Hierarchical Modeling

Symbols and instances

Most graphics APIs support a few geometric primitives:

- spheres, cubes, cylinders
- these procedures define points for you, but they're still just points $P$

These symbols are instanced using an instance transformation.

- the points are originally defined in a local coordinate system (eg, unit cube)

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

Reading

Required:
- Angel, sections 9.1 – 9.6, 9.8

Optional:
- OpenGL Programming Guide, the Red Book, chapter 3
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 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:

Matrix $M_{\text{model}}$;

```c
main()
{
    ...
    robot_arm();
    ...
}

robot_arm()
{
    $M_{\text{model}} = R_y(\theta)$;
    base();
    $M_{\text{model}} = R_y(\theta)*T(0,h1,0)*R_z(\phi)$;
    lower_arm();
    $M_{\text{model}} = R_y(\theta)*T(0,h1,0)*R_z(\phi)*T(0,h2,0)*R_z(\psi)$;
    upper_arm();
}
```

Do the matrix computations seem a tad wasteful?
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 implementation, better

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

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();
    ...
}
```

```c
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();
}
```

ObjectAxes.cpp

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

```c
figure()
{
    torso();
    glPushMatrix();
    glTranslatef( ... );
    glRotatef( ... );
    head();
    glPopMatrix();
    glPushMatrix();
    glTranslatef( ... );
    glRotatef( ... );
    left_upper_arm();
    glPushMatrix();
    glTranslatef( ... );
    glRotatef( ... );
    left_lower_arm();
    glPopMatrix();
    glPopMatrix();
    ...
}
```

Order of transformations

Let's revisit the very first simple example in this lecture.

To draw the transformed house, we would write OpenGL code like:

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

Note that we are building the composite transformation matrix by starting from the left and postmultiplying each additional matrix.

Global, fixed coordinate system

One way to think of transformations is as movement of points in a **global, fixed coordinate system**

- Build the transformation matrix sequentially from left to right: T, then R, then S
- Then apply the transformation matrix to the object points: multiply all the points in P by the composite matrix TRS
  - this transformation takes the points from original to final positions

```
translated rotated scaled draw
```
Local, changing coordinate system

Another way to think of transformations is as affecting a local coordinate system that the primitive is eventually drawn in.

This is EXACTLY the same transformation as on the previous page, it's just how you look at it.

Draw!

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.

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.
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 the notion of a model tree or DAG can be extended to entire scenes.
- How OpenGL transformations can be used in hierarchical modeling.
- How keyframe animation works.