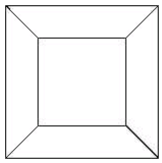
3D OpenGL Basic Models
Hierarchical Modeling

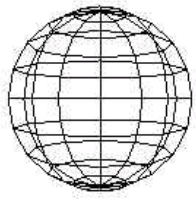
GLUT 3D Models

- Two main categories
  - Wireframe Models
  - Solid Models
- Basic Shapes
  - Cube: glutWireCube( ), glutSolidCube( )
  - Cone: glutWireCone( ), glutSolidCone( )
  - Sphere, Torus, Tetrahedron
- More advanced shapes
  - Octahedron, Dodecahedron, Icosahedron
  - Teapot (symbolic)

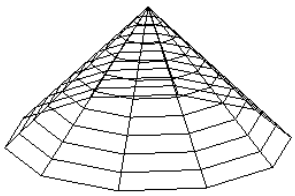
Basic 3D GLUT Objects



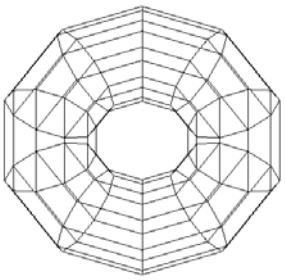
`glutWireCube(1.0);`



`glutWireSphere(0.5,10,10);`

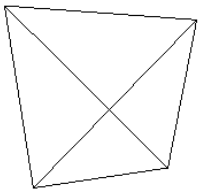


`glutWireCone(1, 1, 10, 10);`

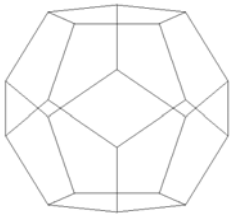


`glutWireTorus(0.5,1.5,10,10)`

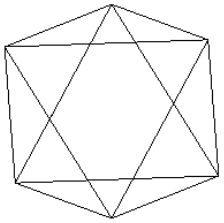
GLUT Platonic Objects



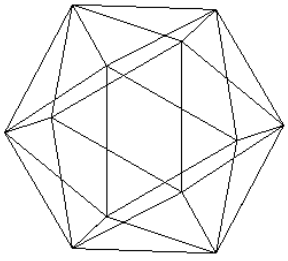
`glutWireTetrahedron();`



`glutWireDodecahedron();`

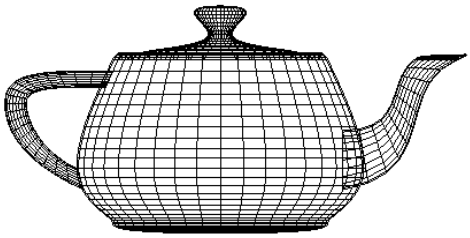


`glutWireOctahedron();`



`glutWireIcosahedron();`

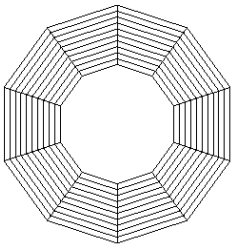
GLUT Object (Symbolic)



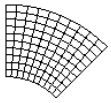
```
glutWireTeapot(1.0);
```

GLU Models

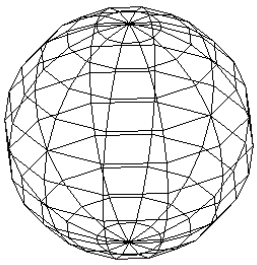
- GLUT does not provide a Cylinder object
- GLU provides so called ***Quadrics*** objects:



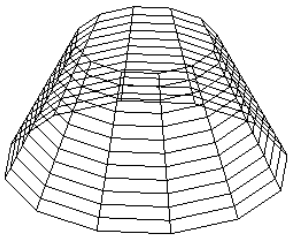
disk



partial disk



sphere



cylinder

Quadrics

- Algebraic surface
  - f(x,y,z) is the sum of polynomials in x, y, z
- Quadric surface
  - Algebraic surfaces with a degree up to 2
  - Examples: x, y, xy, z^2, but not xy^2
- Quadrics:
  - Spheres, disks, cones
  - At most 2 intersections with lines
  - Approximate quadrics surface by polygons

gluCylinder

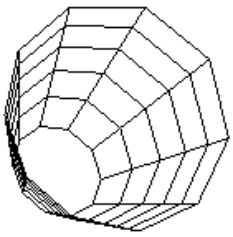
Three steps to create a cylinder:

1. Create a GLU quadric object → sphere, cylinder, disk, partial disk

   GLUquadricObj *p = gluNewQuadric();
2. Set to wire frame mode

   gluQuadricDrawStyle(GLU__LINE);
3. Derive a cylinder object from p

   gluCylinder(p, base, top, height, slice, stacks);

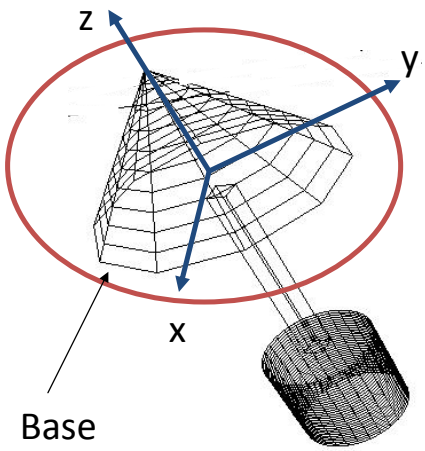


radius at z=0	radius at z=height	number of lines per ring	number of rings

Default position is with base on plane z = 0.

glutWireCone

- Use glutWireCone and gluCylinder to make a lamp

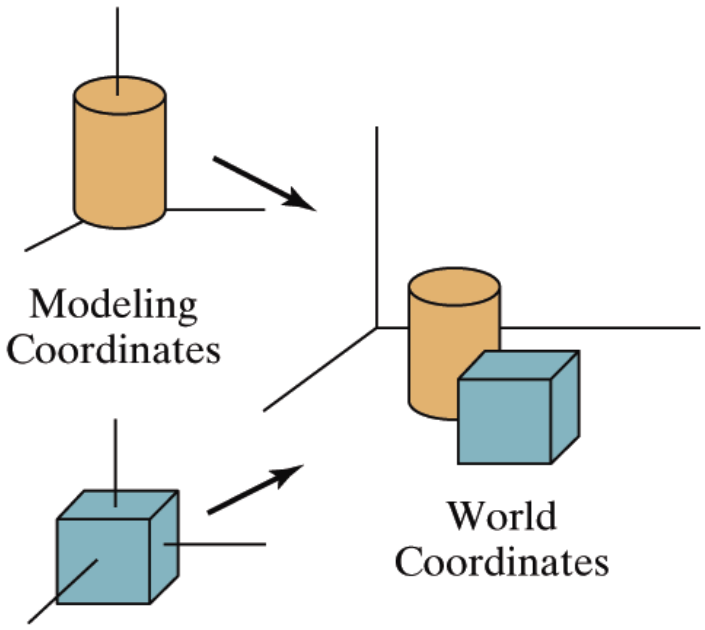


glutWireCone(base, height, slices, stacks)

