Hierarchical Modelling

Lecturer: Rachel McDonnell
Professor of Creative Technologies
Rachel.McDonnell@cs.tcd.ie

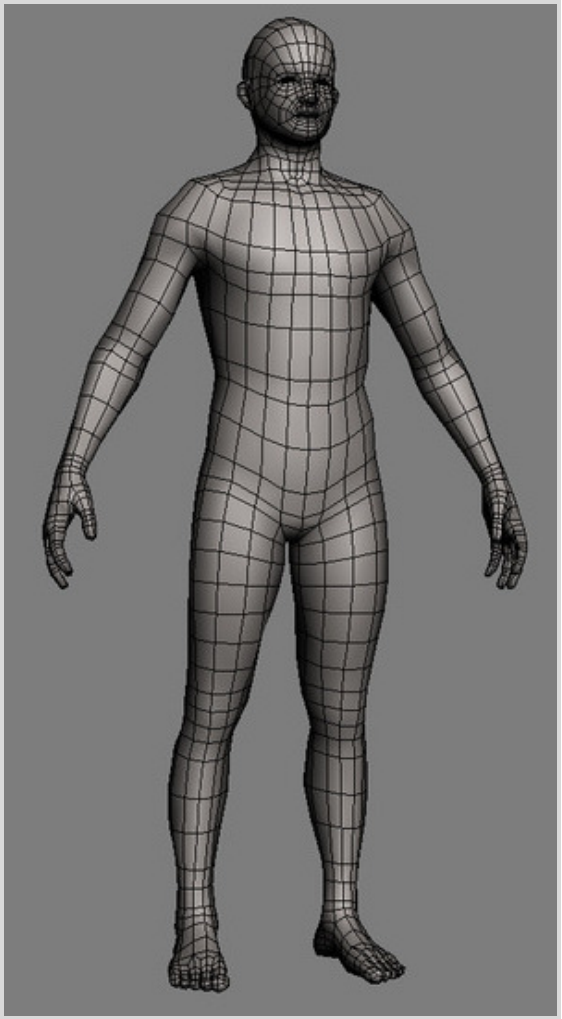
Course www:
https://www.scss.tcd.ie/Rachel.McDonnell/

Character Animation

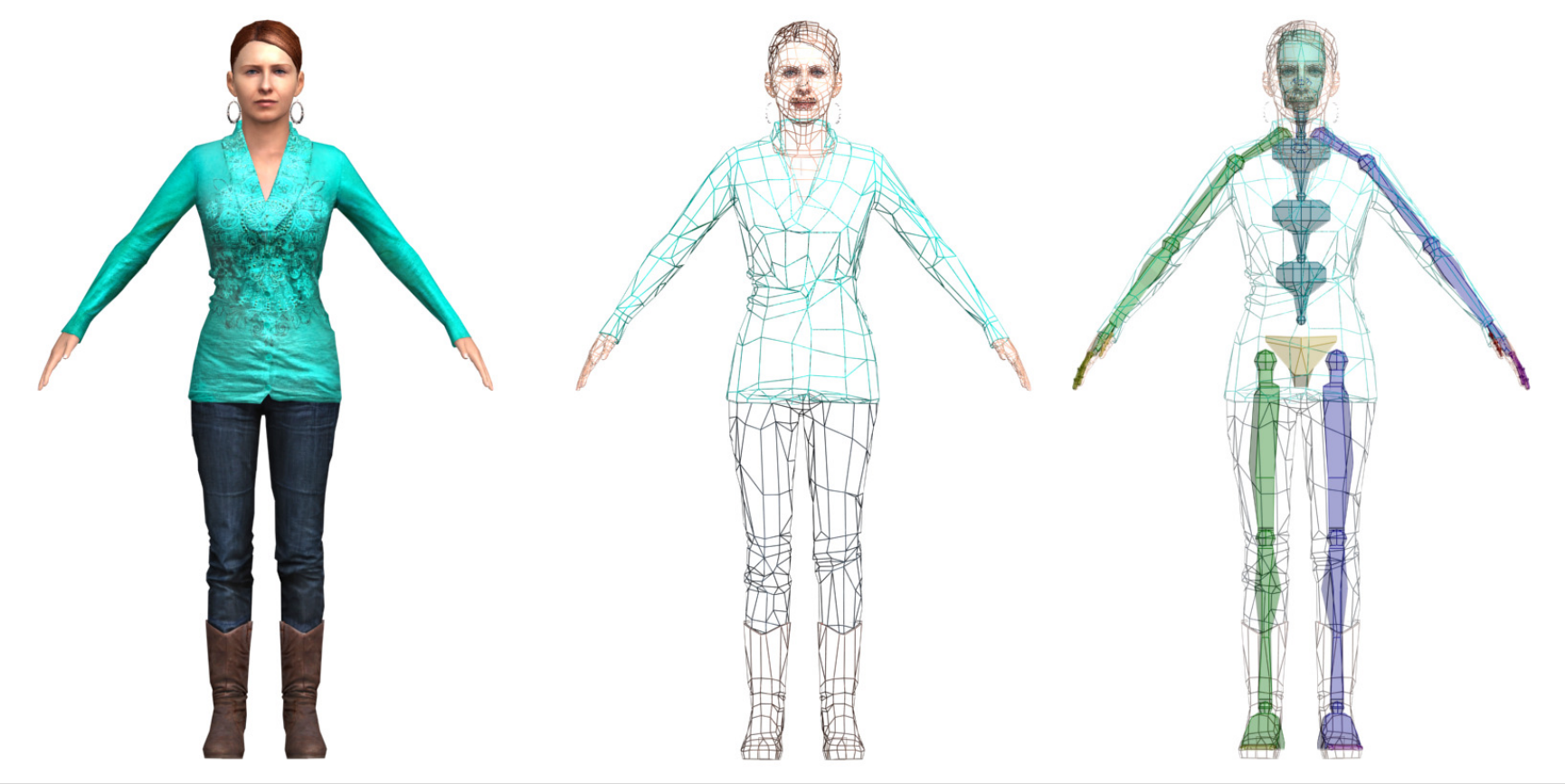
- The task of moving a complex, artificial character in a life-like manner
- Animating characters is a particularly demanding area
- Animated character must move and deform in a manner that is plausible to the viewer

Character Animation

- Animating a character model described as a polygon mesh by moving each vertex in the mesh is impractical
- Instead - specify the motion of characters through the movement of an internal articulated skeleton
  - Movement of the surrounding polygon mesh may then be deduced
- Mesh must deform in a manner that the viewer would expect, consistent with underlying muscle and tissue



