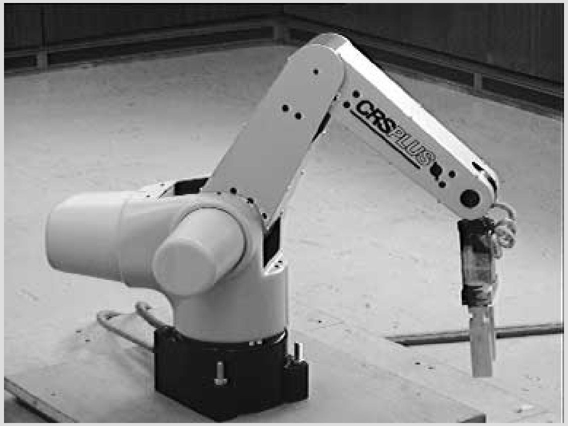
Kinematics

- The branch of mechanics concerned with the motions of objects without regard to the forces that cause the motion



Kinematics vs. Dynamics

Kinematics

Describes the positions of the body parts as a function of the joint angles.

Dynamics

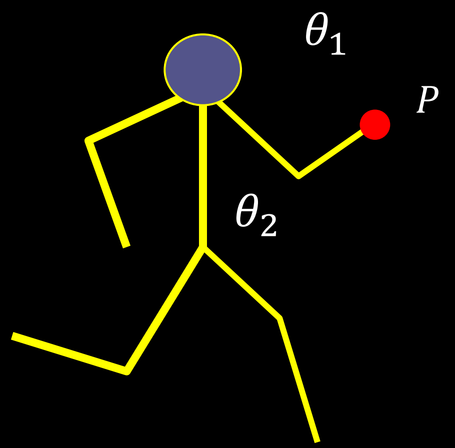
Describes the positions of the body parts as a function of the applied forces.

Kinematics

- Considers only motion
- Determined by positions, velocities, accelerations
- Forward kinematics
  - Low level approach where animator has to explicitly specify all motions of every part of the animated structure
  - Each node in hierarchy inherits movement of all nodes above it
- Inverse kinematics
  - Requires only the position of the ends of the structure
  - Functions as black box - controls detailed movement of entire structure

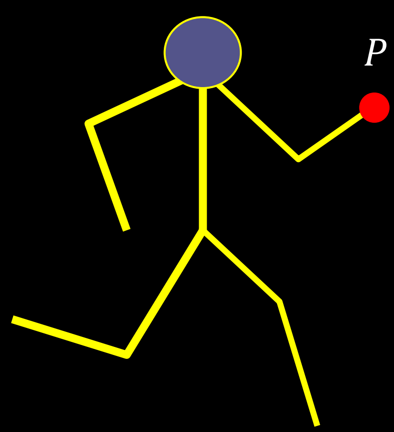
Forward and Inverse Kinematics

- Forward kinematics
  - mapping from joint space to cartesian space
- Inverse kinematics
  - mapping from cartesian space to joint space



Forward Kinematics

$P = f(\theta_1, \theta_2)$

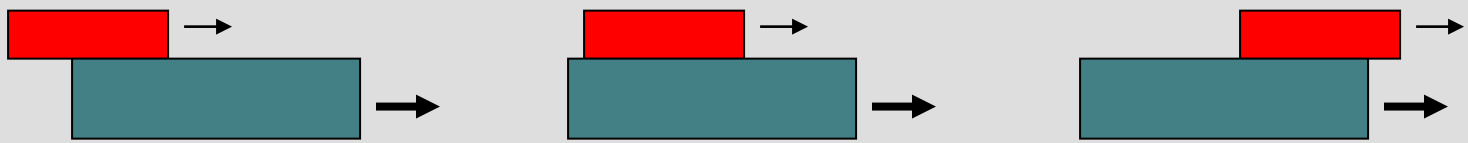


Inverse Kinematics

$\theta_1, \theta_2 = f^{-1}(P)$

Relative Motion

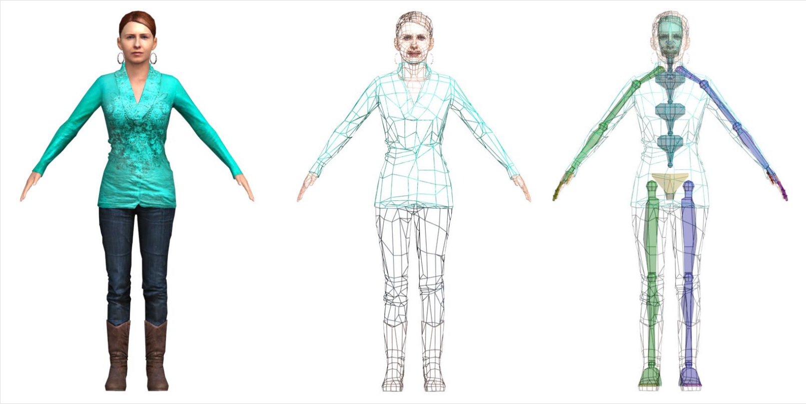
- All motion takes place relative to a local origin.