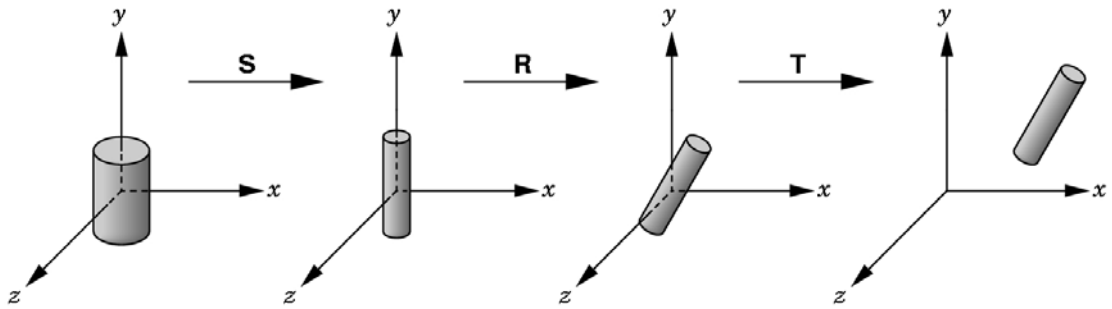
base: the width of its base
height: the height of the cone
slices: number of vertical lines
stacks: number of horizontal lines

A polygonal approximation of a cone.
Default position: base at z=0.

Review: Model to World Frame



Model Frame to World Frame

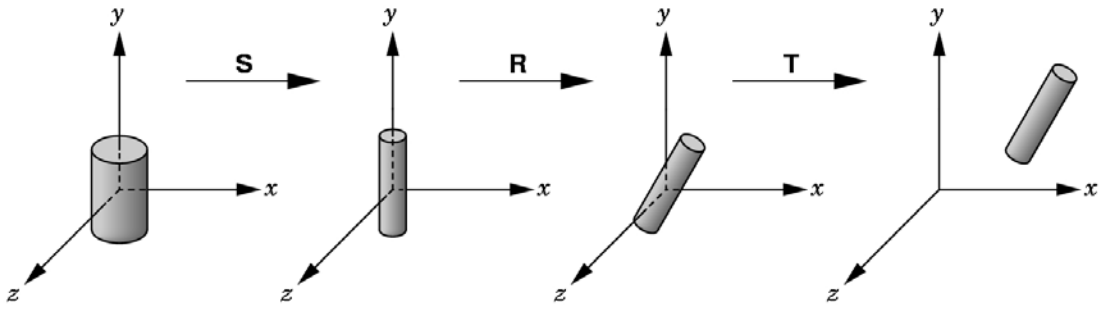


M = T R S

In OpenGL

- Appropriate transformation set up from the model frame (frame of symbols) to the world frame
- Apply it to the MODELVIEW matrix BEFORE EXECUTING the code

Model → World



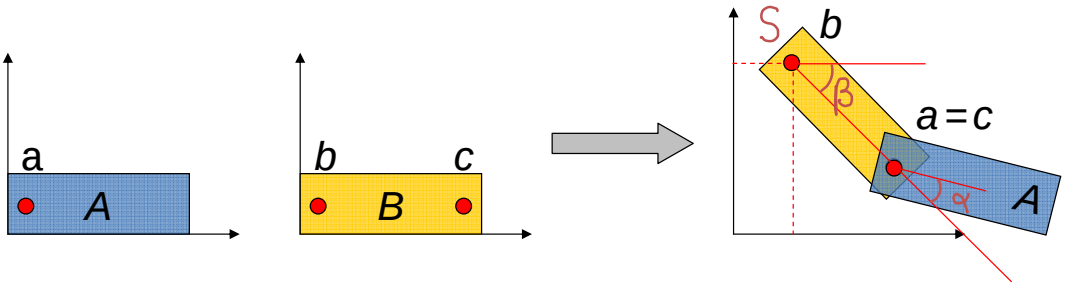

```
glMatrixMode(GL_MODELVIEW);      /* M = T R S */
glLoadIdentity ( );
glTranslatef (....);
glRotatef (...);
glScalef(..);
gluCylinder (.....) /* or other symbol */
```

Hierarchical Models

Articulated Models

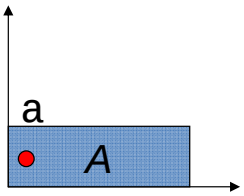
- Rigid parts connected by joints
- [Demo 3D Robot]
- By modeling joint angles, you can do animations
- Question is, given all the join angles, how do you draw the model?

