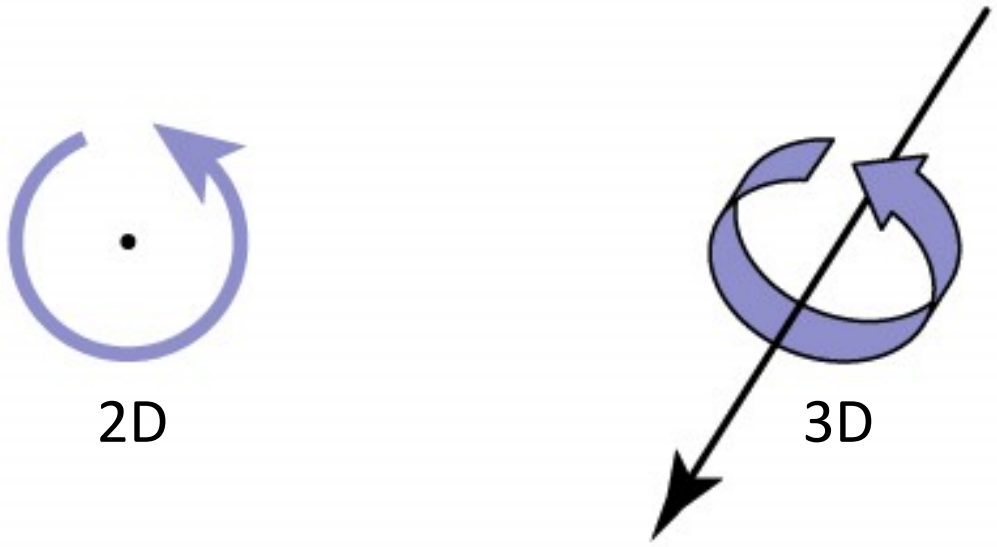
Rotation ~~glRotate(deg, x, y, z)~~

- For example, rotation about the y axis

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

General rotations

- A rotation in 2D is around a point
- A rotation in 3D is around an axis
  - 3D rotation is w.r.t a line through origin, not just about the origin point
  - There are many more 3D rotations than 2D



2D 3D

In 3D, one way to specify a rotation is via a unit vector (the axis) and an angle. Right hand convention: positive rotation is CCW when vector points toward you.

Based on slides by Steve Marschner

Question

- Do any pairs of 3D scaling, rotation, and translation always commute?
- Under what conditions does scaling commute with 3D rotation?
- Under what conditions does translation commute with 3D rotation?

Other Questions

- Derive a matrix for a reflection in the plane containing the origin and having normal n
- Derive a matrix for a reflection in the plane containing point p_0 and having normal n

What are 3D rotations? More precisely…

- 3D Rotations are the set of linear transformations
$$\{R \in \mathbb{R}^{3\times3} : R^T = R^{-1}, \det(R) = +1\}$$
which form a group with matrix multiplication as the binary group operation (i.e., operation that combines two elements)
  - Closed under the group operation
  - Exist identity element (the identity matrix)
  - Exist an inverse for every group element
  - Associative, $A(BC) = (AB)C$
- This group is the "special orthogonal group" $SO(3)$.
- Each different 3D orientation is specified only once in this group.

Based on slides by Steve Marschner

Specifying rotations

- In 2D, a rotation just has an angle
- How many ***Degrees Of Freedom***, i.e., DOFs?
  - $R^T = R^{-1}, \ \det(R) = +1$ does not perhaps give much intuition, but
  - Knowing that an ***angle*** about an ***axis direction*** lets us specify any rotation with 3 DOFs
    - This is how you specify a rotation with glRotate in OpenGL

Based on slides by Steve Marschner

Specifying Rotations, Euler Angles

- Can specify a rotation matrix with Euler angles
  - Stack up three coordinate axis rotations, for instance

$$R = R_x(\theta_x)R_y(\theta_y)R_z(\theta_z)$$

    - Can see z rotation as being about the **object frame**
    - Can see x rotation as being about the **canonical frame**
    - The y rotation is about an intermediate frame
  - Why not another order? (more on this shortly)
  - Also, problem of gimbal lock !! (more on this shortly)

Euler's Theorem

- Any two independent orthonormal coordinate frames can be related by a sequence of rotations (not more than three) about coordinate axes, where no two successive rotations may be about the same axis.
  - Have 12 possible sequences!

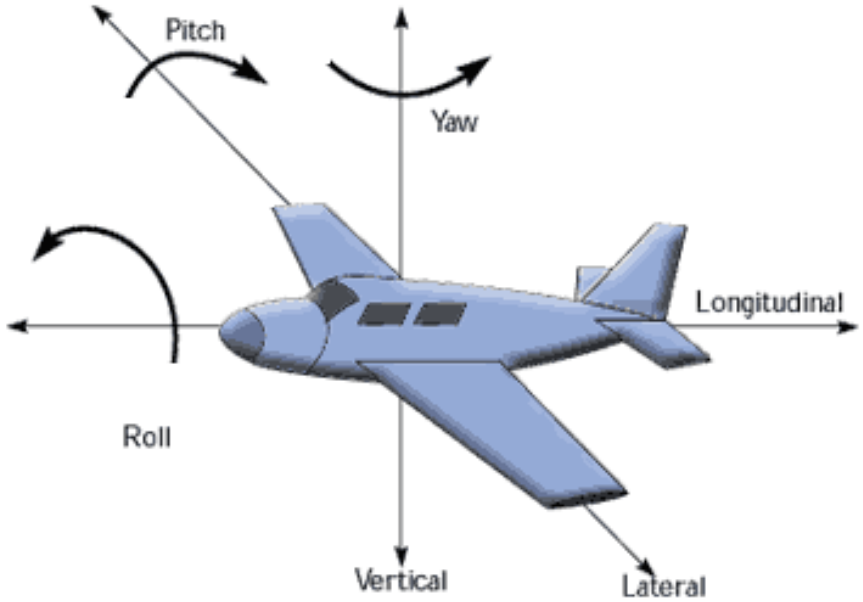
XYZ	**XZY**	**XYX**	**XZX**	**YXZ**	**YZX**
YXY	**YZY**	**ZXY**	**ZYX**	**ZXZ**	**ZYZ**

- Given some Euler angles, not knowing the order or if they map frame A to B or vice versa, then there is 24 possibilities!

