Hierarchical Modeling

Symbols and instances

Most graphics APIs support a few geometric primitives:

- spheres
- cubes
- cylinders

These symbols are \textbf{instanced} using an \textbf{instance transformation}.

\textbf{Q:} What is the matrix for the instance transformation above?

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 $\psi$

(Note that the angles are set to zero in the figure; i.e., the parts are shown in their “default” positions.)

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();
    . . .
}
```

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

```c
main()
{
    . . .
    $M_{\text{model}} = \text{Identity}();$
    robot_arm();
    . . .
}
```

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

Robot arm implementation, OpenGL

OpenGL maintains a global state matrix called the model-view matrix, 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();
}
```
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

```
figure()
{
  torso();
  glPushMatrix();
  glTranslate( ... );
  glRotate( ... );
  head();
  glPopMatrix();
  glPushMatrix();
  glTranslate( ... );
  glRotate( ... );
  left_upper_arm();
  glPushMatrix();
  glTranslate( ... );
  glRotate( ... );
  left_lower_arm();
  glPopMatrix();
  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.
**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 graph diagram]