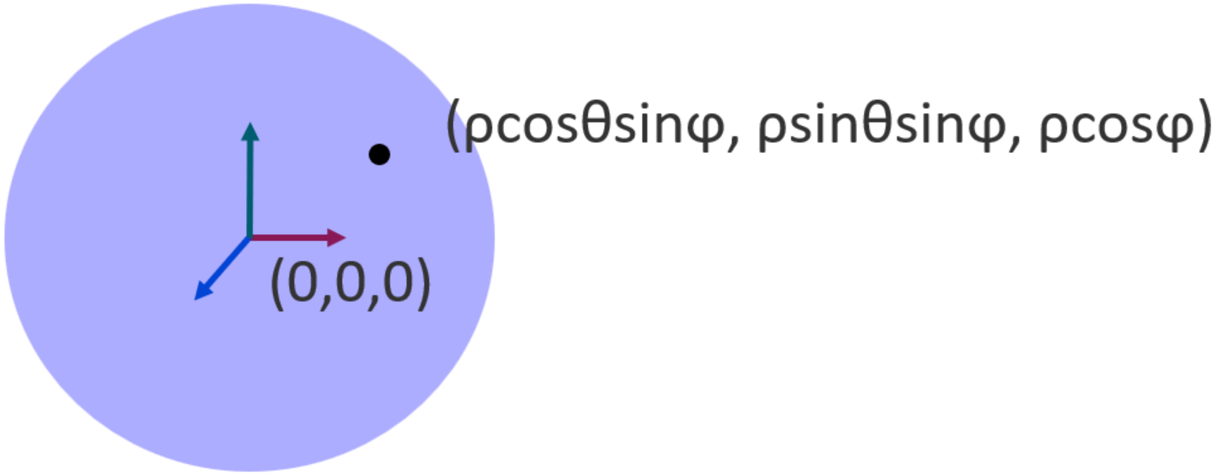
University of Minnesota
Driven to Discover®

Scene Hierarchy

CSCI 4611: Programming Interactive Computer Graphics and Games

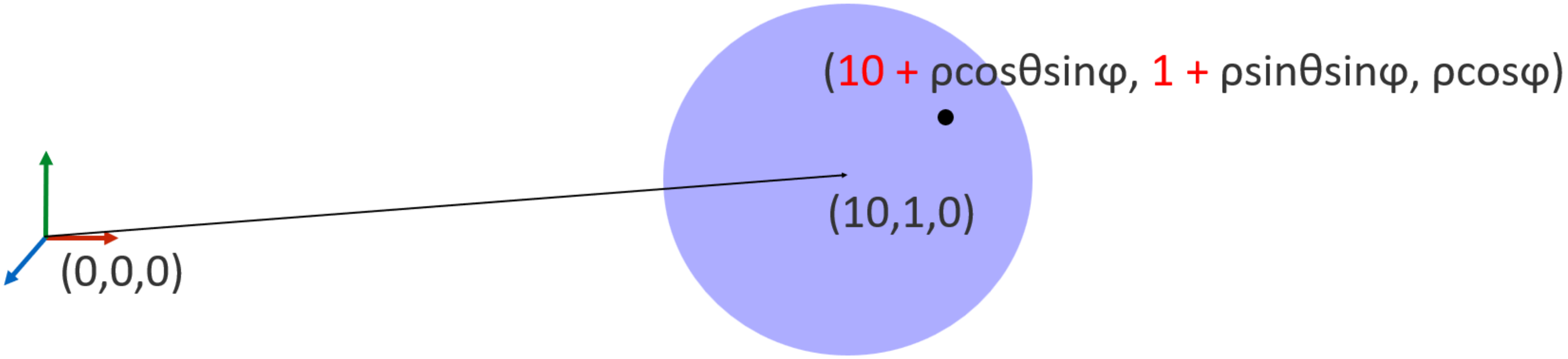
Evan Suma Rosenberg | CSCI 4611 | Fall 2022

