7. Hierarchical Modeling

Reading

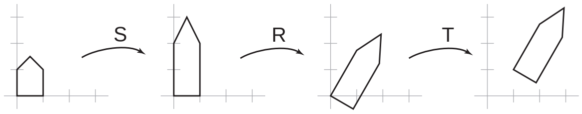
- Angel, sections 8.1 - 8.6
- *OpenGL Programming Guide*, chapter 3

Symbols and instances

Most graphics APIs support a few geometric **primitives**:

- spheres
- cubes
- cylinders

These symbols are **instanced** using an **instance transformation**.



Q: What is the matrix for the instance transformation above?

Instancing in OpenGL

In OpenGL, instancing is created by modifying the **model-view** matrix:

```
glMatrixMode( GL_MODELVIEW );
glLoadIdentity();
glTranslatef( ... );
glRotatef( ... );
glScalef( ... );
house();
```

Do the transforms seem to be backwards? Why was OpenGL designed this way?

Instancing in anti-OpenGL

Suppose OpenGL transforms used left-multiplication.
Take a scene with multiple instances of house:

```
glPushMatrix();
glRotate( ... );
glTranslate( ... );
house();
glPopMatrix();

glPushMatrix();
glRotate( ... );
glTranslate( ... );
house();
glPopMatrix();
```

How would I make all the houses twice as tall?

Instancing in real OpenGL

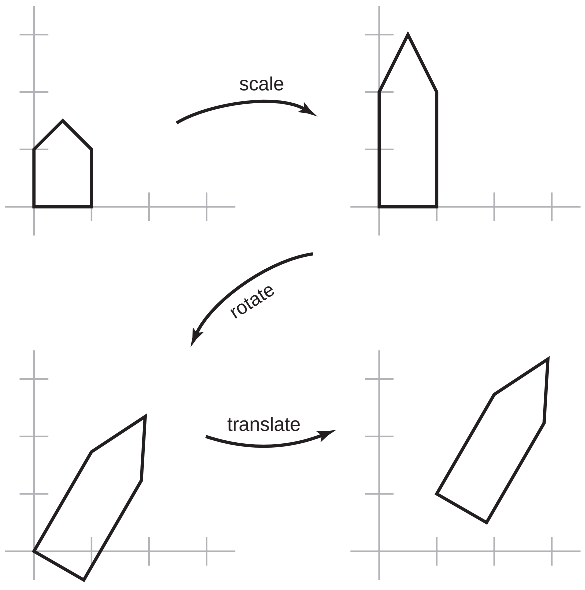
The advantage of right-multiplication is that it places the *earlier* transforms *closer* to the primitive.

```
glPushMatrix();
glTranslate( ... );
glRotate( ... );
house();
glPopMatrix();

glPushMatrix();
glTranslate( ... );
glRotate( ... );
house();
glPopMatrix();
```

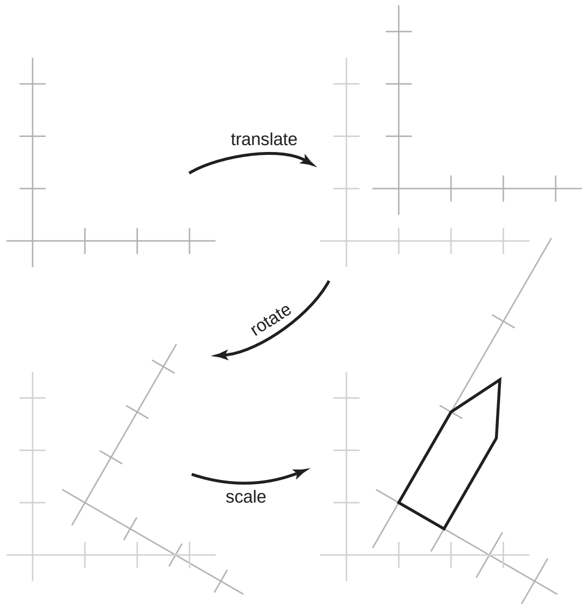
Global, fixed coordinate system

OpenGL's transforms, logical as they may be, still *seem backwards*. They are, if you think of them as transforming the object in a **fixed** coordinate system.



Local, changing coordinate system

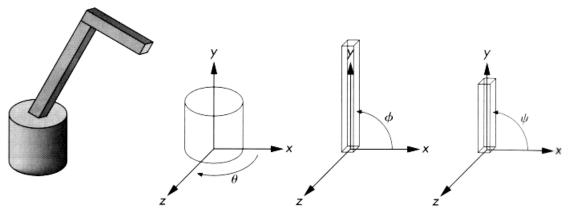
Another way to view transformations is as affecting a *local coordinate system* that the primitive is drawn in. Now the transforms appear in the "right" order.



