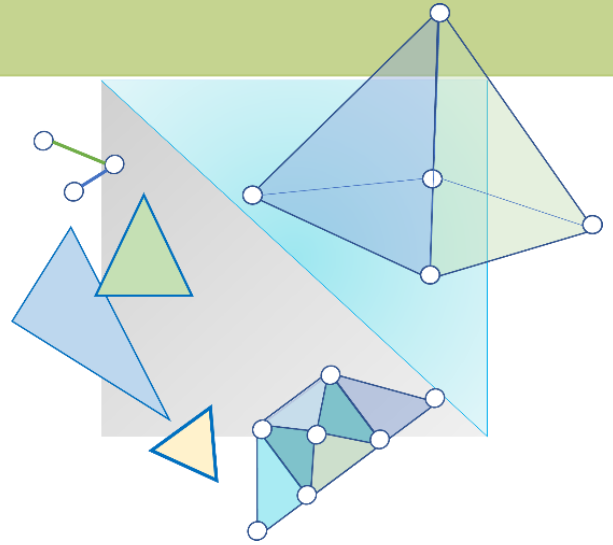
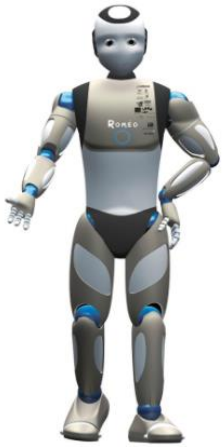
CITS3003 Graphics & Animation

Lecture 20:
Hierarchical Modelling



Objectives

- Symbols and Instances
- Introduce hierarchical models
  - Tree and DAG models
  - Tree Traversal



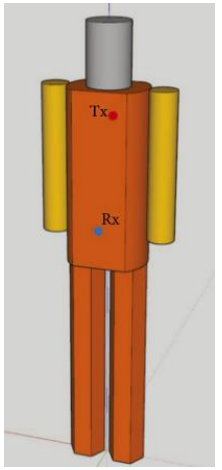
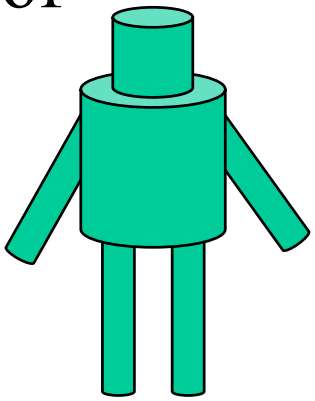


Image source: link

Complex objects – How to represent them?

- Most Graphic API's only support a limited types of primitives:
  - Triangles
  - Cubes
  - Spheres

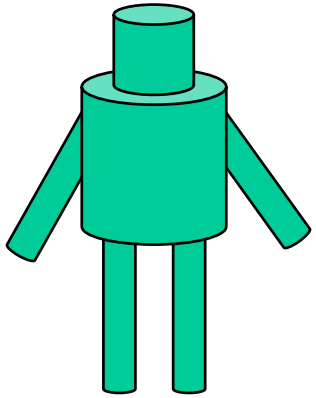


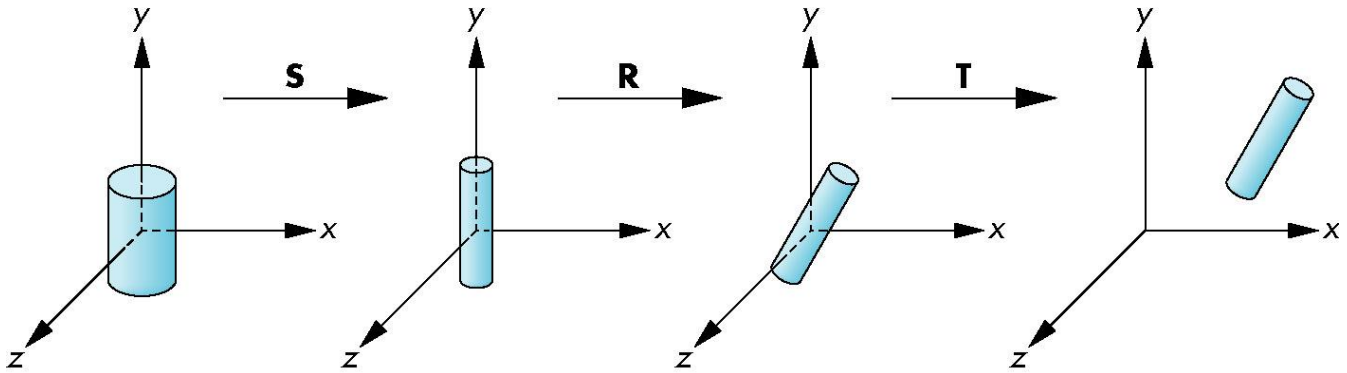
- A complex object can often be decomposed into simpler primitive parts. We will refer to these primitive parts as ***symbols***.

- To render a complex object, we can transform these parts separately to fit the object.

