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?
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:

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

main()
{
    
    robot_arm();
    
}

robot_arm()
{
    $M_{\text{model}} = R_y(\theta)$;
    base();
    $M_{\text{model}} = R_y(\theta) \cdot T(0, h_1, 0) \cdot R_z(\phi)$;
    upper_arm();
    $M_{\text{model}} = R_y(\theta) \cdot T(0, h_1, 0) \cdot R_z(\phi) \cdot T(0, h_2, 0) \cdot R_z(\psi)$;
    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:

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

main()
{
    
    robot_arm();
    
}

robot_arm()
{
    $M_{\text{model}} = \text{Identity}();$
    base();
    $M_{\text{model}} = R_y(\theta)$;
    base();
    $M_{\text{model}} = T(0, h_1, 0) \cdot R_z(\phi)$;
    upper_arm();
    $M_{\text{model}} = T(0, h_2, 0) \cdot R_z(\psi)$;
    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.

main()
{
    
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();
    robot_arm();
    
}

robot_arm()
{
    glRotatef( $\theta$, 0.0, 1.0, 0.0 );
    base();
    glTranslatef( 0.0, $h_1$, 0.0 );
    glRotatef( $\phi$, 0.0, 0.0, 1.0 );
    lower_arm();
    glTranslatef( 0.0, $h_2$, 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();
  ...
}
```

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]