Hierarchical Modeling

Brian Curless
CSE 457
Spring 2016

Reading
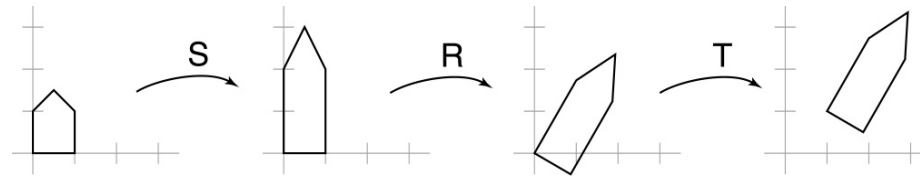
Required:

 ◆ 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?
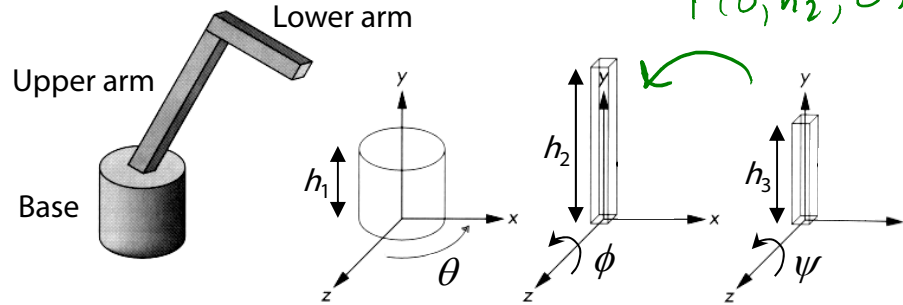
$$M \neq S R T$$

$$M = T R S$$

3D Example: A robot arm

Let's build a robot arm out of a cylinder and two cuboids, with the following 3 degrees of freedom:
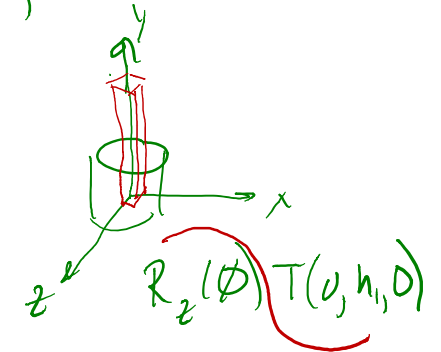
- Base rotates about its vertical axis by θ
- Upper arm rotates in its xy-plane by ϕ
- Lower arm rotates in its xy-plane by ψ

$R_x(\cdot) \quad R_y(\cdot) \quad R_z(\cdot)$

$T(\cdot, \cdot, \cdot)$

$T(0, h_2, 0) \, R_z(\psi)$

$R_z(\phi) \, T(0, h_1, 0)$

Lower arm

Upper arm

Base

[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?
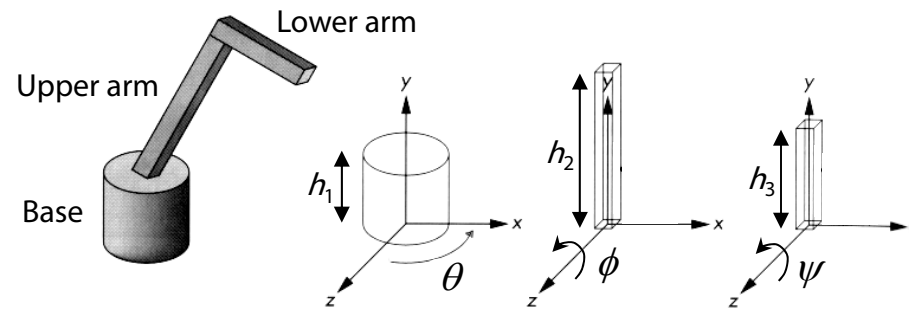
Q: What matrix for the upper arm?

Q: What matrix for the lower arm?