Complex objects – How to represent them?

- We must define an ***instance transformation*** for each part, e.g., a cylinder of radius=1 and height=1 is transformed to give a rod, which models an arm of a robot.

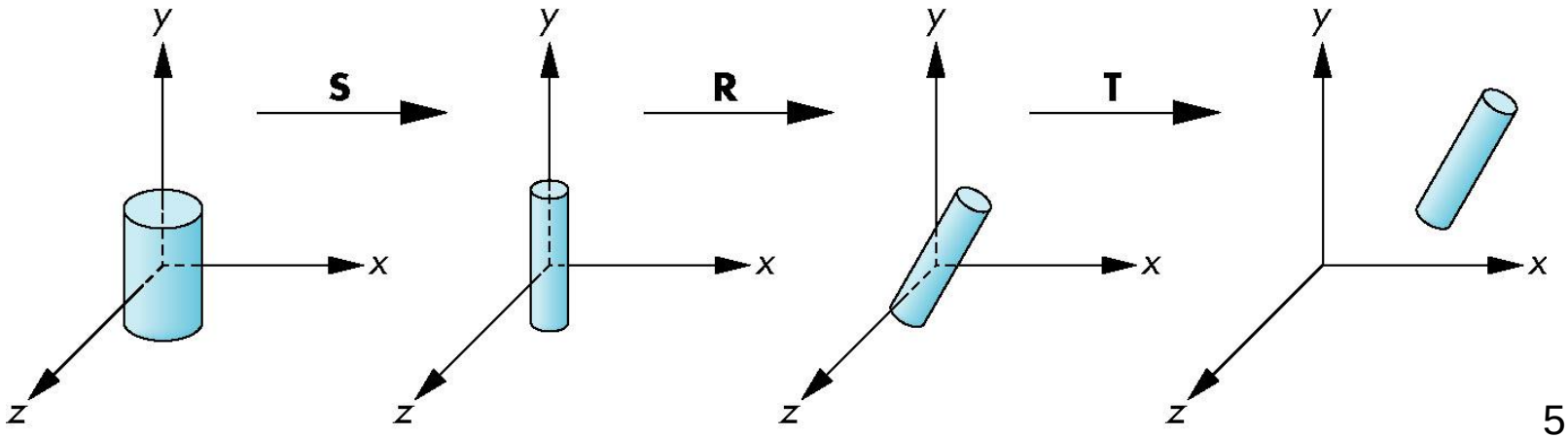




Complex objects – How to represent them?

- The instance below can be represented by one transformation of the form $M=TRS$.

```
mat4 instance;
mat4 model_view;
instance = Translate(dx, dy, dz)*RotateZ(rz)*RotateY(ry)*RotateX(rx)*Scale(sx, sy, sz);
model_view = model_view*instance;
cylinder(); /* or some other symbol */
```

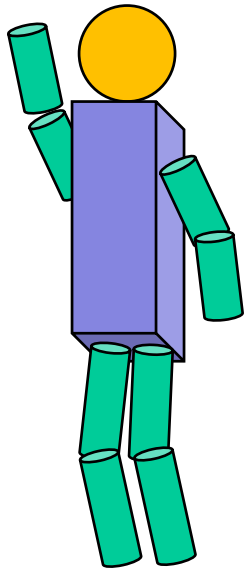


Symbol-Instance Table

- Non-hierarchical approach to modelling a complex object:
  - Collection of symbols and their instances
- For the example before, we would probably have **6** different instances of the cylinder to represent the head, torso, left/right arms, left/right legs of the robot.
- We can store each symbol instance into a symbol-instance table for the complex object by assigning a number to each symbol and storing the parameters for the instance transformation.

2nd and 3rd instances of symbol 1

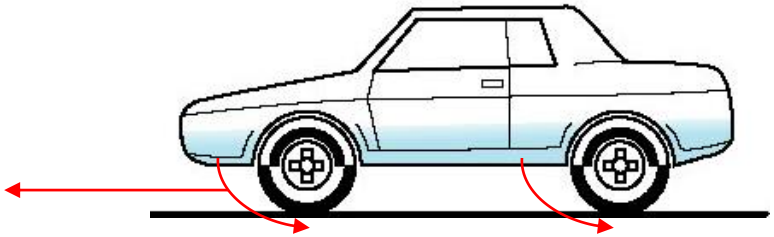
Symbol	Scale	Rotate	Translate
1	s_x, s_y, s_z	$\theta_x, \theta_y, \theta_z$	d_x, d_y, d_z
2			
3			
1			
1			
.			
.			