Review: 3D Transformations



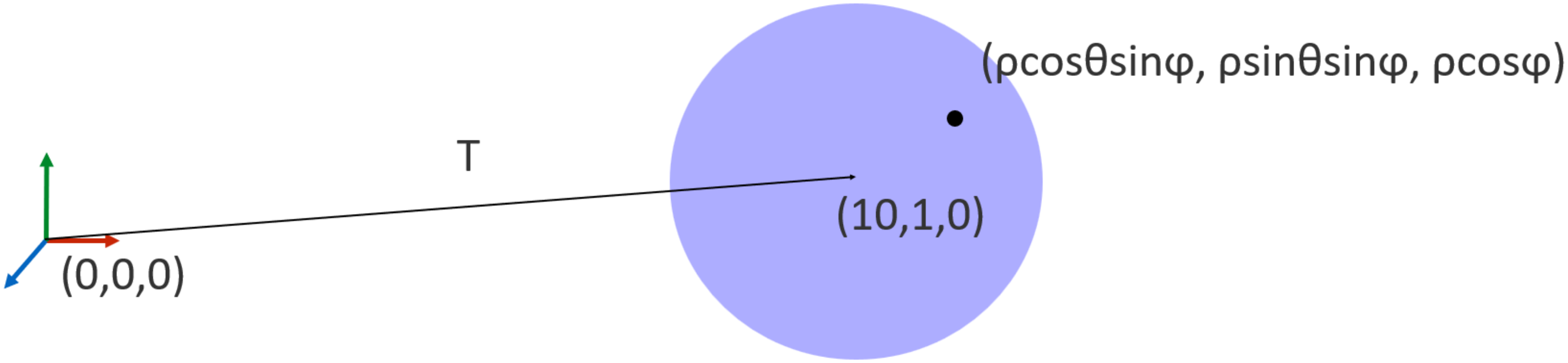
When we defined the (x,y,z) coordinates for the globe
in assignment 3, we placed the center of the sphere at (0,0,0).

Review: 3D Transformations



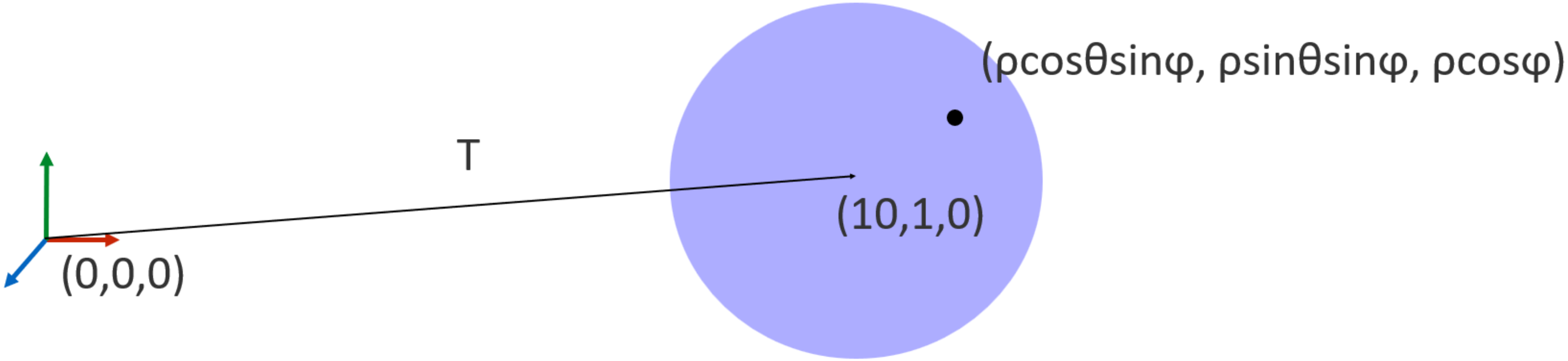
If we wanted the sphere to be centered at (10,1,0), we could change the math we used to define the vertices, but...

Review: 3D Transformations



We have already learned that it is easier to use a transformation matrix.

Review: 3D Transformations

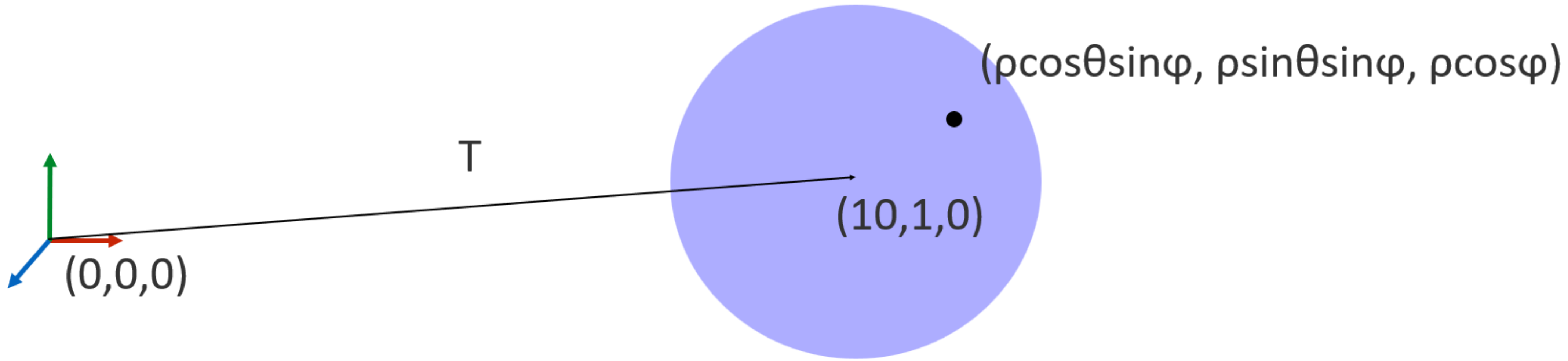


What actually happens when we use a transformation matrix?