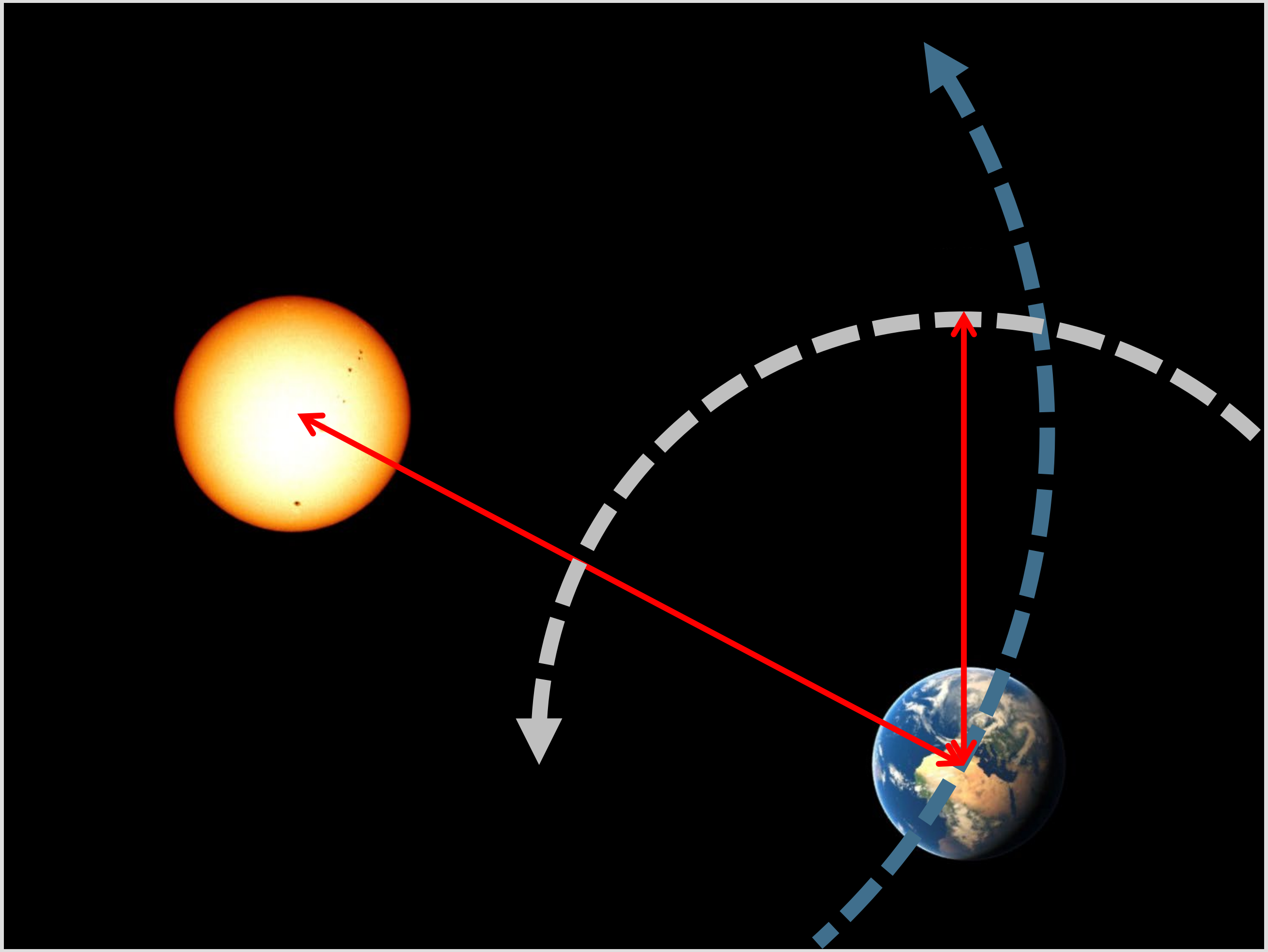
  - Ex: throwing a ball to a friend as you both ride in a train.
- The term *local origin* refers to the (0,0,0) that you've chosen to measure motion from.
- The local origin may be moving relative to some greater frame of reference.