Relationship of Parts in a Car Model

- The problem with a **symbol-instance table** is it does not show relationships between different parts of the complex model

- Consider the modelling of a car:
  - Chassis + 4 identical wheels
  - Two symbols

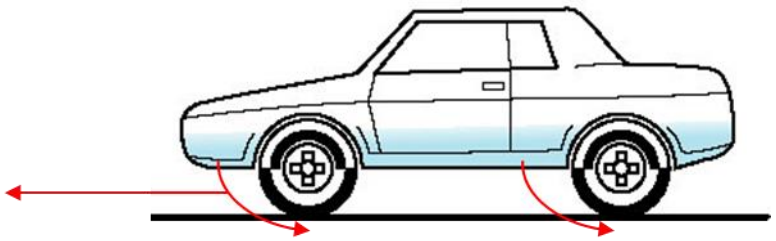


- Rate of forward motion determined by rotational speed of wheels

Relationship of Parts in a Car Model (cont.)

- In pseudocode, our rendering of the car might look like this:

```
{
    float s; /* speed */
    float d[3]; /* direction */
    float t; /* time */

    /* determine speed and direction at time t*/
    draw_right_front_wheel(s,d);
    draw_left_front_wheel(s,d);
    draw_right_rear_wheel(s,d);
    draw_left_rear_wheel(s,d);
    draw_chassis(s,d);
}
```



- It fails to show the relationships of the different parts of the car well (e.g., the wheels do not rotate independently; the chassis must move together with the wheels).

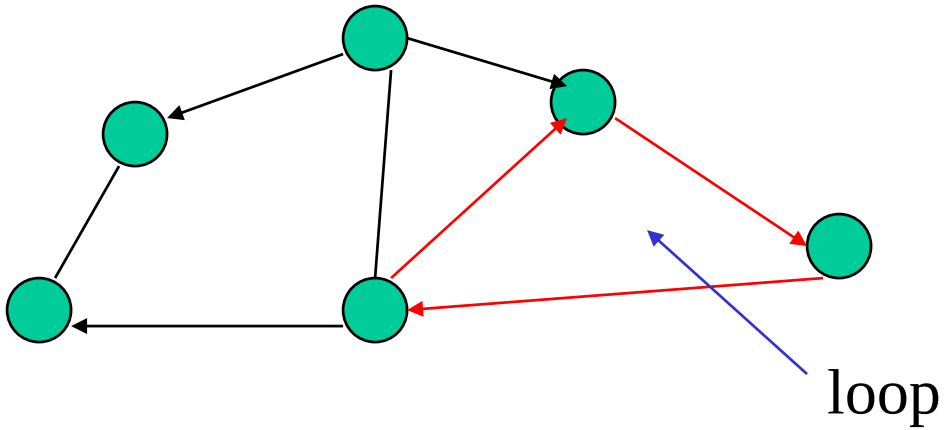
- **Question:** Can we use a graph to represent the relationships?

Graphs

- We can represent the relationship between parts of a model with graphs
- A graph consists of a number of ***nodes*** and ***edges*** (or ***links***)

- An edge connects a pair of nodes
  - Edges can be directed or undirected

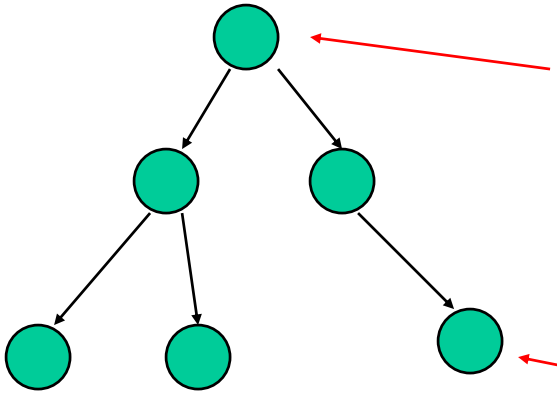
- A ***cycle*** graph: is a directed path that has at least one loop



Trees

- A graph in which each node (except the root) has exactly one parent node but
  - may have multiple children
  - may have no children
    (such nodes are known as leaf or terminal node)

