Hierarchical Modeling

CSE 457, Autumn 2003
Graphics

http://www.cs.washington.edu/education/courses/457/03au/

References


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)

```
translated  rotated  scaled  points to draw
```

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 $\theta$
- 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? the upper arm? the lower arm?

Robot arm implementation

The robot arm could be displayed by using a global matrix and recomputing it at each step:

```c
Matrix M_model;

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

robot_arm() {
    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 just a tad wasteful?
Instead of recalculating the global matrix each time, we could just update it as we go along:

```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,h2,0)*R_z(psi);
    lower_arm();
}
```

OpenGL maintains a global state matrix called the **model-view matrix**.

```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 );
    upper_arm();
    glTranslatef( 0.0, h2, 0.0 );
    glRotatef( psi, 0.0, 0.0, 1.0 );
    lower_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)

![Figure from Angel]
Another example: human figure

Q: What's a sensible way to traverse this tree?

Human figure implementation

- We can also design code for drawing the human figure, with a slight modification due to the branches in the tree:

```plaintext
figure()
    torso();
    M_save = M_model;
    M_model *= T( . . ) * R( . . );
    head();
    M_model = M_save;
    M_model *= T( . . ) * R( . . );
    left_upper_arm();
    M_model *= T( . . ) * R( . . );
    left_lower_arm();
    M_model = M_save;
    M_model *= T( . . ) * R( . . );
    left_hand();
    push(M_model);
    M_model *= T( . . ) * R( . . );
    left_forefinger();
    M_model = pop(M_model);
    M_model *= T( . . ) * R( . . );
    left_thumb();
    M_model = pop(M_model);
    M_model *= T( . . ) * R( . . );
    ...
```

Figure with hand

What if we add a hand?

```plaintext
figure()
    torso();
    M_save = M_model;
    M_model *= T( . . ) * R( . . );
    head();
    M_model = M_save;
    M_model *= T( . . ) * R( . . );
    left_upper_arm();
    M_model *= T( . . ) * R( . . );
    left_lower_arm();
    M_model *= T( . . ) * R( . . );
    left_hand();
    M_save2 = M_model;
    M_model *= T( . . ) * R( . . );
    left_thumb();
    M_model = M_save2;
    M_model *= T( . . ) * R( . . );
    left_forefinger();
    M_model = M_save2;
    ...
```

Is there a better way to keep track of piles of matrices that need to be saved, modified, and restored?

Push and pop

```plaintext
figure()
    torso();
    push(M_model);
    M_model *= T( . . ) * R( . . );
    head();
    M_model = pop(M_model);
    push(M_model);
    M_model *= T( . . ) * R( . . );
    left_upper_arm();
    M_model *= T( . . ) * R( . . );
    left_lower_arm();
    M_model *= T( . . ) * R( . . );
    left_hand();
    push(M_model);
    M_model *= T( . . ) * R( . . );
    left_forefinger();
    M_model = pop(M_model);
    push(M_model);
    M_model *= T( . . ) * R( . . );
    left_thumb();
    M_model = pop(M_model);
    push(M_model);
    M_model *= T( . . ) * R( . . );
    ...
```
Push and pop, OpenGL

```c
figure() {
torso();
glPushMatrix();
glTranslate(...);
glRotate(...);
head();
glPopMatrix();
glPushMatrix();
glTranslate(...);
glRotate(...);
left_upper_arm();
glTranslate(...);
glRotate(...);
left_lower_arm();
glTranslate(...);
glRotate(...);
left_hand();
glPushMatrix();
  glTranslate(...);
glRotate(...);
left_thumb();
glPopMatrix();
glPushMatrix();
  glTranslate(...);
glRotate(...);
left_forefinger();
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.

Kinematics and dynamics

- Definitions:
  - **Kinematics:** how the positions of the parts vary as a function of the joint angles.
  - **Dynamics:** how the positions of the parts vary as a function of applied forces.
- Questions:
  - **Q:** What do the terms *inverse kinematics* and *inverse dynamics* mean?
  - **Q:** Why are these problems more difficult?

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**.

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:

```plaintext
glMatrixMode( GL_MODELVIEW );
glLoadIdentity();
glTranslatef( ... );
glRotatef( ... );
glScalef( ... );
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

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.
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 keyframe animation works.
  » How transforms can be thought of as affecting either the geometry, or the coordinate system which it is drawn in.