- Before we draw each vertex, it gets multiplied by the transformation matrix.
- We don't actually change all the numbers that define the vertices of the sphere to create a new model. The multiplication happens in real-time while drawing.

  So, even though the sphere may move around in the scene, the vertices are still defined relative to its center.

Review: 3D Transformations



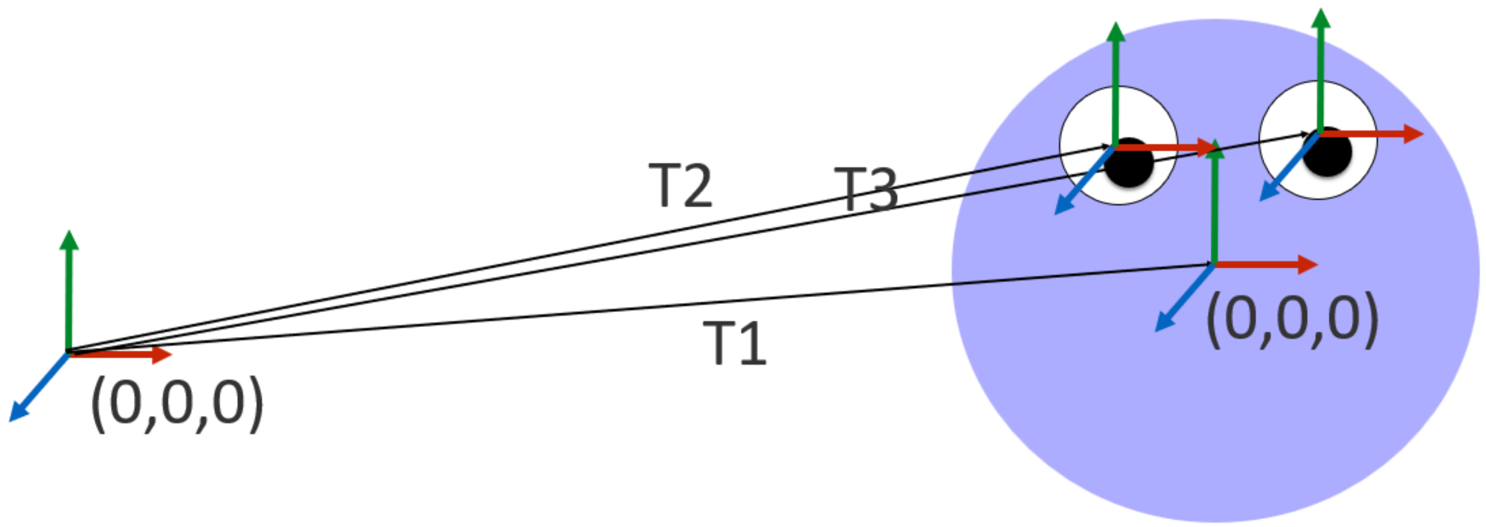
It makes sense then to think of the sphere as having its own (0,0,0) that travels with it.

This is the sphere's **local coordinate system**.

This is the first new concept of the day…

In computer graphics, there will always be one **world coordinate system** or "global" (0,0,0), but we can think of each object as having its own "object coordinate system" or "local" (0,0,0).