Making an Articulated Arm

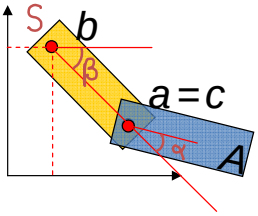


- A minimal 2D jointed object:
  - Two pieces, *A* (“forearm”) and *B* (“upper arm”)
  - Attach point c on *B* to point *a* on *A* (“elbow”)
- Desired parameters:
  - shoulder position *S* (point at which *b* winds up)
  - shoulder angle β (*A* and *B* rotate together about *b*)
  - elbow angle α (*A* rotates about *a*, which stays attached to *c*)

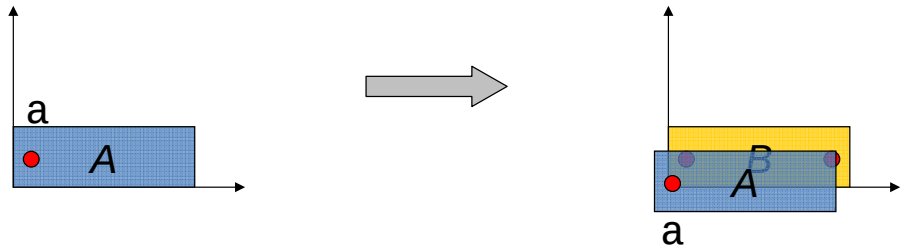
Making an Arm, step 1



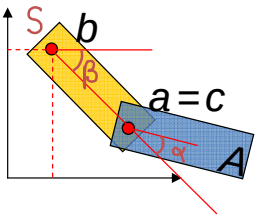
- Start with *A* and *B* in their untransformed configurations (*B* is hiding behind *A*)
- First apply a series of transformations to *A*, leaving *B* where it is…



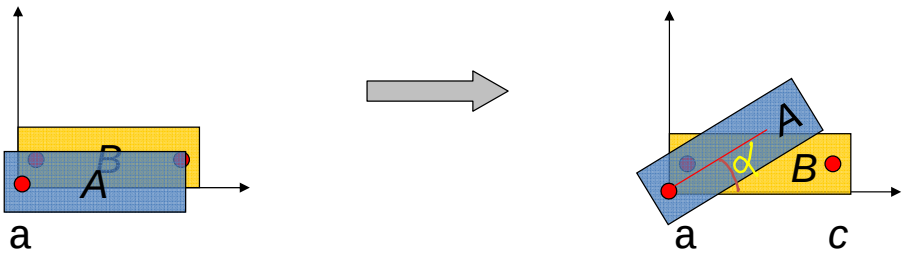
Making an Arm, step 2



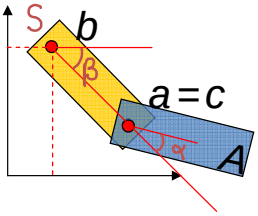
- Translate by -*a*, bringing *a* to the origin
- You can now see B peeking out from behind A



