

# Hierarchical Modeling

CSE 457

# Reading

Required:

- ◆ Angel, sections 8.1 – 8.6, 8.8

Optional:

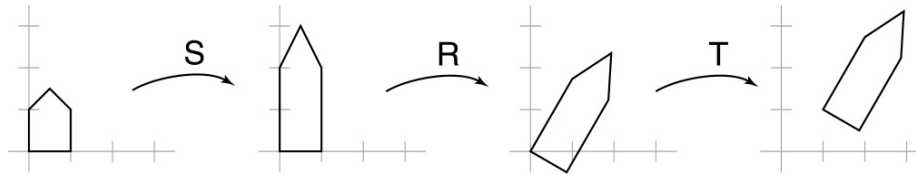
- ◆ *OpenGL Programming Guide*, chapter 3

# Symbols and instances

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

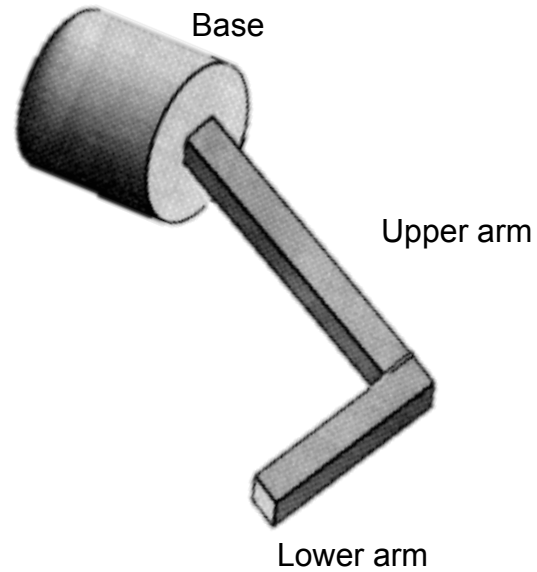
- ◆ spheres
- ◆ cubes
- ◆ cylinders

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

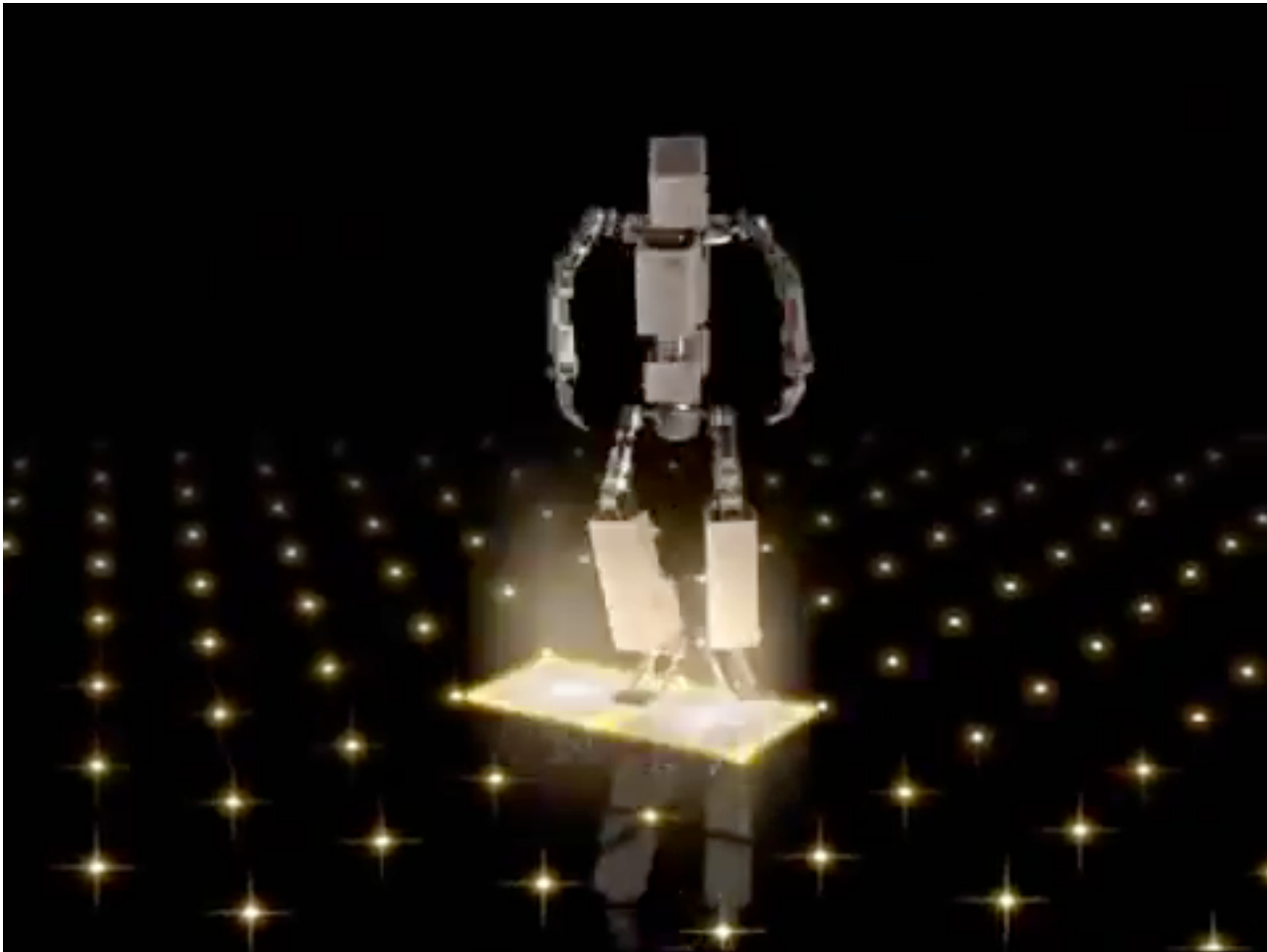