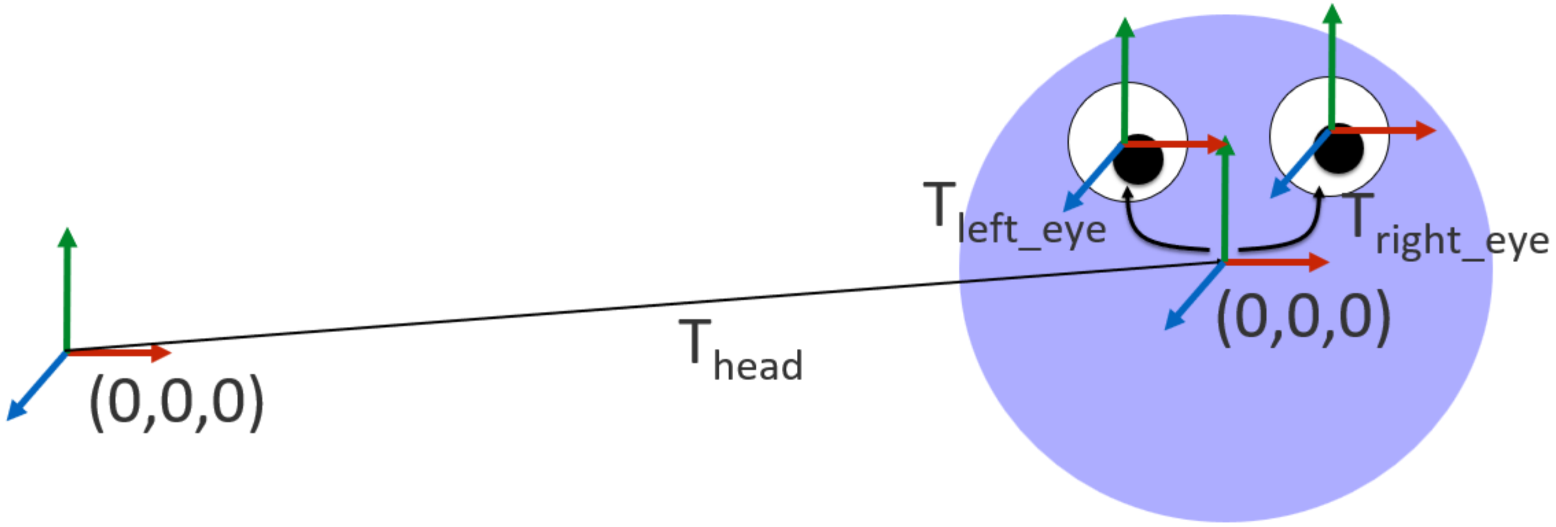
What if the model has multiple parts?



We will need transformations for each part that moves separately.

We could certainly specify all these transformations relative to the world coordinate system.

What if the model has multiple parts?

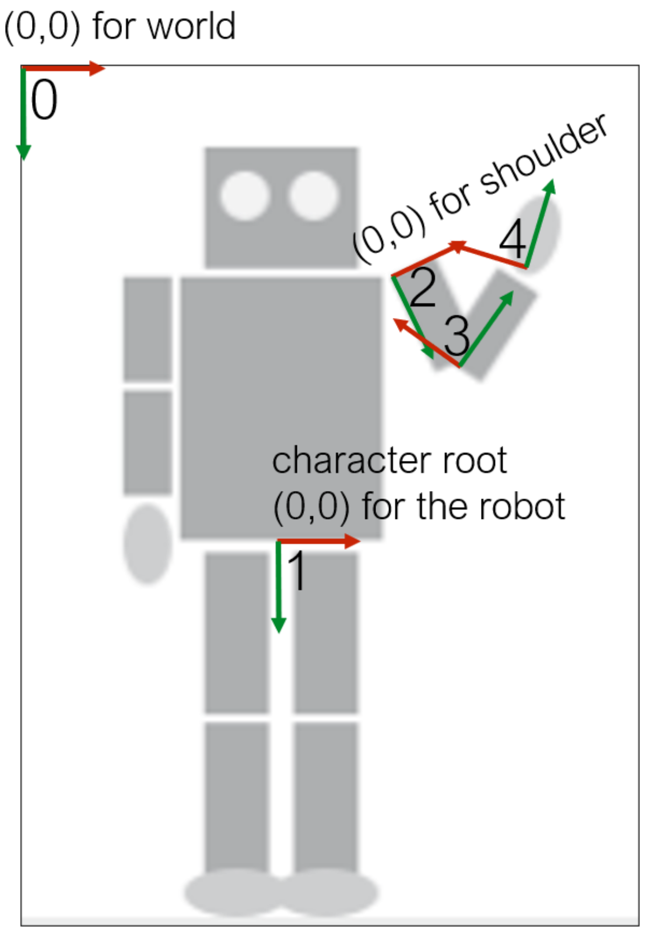


What if we specify the eye transformations relative to the head's local coordinate system?

- The coordinates for the eyes are easier to define.
- We can still move each eye independently if we want.

  When the head moves, the eyes will move with it, as if they are attached.

