Hierarchical Modeling

Adriana Schulz
CSE 457
Fall 2019

Reading

Optional:

- Angel, sections 8.1 – 8.6, 8.8

Further reading:

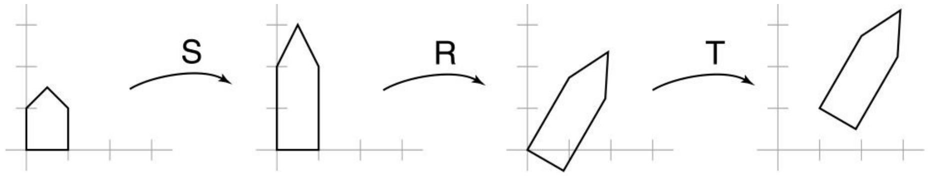
- *OpenGL Programming Guide*, chapter 3

Symbols and instances

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

- spheres
- cubes
- cylinders

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

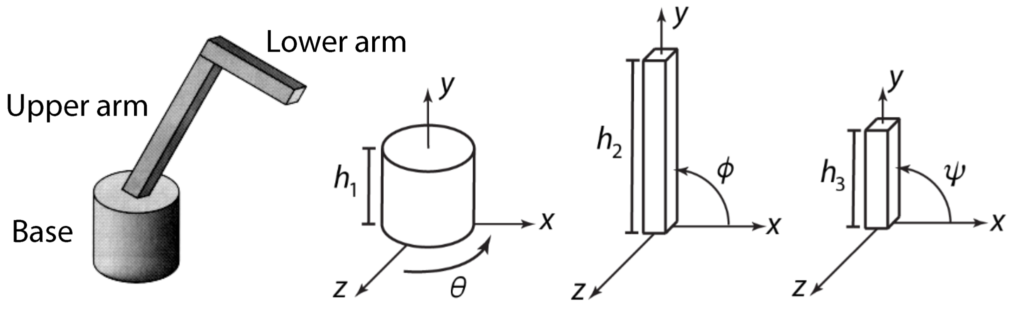


Q: What is the matrix for the instance transformation above?

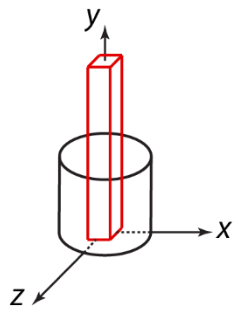
3D Example: A robot arm

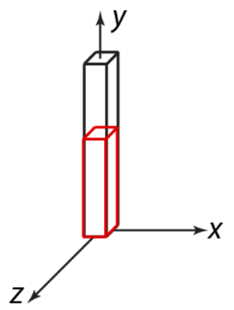
Let's build a robot arm out of a cylinder and two cuboids, with the following 3 degrees of freedom:

- 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



[Angel, 2011]





(Note that the angles are set to zero in the figures on the right; i.e., the parts are shown in their "default" positions.)

Suppose we have transformations $R_x(\cdot), R_y(\cdot), R_z(\cdot), T(\cdot\,, \cdot, \cdot)$.

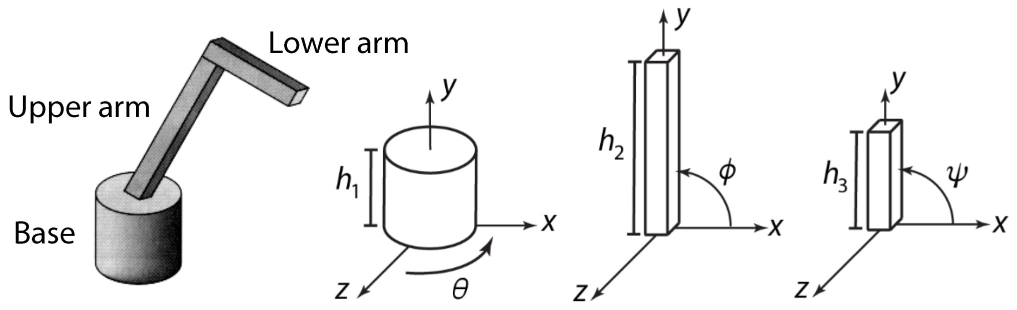
Q: What matrix do we use to transform the base?

Q: What matrix product for the upper arm?

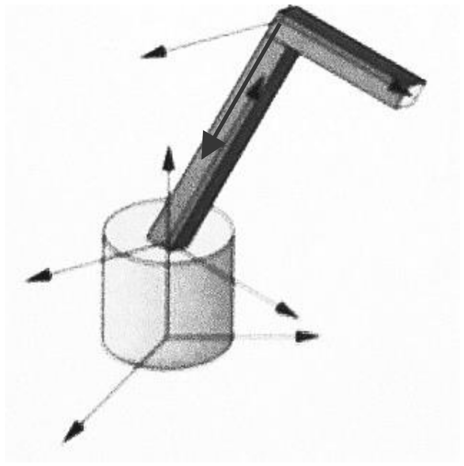
Q: What matrix product for the lower arm?