**Q:** What is the matrix for the instance transformation above?

## 3D Example: A robot arm



Have to be constrained via a hierarchical model

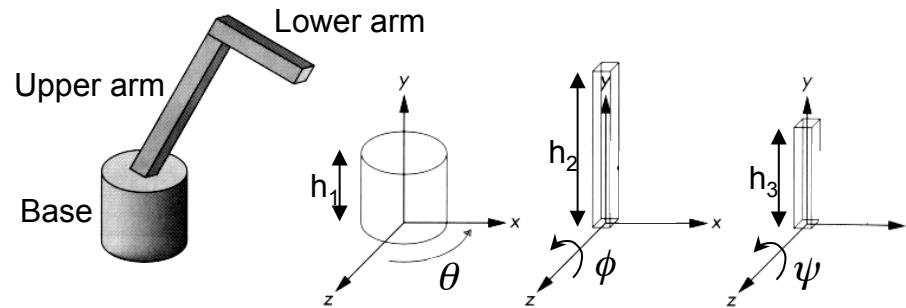


“Number One” Playgroup – Duran Duboi

## 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  $\psi$



[Angel, 2011]

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

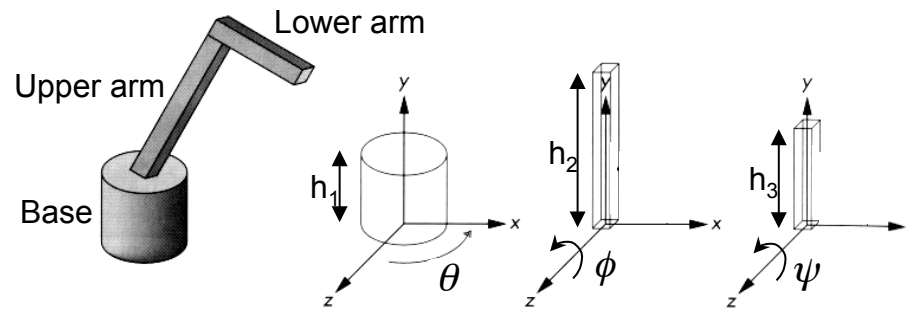
**Q:** What matrix do we use to transform the base?