Hierarchical Transforms

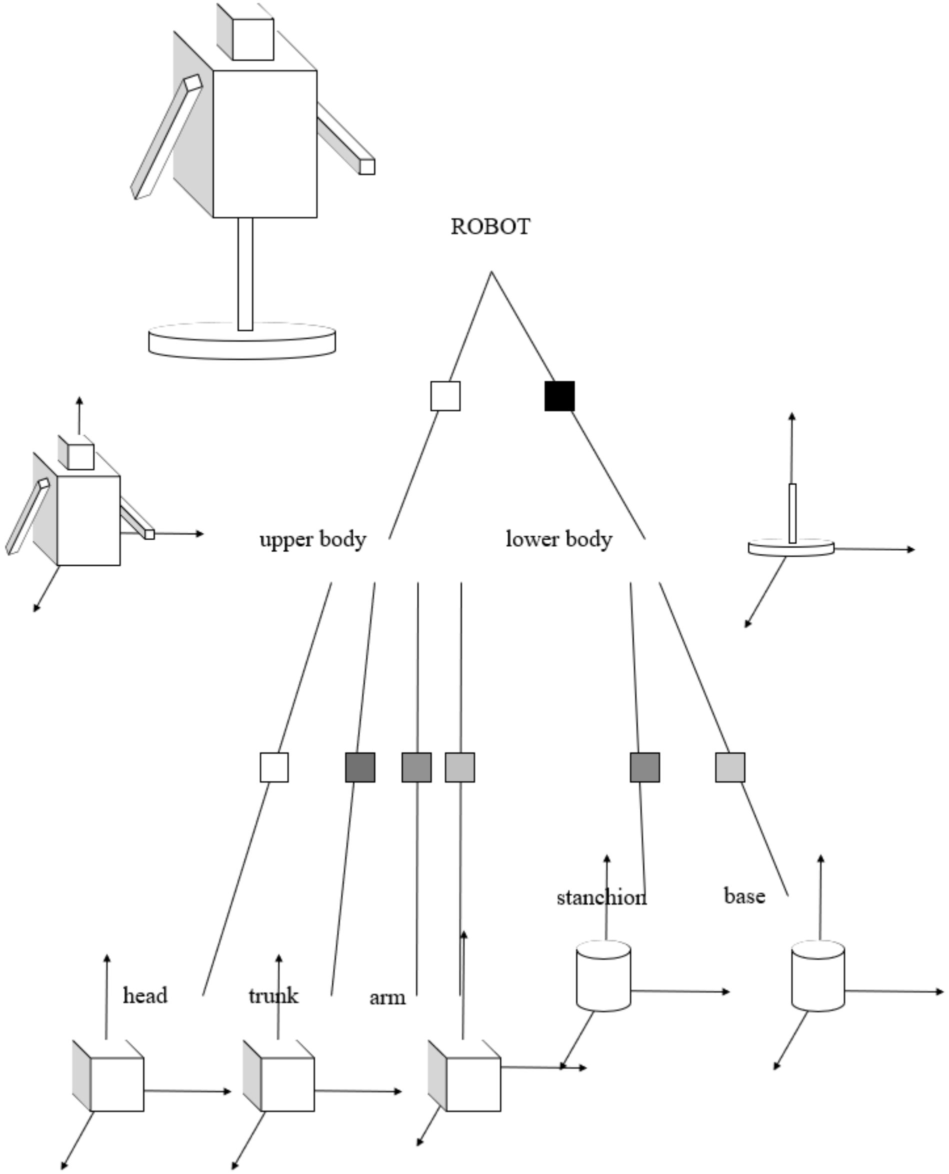


Animated characters in both 2D and 3D graphics use **hierarchical transformations**.

