10. Hierarchical Modeling

**Reading**

- Angel, sections 9.1 - 9.6
  [reader pp. 169-185]
- *OpenGL Programming Guide*, chapter 3
  - Focus especially on section titled “Modelling Transformations”.
Symbols and instances

Most graphics APIs support a few geometric primitives:
- spheres
- cubes
- triangles

These symbols are instanced using an instance transformation.

Use a series of transformations

Ultimately, a particular geometric instance is transformed by one combined transformation matrix:

But it's convenient to build this single matrix from a series of simpler transformations:

We have to be careful about how we think about composing these transformations.

(Mathematical reason: Transformation matrices don't commute under matrix multiplication)
Two ways to compose xforms

Method #1:
Express every transformation with respect to global coordinate system:

Method #2:
Express every transformation with respect to a "parent" coordinate system created by earlier transformations:

#1: Xform for global coordinates

FinalPosition = M₁ * M₂ * … * Mₙ * InitialPosition

Note: Positions are column vectors:

\[
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
\]
#2: Xform for coordinate system

FinalPosition = $M_1 \times M_2 \times \ldots \times M_n \times \text{InitialPosition}$

Translate/Rotate:
FROM previous coord sys
TO new one

with transformation expressed in the ‘previous’ coordinate system.
Connecting primitives

3D Example: A robot arm

Consider this robot arm with 3 degrees of freedom:

- Base rotates about its vertical axis by $\theta$
- Upper arm rotates in its $xy$-plane by $\phi$
- Lower arm rotates in its $xy$-plane by $\psi$

Q: What matrix do we use to transform the base?
Q: What matrix for the upper arm?
Q: What matrix for the lower 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();
}
```

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 by concatenating matrices on the right:

```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,h1,0)*R_z(phi) *T(0,h2,0)*R_z(psi);
    lower_arm();
}
```
Robot arm implementation, OpenGL

OpenGL maintains a global state matrix called the **model-view matrix**, which is updated by concatenating matrices on the right.

```c
main()
{
    . . .
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();
    robot_arm();
    . . .
}

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)

How might we draw the tree for the robot arm?
A complex example: human figure

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

Human figure implementation, OpenGL

```cpp
figure()
{
    torso();
    glPushMatrix();
    glTranslate( ... );
    glRotate( ... );
    head();
    glPopMatrix();
    glPushMatrix();
    glTranslate( ... );
    glRotate( ... );
    left_upper_arm();
    glPushMatrix();
    glTranslate( ... );
    glRotate( ... );
    left_lower_arm();
    glPopMatrix();
    glPopMatrix();
    right_upper_leg();
    . . .
}
```
**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.

---

**Key-frame animation**

The most common method for character animation in production is **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.

```
Scene
  /   \
\___/   \
Camera
     /   \
Light1 Light2
     |   |
Object1 Object2 Object3
```