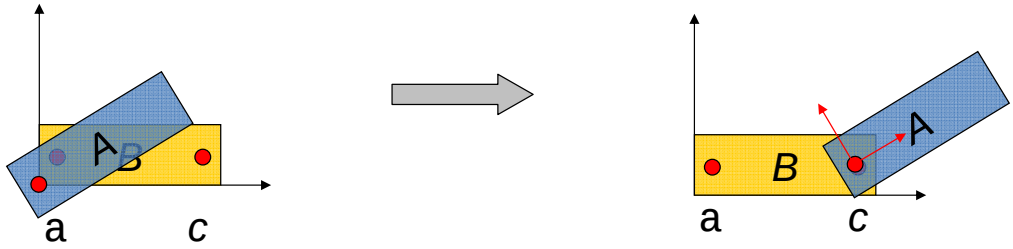
Making an Arm, step 3



- Next, we rotate *A* by the "elbow" angle α

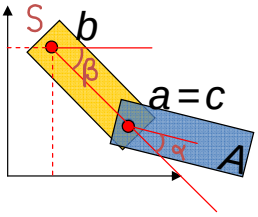


Making an Arm, step 4

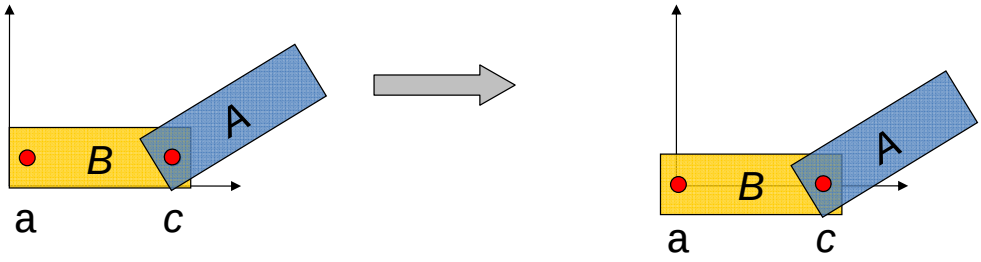


- Translate *A* by *c*, bringing *a* and *c* together to form the elbow joint

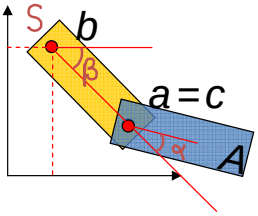
- We can regard *c* as the origin of the *lower arm coordinate system*, and regard A as being in this coordinate system.



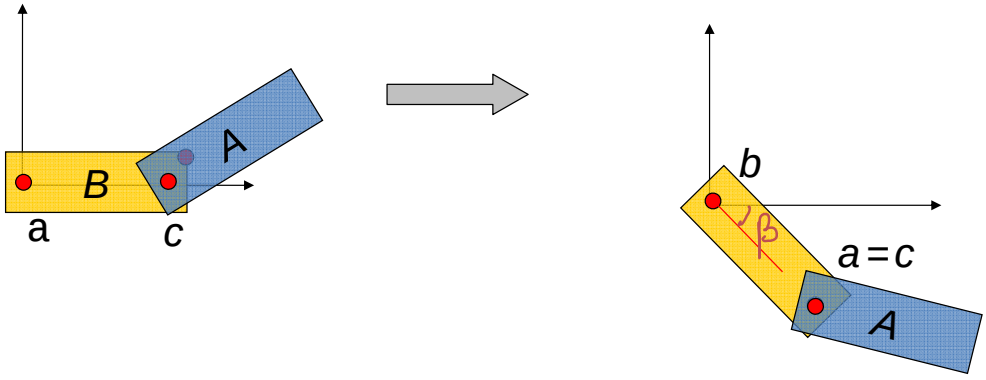
Making an Arm, step 5



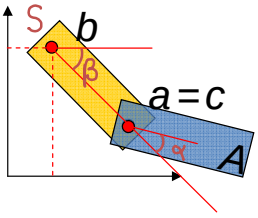
- From now on, each transformation applies to *both A* and *B* (This is important!)
- Translate by -*a*, bringing *a* to the origin
  - *A* and *B* both move together, so the elbow doesn't separate!