3D Example: A robot arm

Consider this robot arm with 3 degrees of freedom:

- Base rotates about its vertical axis by θ
- Lower arm rotates in its *xy*-plane by ϕ
- Upper arm rotates in its *xy*-plane by ψ



Q: What matrix do we use to transform the base?

Q: What matrix for the lower arm?

Q: What matrix for the upper arm?

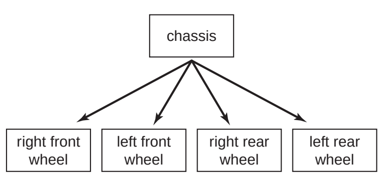
Robot arm implementation

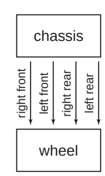
The robot arm can be displayed by altering the model-view matrix incrementally:

```
robot_arm()
{
    glRotatef( theta, 0.0, 1.0, 0.0 );
    base();
    glTranslatef( 0.0, h1, 0.0 );
    glRotatef( phi, 0.0, 0.0, 1.0 );
    lower_arm();
    glTranslatef( 0.0, h2, 0.0 );
    glRotatef( psi, 0.0, 0.0, 1.0 );
    upper_arm();
}
```

Hierarchical modeling

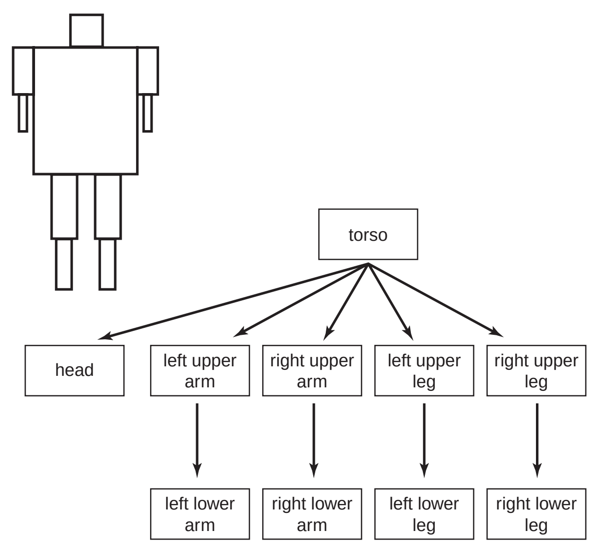
Hierarchical models can be composed of instances using trees or DAGs:





- edges contain geometric transformations
- nodes contain geometry (and possibly drawing attributes)

A complex example: human figure



Q: What's the most sensible way to traverse this tree?

Human figure implementation

The traversal can be implemented by saving the model-view matrix on a stack:

```
figure()
{
    torso();
    glPushMatrix();
        glTranslate( ... );
        glRotate( ... );
        head();
    glPopMatrix();
    glPushMatrix();
        glTranslate( ... );
        glRotate( ... );
        left_upper_leg();
        glTranslate( ... );
        glRotate( ... );
        left_lower_leg();
    glPopMatrix();
    . . .
}
```

Animation

The above examples are called **articulated models**:

- rigid parts
- connected by joints

They can be animated by specifying the joint angles (or other display parameters) as functions of time.

Kinematics and dynamics

Definitions:

- **Kinematics:** how the positions of the parts vary as a function of the joint angles.
- **Dynamics:** how the positions of the parts vary as a function of applied forces.

Questions:

Q: What do the terms **inverse kinematics** and **inverse dynamics** mean?

Q: Why are these problems more difficult?

Key-frame animation

One way to get around these problems is to use **key-frame animation**.

- Each joint specified at various **key frames** (not necessarily the same as other joints)
- System does interpolation or **in-betweening**

Doing this well requires:

- A way of smoothly interpolating key frames: **splines**
- A good interactive system
- A lot of skill on the part of the animator

Scene graphs

The idea of hierarchical modeling can be extended to an entire scene, encompassing:

- many different objects
- lights
- camera position

This is called a **scene tree** or **scene graph**.

Summary

Here's what you should take home from this lecture:

- All the **boldfaced terms**.
- How primitives can be instanced and composed to create hierarchical models using geometric transforms.
- How transforms can be thought of as affecting either the geometry, or the coordinate system which it is drawn in.
- How the notion of a model tree or DAG can be extended to entire scenes.
- How keyframe animation works.