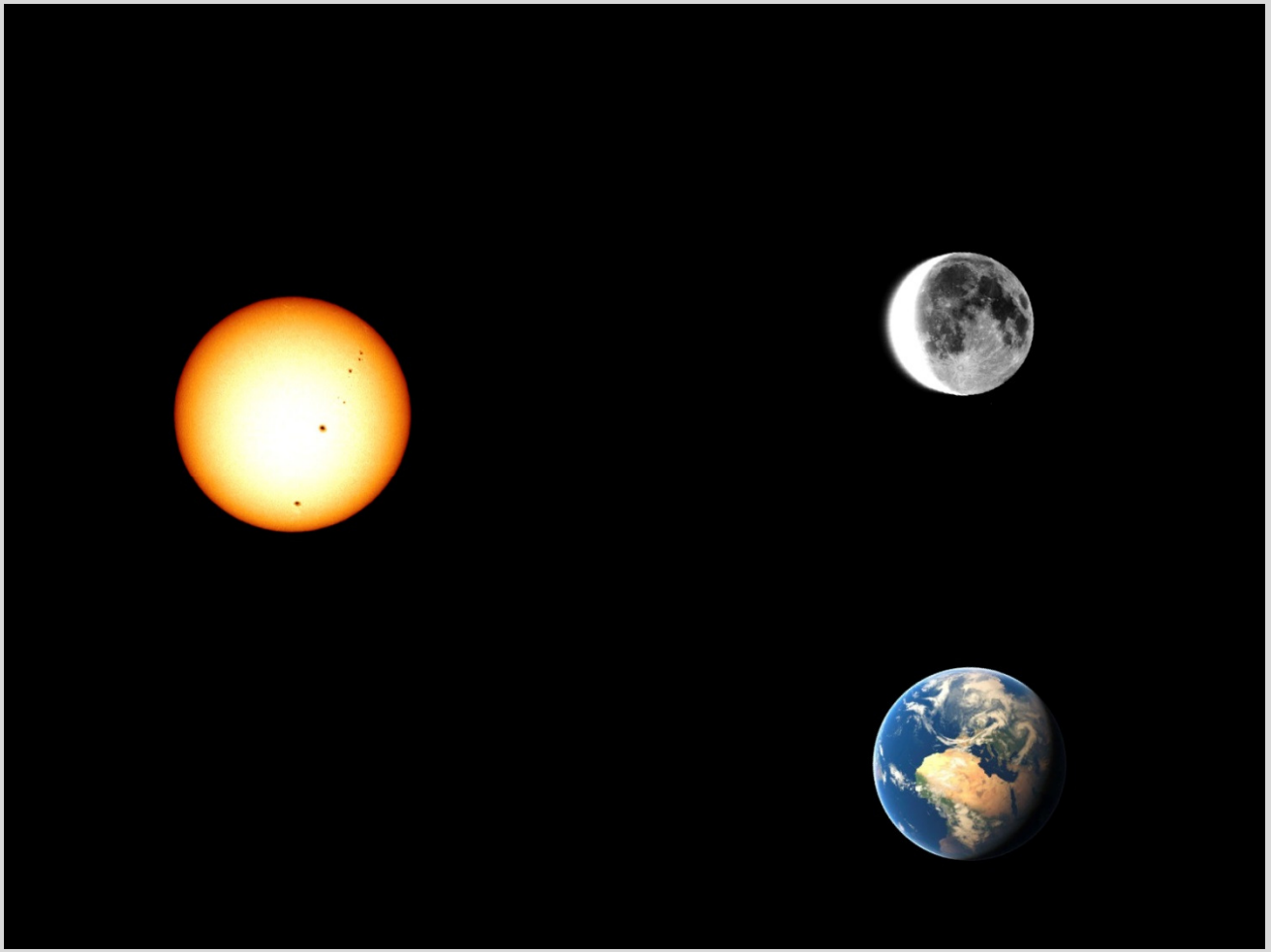
Skeleton



Rigid Body Limitations

- Consider human joints:
  - When they bend, the body shape bends as well
    - No distinct parts
- We cannot represent this with rigid bodies
  - Or the pieces would separate, where there should be stretching or compression

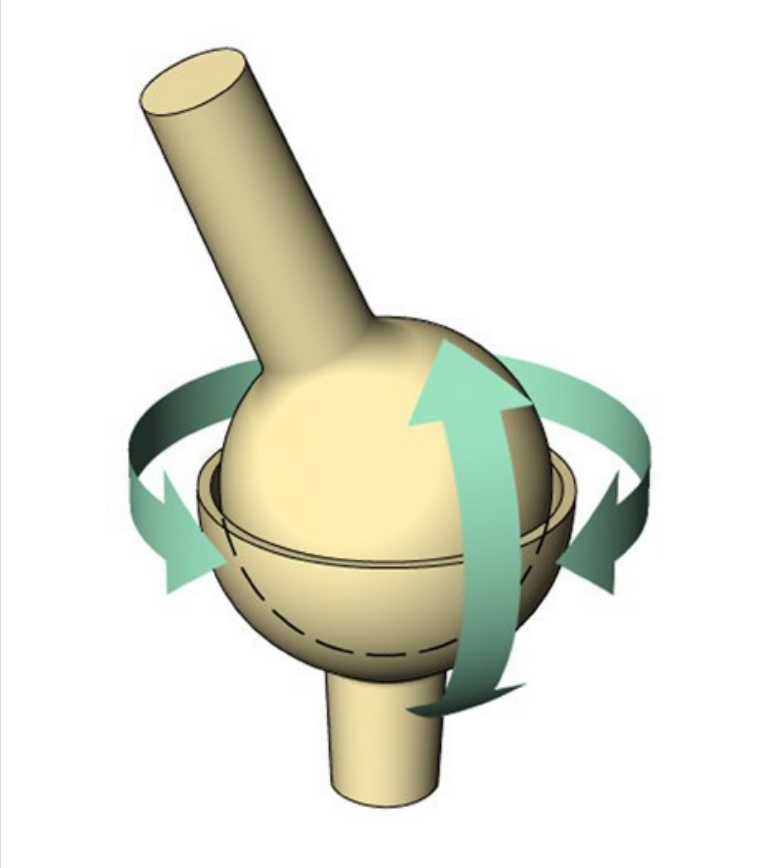
Relative Motion

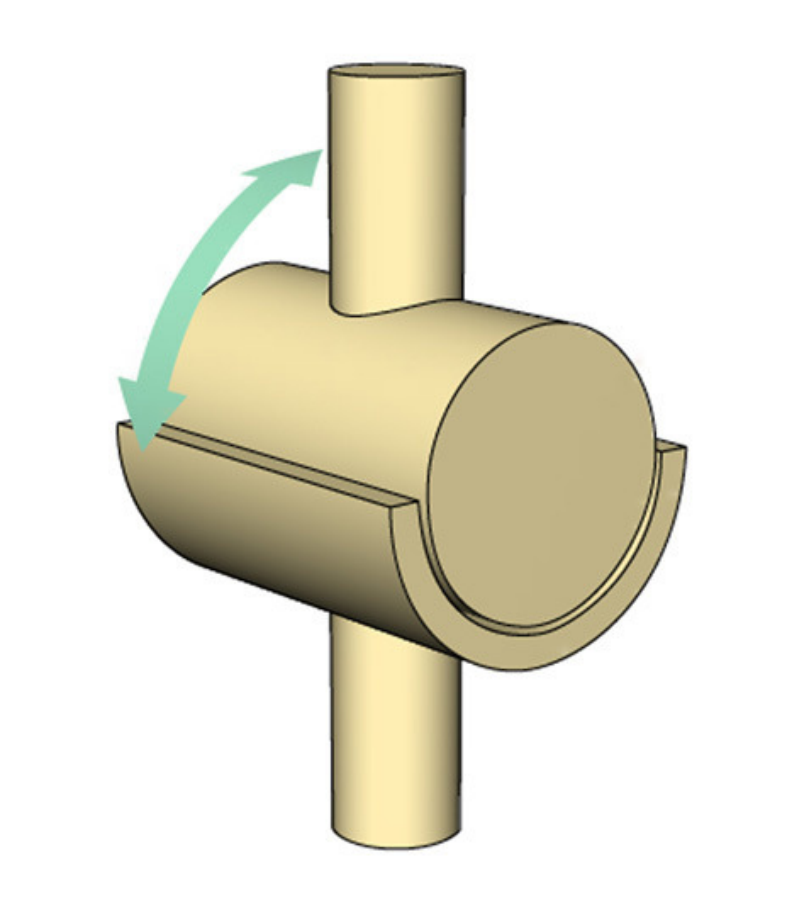


Relative Motion

- Interested in animating objects whose motion is relative to another object
- Such a sequence is called a *motion hierarchy*
- Components of a hierarchy represent objects that are physically connected or *linked*
- In some cases, motion can be restricted
  - Reduced dimensionality
  - Hierarchy enforces constraints
- Two approaches for animating figures defined by hierarchies: forward & inverse kinematics