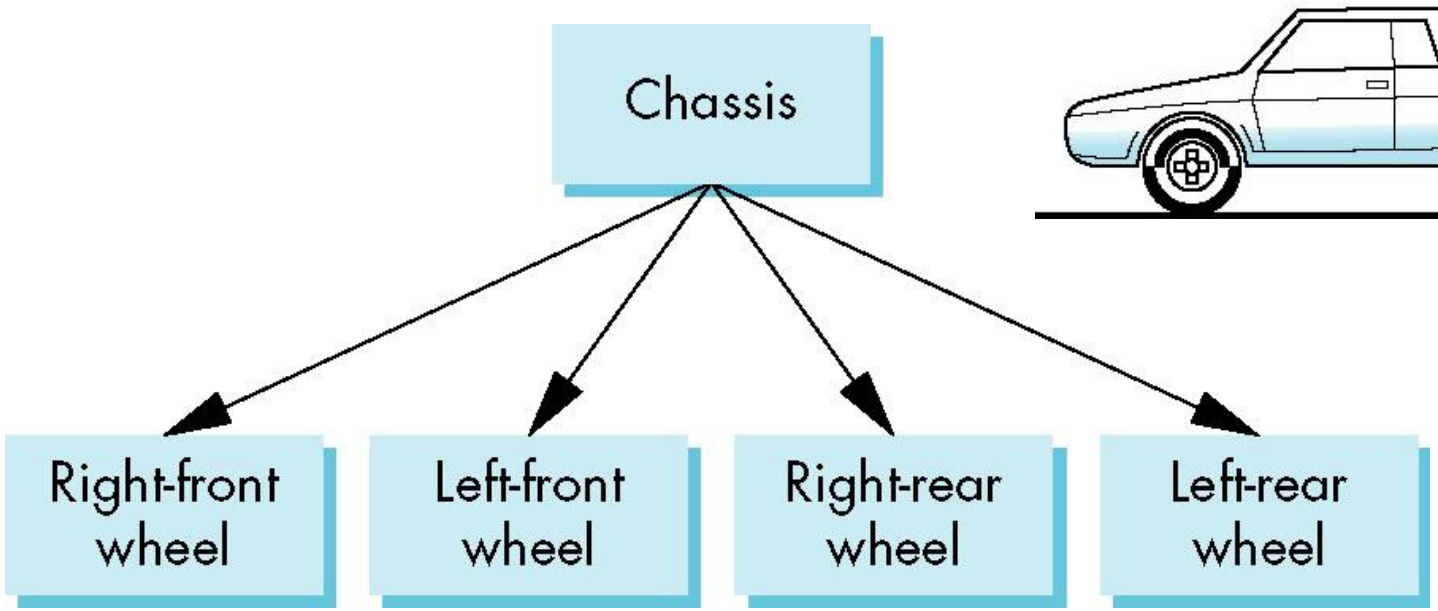
This implicitly imposes that the graph cannot have loops



root node (has no parent node)

leaf node

Tree Model of Car



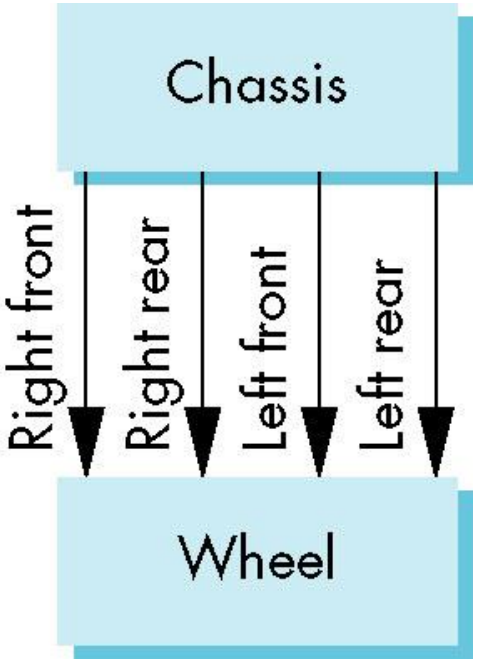
Modeling with Trees

To model our complex object using a tree data structure, we

- Must decide what information to place in the nodes and what to put in the edges
- For nodes, define
  - what to draw, and
  - pointers to all the child nodes
- For edges, we may have
  - information on incremental changes to transformation matrices (which can also be stored in the nodes)

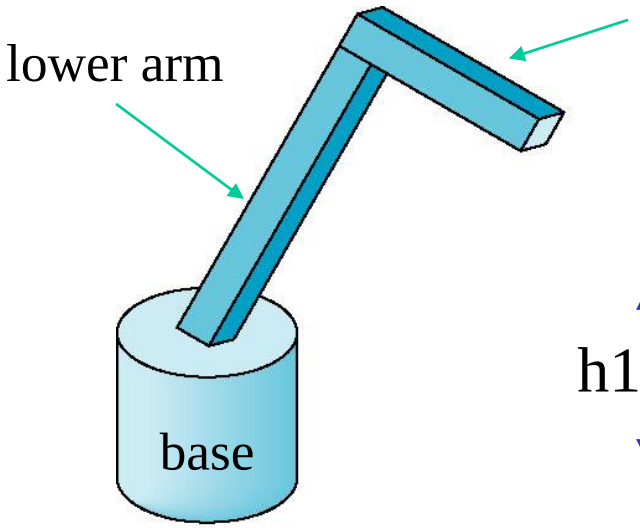
DAG Model

- If we use the fact that all the wheels are identical, we get a ***directed acyclic graph*** (**DAG**)
  - Not much different than dealing with a tree