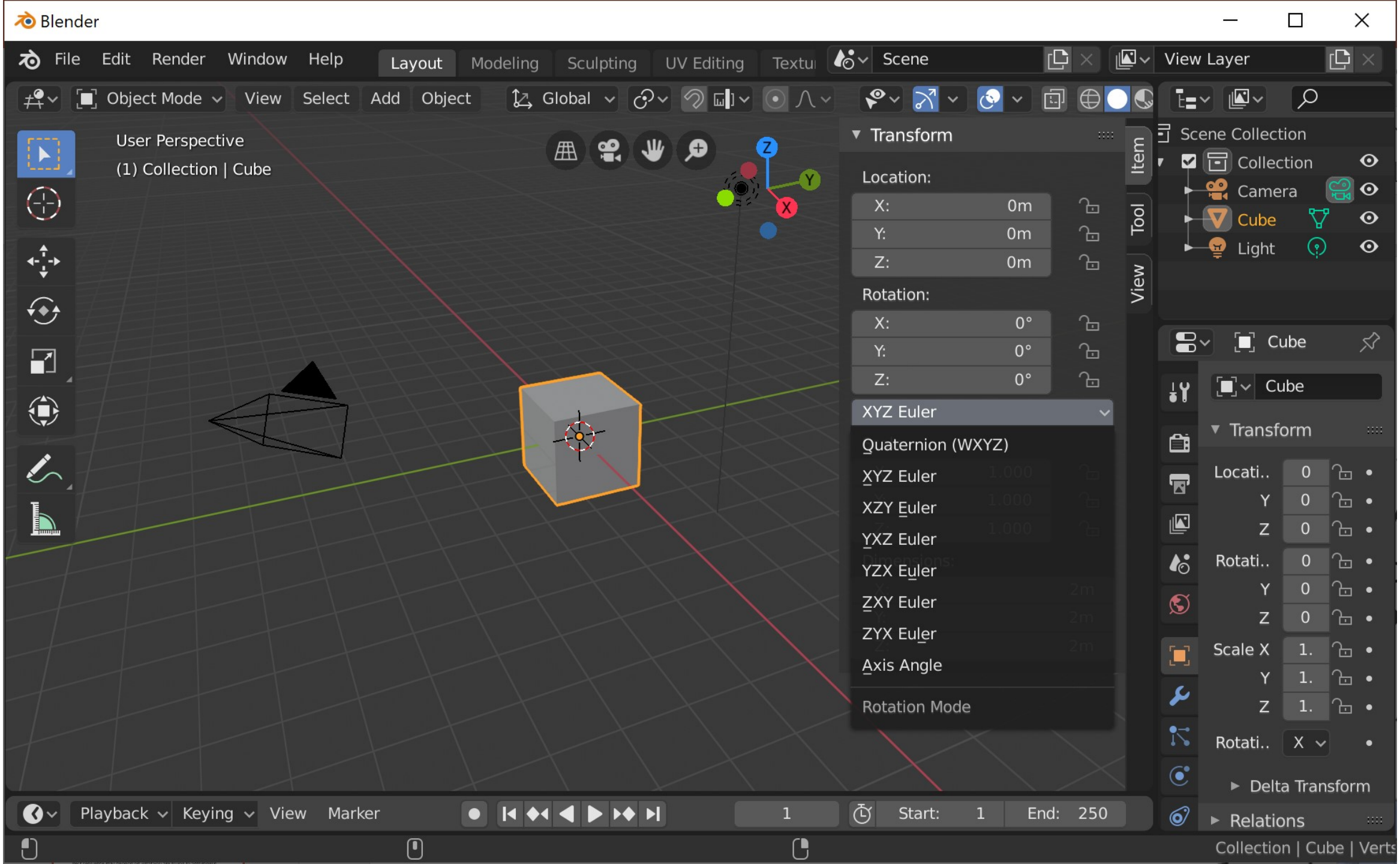
Based on slides by Steve Marschner

Euler Angles

- There are no conventions, different orders in different industries, and order is often customizable (e.g., in software like Maya, Blender).
- There may be some practical differences between orderings and the best sequence may depend on what you are trying to accomplish.
  - In situations where there is a definite ground plane, **Euler angles can actually be an intuitive representation** (e.g., roll pitch yaw of a vehicle)

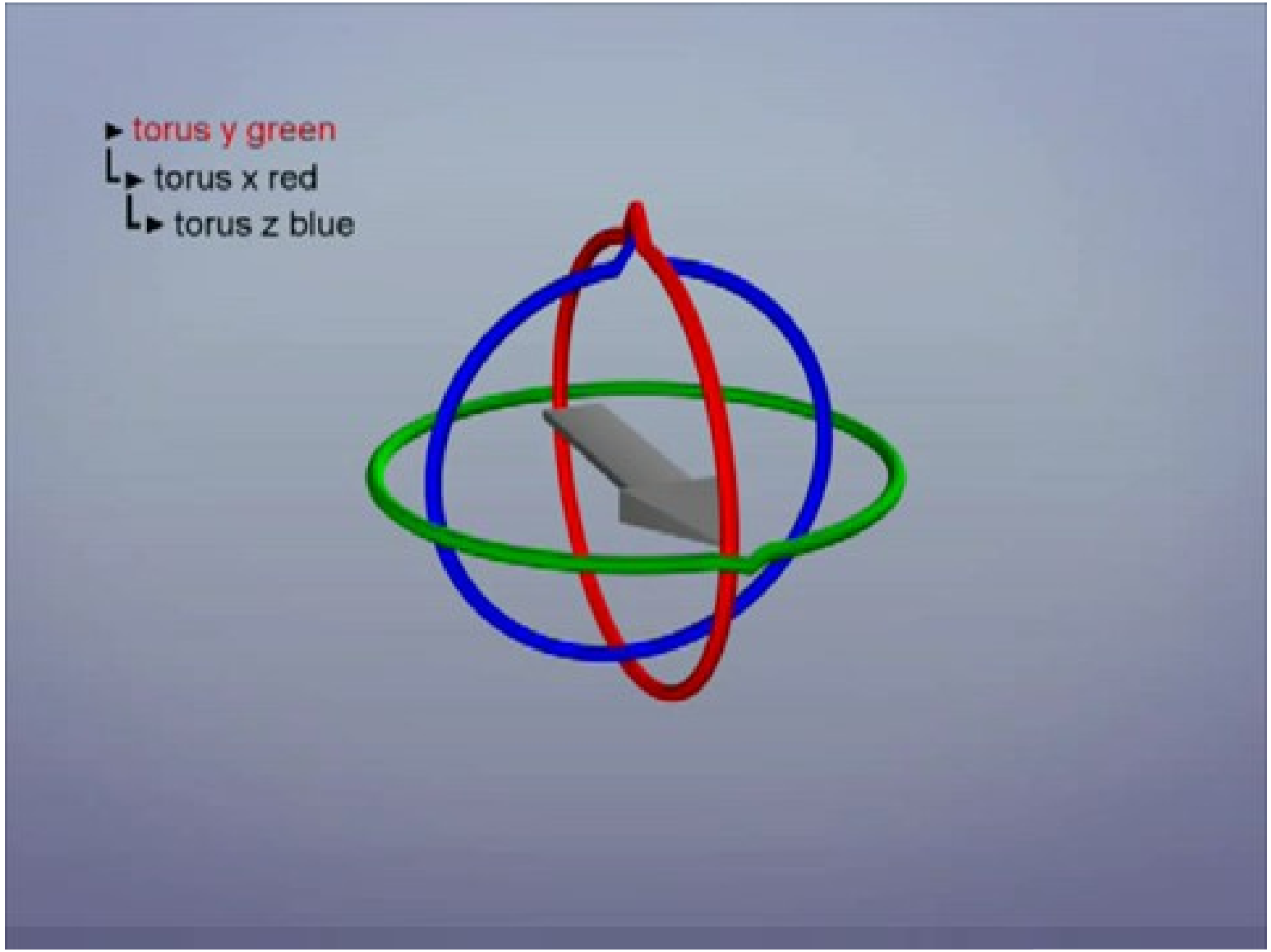


Based on slides by Steve Marschner


Blender
File Edit Render Window Help
Layout Modeling Sculpting UV Editing
Scene
View Layer
Object Mode View Select Add Object
Global
User Perspective
(1) Collection | Cube
Transform
Location:
X: 0m
Y: 0m
Z: 0m
Rotation:
X: 0°
Y: 0°
Z: 0°
XYZ Euler
Quaternion (WXYZ)
XYZ Euler
XZY Euler
YXZ Euler
YZX Euler
ZXY Euler
ZYX Euler
Axis Angle
Rotation Mode
Item
Tool
View
Scene Collection
Collection
Camera
Cube
Light
Cube
Transform
Locati..
Rotati..
Scale X
Delta Transform
Relations
Playback Keying View Marker
1
Start: 1
End: 250
Collection | Cube | Verts

http://www.youtube.com/watch?v=zc8b2Jo7mno

Euler Angles – Gimbal Lock



d on slides by Steve Marschner

Interpolating Euler Angles

- One can simply interpolate between the three values independently
- This will result in the interpolation following a different path depending on which of the 12 schemes you choose
- This may or may not be a problem, depending on your situation
- Interpolating near singularities is problematic
- Note: when interpolating angles, remember to check for crossing the +180/-180 degree boundaries

