Hierarchical Modeling

Brian Curless
CSE 557
Fall 2014

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?

\[
M = SRT
\]

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?
3D Example: A robot arm

An alternative interpretation is that we are taking the original coordinate frames...

...and translating and rotating them into place:

Robot arm implementation

The robot arm can be displayed by keeping a global matrix and computing it at each step:

Matrix M, M_model, M_view;

main()
{
    . . .
    M_view = compute_view_transform();
    robot_arm();
    . . .
}

robot_arm()
{
    M_model = R_y(theta);
    M = M_view*M_model;
    base();
    M_model = R_y(theta)*T(0,h1,0)*R_z(\phi);
    M = M_view*M_model;
    upper_arm();
    M_model = R_y(theta)*T(0,h1,0)*R_z(\phi)*T(0,h2,0)*R_z(\psi);
    M = M_view*M_model;
    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_modelview;

main()
{
    . . .
    M_modelview = compute_view_transform();
    robot_arm();
    . . .
}

robot_arm()
{
    M_modelview *= R_y(theta);
    base();
    M_modelview *= T(0,h1,0)*R_z(\phi);
    upper_arm();
    M_modelview *= T(0,h2,0)*R_z(\psi);
    lower_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 );
    Matrix M = compute_view_xform();
    glLoadMatrixf( M );
    robot_arm();
    ...}

robot_arm()
{
    glRotatex{ theta, 0.0, 1.0, 0.0 }; base();
    glTranslatef( 0.0, h1, 0.0 );
    glRotatex{ phi, 0.0, 0.0, 1.0 }; lower_arm();
    glRotatex{ 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?

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

**Human figure implementation, OpenGL**
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: **spline**  
- A good interactive system  
- A lot of skill on the part of the animator