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

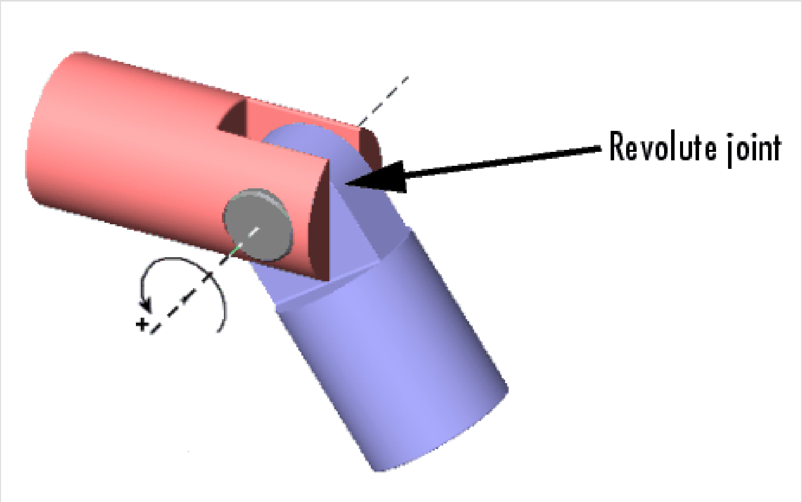


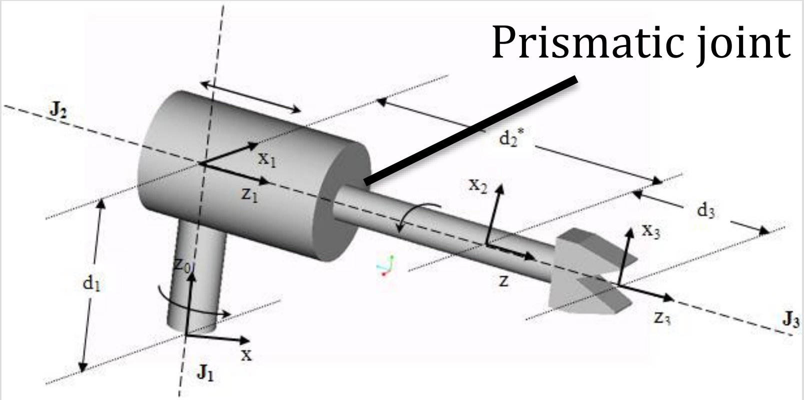
Relative Motion



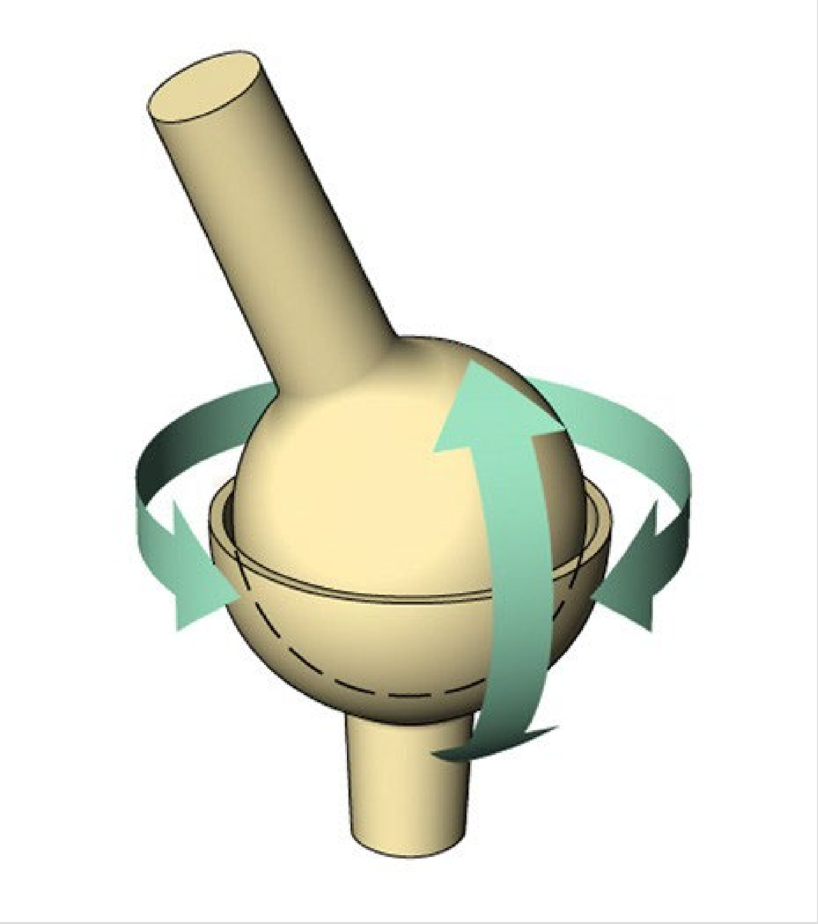
Hierarchies

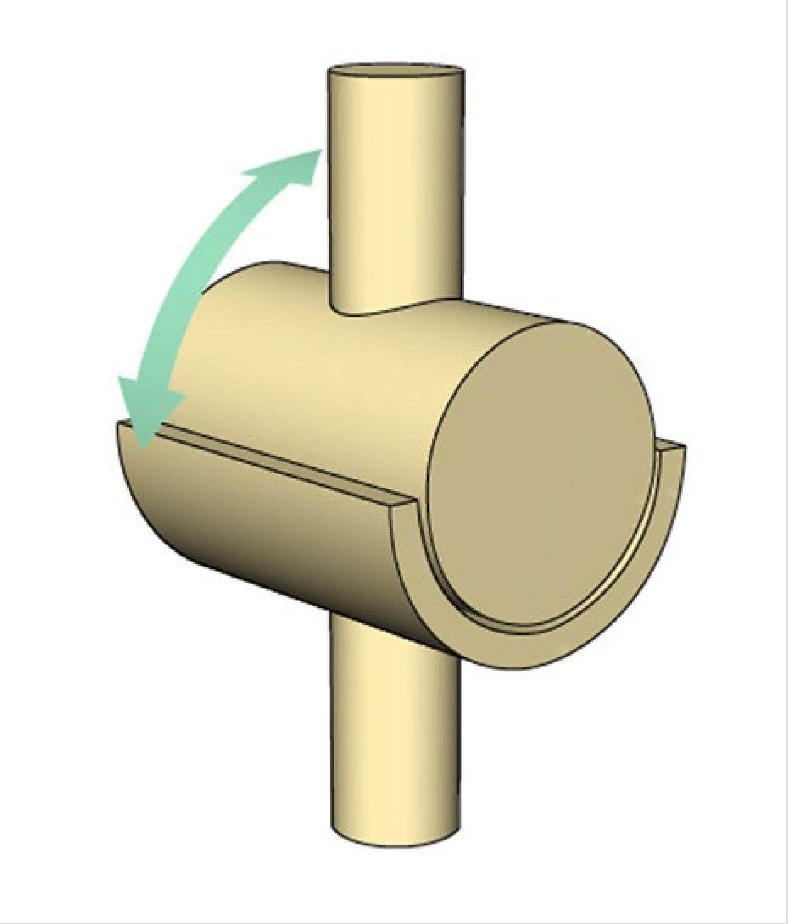
- Useful for modelling
  - Animals
  - Humans
- Field of Robotics
  - Manipulators
  - Joints
  - Links
  - End effectors
  - Frame
- Animation
  - mostly interested in rotational joints





Degrees of Freedom





Degrees of Freedom

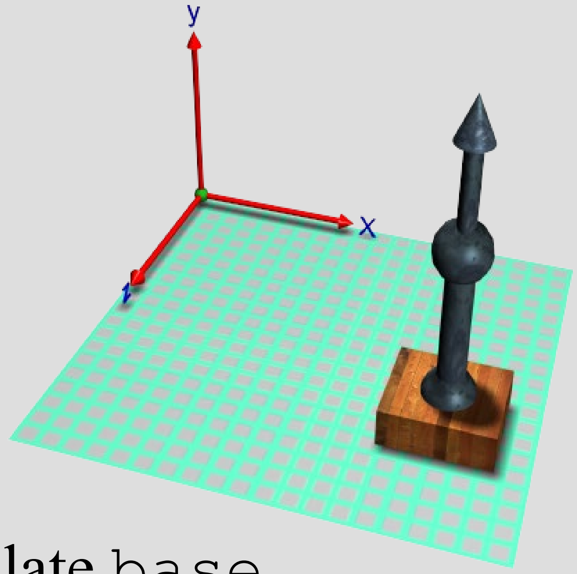
- Root: 3 translational DOF + 3 rotational DOF
- Rotational joins are commonly used
- Each joint can have up to 3 DOF
  - Shoulder: 3 DOF
  - Wrist: 2 DOF
  - Knee: 1 DOF

Data structure

- Hierarchical linkages to create model
  - Tree structure of **nodes** connected by **arcs**
  - The highest node of the tree is the root node
  - Position and orientation of root is known in **global** coordinate system
  - All other node positions will be located **relative** to the root node

Root Node

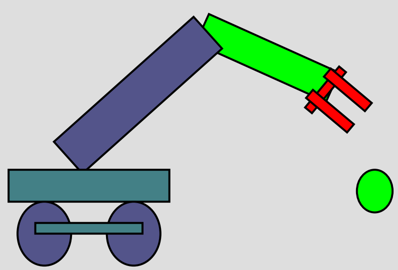
- Represents a global transformation
- Indirectly all other nodes get this transform
- Changing the global transformation of the root will reposition the entire structure in the global coordinate system

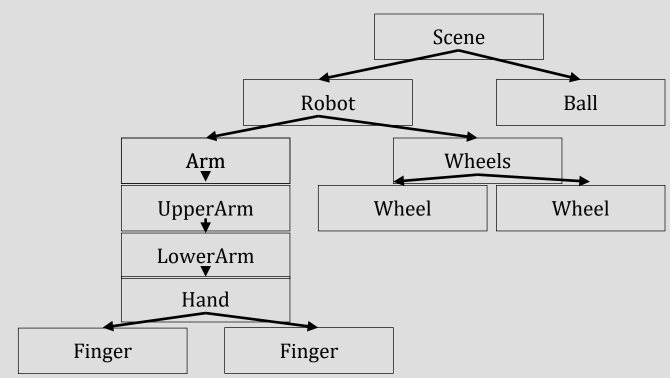


translate base

Data structure

- A ***node*** is any element in the graph
  - A ***child node*** is any node which is an immediate descendent of the node being discussed
  - The ***parent node*** is the node from which the node being discussed descends
  - The ***root node*** is the ancestor of all other nodes in the scene, and has no parent.
  - A node with no child is a ***leaf node***





Child Nodes

- Initial position transformation
  - Rotate and translate the object into its position of attachment ***relative*** to its parent
  - -> neutral position wrt parent
- Other transformation
  - Variable information responsible for the actual joint articulation