3D Example: A robot arm

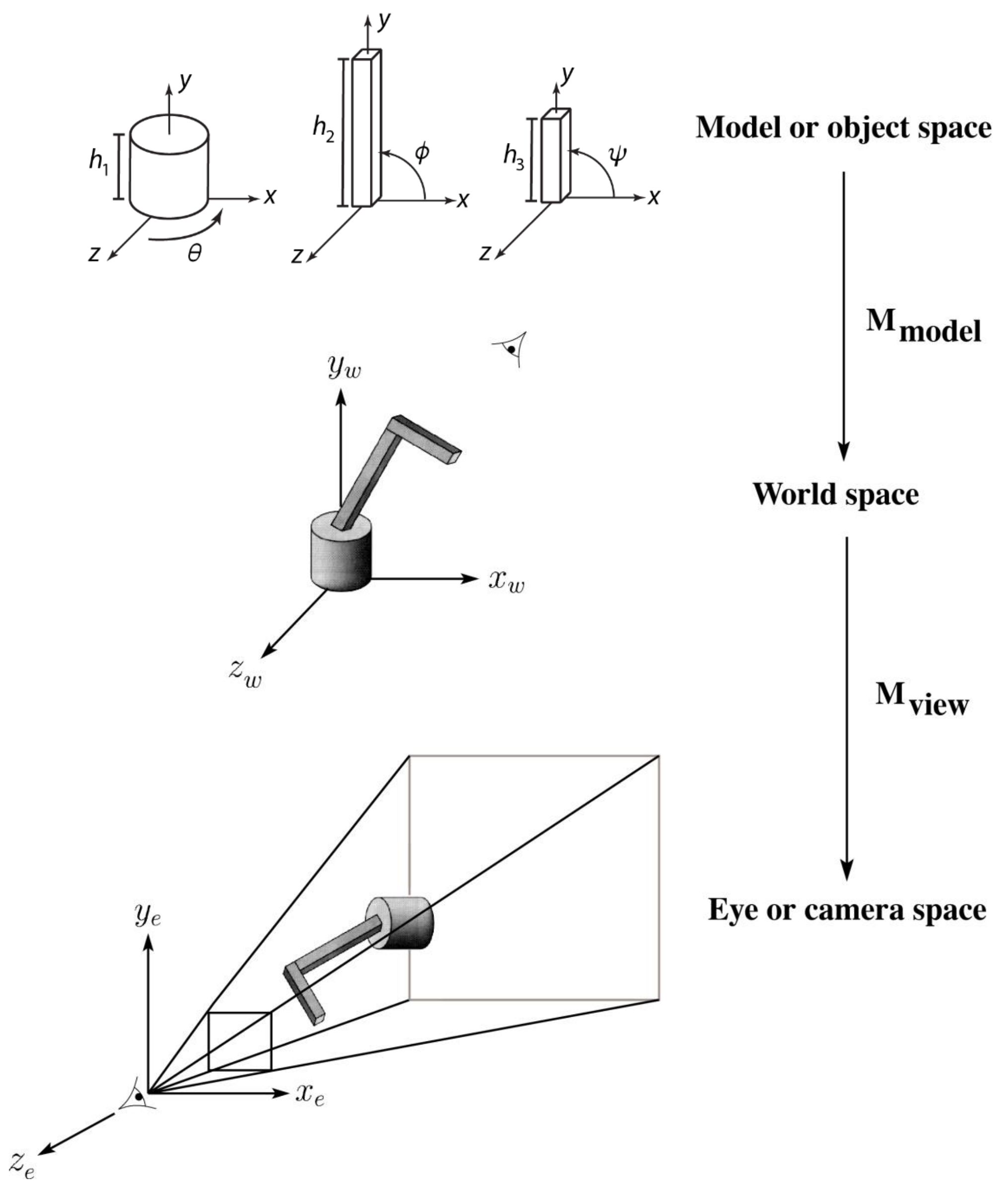
An alternative interpretation is that we are taking the original coordinate frames…



…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, 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?

