3D Hierarchical Modelling

Intended Learning Outcomes

- Understand the need of hierarchical structuring for building articulated 3D objects
- Able to compute the relative coordinate transform between component parts
- Able to represent an articulated 3D object as a hierarchical structure using OpenGL

Problem:

- Given a large number of graphics models which form parts of a whole object, it is cumbersome to animate each part by individual commands

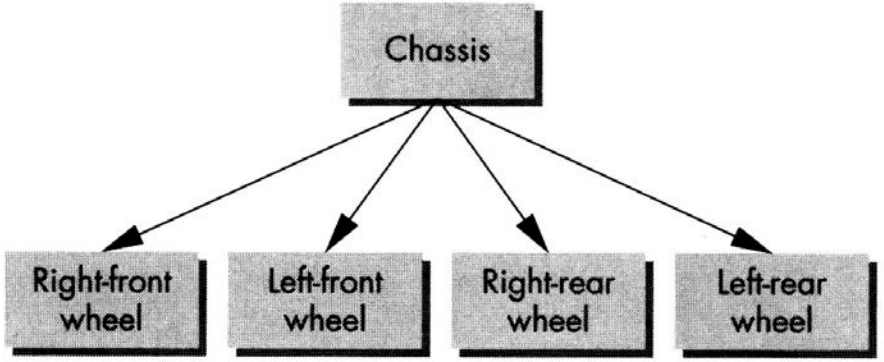
Example : Animate a car moving at a speed of 20 miles and in direction (2, 3, 4)

```
main ()
{
      float s     = 20.0; /* speed */
      float d[3] = {2.0, 3.0, 4.0}; /* direction */

      draw_chassis (s, d);

      draw_right_front_wheel (s, d);
      draw_left_front_wheel   (s, d);
      draw_right_rear_wheel  (s, d);
      draw_left_rear_wheel    (s, d);

}
```



Tree with directed edge

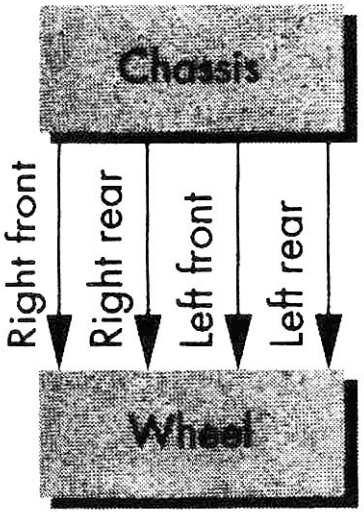
Bad Programming - Redundancy: the 4 draw wheel functions can be replaced by one function

Introduction of hierarchical structures

- Use *relative transformation* to link the movements of different parts
- Use a single function for a unique (single) part

Directed Acyclic Graph (DAG)

- DAG is a graph with directed arc but no cycle
- It is a tree but additional allows more than one arc from one node to another node



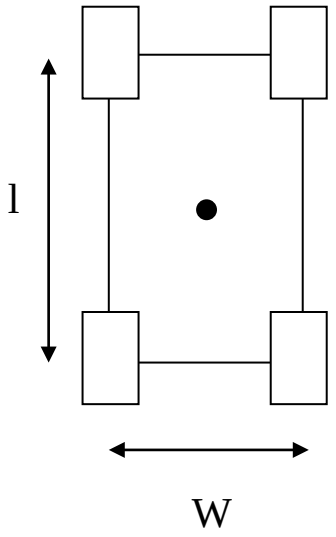
Revised program

```
main ()
{
      float s = 20.0;
      float d[3] ={2.0, 3.0, 4.0};
      float w = 2.0, l = 4.0;        // width and length of the car

      draw_chass (s, d);

      glTranslatef ( w/2 , l/2, 0 );    // position the right front wheel
      draw_wheel (s, d);
     glTranslatef ( -w,     0,  0 );    // position the left front wheel
     draw_wheel (s, d);
     glTranslatef (  0,     -l ,  0 );    // position the left rear wheel
     draw_wheel (s, d);
     glTranslatef ( w,     0,   0  );     // position the right rear wheel
     draw_wheel (s, d);
}
```

Let the initial coordinate system be the centroid of the car



We can make it more systematic by formally introducing coordinate system change, which we do below