Child Nodes

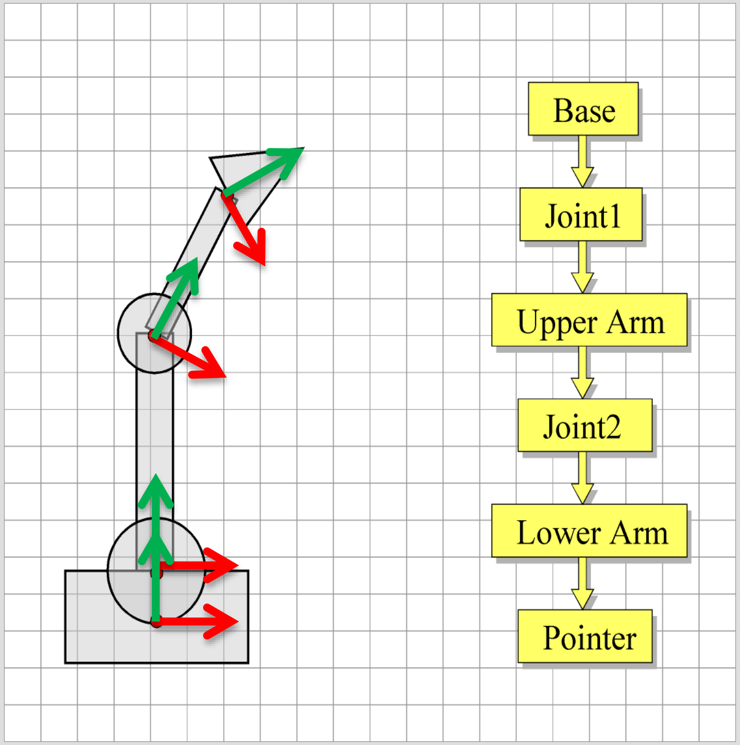
- The vertices of a particular object can be transformed to their final positions by *concatenating* the transformations higher up the tree and applying the composite transformation matrix to the vertices

Hierarchical Modeling *meets* Transforms

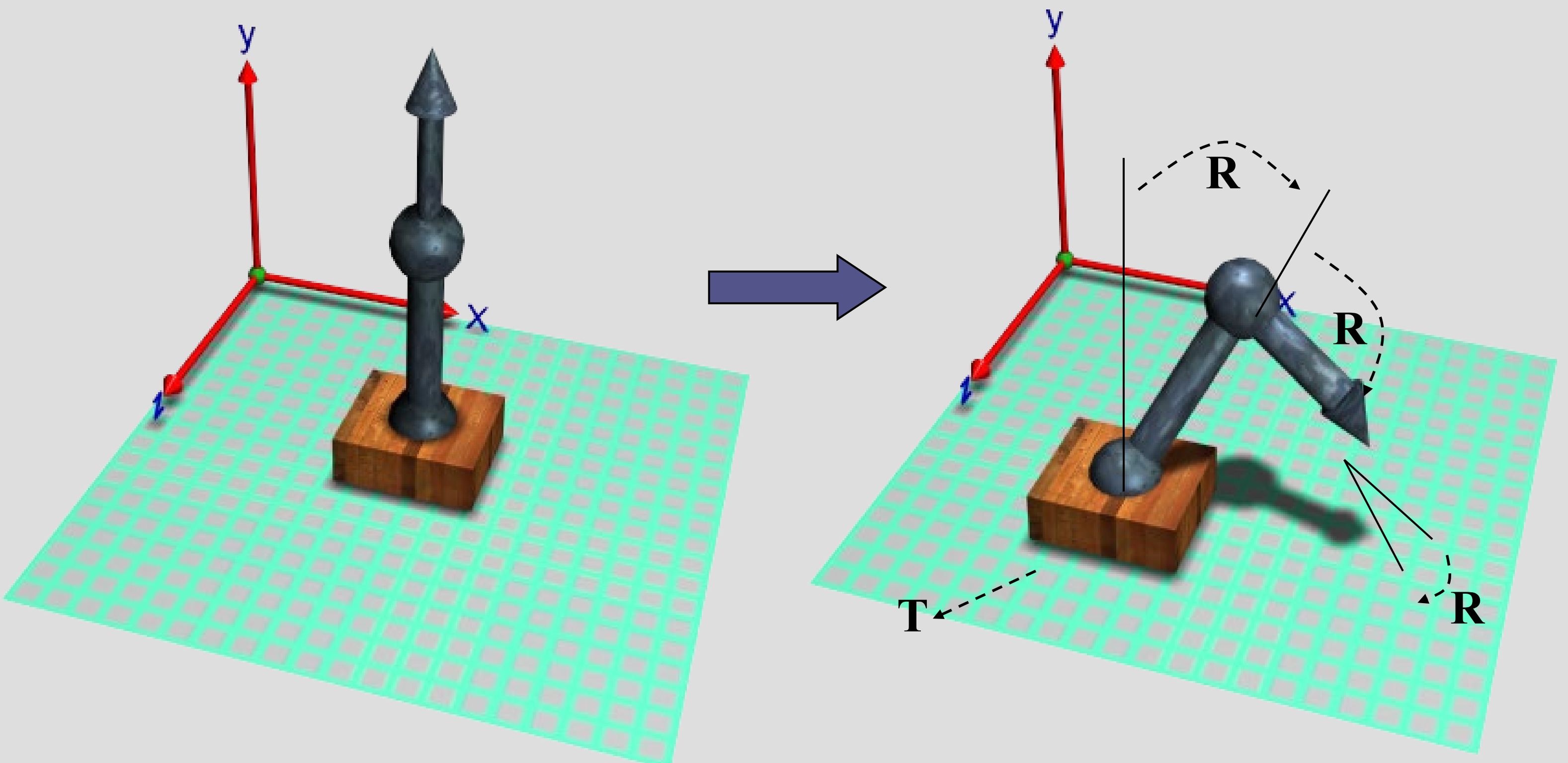
- Each object in your scene knows where it is
  - But you don't have to store its location and orientation relative to the center of the world
  - You can store its location and orientation ***relative to its parent***
  - The *other* great strength of hierarchical modeling is that moving the parent repositions all children without effort

Hierarchical Transformations

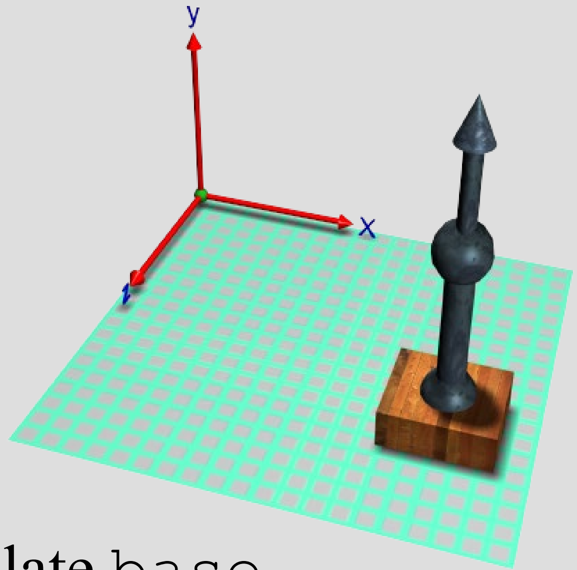
- For geometries with an implicit *hierarchy* we wish to associate local frames with sub-objects in the assembly
- *Parent-child frames* are related via a transformation
- Transformation linkage is described by a *tree*
- Each node has its own *local co-ordinate system*.