Degrees of Freedom



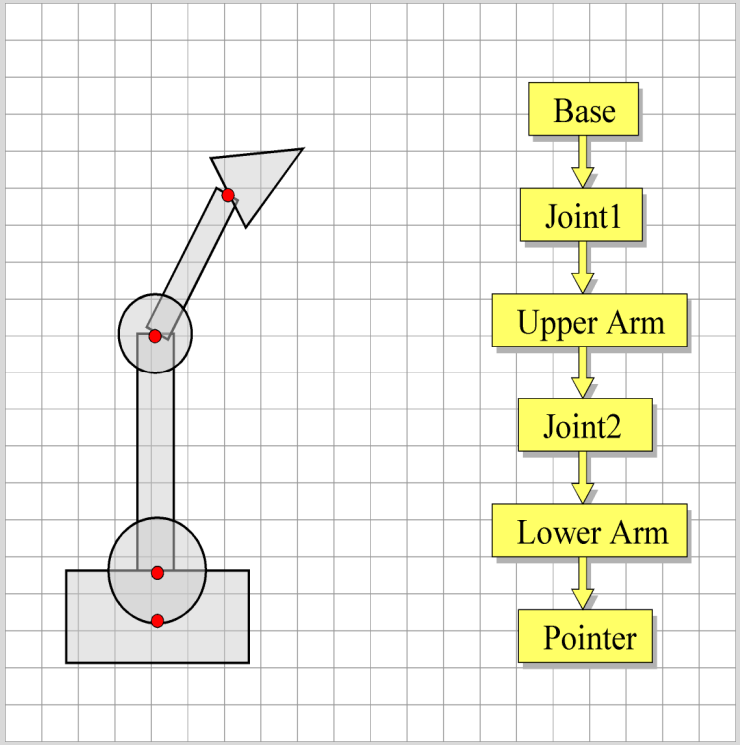


Model Transformations

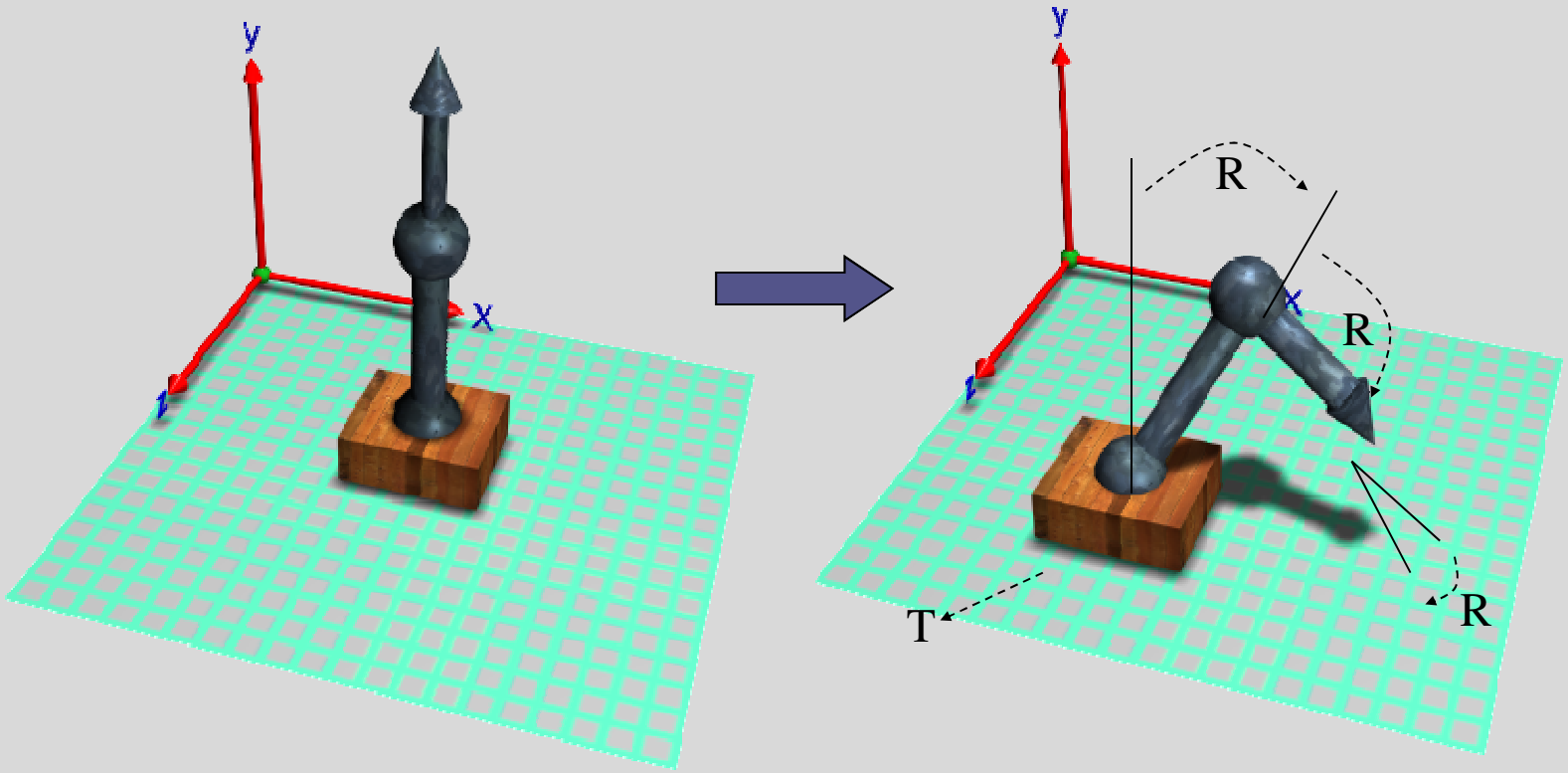
- A "local frame view" is usually adopted as it extends naturally to the specification of *hierarchical model frames*.
- This allows creation of *jointed assemblies*
  - articulated figures (animals, robots etc.)
- In the hierarchical model, each sub-component has its own *local frame*.
- Changes made to the *parent frame* are propagated down to the *child frames* (thus all models in a branch are globally controlled by the parent).
- This simplifies the specification of *animation*.

Hierarchical Transformations

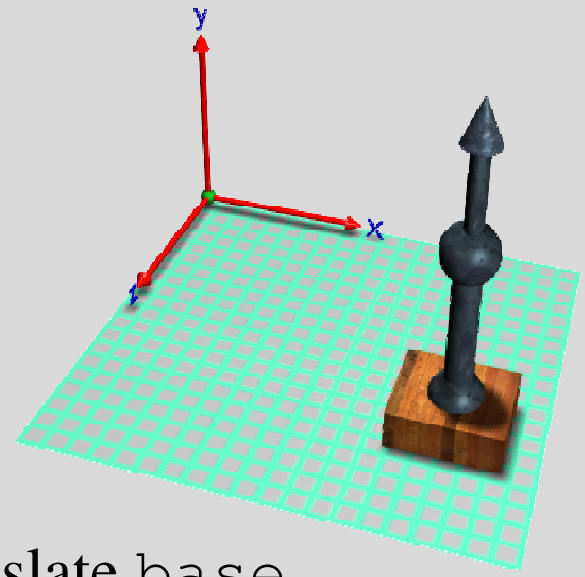
- For geometries with an implicit *hierarchy* we wish to associate local frames with sub-objects in the assembly.
- *Parent-child frames* are related via a transformation.
- Transformation linkage is described by a *tree*:
- Each node has its own *local co-ordinate system*.