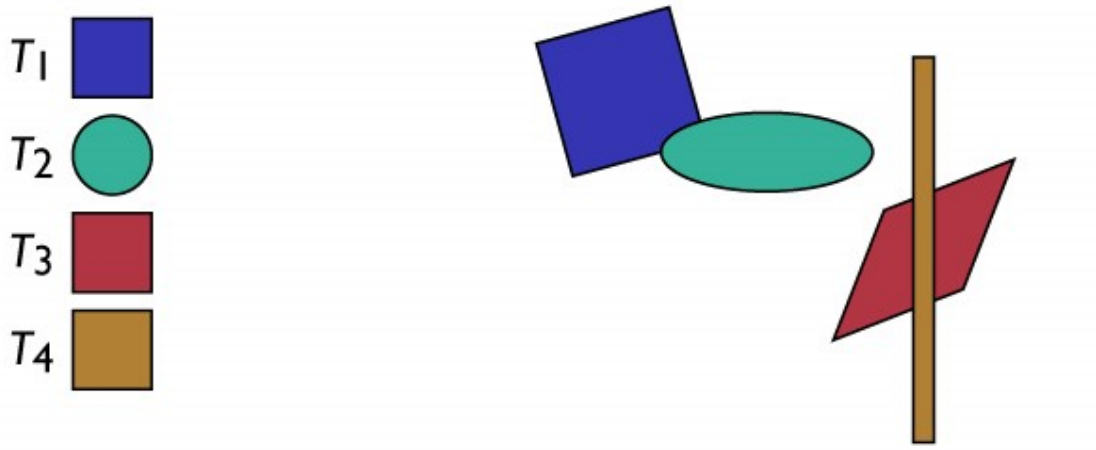
Euler Angles - Summary

- Euler angles are used in a lot of applications, but they tend to require some rather arbitrary decisions
- They also do not interpolate in a consistent way (but this isn't always bad)
- They suffer from Gimbal lock and related problems
- Two rotations are combined by multiplication, not by addition of the Euler Angles!
- Conversion to/from a matrix requires several trigonometry operations
- They are compact (requiring only 3 numbers)

Scene Graphs

Data structures with transforms

- Representing a drawing (“scene”)
- List of objects
- Transform for each object
  - Can use minimal primitives: ellipse is transformed circle
  - Transform applies to points of object



Based on slides by Steve Marschner

Example

- Can represent drawing with flat list
  - But editing operations require updating many transforms

Based on slides by Steve Marschner

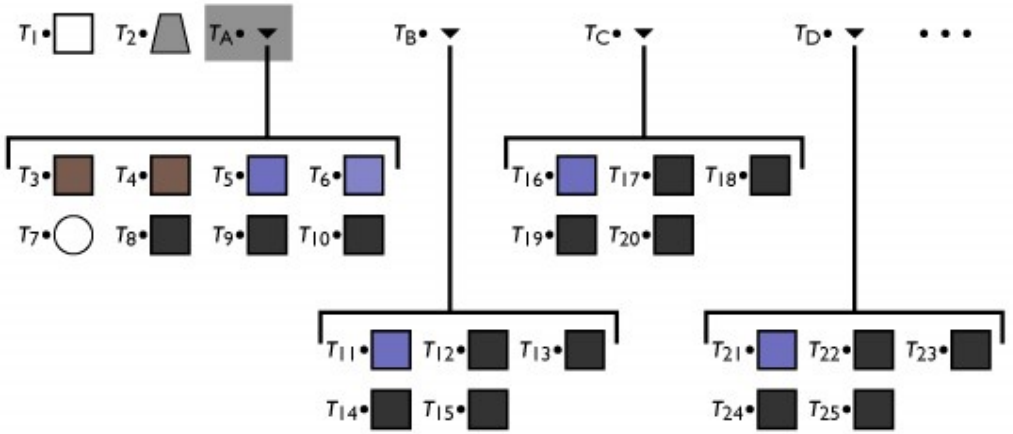
Groups of objects

- Treat a set of objects as one
- Introduce new object type: group
  - contains list of references to member objects
- This makes the model into a tree
  - Interior nodes = groups
  - Leaf nodes = objects
  - Edges = membership of object in group

Based on slides by Steve Marschner

Example

- Add group as a new object type
  - Lets the data structure reflect the drawing structure
  - Enables high-level editing by changing just one node

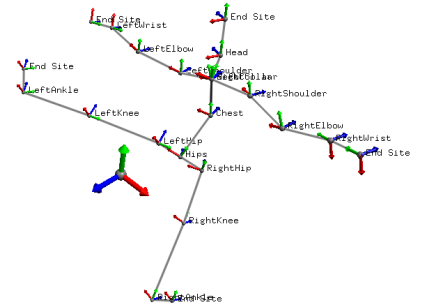


Based on slides by Steve Marschner

Another example… BVH motion capture

- Hierarchy definition
  - Offset gives child in parent frame
  - Channels specify degrees of freedom
  - End sites provide an indication of the extent of the last limb
- Motion definition
  - Number of frames, and step size
  - All channels, in order
  - Euler angles in degrees (always?)
- It’s an old file format!
  - Units may vary (cm in this example)

```
HIERARCHY
ROOT Hips
{
   OFFSET 0.000000 0.000000 0.000000
   CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
      JOINT Chest
      {
         OFFSET 0.000000 26.452015 0.000000
         CHANNELS 3 Zrotation Xrotation Yrotation
         JOINT Neck
         {
            OFFSET 0.000000 18.643064 0.000000
            CHANNELS 3 Zrotation Xrotation Yrotation
            JOINT Head
            {
               OFFSET 0.000000 10.388890 0.000000
               CHANNELS 3 Zrotation Xrotation Yrotation
               End Site
               {
                  OFFSET 0.000000 16.055559 0.000000
               }
            }
         }
         JOINT LeftCollar
```




```
      }
   }
}
MOTION
Frames:    7931
Frame Time:    0.010000
-7.192307   92.067162   18.585640   -2.583095   -44.001091   174.348663   -3.655392   55.8
-7.192307   92.067131   18.585627   -2.583080   -44.001011   174.348694   -2.449191   55.8
```