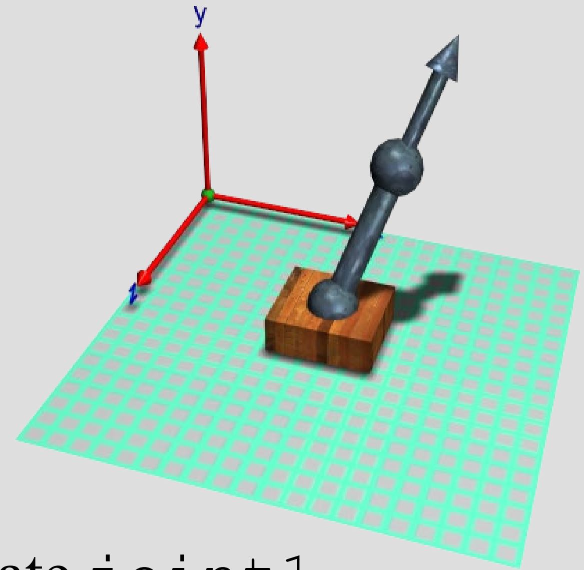
Hierarchical Transformations



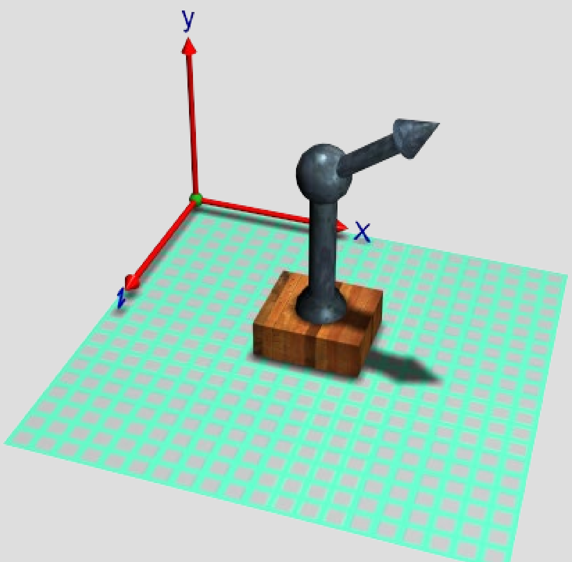
Hierarchical transformation allow independent control over sub-parts of an assembly