Moving a Robot Arm – a 3 level hierarchy

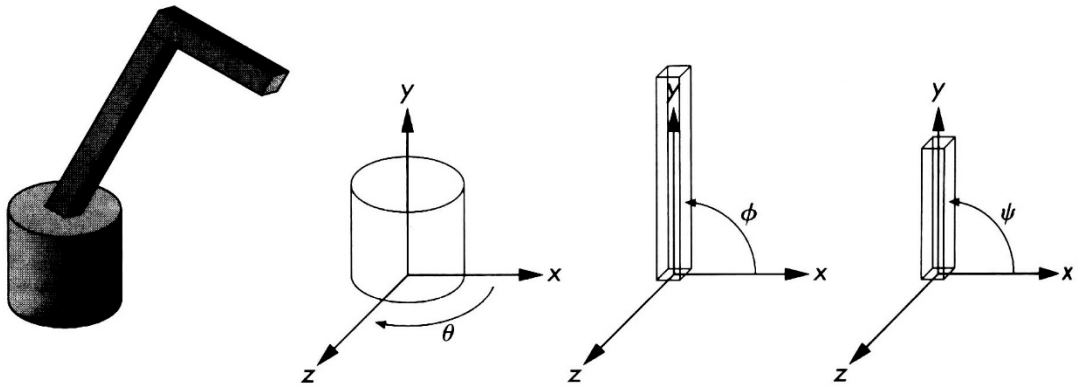
Parts : **base B** (cylinder),
lower arm La (rectangular box)
upper arm Ua (rectangular box)

Arm has 3 degree of freedom:

B rotate about Y by θ

La rotate about Z by ϕ

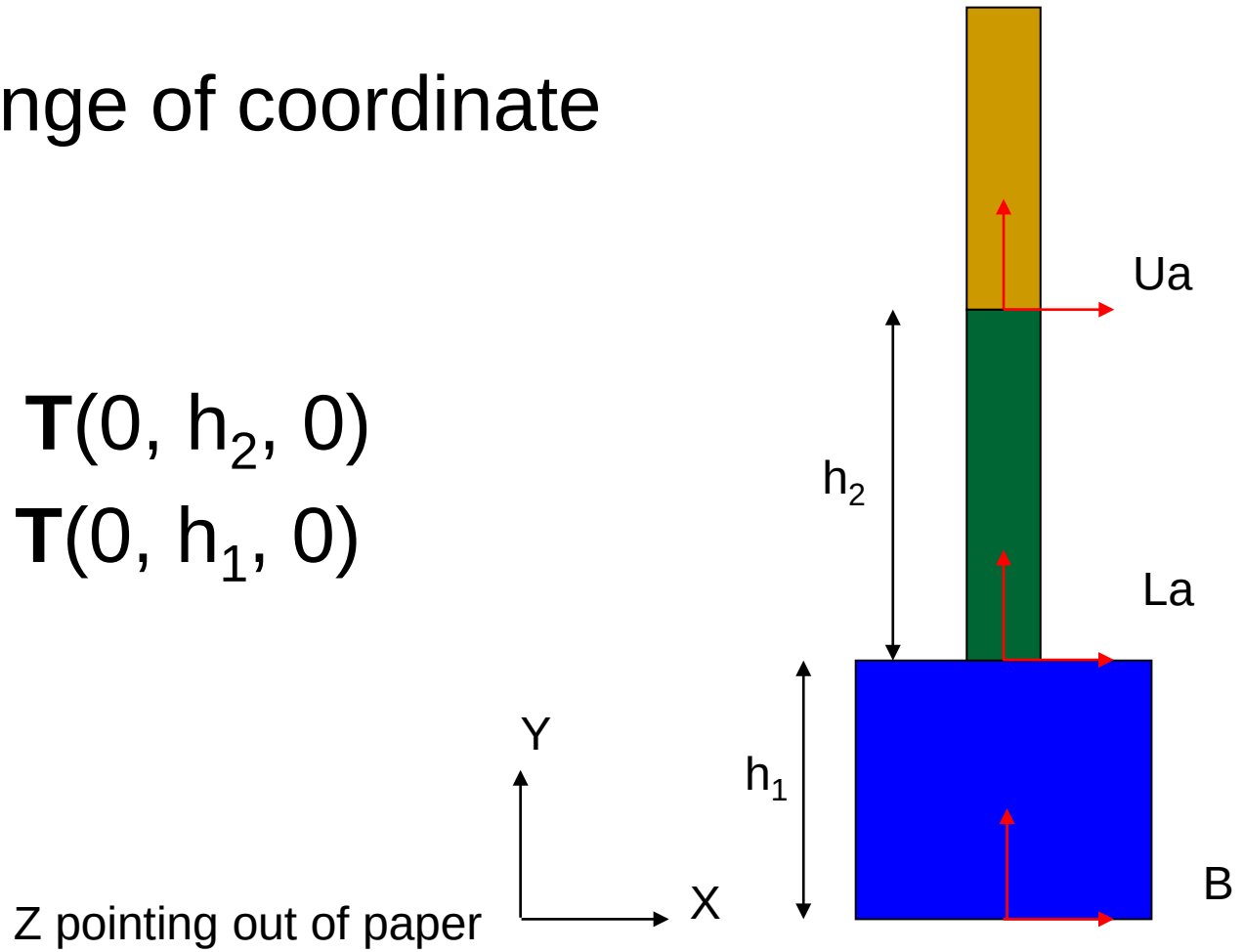
Ua rotate about Z by ψ



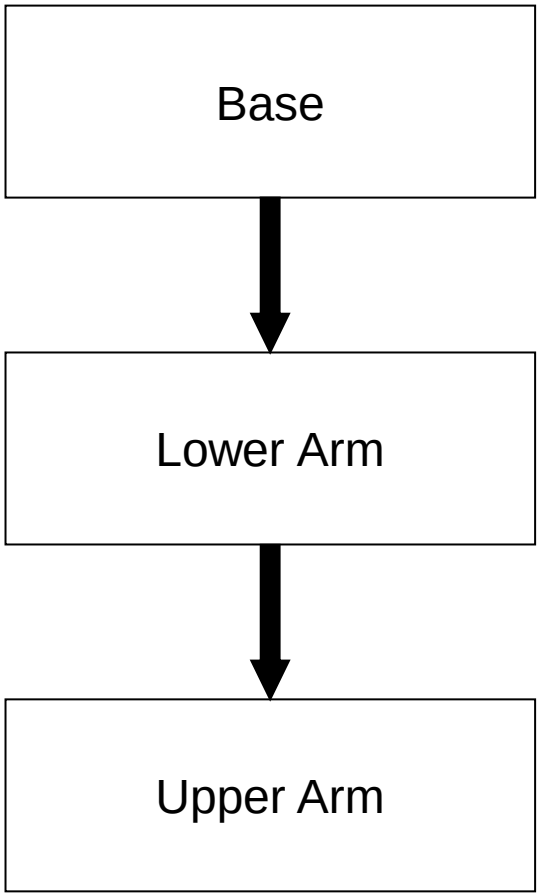
Relative Coordinate Transformations

- Use Change of coordinate system:

- $\mathbf{M}_{La \leftarrow Ua} = \mathbf{T}(0, h_2, 0)$
- $\mathbf{M}_{B \leftarrow La} = \mathbf{T}(0, h_1, 0)$



DAG



Write a program to …

- Rotate the robot arm about its base by θ, then about its lower arm by ϕ, then about its upper arm by ψ
- when rotating the whole arm, everything should move; but when rotating the lower arm, only it and the upper arm should move; when rotating the upper arm, only the upper arm should move.
- Solve this using a hierarchy concept

Program

robot_arm ()
{

glRotatef (theta, 0.0, 1.0, 0.0); // $\mathbf{R}_y(\theta)$ *rotate the whole robot arm*

// *each point of whole robot arm will be pre-multiplied by* $\mathbf{R}_y(\theta)$
base ();

glTranslatef (0.0, h_1*, 0.0);* // $\mathbf{M}_{B\leftarrow La}$ *changes lower arm coord. sy. to base coord. sy.*
glRotatef (phi, 0.0, 0.0, 1.0); // $\mathbf{R}_z(\phi)$ *rotate the lower arm*