**Q:** What matrix for the upper arm?

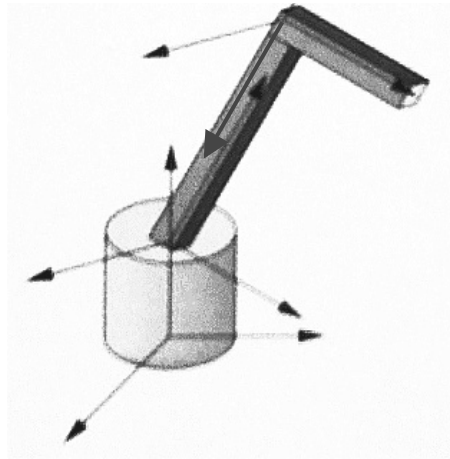
**Q:** What matrix for the lower arm?

## 3D Example: A robot arm

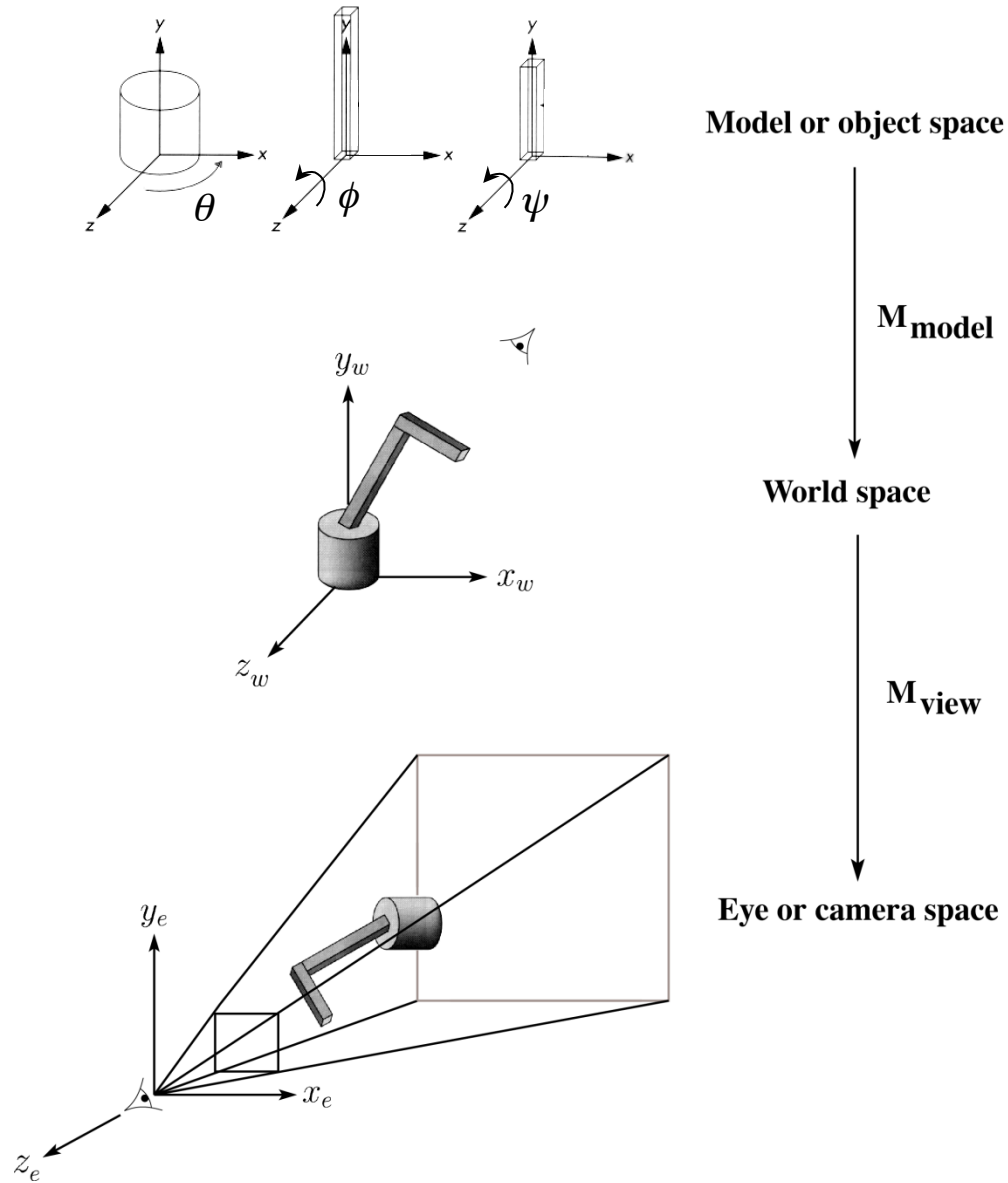
An alternative interpretation is that we are taking the original coordinate frames...