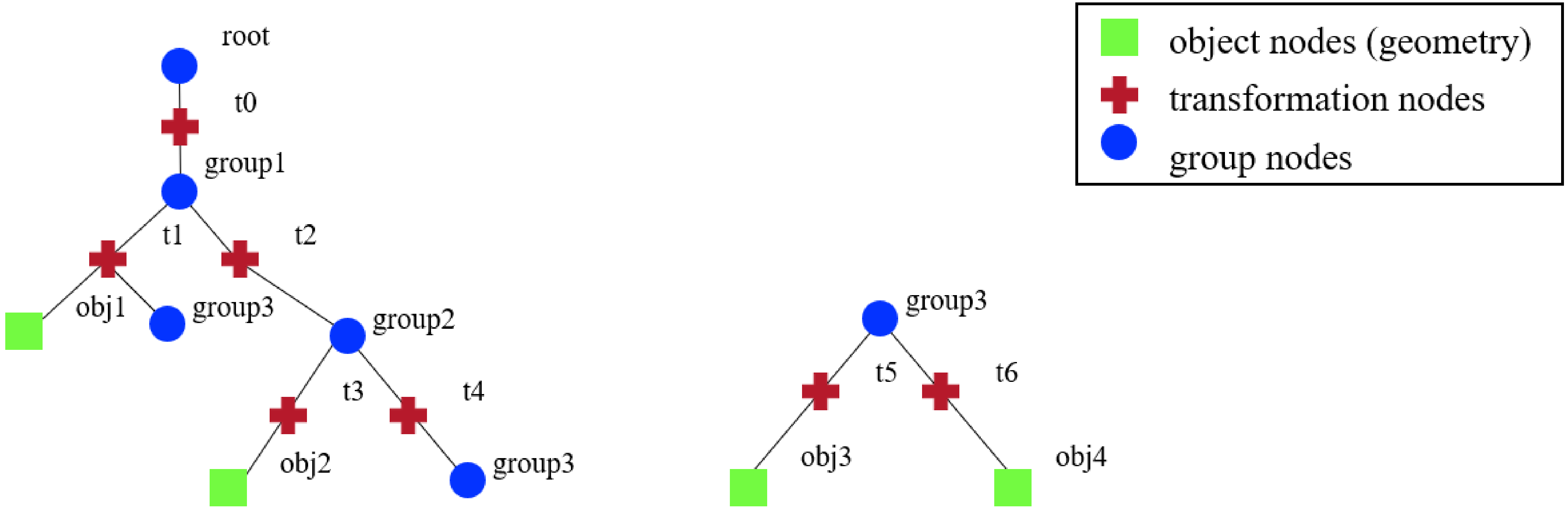
Scene Graphs

A **scene graph** is a data structure used to represent a hierarchical scene or geometry in computer graphics.

Notice that the leaf nodes here are all just basic shapes (unit cube, unit cylinder).



Transformations in Scene Graphs



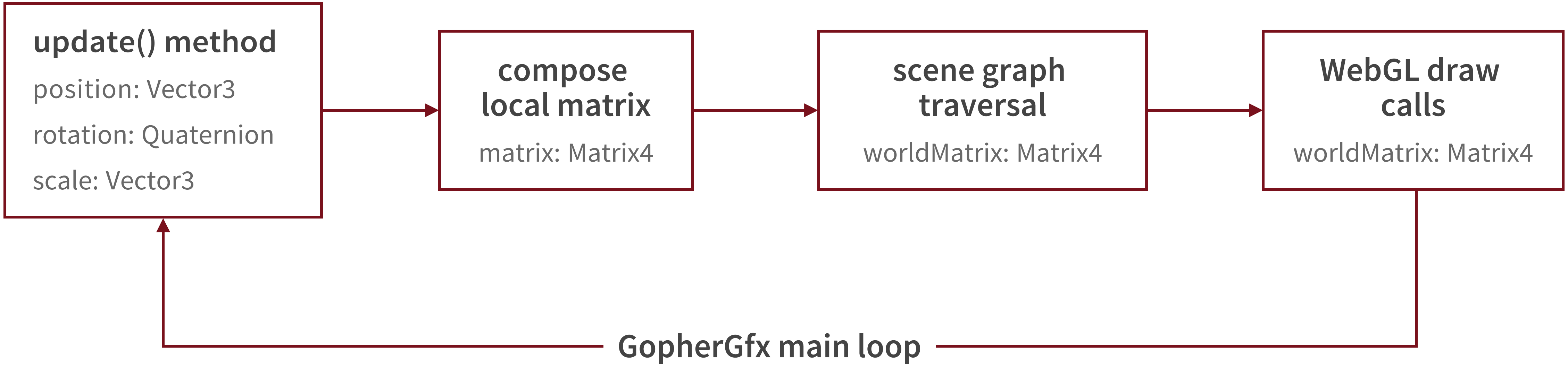
- Transformations affect all child nodes
- Sub-trees (group nodes) can be reused

  Instances of a group can have different transformations applied to them (e.g. group3 is used twice)

Composing Transforms in Scene Graphs

- Transformation nodes specify a matrix that handles the transformation
- To determine final **composite transformation matrix** (CTM) for an object node:
  - Compose all parent transformations during preorder graph traversal
  - Exact details of how this is done varies from package to package

GopherGfx Main Loop

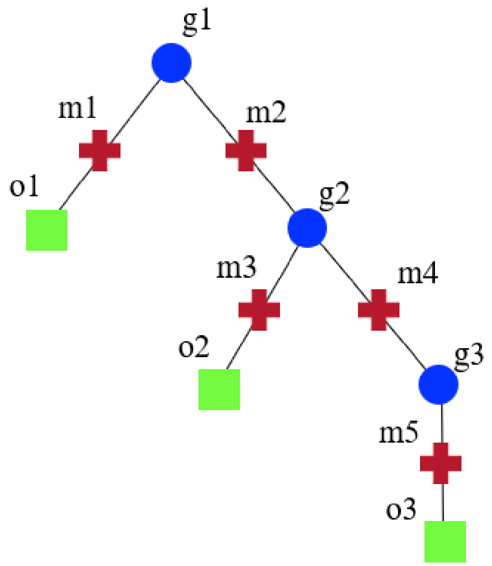


Composing Transforms in Scene Graphs

for o1, CTM = m1

for o2, CTM = m2* m3

for o3, CTM = m2* m4* m5



g: group nodes

m: matrices of transform nodes

o: object nodes

for a vertex v defined in o3's local coordinate system, its position v' in the world coordinate system would be:

v' = CTM*v = (m2*m4*m5)*v

How to implement this in code?