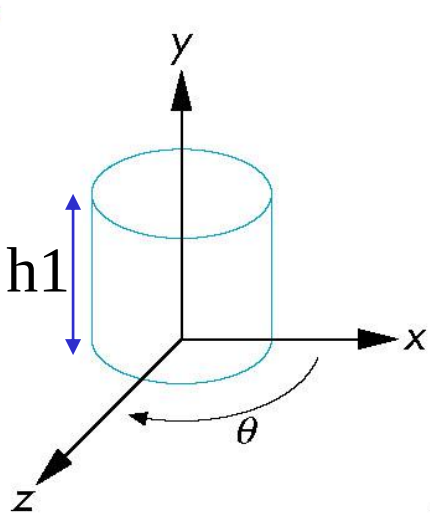
One or more edges between two nodes

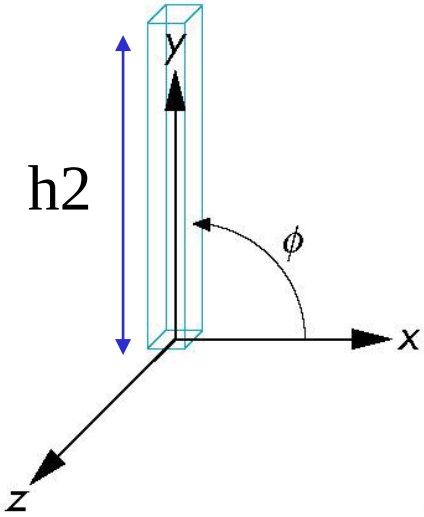
Example: A Robot Arm

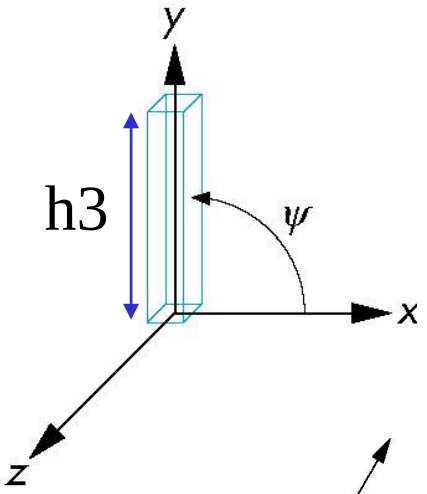


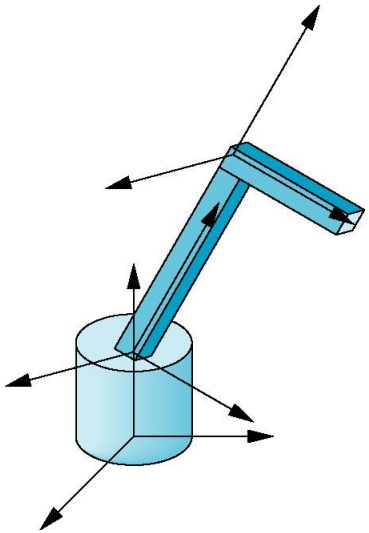
robot arm

parts in their own coordinate systems









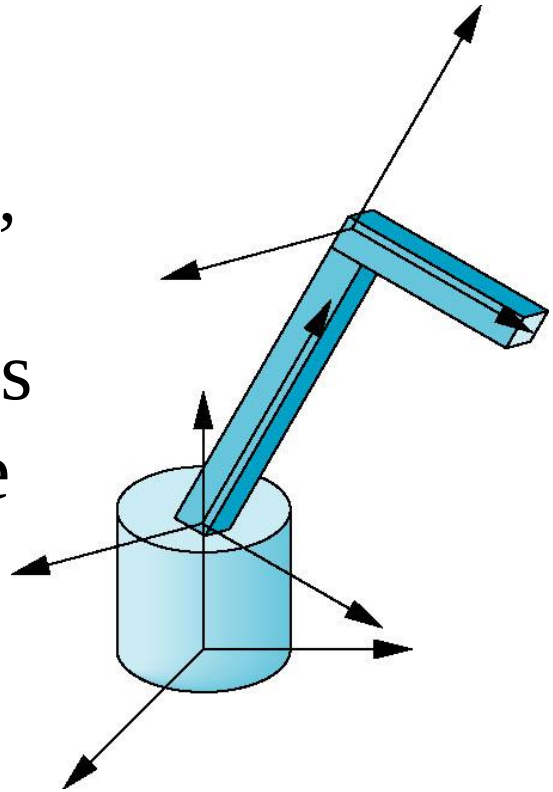
We can model it with only two parallelepipeds and a cylinder

Articulated Models

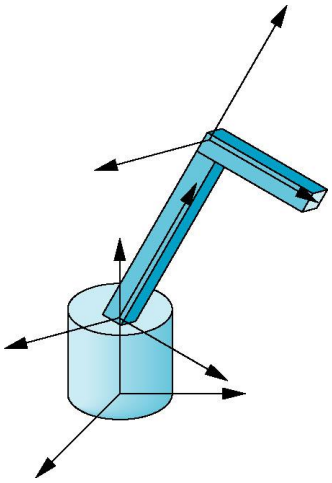
- The robot arm is an example of an ***articulated model*** where,

  - adjacent parts are connected at a joint,

  articulated objects can be defined as objects composed of more than one rigid parts connected by joints