Based on slides by Steve Marschner

```
JOINT LeftElbow
{
   OFFSET 25.525267 0.000000 0.000000
   CHANNELS 3 Zrotation Xrotation Yrotation
   JOINT LeftWrist
```

$$E_{pFromT} = T$$

$$E_{pFromTR1} = T\ R_z$$

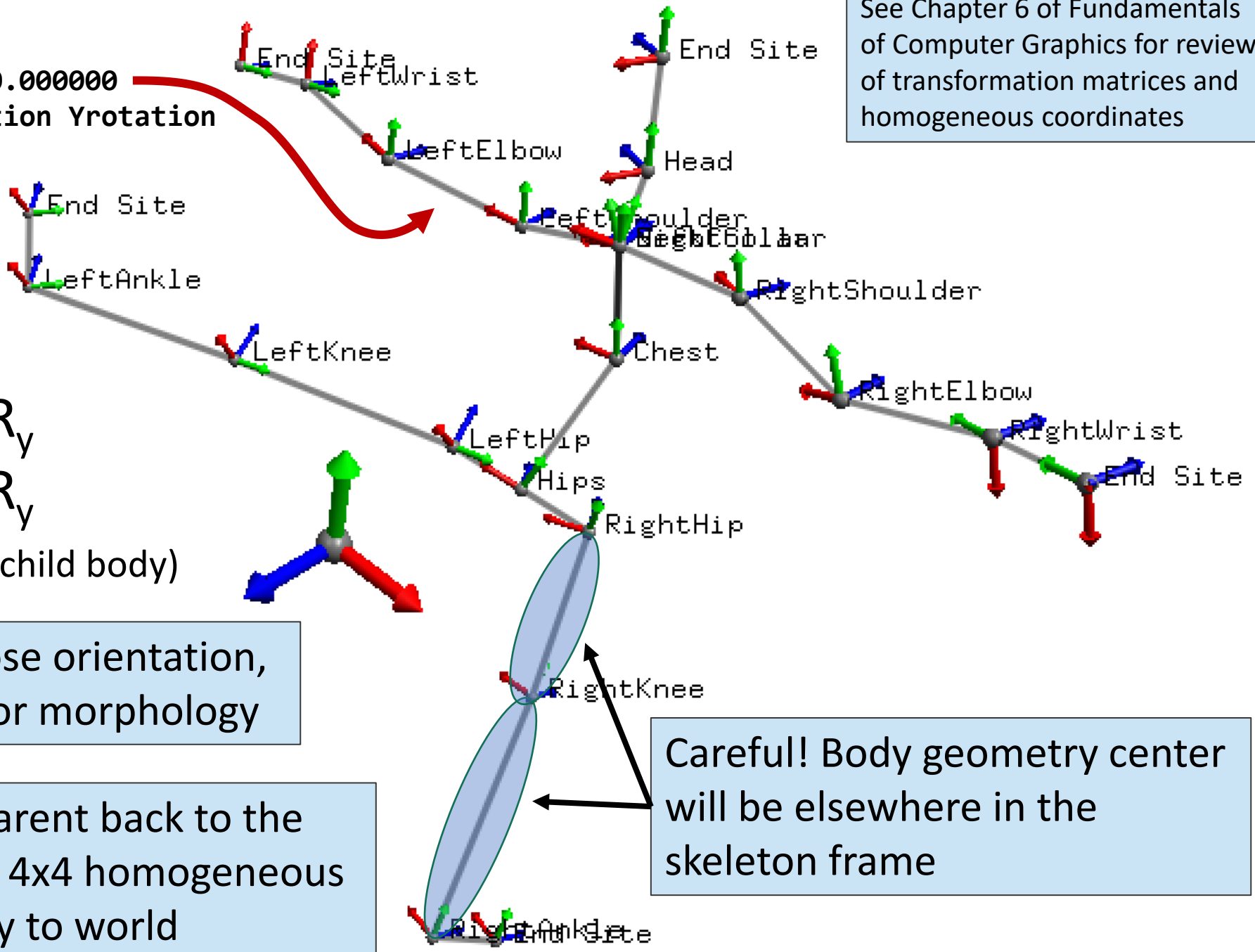
$$E_{pFromTR2} = T\ R_z\ R_x$$

$$E_{pFromTR3} = T\ R_z\ R_x\ R_y$$

$$E_{pFromb} = T\ R_z\ R_x\ R_y$$

(p here means parent, b the child body)

See Chapter 6 of Fundamentals of Computer Graphics for review of transformation matrices and homogeneous coordinates



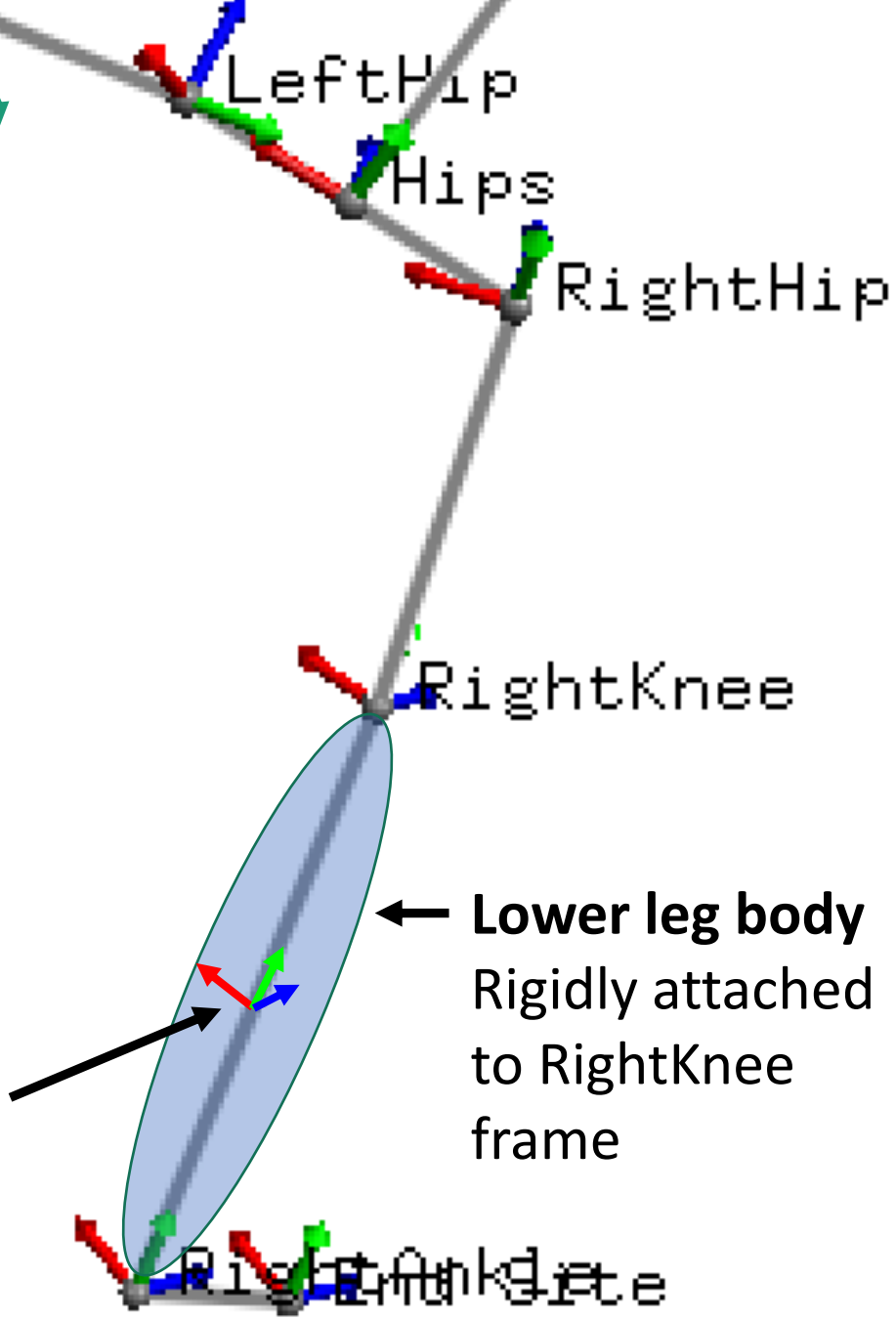
Use ***motion*** data for pose orientation, and ***hierarchy*** offsets for morphology

Careful! Body geometry center will be elsewhere in the skeleton frame

Chain together child to parent back to the world frame for E_{wfromb}, a 4x4 homogeneous transformation from body to world

Alternatively, lots of work on skeletal motion driving (driven by) elastic flesh/muscle FEM