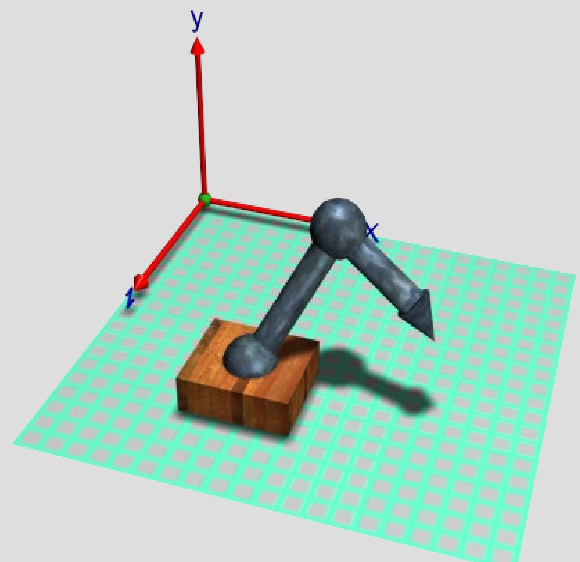
translate `base`



rotate `joint1`

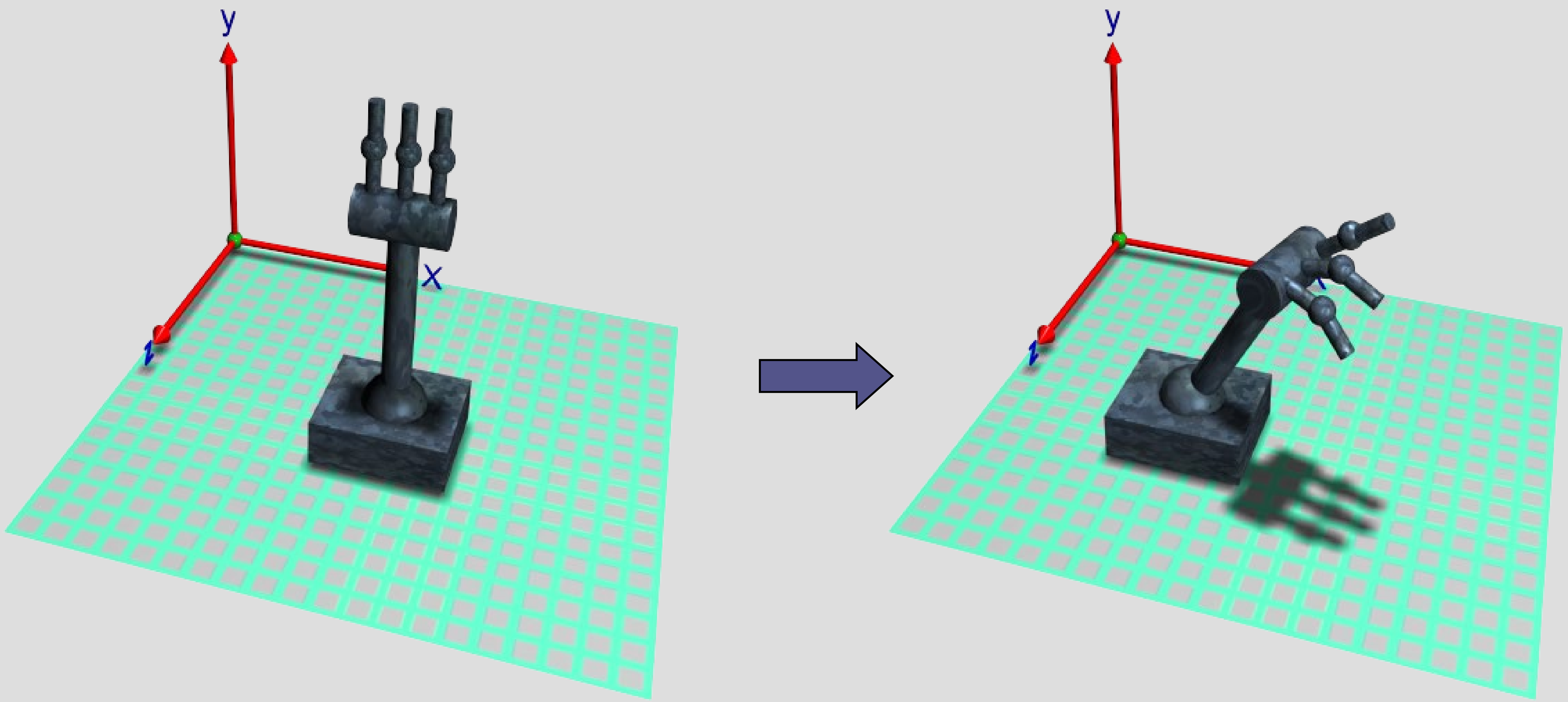


rotate `joint2`



complex hierarchical transformation

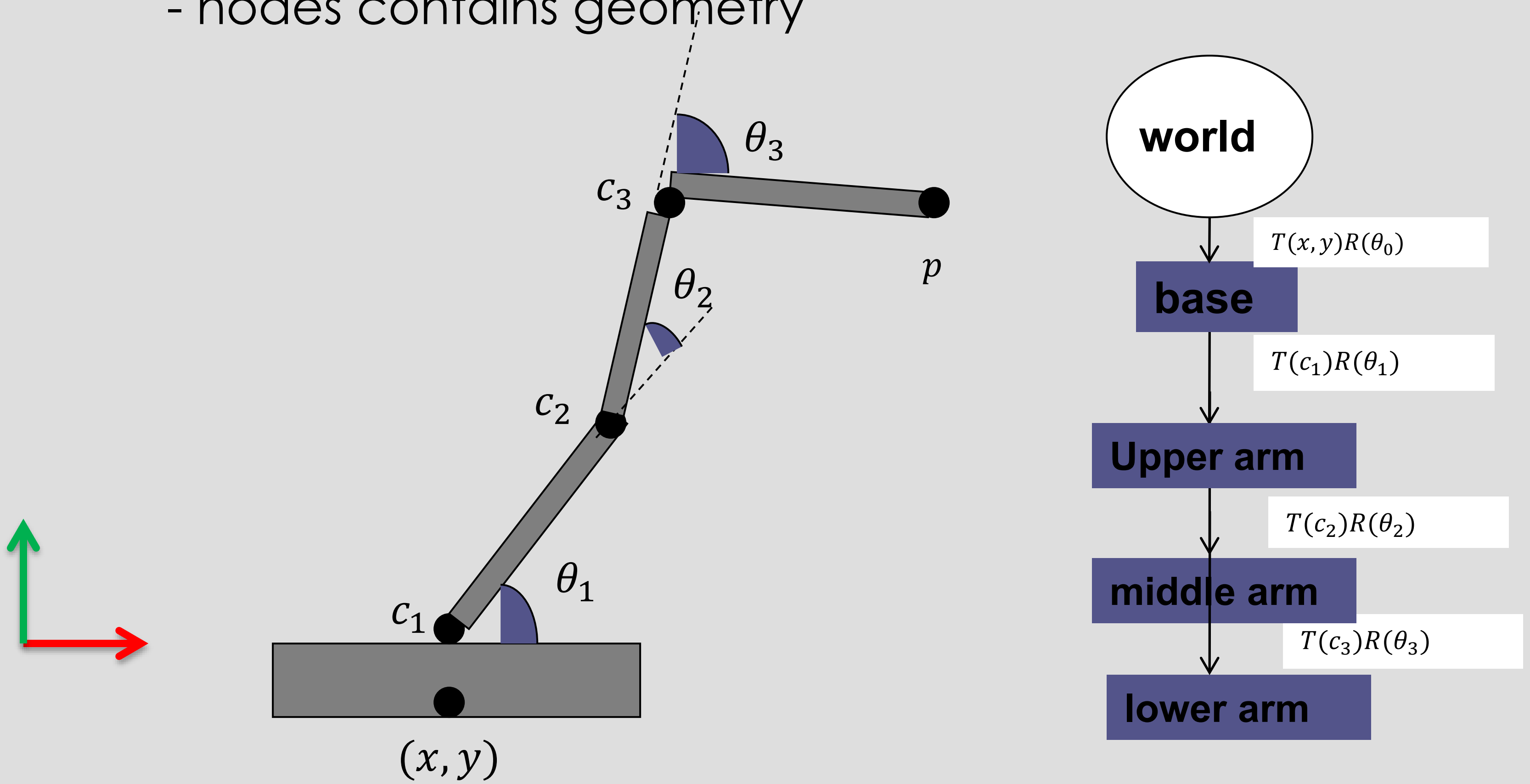
Hierarchical Transformations



Each finger is a child of the parent (wrist)
$\Rightarrow$ independent control over the orientation
of the fingers relative to the wrist