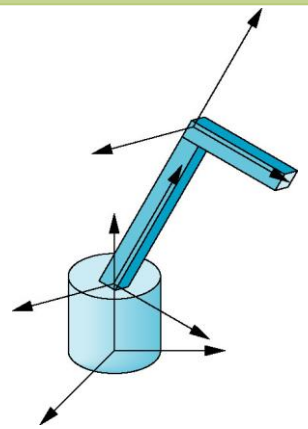


Relationships in Robot Arm



- Base rotates independently
  - Single angle determines position
- Lower arm attached to base
  - Its position depends on the base
  - Must also translate relative to base and rotate about connecting joint
- Upper arm attached to lower arm
  - Its position depends on both base and lower arm
  - Must translate relative to lower arm and rotate about joint connecting to lower arm

Required Matrices



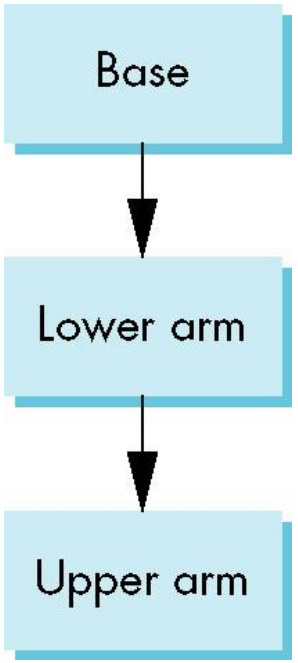
- Rotation of base: $\boldsymbol{R}_b$
  - Apply $\boldsymbol{M} = \boldsymbol{R}_b$ to base
- Translate lower arm relative to base: $\boldsymbol{T}_{lb}$
- Rotate lower arm around joint: $\boldsymbol{R}_{la}$
  - Apply $\boldsymbol{M} = \boldsymbol{R}_{la}\boldsymbol{T}_{lb}\boldsymbol{R}_b$ to lower arm
- Translate upper arm relative to lower arm: $\boldsymbol{T}_{ul}$
- Rotate upper arm around joint: $\boldsymbol{R}_{ua}$
  - Apply $\boldsymbol{M} = \boldsymbol{R}_{ua}\boldsymbol{T}_{ul}\boldsymbol{R}_{la}\boldsymbol{T}_{lb}\boldsymbol{R}_b$ to upper arm

OpenGL Code for Robot

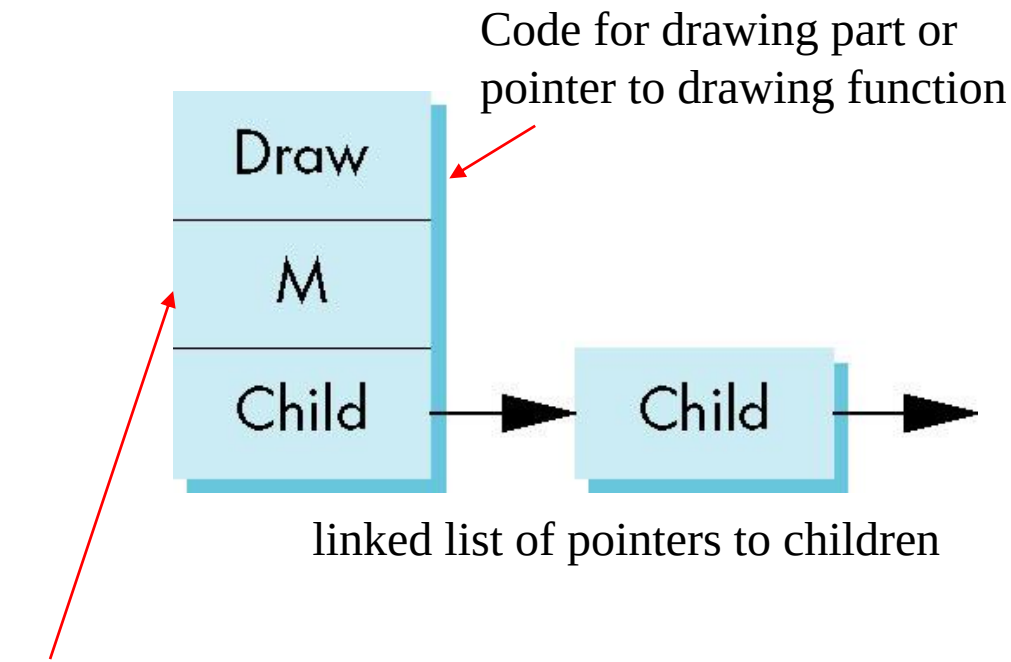
```
mat4 modelMatrix;

void robot_arm()
{
    modelMatrix = RotateY(theta);
    base();
    modelMatrix *= Translate(0.0, h1, 0.0);
    modelMatrix *= RotateZ(phi);
    lower_arm();
    modelMatrix *= Translate(0.0, h2, 0.0);
    modelMatrix *= RotateZ(psi);
    upper_arm();
}
```

Tree Model of a Robotic Arm

- Codes in the nodes of the tree show relationships between parts of the model
  - Can change the "look" of body parts easily without altering relationships
- This is a simple example of tree model
- We want a general code structure for nodes



Possible Code Structure



If we store all the necessary information in the nodes, rather than in the edges, then each node must store at least three items:

1. A pointer to a function that draws the object represented by the node

2. A homogeneous-coordinate matrix that positions, scales, and orients this node (and its children) relative to the node's parent

3. Pointers to children of the node

Generalizations

- Need to deal with multiple children
  - How do we represent a more general tree?
  - How do we traverse such a data structure?