- For animated characters approximated with a collection of ***rigid*** bodies and joints, each body part has…Size and shape (e.g., ellipsoid of semiaxes a, b, and c)
  - Center (e.g., a center of mass)

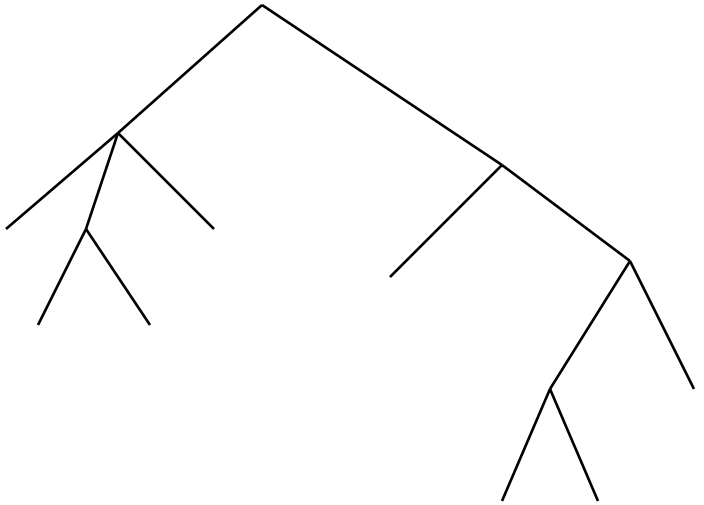


Lower leg body
Rigidly attached to RightKnee frame

Geometry frame at center of mass
Half way down the lower leg, with position easy to specify in the RightKnee coordinate frame

The Scene Graph (tree)

- A name given to various kinds of graph structures (nodes connected together) used to represent scenes
- Simplest form: tree
  - Just saw this
  - Every node has one parent
  - Leaf nodes are identified with objects in the scene



Based on slides by Steve Marschner

Concatenation and hierarchy

- Transforms associated with nodes or edges
- Each transform applies to all geometry below it
  - want group transform to transform each member
  - members already transformed—concatenate
- Frame transform for object is the product of all matrices along path from root
  - Each object's transform describes relationship between its local coordinates and its group's coordinates
  - Frame-to-canonical transform is the result of repeatedly changing coordinates from group to containing group

Based on slides by Steve Marschner

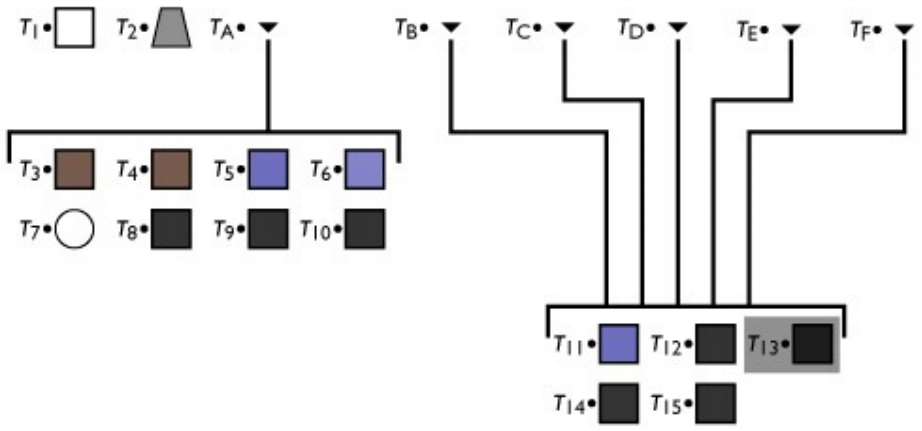
Instances

- Simple idea: allow an object to be a member of more than one group at once
  - Transform different in each case
  - Leads to linked copies
  - Single editing operation changes all instances

Based on slides by Steve Marschner

Example

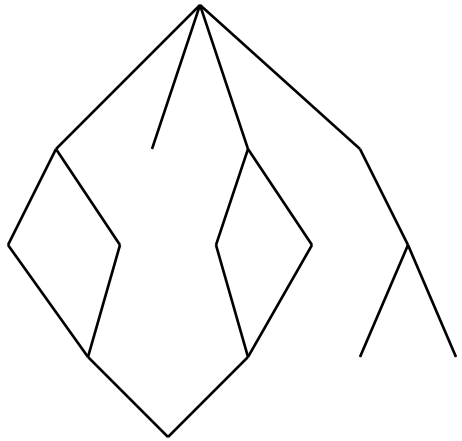
- Allow multiple references to nodes
  - Reflects more of drawing structure
  - Allows editing of repeated parts in one operation



Based on slides by Steve Marschner

The Scene Graph (with instances)

- With instances, there is no more tree
  - An object that is instanced multiple times has more than one parent
- Transform tree becomes DAG
  - **D**irected **A**cyclic **G**raph
  - Group is not allowed to contain itself, even indirectly
- Transforms still accumulate along path from root
  - Now *paths* from root to leaves are identified with scene objects



Based on slides by Steve Marschner

Implementing a hierarchy

- Object-oriented language is convenient
  - Define shapes and groups as derived from single class

```
abstract class Shape {
    void draw();
}
```

```
class Square extends Shape {
    void draw() {
        // draw unit square
    }
}
```

```
class Circle extends Shape {
    void draw() {
        // draw unit circle
    }
}
```


Based on slides by Steve Marschner

Implementing traversal

- Pass a transform down the hierarchy, and before drawing, concatenate

```
abstract class Shape {
    void draw(Transform t_c);
}
```

```
class Circle extends Shape {
    void draw(Transform t_c) {
        // draw t_c * unit circle
    }
}
```

```
class Square extends Shape {
    void draw(Transform t_c) {
        // draw t_c * unit square
    }
}
```

```
class Group extends Shape {
    Transform t;
    ShapeList members;
    void draw(Transform t_c) {
        for (m in members) {
            m.draw(t_c * t);
        }
    }
}
```


Based on slides by Steve Marschner

Implementing traversal, an example

- Modern OpenGL no longer has matrix stack, make our own!

```
abstract class Shape {
    void draw( GL4 gl, Stack<Matrix4f> s );
}
```

```
class Circle extends Shape {
    void draw( Stack<Matrix4f> s ) {
        // pass s.peek() to GLSL (to mult)
        // draw unit circle
    }
}
```

```
class Square extends Shape {
    void draw(Stack<Matrix4f> s ) {
        // pass s.peek() to GLSL (to mult)
        // draw unit square
    }
}
```

```
class Group extends Shape {
    Matrix4f T;
    ShapeList members;
    void draw( GL4 gl, Stack<Matrix4f> s ) {
        Matrix4f M = new Matrix4f();
        M.mult( s.peek(), T );
        s.push( M );
        for ( m in members ) {
            m.draw( gl, s );
        }
        s.pop();
    }
}
```

Is new Matrix4f() a good idea in the draw call? What could be done differently?