- Under the hood, draw routine(s) must keep track of the Current Transformation Matrix (CTM).
- When moving "down" the scene graph from a parent node to a child, the CTM must be updated:

  M_{child} = CTM * M

  where M transforms points in the **child coordinate system** into the **parent coordinate system**
- When moving "back up" the graph from a child node to its parent, you must "undo" that transformation:

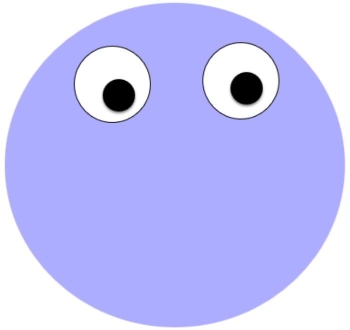
  CTM = M_{child} * M^{-1}

How to implement this in code?

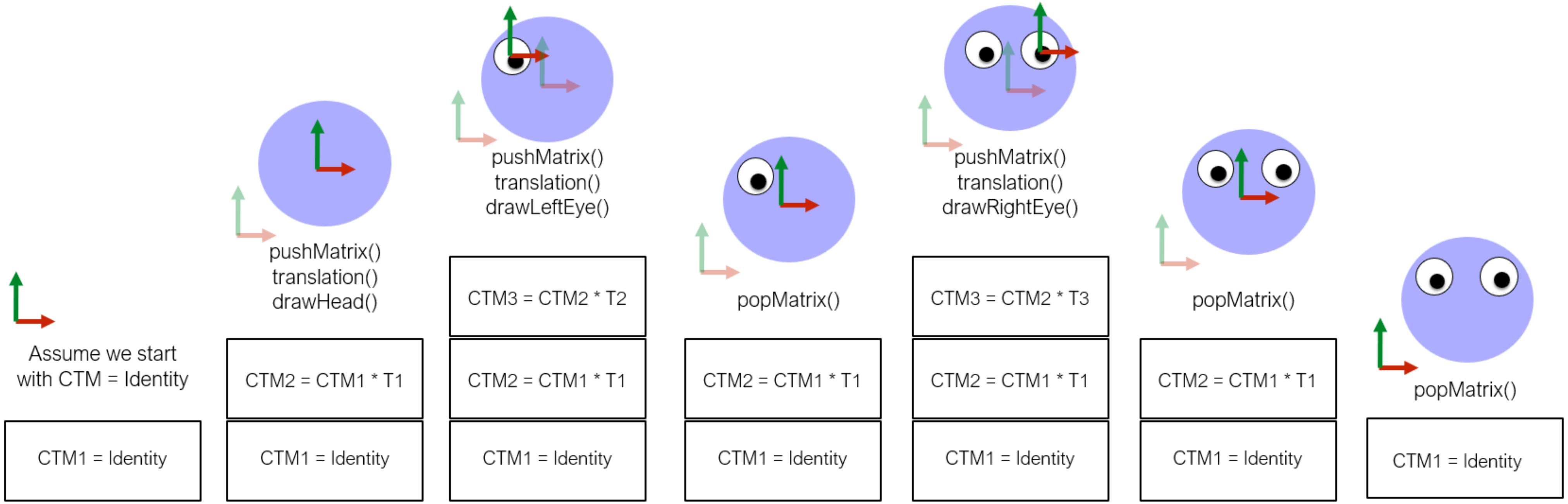
A lot of saving and restoring of the CTM needed to do this.

This is a perfect opportunity to use concepts from algorithms and data structures courses to make this easier for programmers!

Option 1: Use a Stack Data Structure



- You could create your own stack to store an array of Matrix4s.
- This was so common that was built directly into "old school" OpenGL.
- Modern OpenGL and WebGL use a different approach.



Option 2: Use a Recursive Draw Function

Example pseudocode:

```
DrawBodyPart(int bodyPartId, Matrix4 CTM)
{
  // step 1: draw the current body part using the CTM

  // step 2: draw all the child body parts
      foreach child body part
          find M – the child to parent transformation
          DrawBodyPart(childId, CTM * M)
  }

// start drawing at the root node
DrawBodyPart(rootId, identityMatrix)
```

Option 2: Use a Recursive Draw Function

```
foreach child body part
    find M – the child to parent transformation
    DrawBodyPart(childId, CTM * M)
```

M is labeled as the **child to parent transformation** as we traverse this scenegraph (even though we are moving from parent to child) because I want to always be thinking about what is happening to the vertices.

Vertices defined in the child's coordinate system need to get transformed into the current coordinate system.

So, M is the transformation matrix that transforms points defined in the local coordinate system of the child to the local coordinate system of the parent.