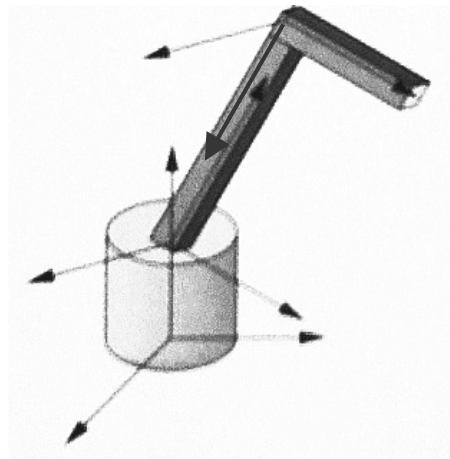$R_y(\theta) \, T(0, h_1, 0) \, R_z(\phi) \, T(0, h_2, 0) \, R_z(\psi)$

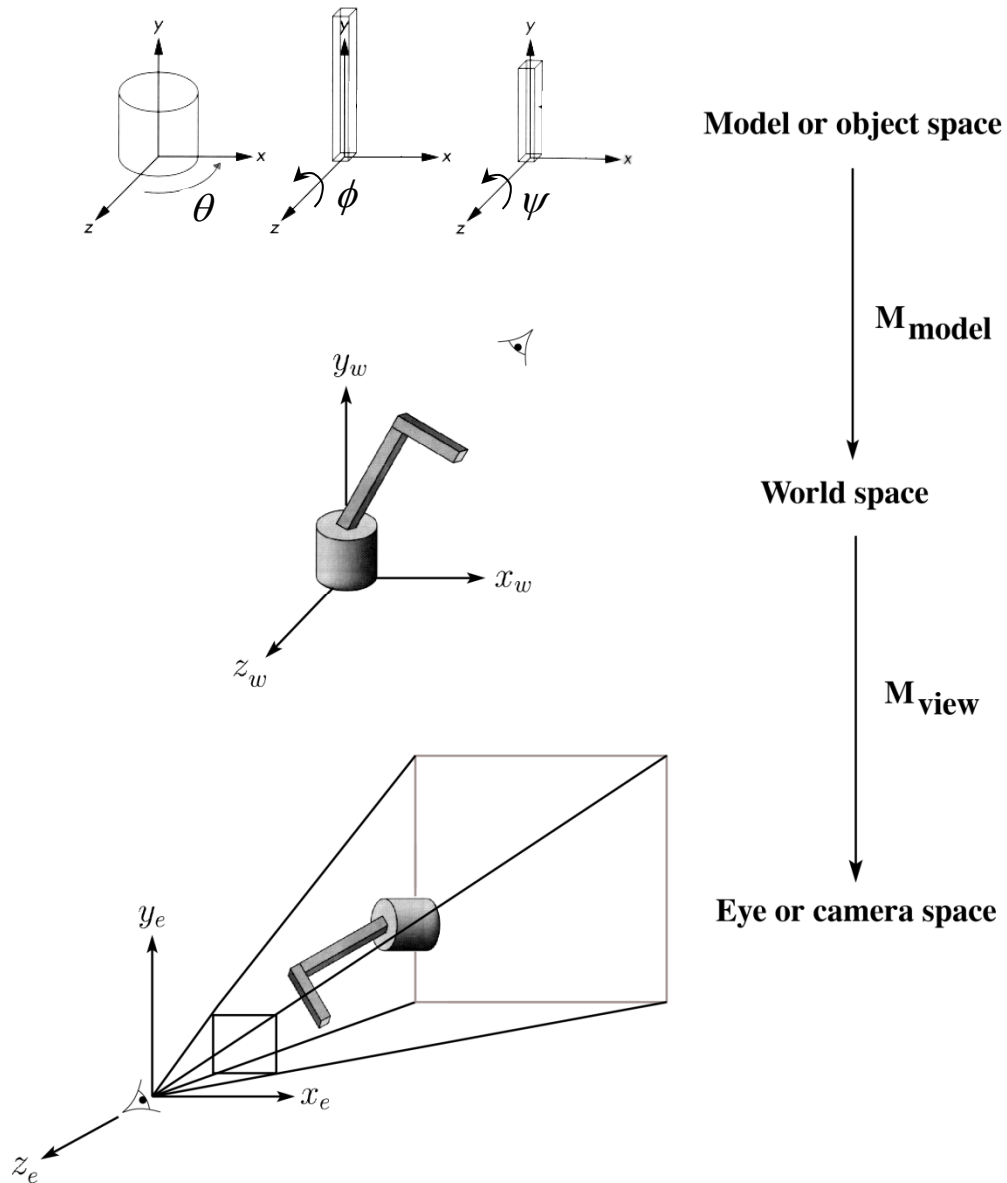base

upper arm

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:

From parts to model to viewer

Model or object space

$\mathbf{M}_{model}$

World space

$\mathbf{M}_{view}$

Eye or camera space

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();