Based on slides by Steve Marschner

Basic Scene Graph operations

- Editing a transformation
  - Good to present usable UI
- Getting transform of object in canonical (world) frame
  - Traverse path from root to leaf
- Grouping and ungrouping
  - Can do these operations without moving anything
  - Group: insert identity node
  - Ungroup: remove node, push transform to children
- Re-parenting
  - Move node from one parent to another
  - Can do without altering position

Based on slides by Steve Marschner

Adding more than geometry

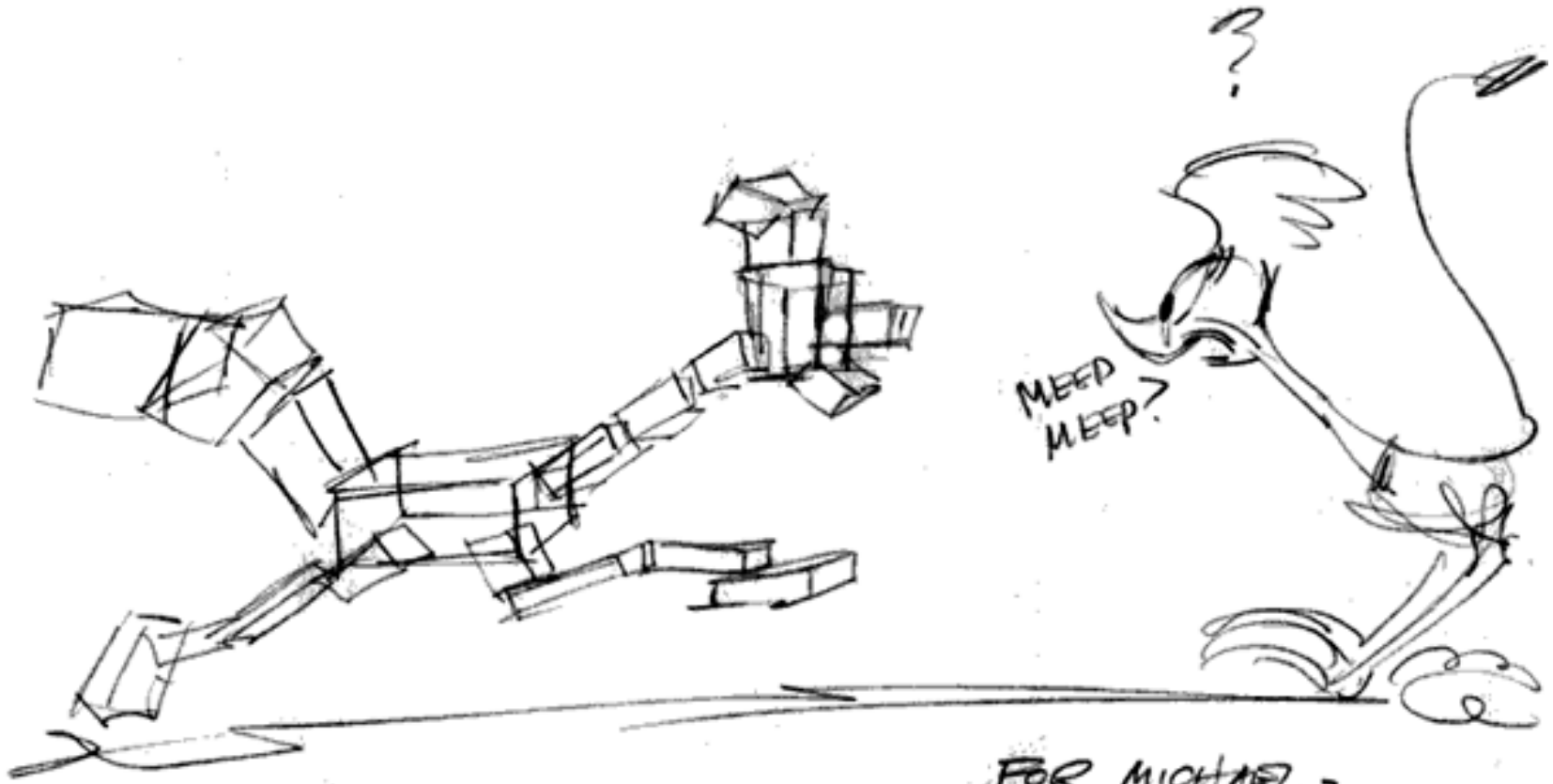
- Objects have properties besides shape
  - Color, shading parameters
  - Approximation parameters (for instance, precision of subdividing curved surfaces into triangles)
  - Behavior in response to user input
  - …
- Setting properties for entire groups is useful
  - Paint entire window green
- Many systems include some kind of property nodes
  - In traversal they are read as, e.g., “set current color”
- Scene graphs also typically include cameras, lights, background, and other information

Based on slides by Steve Marschner

Scene Graph variations

- Where transforms go
  - in every node?
  - in group nodes only?
  - in special Transform nodes?
- Tree vs DAG
- Nodes for cameras and lights?

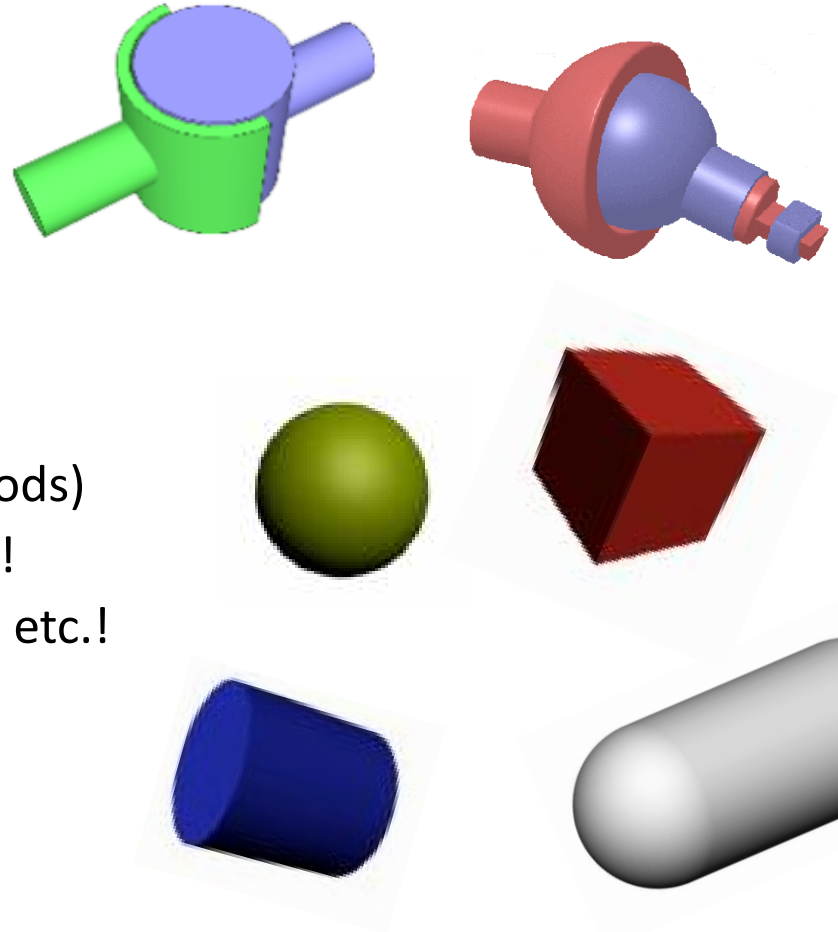
Based on slides by Steve Marschner



FOR MICHAEL -
WITH ADMIRATION
Chuck Jones
1985

Design, Posing, and Key Frame Animation

- Joints and bones, it is useful to separate joint transforms from geometry (and geometry transforms)
  - Joint types:
    - Hinge / Rotary, a given axis about a given point
    - Ball / Spherical, XYZ Euler, about a given point
    - Free, e.g., both translation and XYZ Euler angles
  - Bones / geometry:
    - Cube, Sphere, Cylinder, Cone, etc… (see glut methods)
    - Translated and scaled with respect to parent node!
    - Combine shapes (e.g., capsules), different colours, etc.!



Based on slides by Steve Marschner

Design, Posing, and Key Frame Animation

- What should be the root?
  - A foot? Hips? Torso? Head?
- Want to expose parameters necessary to pose
  - Typically only want to edit joint angles, while geometry may remain fixed (e.g., bone lengths)
  - Reasonable limits on parameters?
- Want meaningful poses produced by interpolation of parameters (gimbal lock?)
- Many options for organizing DAG and classes!