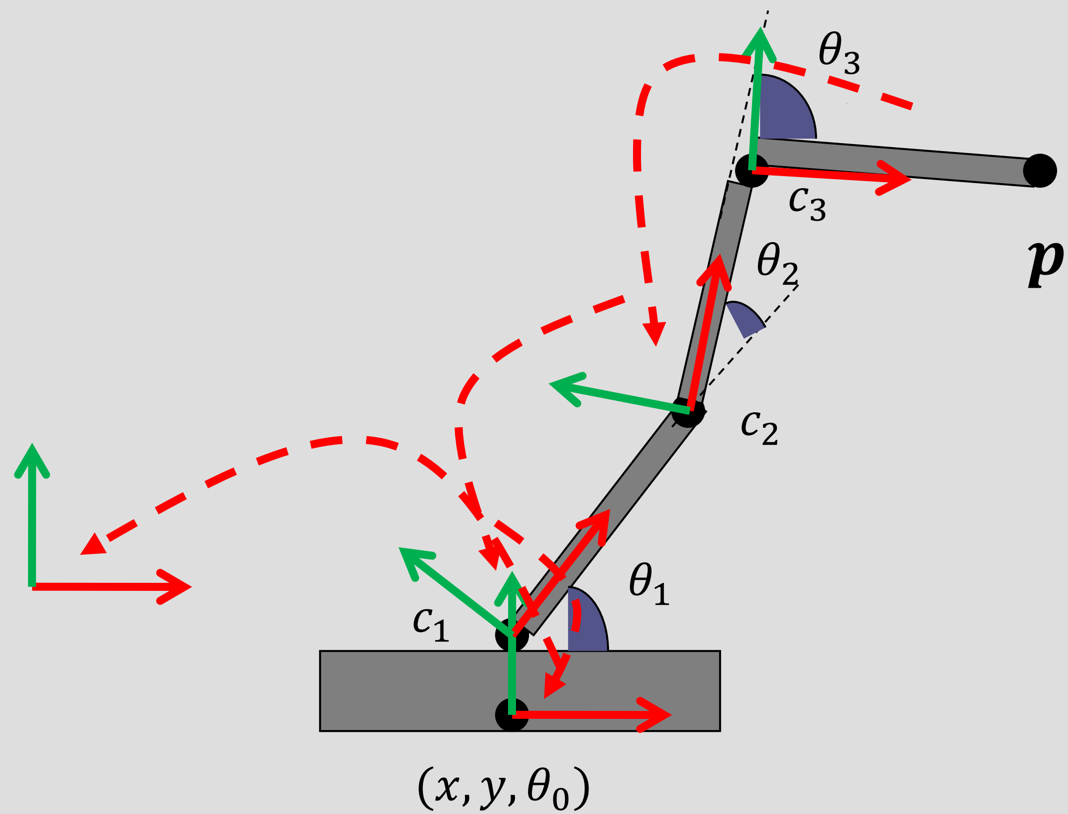
Hierarchical Modeling

- Hierarchical model can be composed of instances using trees or directed acyclic graphs (DAGs)
  - edges contains geometric transformations
  - nodes contains geometry



Lamp

- What's the current coordinate $\boldsymbol{p}$?



$$\boldsymbol{p} = \boxed{T(x,y)R(\theta_0)T(c_1)R(\theta_1)T(c_2)R(\theta_2)T(c_3)R(\theta_3)p_0}$$

Lamp Implementation

- The lamp can be displayed by computing a global matrix and computing it at each step

```
Matrix M_model;
Main()
{ ...
 M_model=Identity()
 lamp();
}
```

```
lamp()
{
  M_model = T(x,y)R(θ0)
  base();
  M_model = T(x,y)R(θ0)T(c1)R(θ1)
  upper_arm();
  M_model = T(x,y)R(θ0)T(c1)R(θ1)T(c2)R(θ2)
  middle_arm();
  M_model = T(x,y)R(θ0)T(c1)R(θ1)T(c2)R(θ2)T(c3)R(θ3)
   lower_arm();
}
```

$T(x,y)R(\theta_0)$

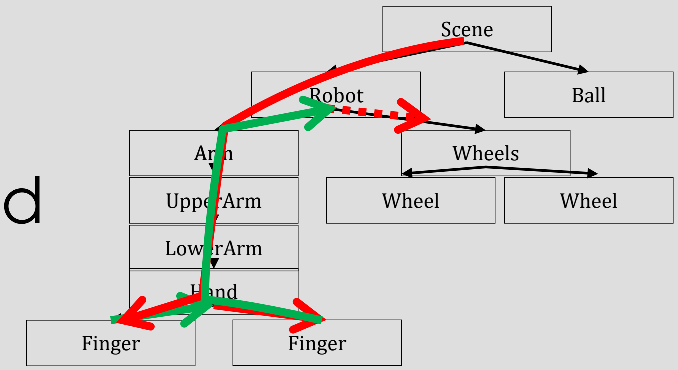
$T(x,y)R(\theta_0)T(c_1)R(\theta_1)$

$T(x,y)R(\theta_0)T(c_1)R(\theta_1)T(c_2)R(\theta_2)$

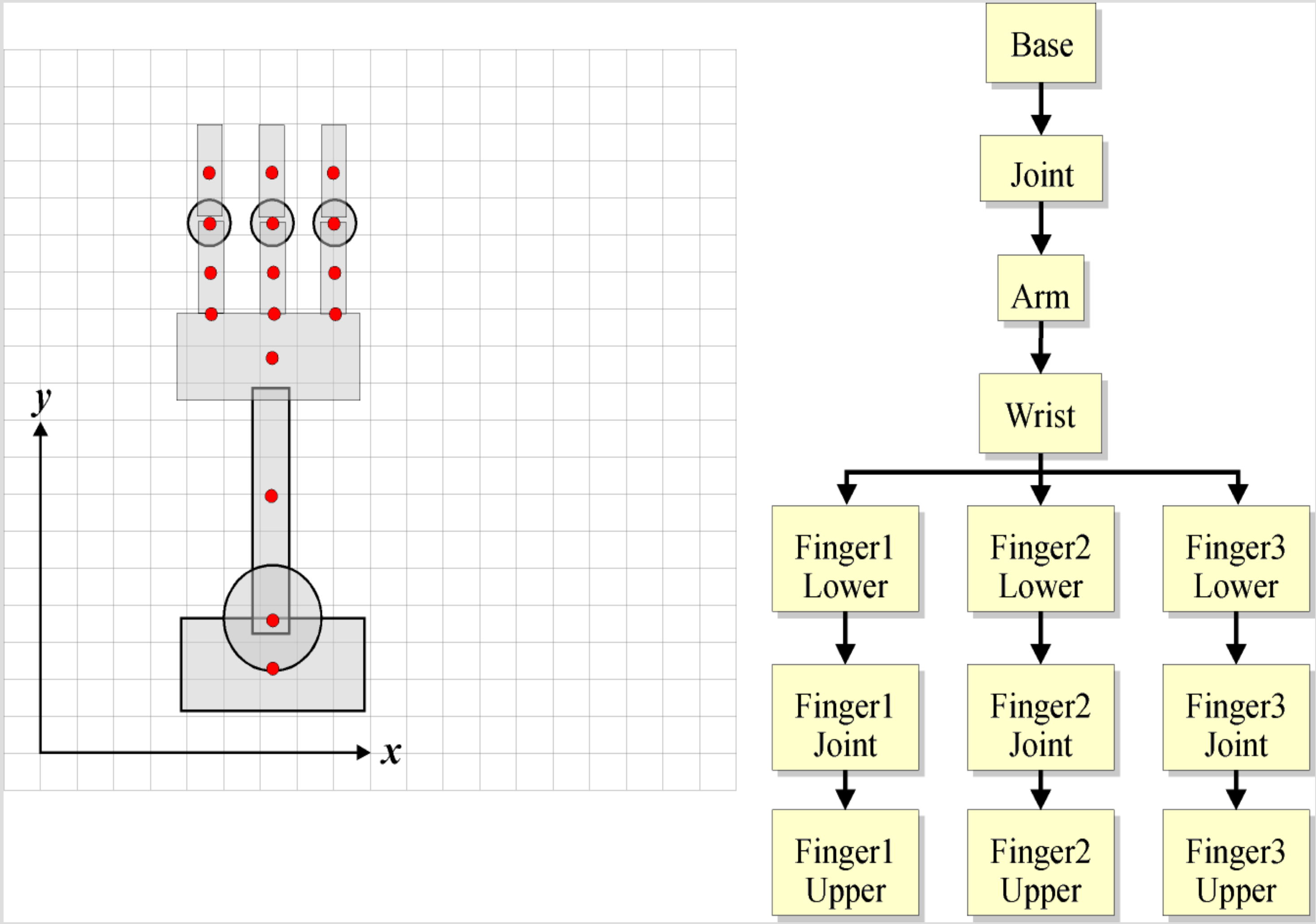
$T(x,y)R(\theta_0)T(c_1)R(\theta_1)T(c_2)R(\theta_2)T(c_3)R(\theta_3)$