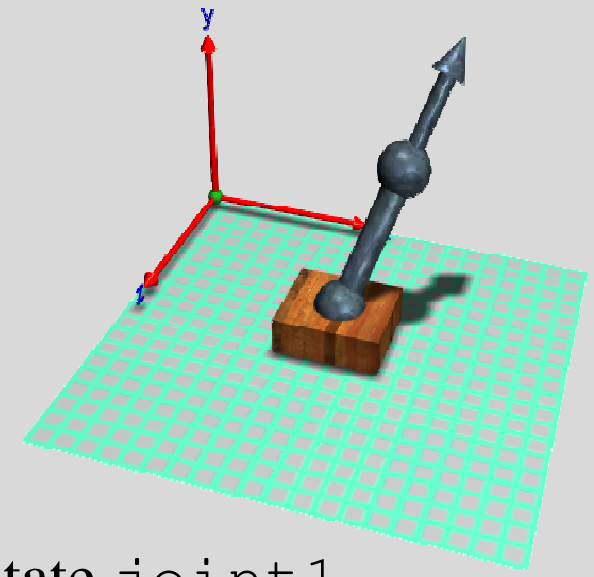
Hierarchical Transformations



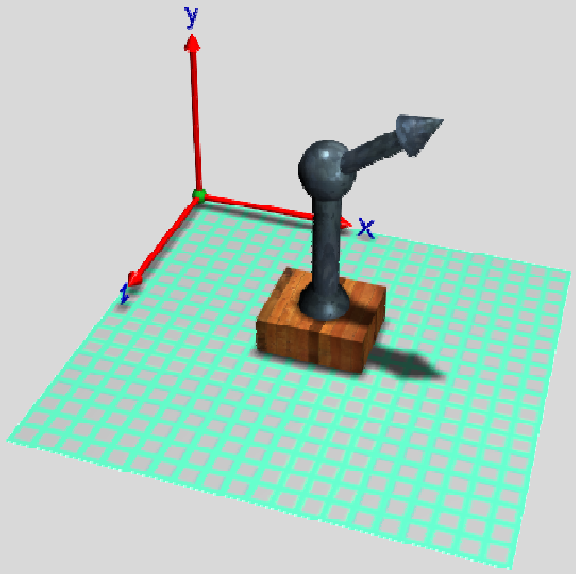
Hierarchical transformation allow independent control over sub-parts of an assembly