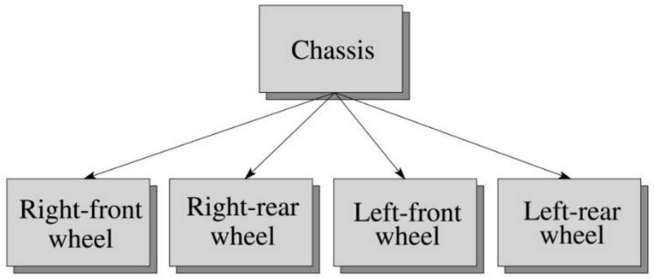
Robot arm implementation, better

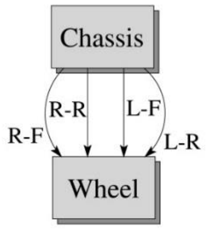
Instead of recalculating the global matrix each time, we can just update it *in place* by concatenating matrices on the right:

```
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();
}
```

Hierarchical modeling

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

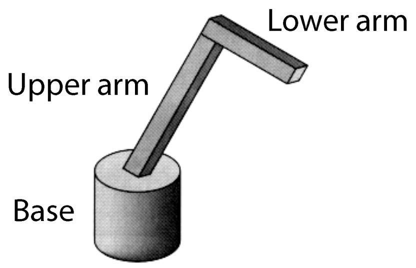




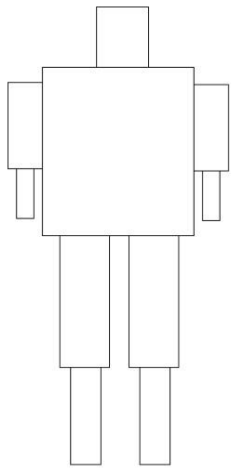
- edges contain geometric transformations
- nodes contain geometry (and possibly drawing attributes)

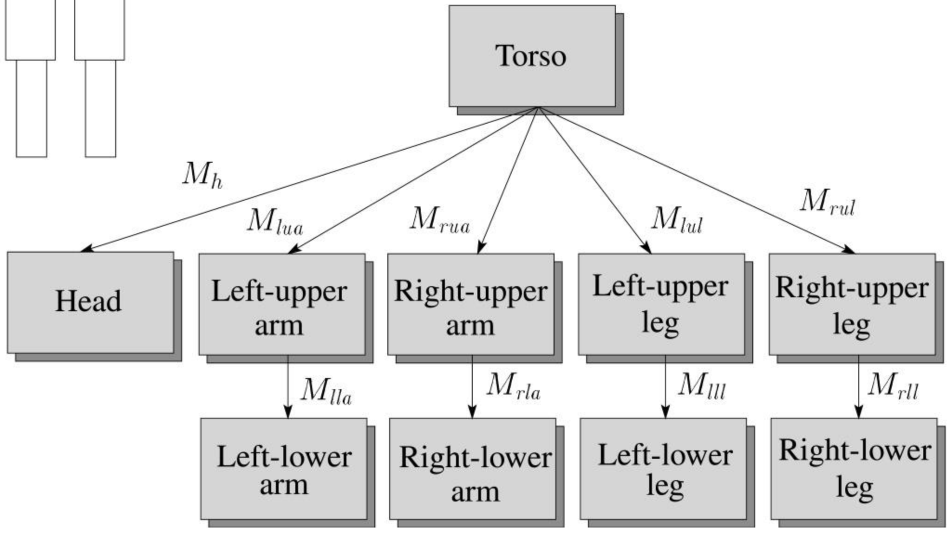
We will use trees for hierarchical models.

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?

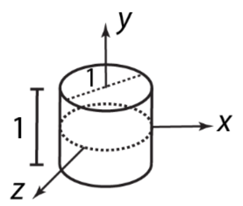
Using canonical primitives

Consider building the robot arm again, but this time the building blocks are canonical primitives like a unit cylinder and a unit cube. We can use transformations like $T(t_x,t_y,t_z)$, $S(s_x,s_y,s_z)$, $R_y(\theta)$, etc.

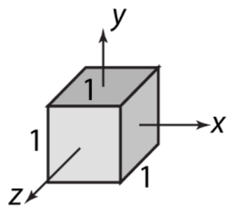
What additional transformations are needed?

What does the hierarchy look like now?