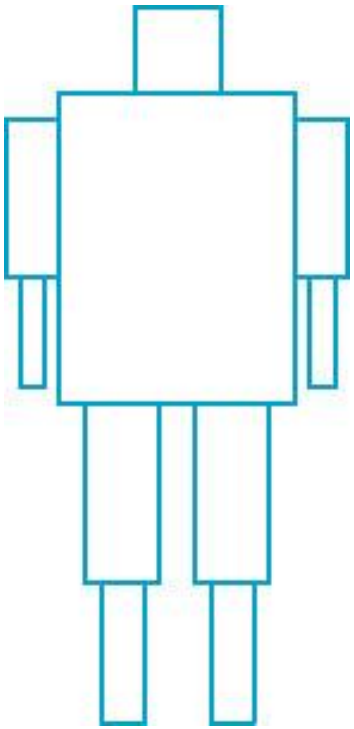
- Animation
  - How to use dynamically?
  - Can we create and delete nodes during execution?

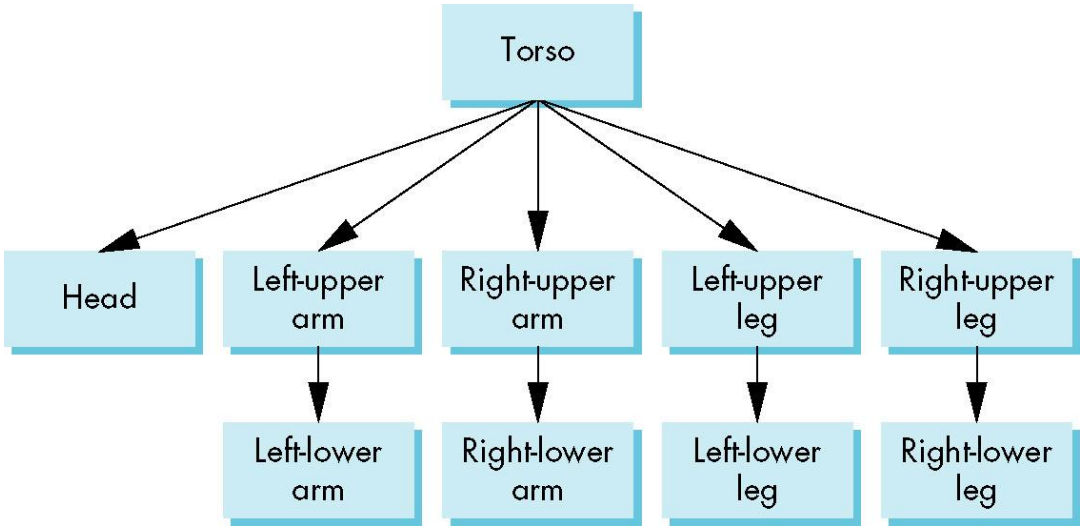
Humanoid Figure

The humanoid robot has 10 body parts:

torso, head, left-upper and left-lower arms,

right-upper and right-lower arms, left-upper and left-lower legs, right-upper and right-lower legs.