...and translating and rotating them into place:



# From parts to model to viewer





# Robot arm implementation

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

```
Matrix M_model;
Matrix M_view;

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

robot_arm()
{
    M_model = M_view*R_y(theta);
    base();
    M_model = M_view*R_y(theta)*T(0,h1,0)*R_z(phi);
    upper_arm();
    M_model = M_view*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_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();
}
```

# Robot arm implementation, OpenGL

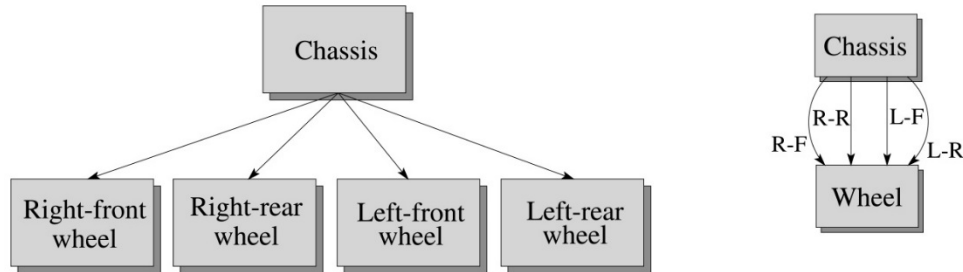
OpenGL maintains a global state matrix called the **model-view matrix**, which is updated by concatenating matrices on the *right*.

```
main()
{
    . . .
    glMatrixMode( GL_MODELVIEW );
    Matrix M = compute_view_xform();
    glLoadMatrixf( M );
    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

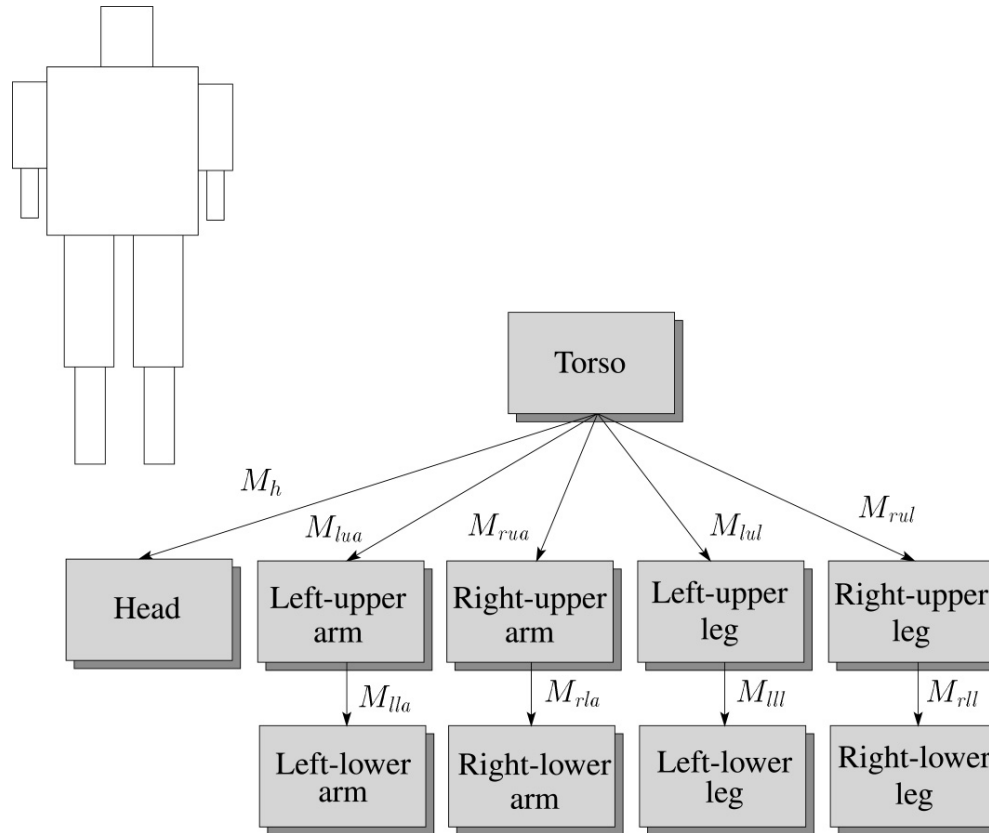
Hierarchical models can be composed of instances using trees or DAGs:



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

How might we draw the tree for the robot arm?

## A complex example: human figure



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

## Implementing hierarchies:

A matrix stack that you can push/pop (LIFO).

Recursive algorithm that descends the model tree:

- Load identity matrix
- For each node:
  - Push a new matrix onto stack
  - Concatenate transformations onto current
  - Recursively descend the tree
  - Pop matrix out of stack
- For each leaf node:
  - Draw using the current transformation matrix

# 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();
    . . .
}
```

