

Hierarchical Modeling

1

Reading

Required:

- ♦ Angel, sections 10.1 – 10.6, 10.8

Optional:

- ♦ *OpenGL Programming Guide*, chapter 3

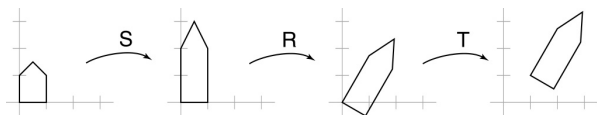
2

Symbols and instances

Most graphics APIs support a few geometric **primitives**:

- ♦ spheres
- ♦ cubes
- ♦ cylinders

These symbols are **instanced** using an **instance transformation**.



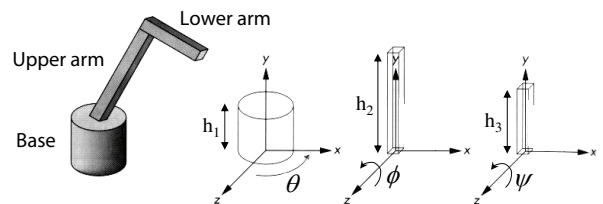
Q: What is the matrix for the instance transformation above?

3

3D Example: A robot arm

Consider this robot arm with 3 degrees of freedom:

- ♦ Base rotates about its vertical axis by θ
- ♦ Upper arm rotates in its xy -plane by ϕ
- ♦ Lower arm rotates in its xy -plane by ψ



(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?

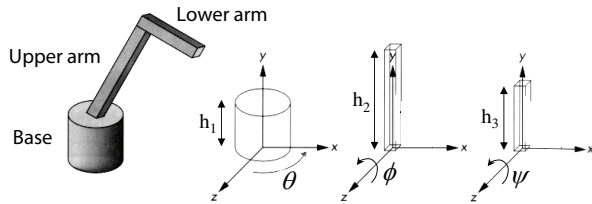
Q: What matrix for the upper arm?

Q: What matrix for the lower arm?

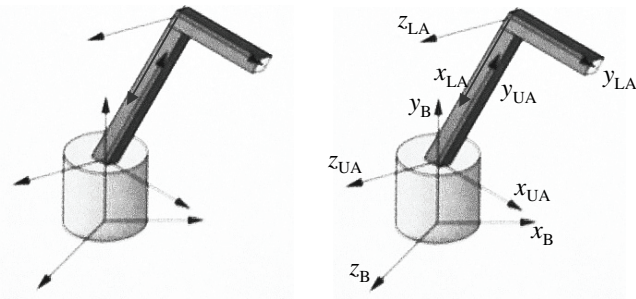
4

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_model;

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

robot_arm()
{
    M_model = R_y(theta);
    base();
    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 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_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();
}
    
```

Robot arm implementation, OpenGL

OpenGL maintains a global state matrix called the **model-view matrix**, which is updated by concatenating matrices on the **right**.

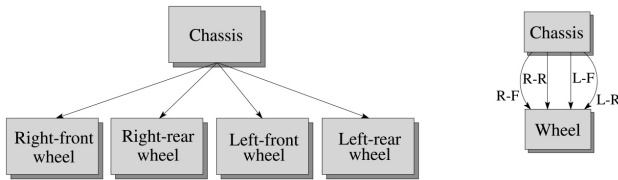
```

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

robot_arm()
{
    glRotatef( theta, 0.0, 1.0, 0.0 );
    base();
    glTranslatef( 0.0, 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:

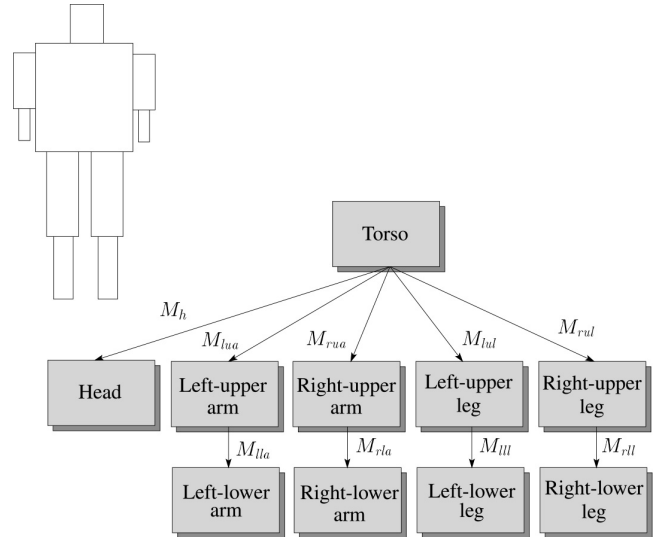


- ♦ edges contain geometric transformations
- ♦ nodes contain geometry (and possibly drawing attributes)

How might we draw the tree for the robot arm?

9

A complex example: human figure



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

10

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();
    ...
}
```

11

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.

12

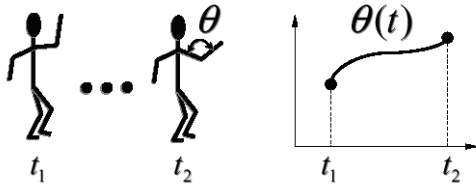
Key-frame animation

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



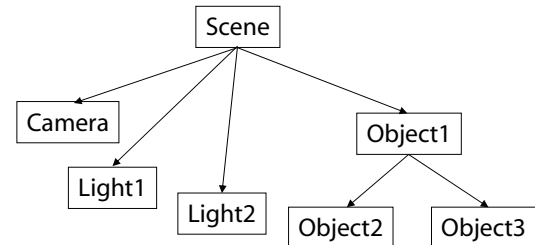
13

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**.



14

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.

15