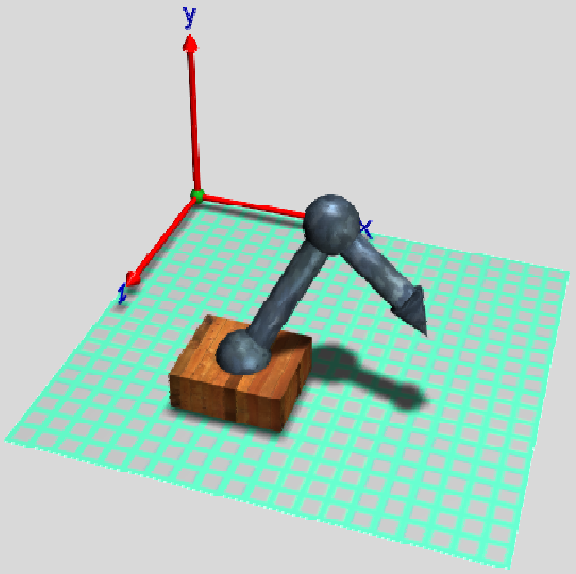
translate `base`



rotate `joint1`

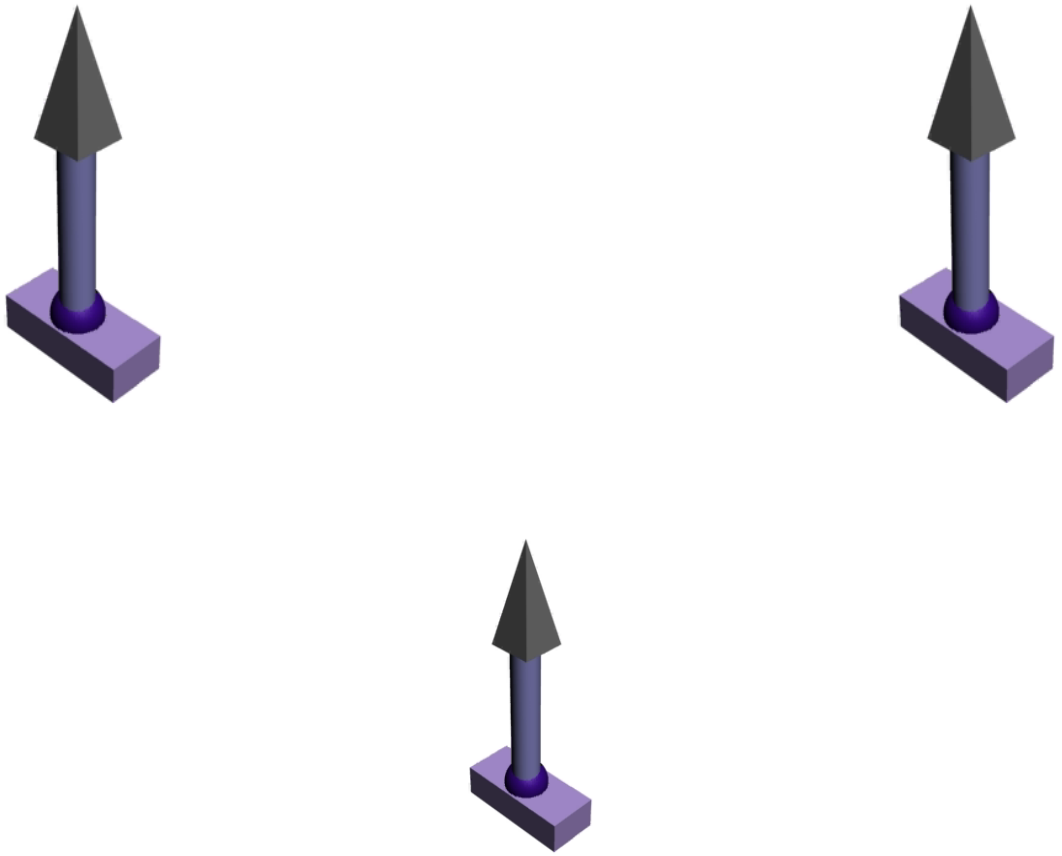


rotate `joint2`



complex hierarchical transformation

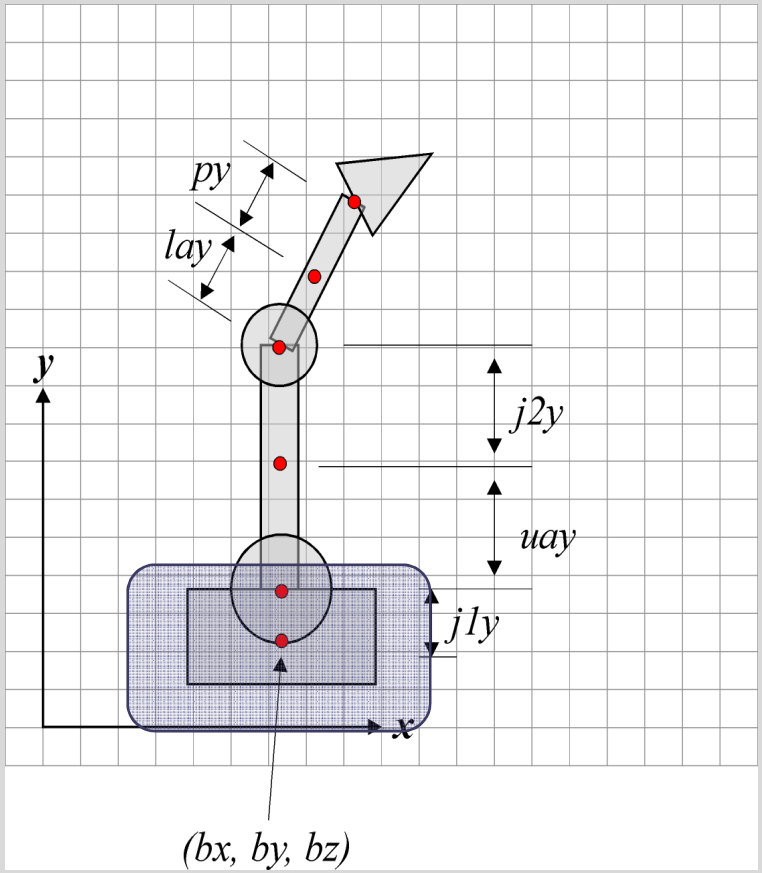
Hierarchical Transformations



OpenGL® Implementation

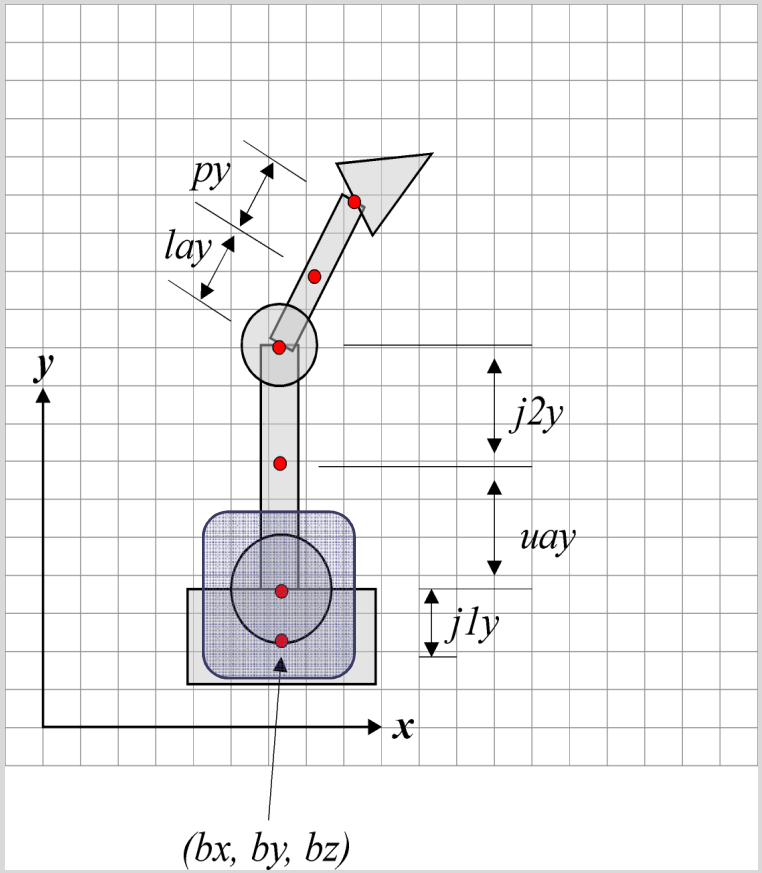
```
local1 = identity_mat4 ();
local1 = rotate(base_orientation) * local1;
local1 = translate(bx, by, bz) * local1;
global1 = local1;

updateUniformVariables(model matrix = global1);
drawBase();
```

```
local2 = identity_mat4 ();
local2 = rotate(joint1_orientation) * local2;
local2 = translate(0, j1y, 0) * local2;
global2 = local1*local2;

updateUniformVariables(model matrix = global2);
drawJoint1();

local3 = identity_mat4 ();
local3 = rotate(upperArm_orientation) * local3;
local3 = translate(0, uay, 0) * local3;
global3 = local1*local2*local3;

updateUniformVariables(model matrix = global3);
drawUpperArm();

etc.
```



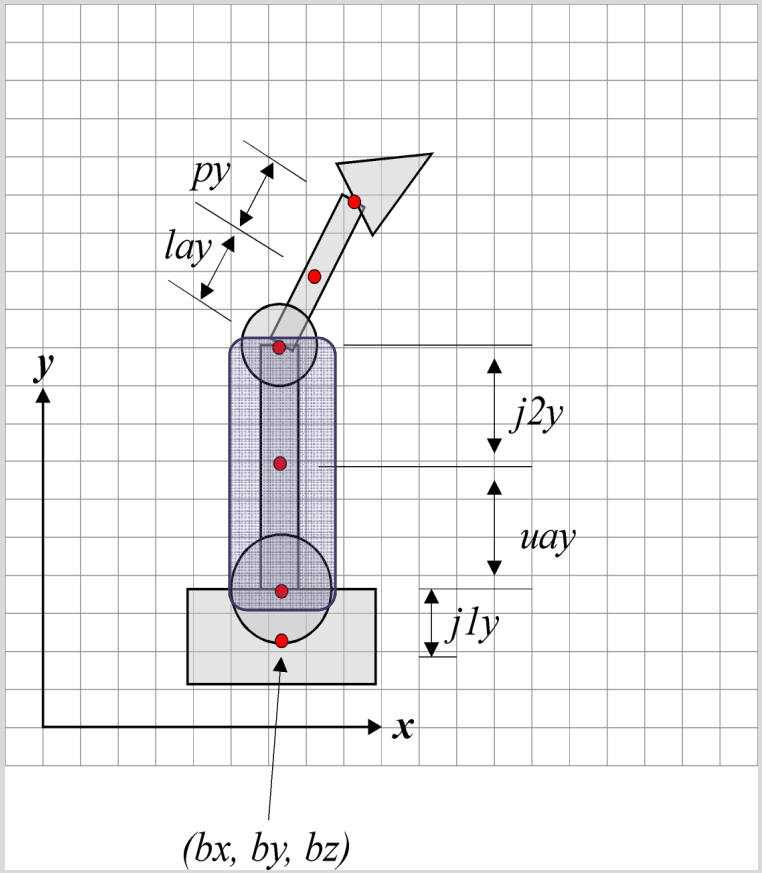
OpenGL® Implementation

```
local1 = identity_mat4 ();
local1 = rotate(base_orientation) * local1;
local1 = translate(bx, by, bz) * local1;
global1 = local1;

updateUniformVariables(model matrix = global1);
drawBase();

local2 = identity_mat4 ();
local2 = rotate(joint1_orientation) * local2;
local2 = translate(0, j1y, 0) * local2;
global2 = local1*local2;

updateUniformVariables(model matrix = global2);
drawJoint1();

local3 = identity_mat4 ();
local3 = rotate(upperArm_orientation) * local3;
local3 = translate(0, uay, 0) * local3;
global3 = local1*local2*local3;

updateUniformVariables(model matrix = global3);
drawUpperArm();

etc.
```