# 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.



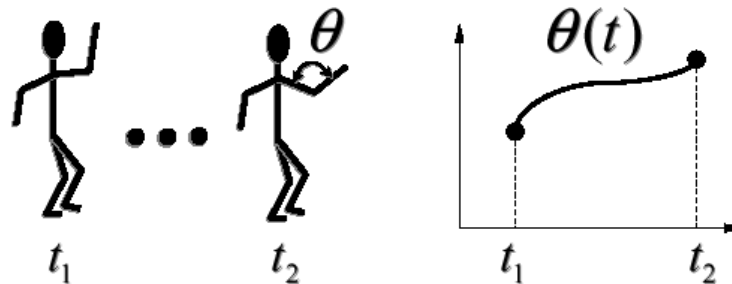
# Key-frame animation

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

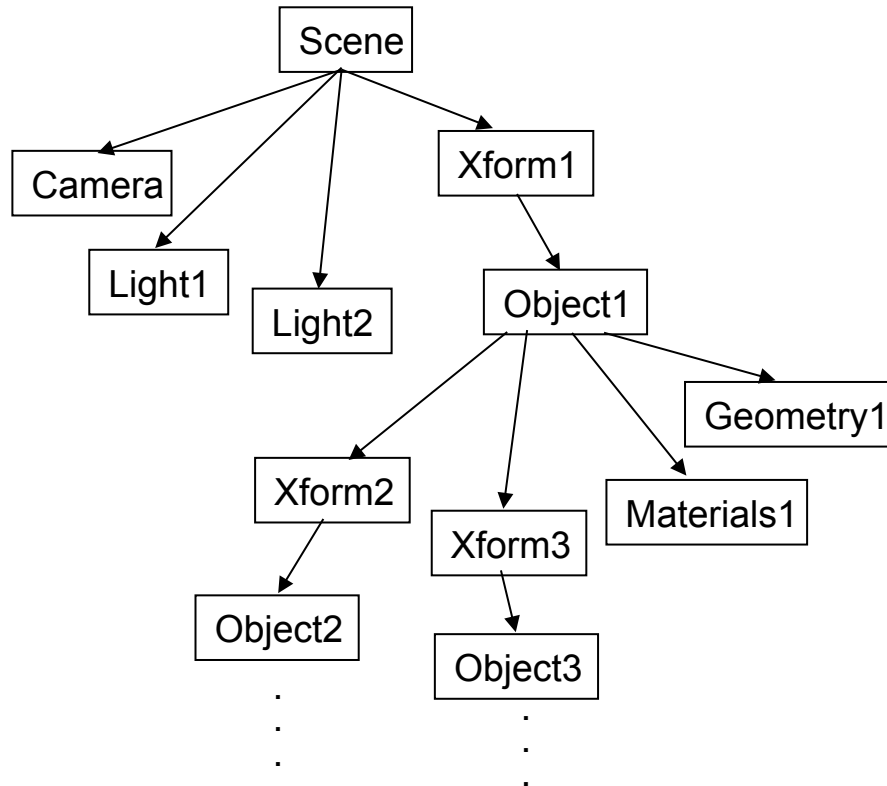


# Scene graphs

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**.



## 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.