Discuss!

Based on slides by Steve Marschner

More on Rotations

Quaternions

Interpolation

Trackball Interaction

Group of Unit Quaternions

- Quaternions are like complex numbers, but with three imaginary parts

$$ijk = i^2 = j^2 = k^2 = -1$$

- Example multiplication:

$$A = a_0 + a_1 i + a_2 j + a_3 k$$
$$B = b_0 + b_1 i + b_2 j + b_3 k$$
$$AB = a_0 b_0 - a_1 b_1 - a_2 b_2 - a_3 b_3 +$$
$$(a_0 b_1 + a_1 b_0 + a_2 b_3 - a_3 b_2) i + \ldots$$

The set of unit quaternions, combined with quaternion multiplication as a binary operation, form a group which is ***very similar*** to the rotation group $SO(3)$, except that each 3D orientation appears twice! It is a double covering.

Based on slides by Steve Marschner

Quaternions as Rotations

- Related to a rotation by θ on unit length axis (x, y, z)

$$\mathrm{q} = (c, sx, sy, sz) \equiv c + sxi + syj + szk$$

where $c = \cos\left(\frac{\theta}{2}\right)$ and $s = \sin\left(\frac{\theta}{2}\right)$

- Why $\theta/2$?
  - This means that q and –q are same rotation, and the angle between two unit quaternions in 4D space is half the angle between the 3D orientations that they represent
- Composition by multiplication (rules on previous slide)
- Inverse of unit quaternions similar to complex conjugation
  - If $q = (w, x, y, z)$ is unit length, then $q^{-1} = (w, -x, -y, -z)$
  - Vector (x, y, z) transforms as qvq^{-1} where $v = 0 + ix + jy + kz$ and the result is the imaginary part

Questions

- Cost of composing rotations:
  - Matrix? Quaternion?
- Cost of transforming vectors:
  - Matrix? Quaternion?

Comparison of Rotation representations

- Rotation matrix
  - Used widely, 3 by 3 matrix with $R^T R = I$ and $\det(R) = +1$
- Euler angles
  - Used widely, often simple and convenient, also problematic
- Quaternion
  - Used widely, compact storage, efficient and nice mathematical properties…
- Axis and Angle
  - Used widely, similar to quaternions
- Axis scaled by angle
  - Save storage space, costly to normalize!

Based on slides by Steve Marschner

Important issues for different rotation representations

- Storage
  - How much memory?
- Composition
  - Is it possible?
  - What's the cost?
  - What's the cost of transforming points and vectors?
- Conversion between representations?
  - Easy or Hard?

Numerical problems?

- Suppose we need to compose hundreds or thousands of transformations?
- Does this ever happen?
- What can go wrong?
- How can we fix it?

- Interpolation
  - Smoothly interpolating between two rotations is important for animation!

Based on slides by Steve Marschner

Interpolation

- ***Linear interpolation*** between two points **a** and **b**

  Lerp(t,**a**,**b**) = (1-t)**a** + (t)**b**

  where t ranges from 0 to 1
- Note that the Lerp operation can be thought of as a weighted average (convex, i.e., $\sum w_i = 1, \;\; w_i > 0$)
- We can also write it in as an additive blend:

  Lerp(t,**a**,**b**) = **a** + t(**b**-**a**)
- What happens if we use linear interpolation with rotations in different representations?

Spherical Linear Interpolation

- We define the spherical linear interpolation of two unit vectors in N dimensional space as

$$\text{Slerp}(t, \boldsymbol{a}, \boldsymbol{b}) = \frac{\sin((1-t)\theta)}{\sin\theta}\boldsymbol{a} + \frac{\sin(t\theta)}{\sin\theta}\boldsymbol{b} \quad \text{where} \quad \theta = \cos^{-1}(\boldsymbol{a}\cdot\boldsymbol{b})$$

- We can use this formula to smoothly interpolate two arbitrary rotations represented as **quaternions**
  - Careful if a dot b is negative! (recall, q and –q are same rotation)
  - What happens in this case? How to fix this?
- Lerp and Slerp are not the only options…

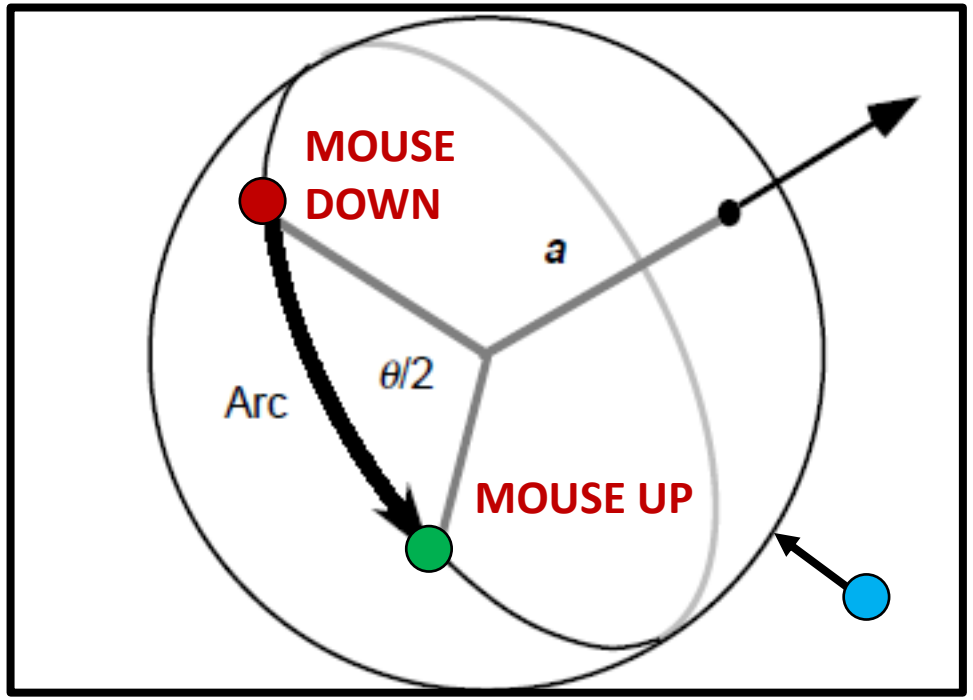
Based on slides by Steve Marschner

Interaction

- What is a good way to rotate an object you are viewing on screen using a mouse?

ArcBall / TrackBall

Ken Shoemake, ARCBALL: A User Interface for Specifying Three-Dimensional Orientation Using a Mouse, Graphics Interface 1992



- Imagine: ball centered on the screen
- screen is at z=0 with axis pointing out of the screen
- Dragging the mouse from one screen point to another rotates the point on the ball
- **How do we compute the rotation?**
  - Axis?
  - Angle?
- When mouse not on ball, project onto ball

- **Fit** specifies size of the ball, radius = min screen dim / fit, a fit = 2 means just touching edges
- **Gain** specifies a modification to the computed arc angle

Coming up with a rotation matrix

- We have seen matrices for coordinate axis rotations
- There are conversion formulas for quaternion and axis angle to matrix
  - What if we want rotation about some random axis?
  - Is there an intuitive / constructive solution?
- Yes! Compute by composing elementary transforms!
  - Transform rotation axis to align with *x* axis
  - Apply rotation
  - Inverse transform back into position
- Just as in 2D this can be interpreted as a similarity transform