OpenGL® Implementation

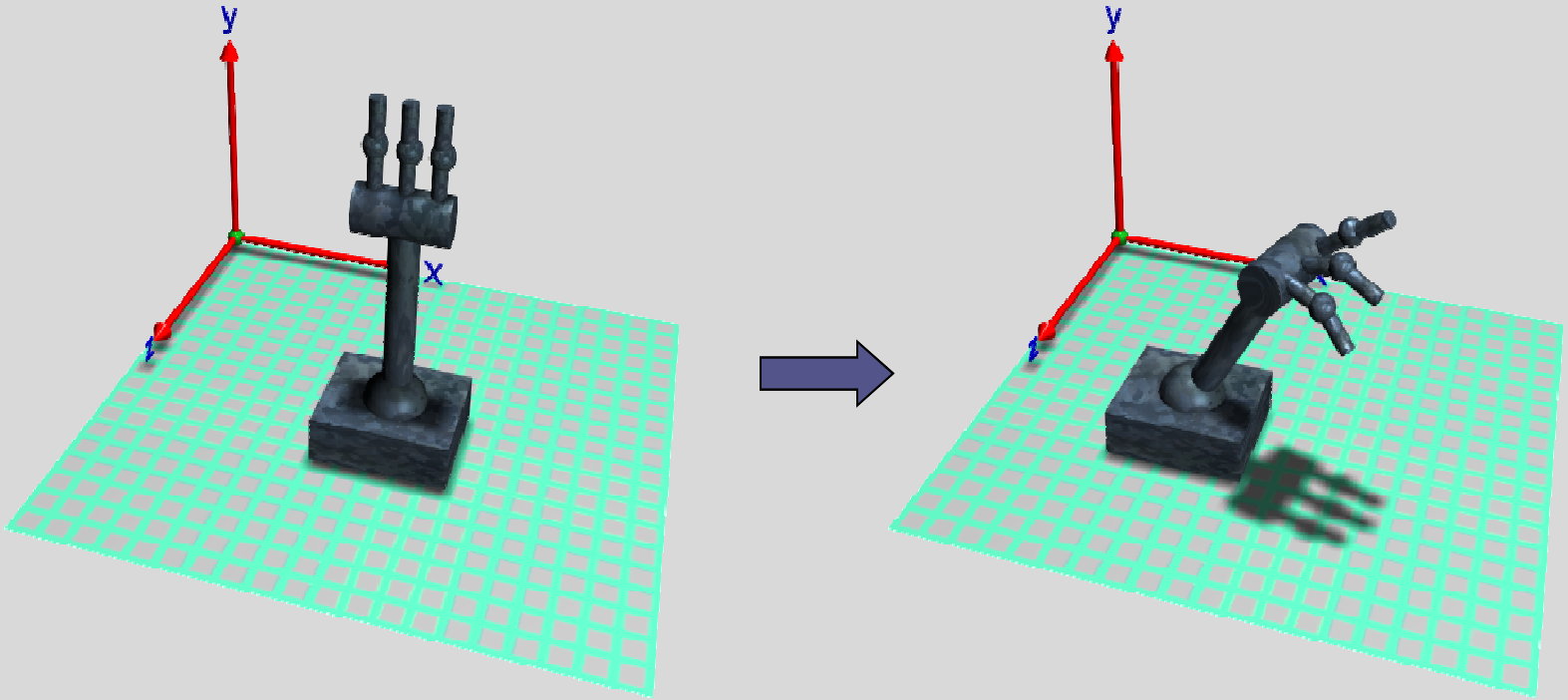
```
local1 = identity_mat4 ();
local1 = rotate(base_orientation) * local1;
local1 = translate(bx, by, bz) * local1;
global1 = local1;

updateUniformVariables(model matrix = global1);
drawBase();

local2 = identity_mat4 ();
local2 = rotate(joint1_orientation) * local2;
local2 = translate(0, j1y, 0) * local2;
global2 = local1*local2;

updateUniformVariables(model matrix = global2);
drawJoint1();

local3 = identity_mat4 ();
local3 = rotate(upperArm_orientation) * local3;
local3 = translate(0, uay, 0) * local3;
global3 = local1*local2*local3;

updateUniformVariables(model matrix = global3);
drawUpperArm();

etc.
```



Hierarchical Transformations

- The previous example had simple *one-to-one* parent-child linkages.
- In general there may be many *child frames* derived from a single parent frame.
- We need to remember the parent frame and return to it when creating new children.
  - Store global transformation as we go

