Hierarchical Modeling

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?

Reading
- Angel, Interactive Computer Graphics, sections 8.1 - 8.6
- Foley, Computer Graphics, Chapter 5.
- OpenGL Programming Guide, chapter 3
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();```

### Connecting Primitives

#### 3D Example: A robot arm

Consider this robot arm with 3 degrees of freedom:
- Base rotates about its vertical axis by $\theta$
- Lower arm rotates in its $xy$-plane by $\phi$
- Upper arm rotates in its $xy$-plane by $\psi$

**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 keeping a global matrix and computing it at each step:

```c
Matrix M_model;
main()
{
    ... robot_arm(); ...
}
robot_arm()
{
    M_model = R_y(theta); base();
    M_model = R_y(theta)*T(0,h1,0)*R_z(phi); upper_arm();
    M_model = R_y(theta)*T(0,h1,0)*R_z(phi)*T(0,h2,0)*R_z(psi); lower_arm();
}
```

**Q:** Do the matrix computations seem wasteful?
**Robot arm implementation, better**

Instead of recalculating the global matrix each time, we can just update it in place:

```c
Matrix M_model;
main()
{
    ...;
    M_model = Identity();
    robot_arm();
    ...
}
robot_arm()
{
    M_model *= R_y(theta);
    base();
    M_model *= T(0,h1,0)*R_z(phi);
    upper_arm();
    M_model *= T(0,h2,0)*R_z(psi);
    lower_arm();
}
```

**Robot arm implementation, OpenGL**

OpenGL maintains a global state matrix called the **model-view matrix**.

```c
main()
{
    ...
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();
    robot_arm(a, b, c);
    ...
}
robot_arm(theta, phi, psi)
{
    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:

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

![Joint angles as functions of time](image)

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:

- 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.
Inverse Rotation

$R(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$

$R^{-1}(\theta) = ?$

Rotation in 3D

What about the inverses of 3D rotations?

$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$

$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$

$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Rotation around arbitrary point

Reflection around arbitrary axis
Reflection around arbitrary axis

Rotation that aligns 3 orthonormal vectors with the principal axes