Building general rotations

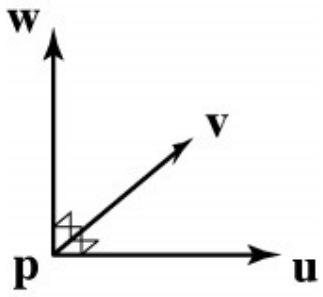
- Using elementary transforms you need three
  - translate axis to pass through origin
  - rotate about *y* to get into *x*-*y* plane
  - rotate about *z* to align with *x* axis
- Alternative: construct frame and change coordinates
  - choose *p*, *u*, *v*, *w* to be orthonormal frame with *p* and *u* matching the rotation axis
  - apply similarity transform $T = F\, R_x(\theta)\, F^{-1}$

Based on slides by Steve Marschner

Orthonormal frames in 3D

- Useful tools for constructing transformations
- Recall rigid motions
  - affine transforms with pure rotation
  - columns (and rows) form right handed orthonormal basis

$$F = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} & \mathbf{p} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Based on slides by Steve Marschner

Building 3D frames

- Given a vector **a** and a secondary vector **b**
  - The **u** axis should be parallel to **a**; the **u**–**v** plane should contain **b**
    - **u** = **u** / ||**u**||
    - **w** = **u** x **b**; **w** = **w** / ||w||
    - **v** = **w** x **u**
- Given just a vector **a**
  - The **u** axis should be parallel to **a**; don't care about orientation about that axis
    - Same process but choose arbitrary **b** first
    - Good choice is not near **a**, e.g., set smallest entry to 1

Based on slides by Steve Marschner

Axis Angle to Matrix

- Rotation θ around an arbitrary ***unit*** length axis a

$$\begin{bmatrix} a_x^2 + c_\theta(1-a_x^2) & a_x a_y(1-c_\theta) + a_z s_\theta & a_x a_z(1-c_\theta) - a_y s_\theta \\ a_x a_y(1-c_\theta) - a_z s_\theta & a_y^2 + c_\theta(1-a_y^2) & a_y a_z(1-c_\theta) + a_x s_\theta \\ a_x a_z(1-c_\theta) + a_y s_\theta & a_y a_z(1-c_\theta) - a_x s_\theta & a_z^2 + c_\theta(1-a_z^2) \end{bmatrix}$$

Building rotation with axis not through origin

- Alternative: construct frame and change coordinates
  - choose p, u, v, w to be orthonormal frame with p and u matching the rotation axis
  - apply similarity transform $T = F\, R_x(\theta)\, F^{-1}$
  - interpretation: move to x axis, rotate, move back
  - interpretation: rewrite u-axis rotation in new coordinates
  - (each is equally valid)

Based on slides by Steve Marschner

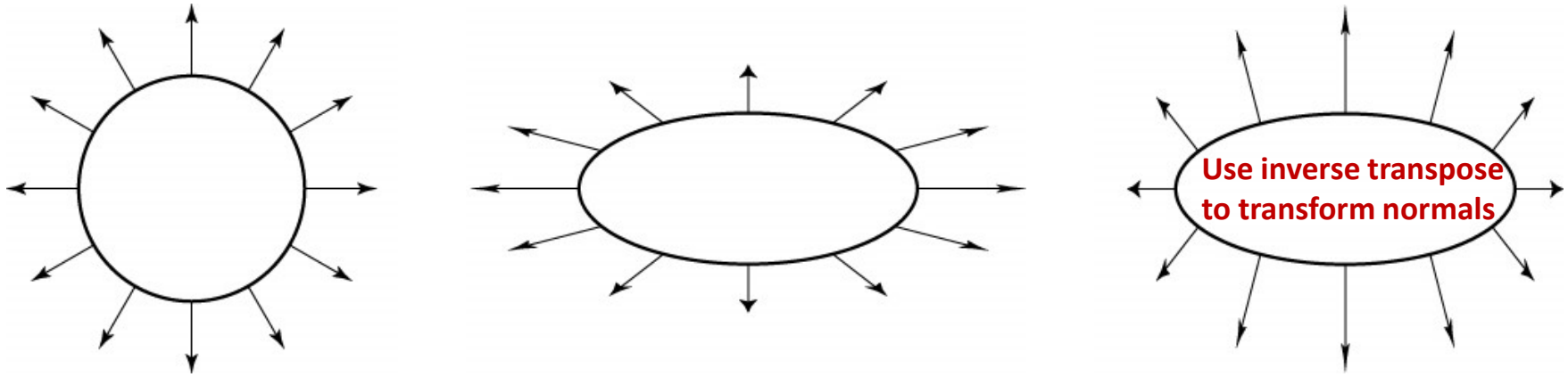
Building transforms from points

- 2D affine transformations have 6 degrees of freedom (DOFs)
  - Number of parameters we must set to define an affine transformation
- Therefore 6 constraints suffice to define the transformation
  - Constrain point p maps to point q (2 constraints at once)
  - Three point constraints add up to constrain all 6 DOFs
  - Can map any triangle to any other triangle
- 3D affine transformation has 12 degrees of freedom
  - Count them by looking at the matrix entries we're allowed to change
- Therefore 12 constraints suffice to define the transformation
  - In 3D, we need 4 point constraints
  - Can map any tetrahedron to any other tetrahedron

Based on slides by Steve Marschner

Transforming normal vectors

- Transforming surface normals
  - Differences of points (e.g., tangents) transform OK
  - Normals are ***covectors***, and do not transform the same way



have: $\mathbf{t} \cdot \mathbf{n} = \mathbf{t}^T \mathbf{n} = 0$

want: $M\mathbf{t} \cdot X\mathbf{n} = \mathbf{t}^T M^T X \mathbf{n} = 0$

so set $X = (M^T)^{-1}$

then: $M\mathbf{t} \cdot X\mathbf{n} = \mathbf{t}^T M^T (M^T)^{-1} \mathbf{n} = \mathbf{t}^T \mathbf{n} = 0$

Based on slides by Steve Marschner

Question

Let $p = (\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0)^T$ be a point in non-homogeneous coordinates on a sphere with center at the origin and with radius equal to 1. Let $n = (\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0)^T$ be the normal vector in non-homogeneous coordinates of the sphere at point p. Given transformations, T, S, R,

$$T = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad S = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad R = \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

(a) What is the position of the point p on the sphere after the sphere is transformed by the product TSR? Show your work by appling each transformation separately rather than computing the product of the three matrices.

(b) What is the normal of the point p on the sphere after the sphere is transformed by the product TSR? Show your work by appling each transformation separately.

Question

- Give an example of a matrix
  - Whose columns and rows are orthogonal
  - Has determinant equal to +1
  - But is ***not*** a rotation matrix

Review and more information FCG

- Textbook (3rd edition)
  - 6.1 - 2D linear transformations, composition
  - 6.2 - 3D linear transformations, transforming normal vectors
  - 6.3 - Translation and affine transformations, homogeneous coordinates
  - 6.5 - Coordinate transformations
- Rotations not covered in much depth in the textbook, but note in particular:
  - 6.2.1 general rotations
  - 17.2.2 Interpolating rotations (gimbal lock, quaternions)

Based on slides by Steve Marschner

Review and more information FCG

- Textbook
  - FCG 12.2 - Scene graphs
  - Directed acyclic graphs and instances not in FCG (but discussed in CAPP 6.6.4)

Based on slides by Steve Marschner

Review and More Information

- Transforming covectors (normals) CGPP 10.12
- Transformations in 3D, Chapter 11, in particular:
  - Euler angles and gimbal lock 11.2
  - Quaternions 11.2.6
  - Trackball and Arcball, 11.6
- CGPP 6.6, Hierarchical modeling using a scene graph
- glPushMatrix and glPopMatrix
  - Convenient but not in the latest spec