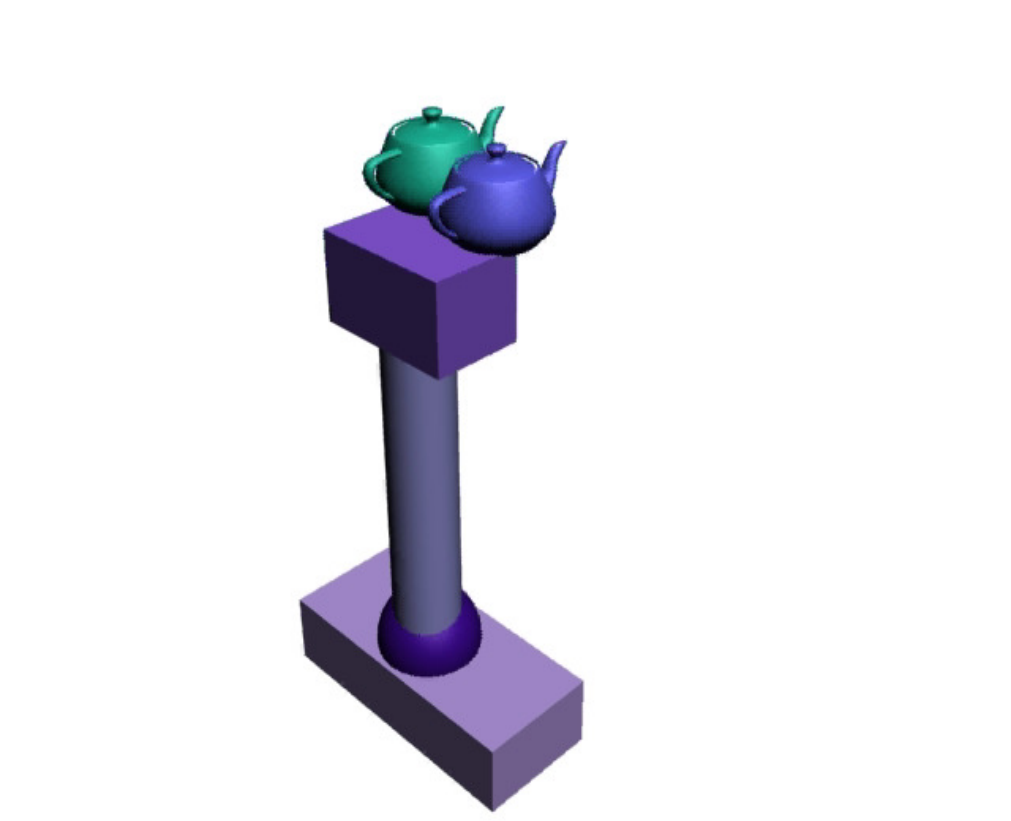
Hierarchical Transformations



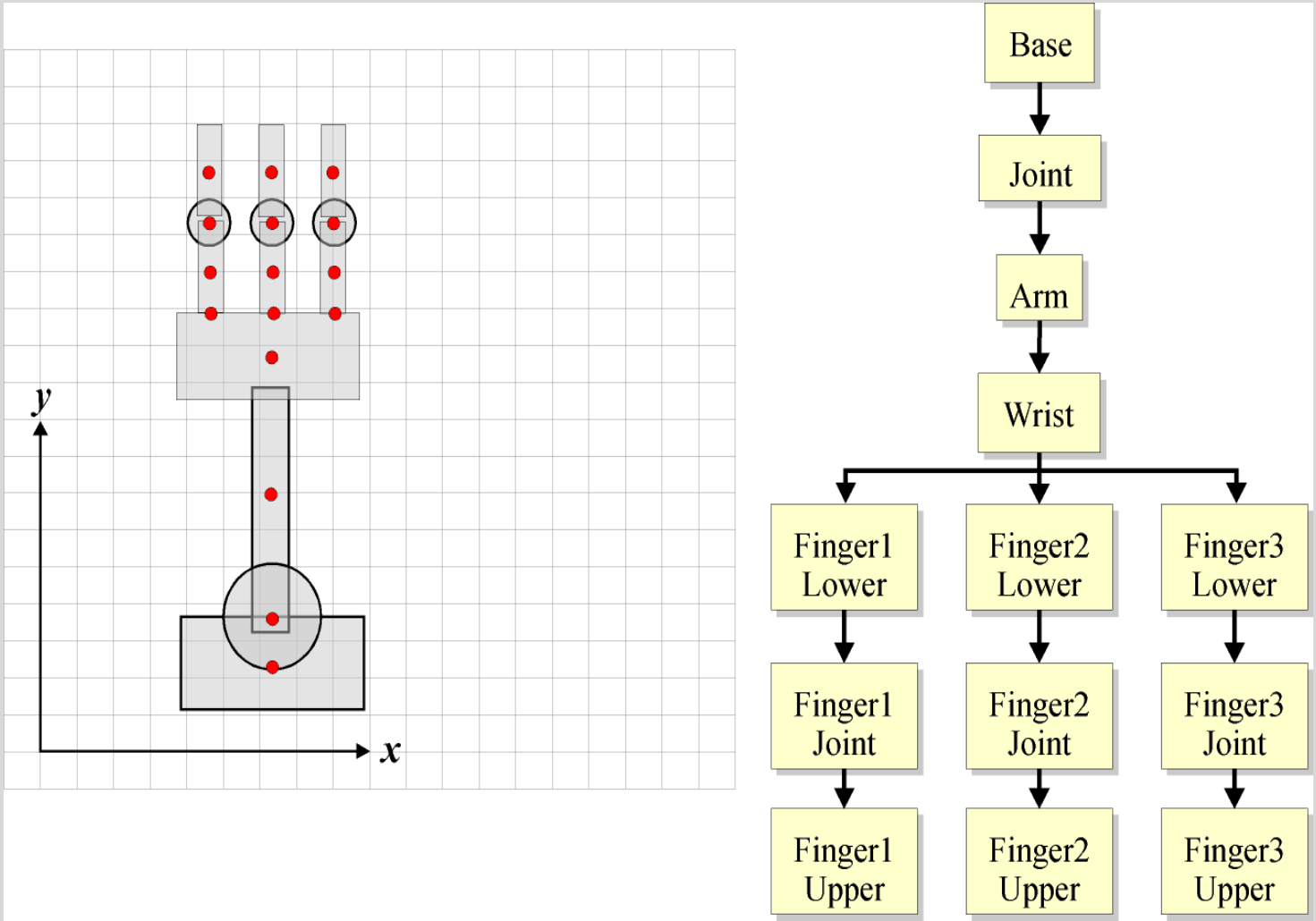
Each finger is a child of the parent (wrist)

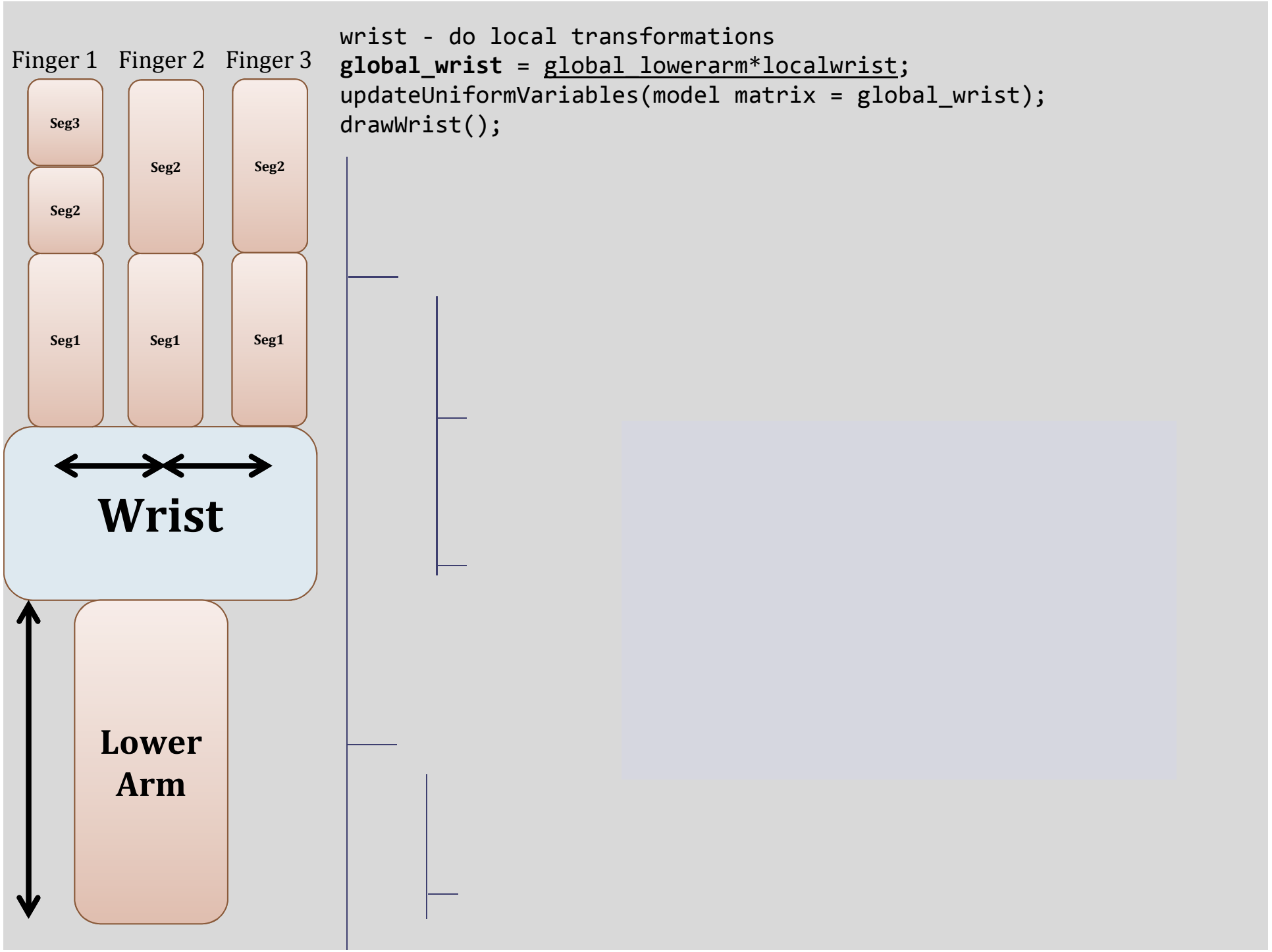
$\Rightarrow$ independent control over the orientation of the fingers relative to the wrist