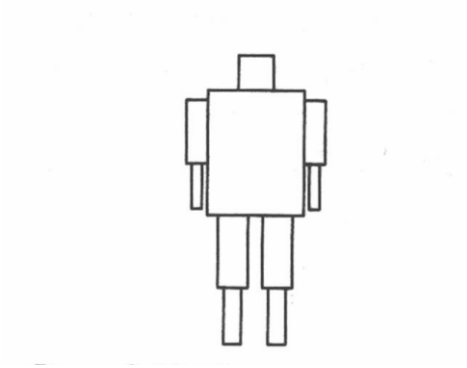
// *each point of lower arm will be pre-multiplied by* $\mathbf{R}_y(\theta)\mathbf{T}(0,h_1,0)\mathbf{R}_z(\phi)$
lower_arm ();

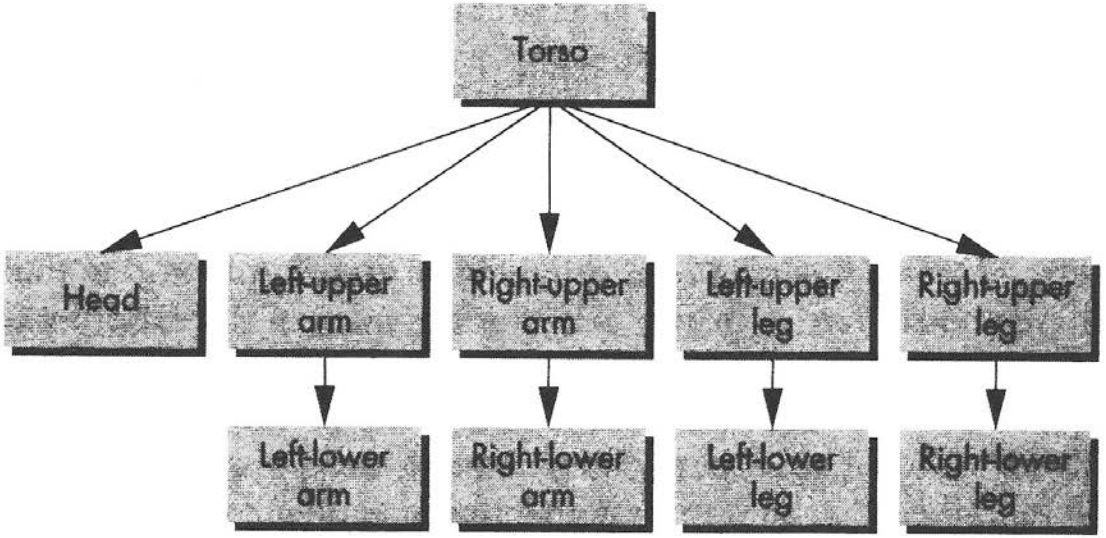
glTranslatef (0.0, h_2*, 0.0);* // $\mathbf{M}_{La\leftarrow Ua}$ *changes upper arm coord. sy. to lower arm coord. sy.*
glRotatef (psi, 0.0, 0.0, 1.0);

// each point of upper arm will be pre-multiplied by $\mathbf{R}_y(\theta)\mathbf{T}(0,h_1,0)\mathbf{R}_z(\phi)\mathbf{T}(0,h_2,0)\mathbf{R}_Z(\psi)$
upper_arm ();
}

Moving a Robot

- Need to organize the hierarchy better
- Solution: use *glPushMatrix* and *glPopMatrix* to store and retrieve intermediate composite relative transformations





Program

```
Robot ()
{
        glPushMatrix ();
        torso;

        glTranslate …
        glRotate …
        head ();

        glPopMatrix ();          // go back to the node of the torso
         glPushMatrix ();

        glTranslate …            // similar technique used here as that
        glRotate …               // used in example 2
        left_upper_arm ();
        glTranslate …
        glRotate …
        left_lower_arm ();

        glPopMatrix ();          // go back to the node of the torso
         glPushMatrix ();

        glTranslate …
        glRotate …
        right_upper_arm ();

                ⋮
}
```

References

- Our exposition follows:

  E. Angel, Interactive Computer Graphics: A Top-down Approach Using OpenGL, 5th Ed. (2009), Ch. 10.1-10.4
- Ch. 11 of text provides an alterative reference.