}
```
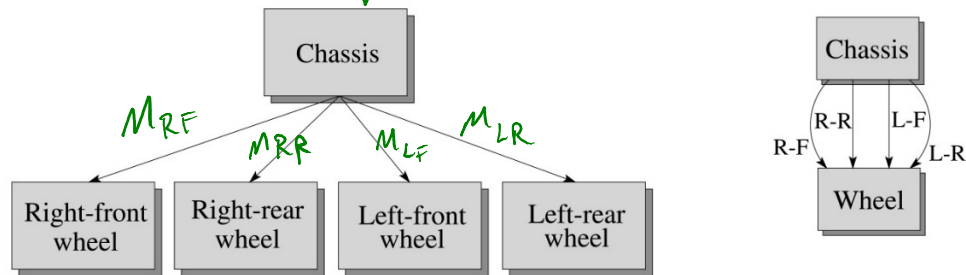
Robot arm implementation, OpenGL

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: M_W {world}

Chassis

M_{RF} M_{RR} M_{LF} M_{LR}

Right-front wheel   Right-rear wheel   Left-front wheel   Left-rear wheel

Chassis

R-R   L-F
R-F   L-R

Wheel

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

M_B {World}

How might we draw the tree for the robot arm?

Base

M_{UA}

Upper arm

M_{LA}

Lower arm

10
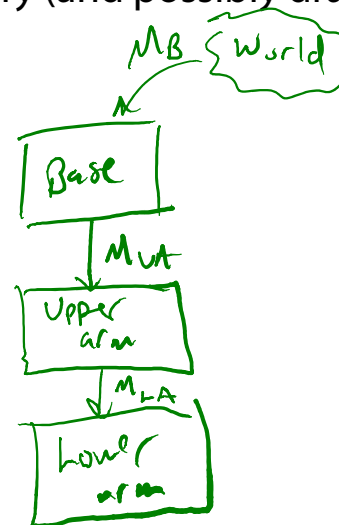
A complex example: human figure

Q: What's the most sensible way to traverse this tree?
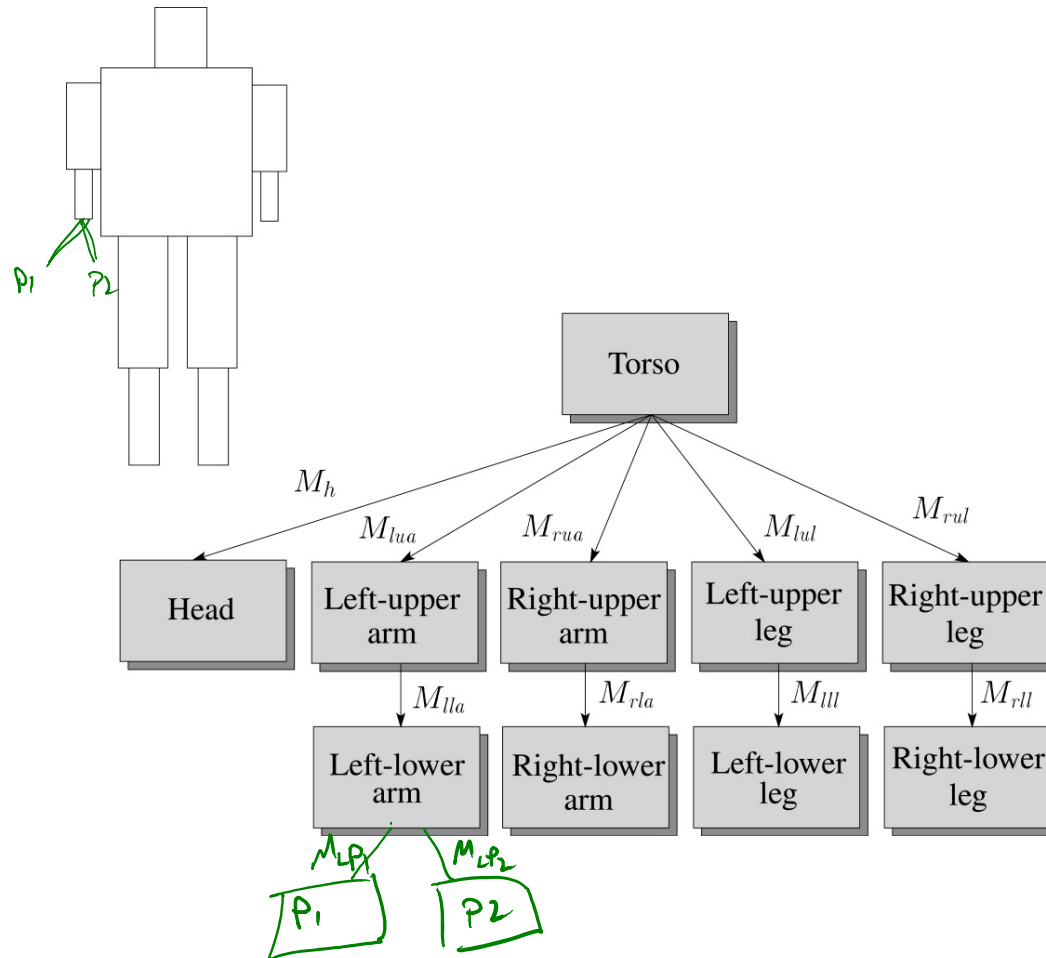
Depth First w/ stack

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

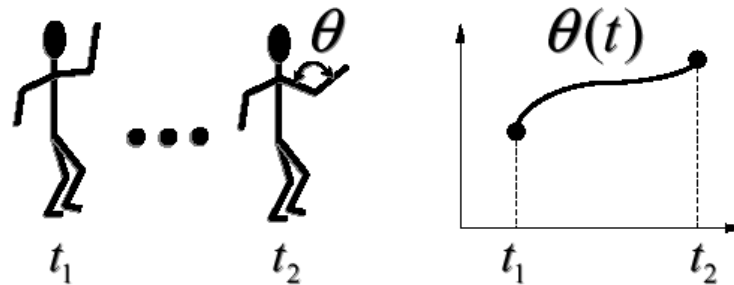The most common method for character animation in production is **key-frame animation**.

- ◆ 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