Hierarchical Transformations

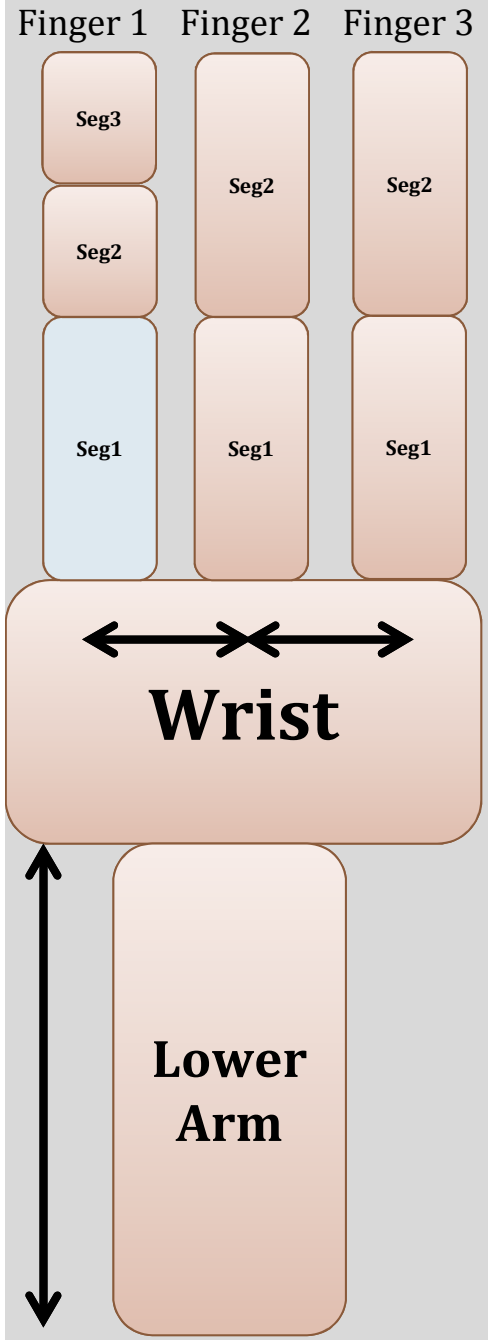


Hierarchical Transformations

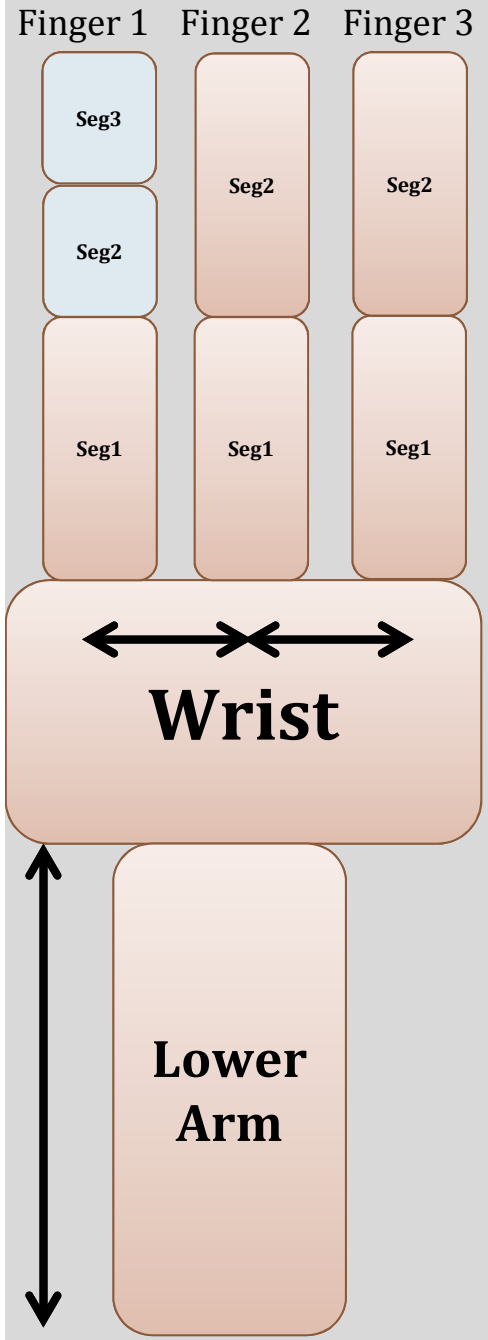




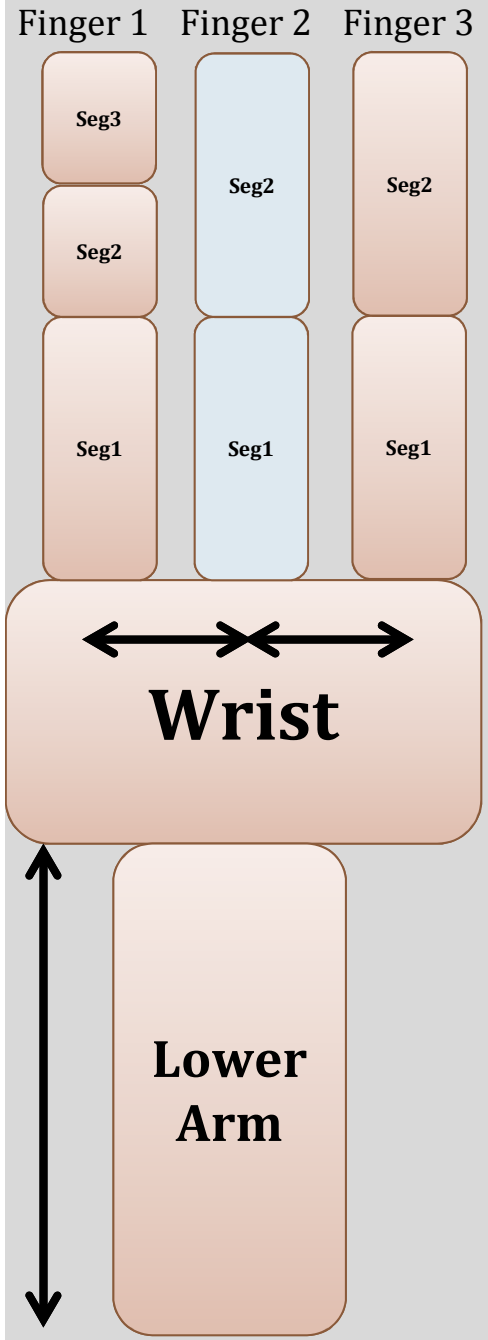

```
wrist - do local transformations
global_wrist = global_lowerarm*localwrist;
updateUniformVariables(model matrix = global_wrist);
drawWrist();
```




```
wrist - do local transformations
global_wrist = global_lowerarm*localwrist;
updateUniformVariables(model matrix = global_wrist);
drawWrist();

    finger1_segment1 – do local transformations
    global_finger1_1 = global_wrist*localfinger1_segment1;
    updateUniformVariables(model matrix = global_finger1_1);
    drawFinger1Segment1();
```




```
wrist - do local transformations
global_wrist = global_lowerarm*localwrist;
updateUniformVariables(model matrix = global_wrist);
drawWrist();

    finger1_segment1 – do local transformations
    global_finger1_1 = global_wrist*localfinger1_segment1;
    updateUniformVariables(model matrix = global_finger1_1);
    drawFinger1Segment1();

        finger1_segment2 – do local transformations
        global_finger1_2 = global_finger1_1 *localfinger1_segment2;
        updateUniformVariables(model matrix = global_finger1_2);
        drawFinger1Segment2();

        finger1_segment3 – do local transformations
        global_finger1_4 = global_finger1_2 *localfinger1_segment3;
        updateUniformVariables(model matrix = global_finger1_3);
        drawFinger1Segment3();
        etc.
```




```
wrist - do local transformations
global_wrist = global_lowerarm*localwrist;
updateUniformVariables(model matrix = global_wrist);
drawWrist();

    finger1_segment1 – do local transformations
    global_finger1_1 = global_wrist*localfinger1_segment1;
    updateUniformVariables(model matrix = global_finger1_1);
    drawFinger1Segment1();

        finger1_segment2 – do local transformations
        global_finger1_2 = global_finger1_1 *localfinger1_segment2;
        updateUniformVariables(model matrix = global_finger1_2);
        drawFinger1Segment2();

        finger1_segment3 – do local transformations
        global_finger1_4 = global_finger1_2 *localfinger1_segment3;
        updateUniformVariables(model matrix = global_finger1_3);
        drawFinger1Segment3();
        etc.

    finger2_segment1 – do local transformations
    global_finger2_1 = global_wrist*localfinger2_segment1;
    updateUniformVariables(model matrix = global_finger2_1);
    drawFinger2Segment1();

        finger2_segment2 – do local transformations
        global_finger2_2 = global_finger2_1 *localfinger2_segment2;
        updateUniformVariables(model matrix = global_finger2_2);
        drawFinger2Segment2();
```

Summary

- Viewing
- Transformations
- Transformations in OpenGL
- Hierarchies
- Next - Lighting!

Lab

- Online today
- Basic Hierarchical animation
- Marked October 27th
- 4 Demonstrators
  - Use upstairs
  - Attend labs!!