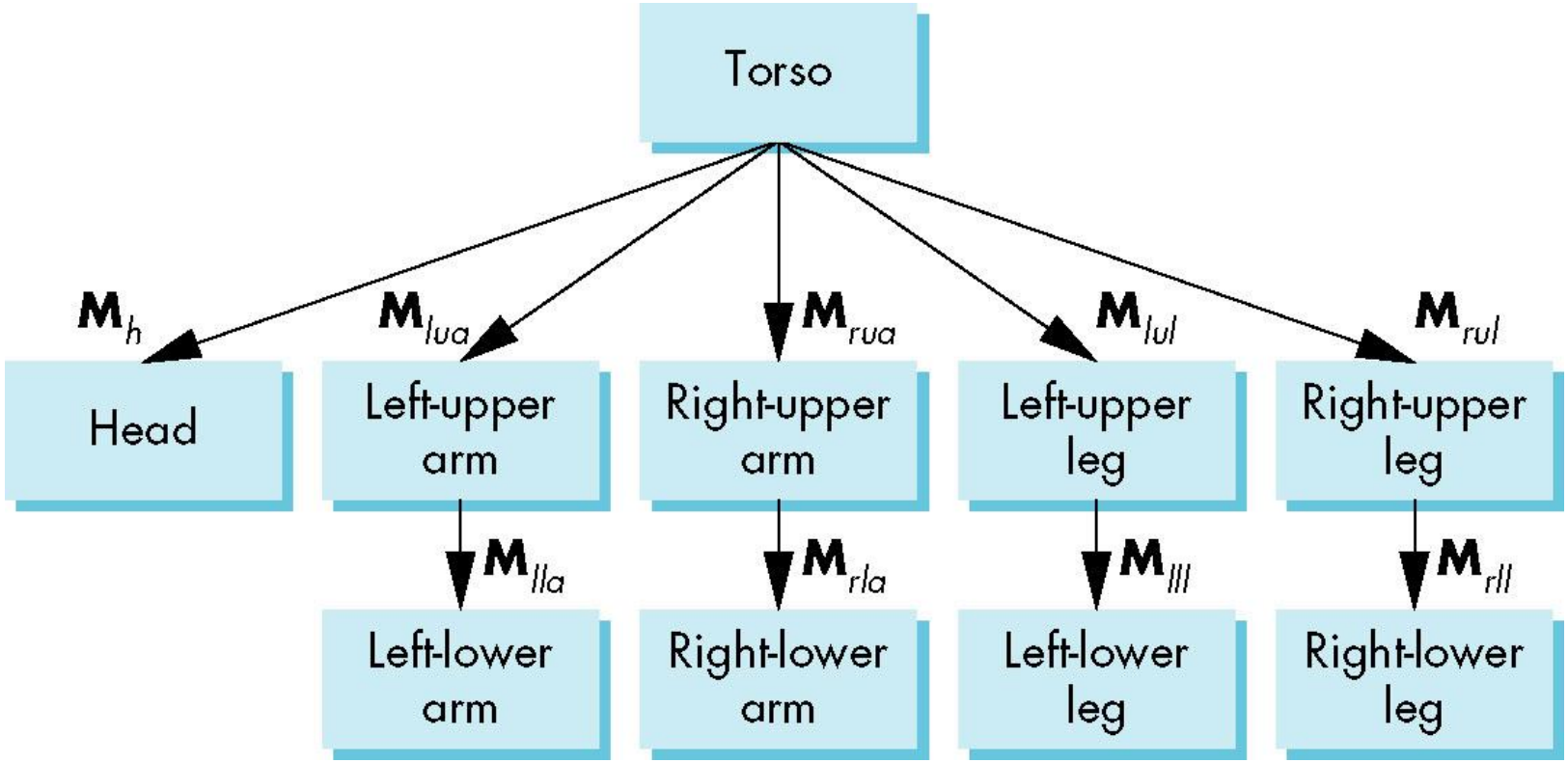




Building the Model

- Can build a simple implementation using quadrics: ellipsoids and cylinders (to represent the body parts)
- Access body parts through function calls:
  - **torso()**
  - **left_upper_arm()**
- The transformation matrix stored in a node describes the position of the node with respect to its parent, e.g.
  - $\boldsymbol{M}_{lla}$ positions left lower arm with respect to left upper arm

Tree with Matrices

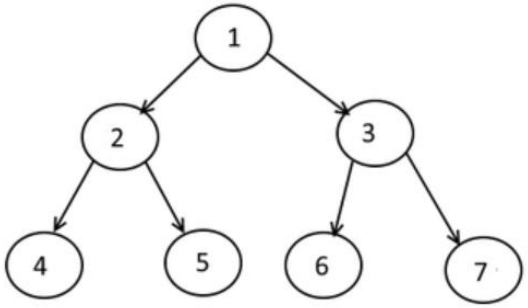


Transformation Matrices

- There are 10 relevant matrices
  - $\boldsymbol{M}$ positions and orients entire figure through the torso which is the root node
  - $\boldsymbol{M}_h$ positions head with respect to torso
  - $\boldsymbol{M}_{lua}$, $\boldsymbol{M}_{rua}$, $\boldsymbol{M}_{lul}$, $\boldsymbol{M}_{rul}$ position upper arms and legs with respect to torso
  - $\boldsymbol{M}_{lla}$, $\boldsymbol{M}_{rla}$, $\boldsymbol{M}_{lll}$, $\boldsymbol{M}_{rll}$ position lower parts of limbs with respect to corresponding upper limbs

Display and Traversal

- The pose of the robot is determined by 11 joint angles (two for the head and one for each other part)
- Display of the tree requires a *tree traversal (depth or width first?)*
  - Visit each node once.
  - Call the *display* function at each node that describes the part associated with the node This involves applying the correct transformation matrix for the position and orientation of the part.



Width first: 1234567
Depth first: 1245367

Further Reading

"Interactive Computer Graphics – A Top-Down Approach with Shader-Based OpenGL" by Edward Angel and Dave Shreiner, 6th Ed, 2012

- Sec. 8.1 *Symbols and Instances*
- Sec. 8.1 *Hierarchical Models*
- Sec. 8.3 *A Robot Arm*
- Sec. 8.4 *Trees and Traversal*
- Sec. 8.5 *Use of Tree Data Structures*
- Sec. 8.6 *Animation*