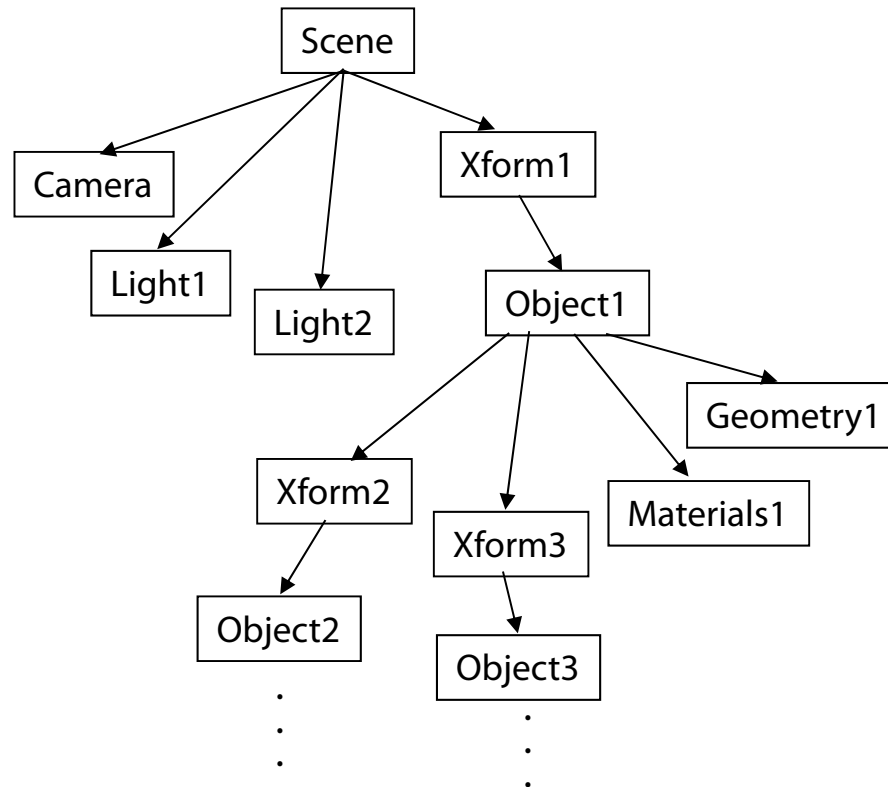- ◆ A lot of skill on the part of the animator

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**.

```
                        Scene
                  /       |      \
                 /        |       \
            Camera        |      Xform1
                 \        |         |
               Light1   Light2   Object1
                                /   |    \
                               /    |     \
                          Xform2  Xform3  Geometry1
                             |      |    Materials1
                          Object2 Xform3
                             .      |
                             .    Object3
                             .      .
                             .      .
                                    .
```

15

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.