Tree Traversal

- Successively applying matrices farther up the hierarchy can transform a point from any position in the tree into world coordinates
  - Depth first pattern from root to leaf node
  - Traversal then backtracks up the tree until an unexplored downward arc is encountered
  - Downward arc is then traversed followed by backtracking
  - Continues until all nodes and arcs have been visited
  - Transformations are concatenated



Scene Graphs



Hierarchical Modeling in C++ and OpenGL

- Minimum contents of a node class
  - A name or ID
  - A pointer to the node's parent in the scene graph
  - A list or array of the node's children
  - The node's position and rotation

```
class Node
{
  string        m_nodeName;
  Node *        m_pParent;
  list<Node*>   m_lChildren;

  Vec           m_rotationAxis;
  float         m_rotationAngle;
  Vec           m_translation;
};
```

Hierarchical Modeling in C++ and OpenGL

- A better node
  - Store the object's transformation in a 4x4 matrix (Instead of storing the node's position and rotation as two separate pieces of data).

```
class Node
{
  string        m_nodeName;
  Node          *m_pParent;
  list<Node*>   m_lChildren;
  Matrix4x4     m_transform;
};
```

Hierarchical Modeling in C++ and OpenGL

- A Tree/ Skeleton Class
- Minimum contents of a Skeleton class
  - Root bone
  - A list or array of all nodes
  - Ability to traverse the structure

Hierarchical Modeling in C++ and OpenGL

- Rendering your node structure
  - The scene graph model is based on the concept of *recursion*.
  - Your display routine will render
    - the current scene graph node
    - then call *itself* (the same routine) to render each of the children of the current node

Hierarchical Modeling in C++ and OpenGL

- The pseudocode of a renderer

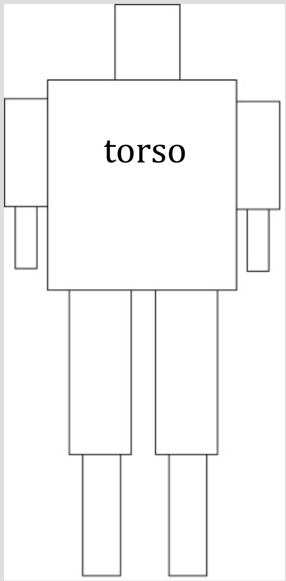
```
void RenderObject(Node *pNode)
{
  nodeGlobal = pNode->getGlobalTransform();
  Send uniform matrix nodeGlobal to shader
  pNode->render();
  for each child of pNode, do
     RenderObject(child);
}

void displayFunction(void)
{
   RenderObject(pSceneRootNode);
}
```

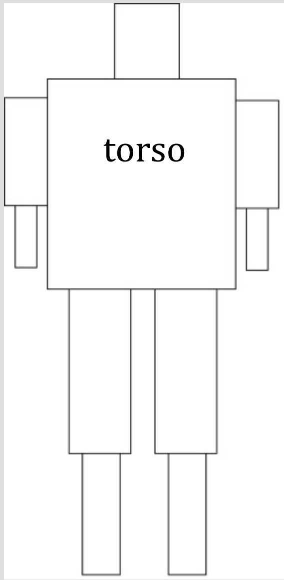
Hierarchical Modeling in C++ and OpenGL

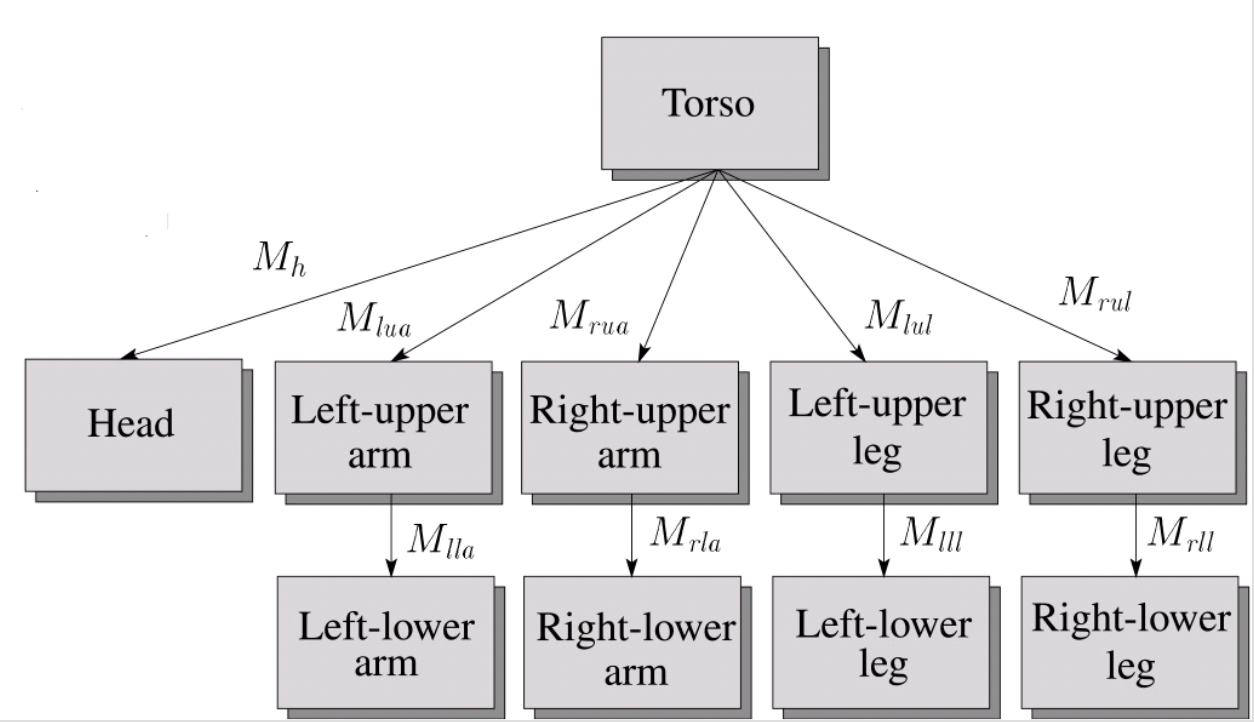
To use hierarchical modeling effectively, you need to create a family of C++ classes to store the objects in your scene graph.

A More Complex Example: Human Figure



A More Complex Example: Human Figure





Animation

- Define key positions interactively
- Specify numeric values and then interpolate
- Forward kinematics
- Can be tedious for the user
- Trial and error process
  - Inverse kinematics