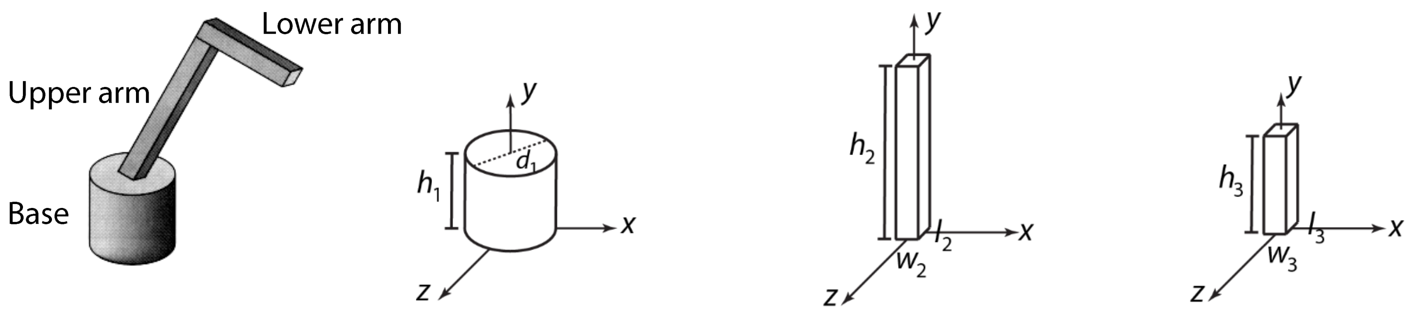
Canonical primitives



Unit cylinder



Unit cube



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.

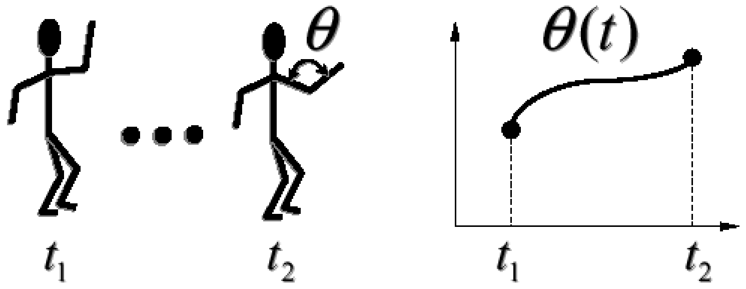
Key-frame animation

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

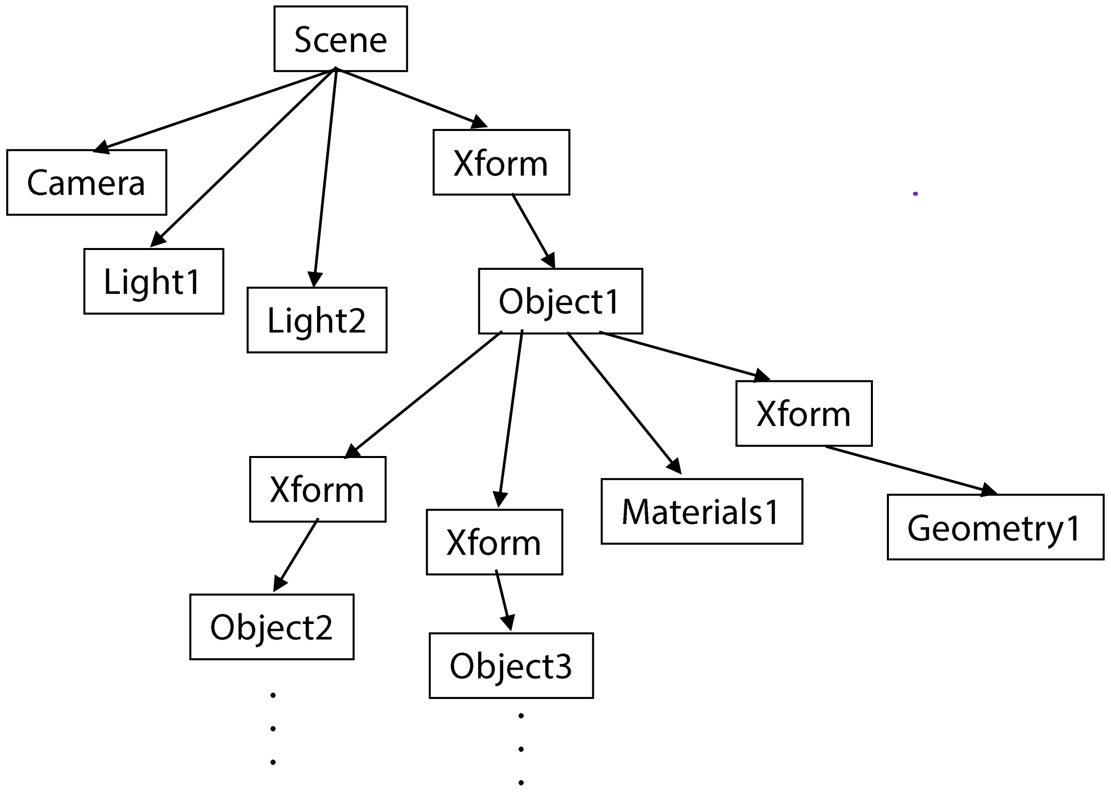


Scene graphs

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.