Option 3: Use a High-Level API (that implements #2)

OpenSceneGraph is an open-source 3D graphics API written in C++.

Core OSG classes:

- Helper classes - *memory management, math classes*
- osg::Nodes - *the internal nodes in the scene graph*
- osg::Drawables - *the leaves of the scene graph that can be drawn*
- osg::State - *the classes that encapsulate OpenGL state*
- Traversers/visitors - *classes for traversing and applying operations on the scene*

Option 3: Use a High-Level API (that implements #2)

Three.js is an open-source 3D graphics API written in JavaScript.

Core Three.js classes:

- THREE.Object3D - *base class for all nodes in the scene graph*
- THREE.Group - *internal nodes of the scene graph*
- THREE.Mesh, THREE.Line, etc... - *subclasses for leaf nodes of the scene graph that can be drawn*

Assignment 4

So You Think Ants Can Dance

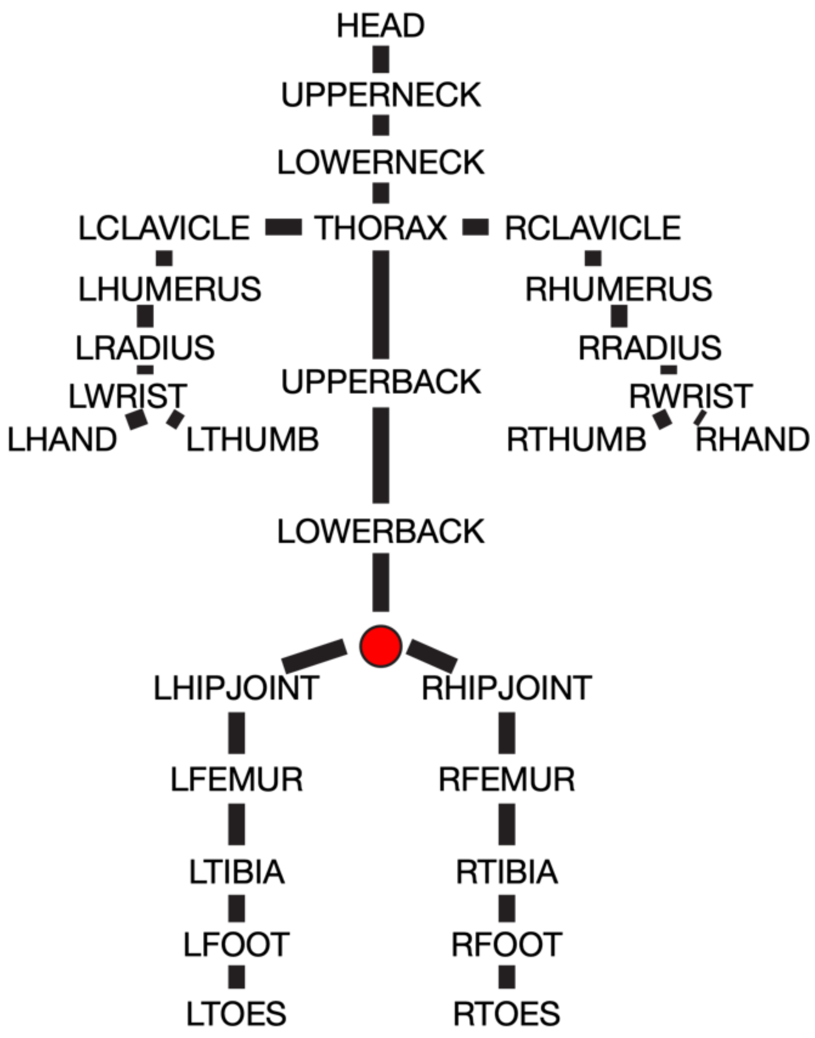
https://csci-4611-fall-2022.github.io/Builds/Assignment-4/

Mocap Data

- Motion capture data from the CMU Mocap database
- 2,605 different motions
- http://mocap.cs.cmu.edu/

CMU Mocap Skeleton

- Each bone has a rest coordinate frame, a direction (in its local coordinate frame), and a length
- Each bone has zero or more children
- Root node is the origin of the skeleton



Learning Objectives

- How transformations can be composed in a hierarchy to create a scene graph and/or, in this case, an animated character within a scene.
- How transformations can be used to scale, rotate, and translate basic shapes (unit cubes, spheres, cylinders, and cones) into more complex scenes and characters.
- How mocap data can be used and manipulated in multiple ways to create different types of animations. For example:
  - How to create a looping animation that smoothly interpolates between the beginning of the motion clip and the end to avoid any discontinuities.
  - How to overlay new motion clips onto a character at runtime, for example, making your character jump in a game when you press a button, or in our case, perform one of a series of cool ballet moves.

How to read and extend some fairly sophisticated computer graphics code.