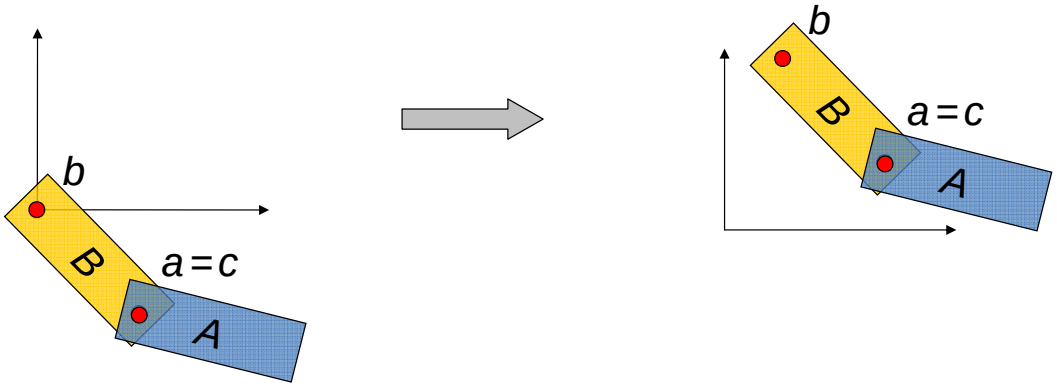
Making an Arm, step 6



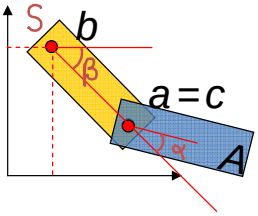
- Next, rotate by the "shoulder" angle -β
  - again, *A* and *B* rotate together



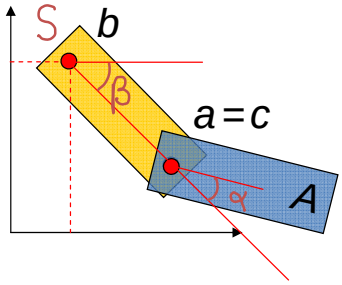
Making an Arm, last step



- Finally, translate by the shoulder position *S*, bringing the arm where we want it
- *b* is at origin of *upper arm coordinate system*

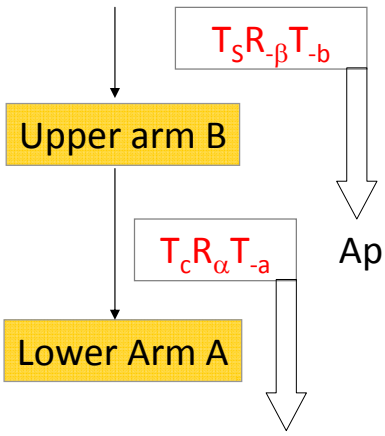


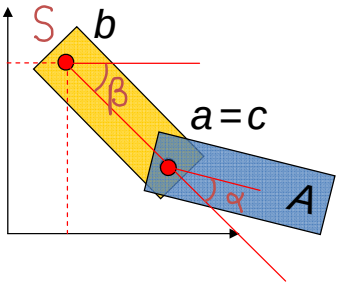
Note that …



- *S*, α, and β are parameters of the model
- But a, b, and c are *structural constants.*
- Changing *S*, α, or β wiggles the arm
- Changing a, b, or c *dismembers it*
  - (useful only in video games!)

Hierarchical Transforms

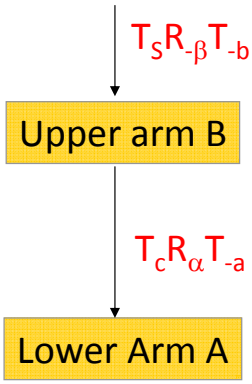




Apply all the way down

Apply all the way down

Hierarchical Transforms in OpenGL

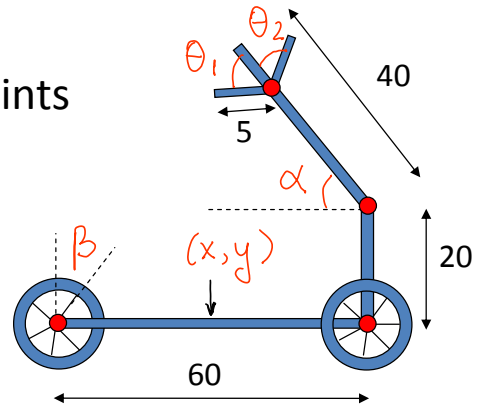


- Down edges:
  - Push, transform, draw
- Up edges:
  - Pop.

```
glLoadIdentity();
glPushMatrix();
        ____________________
        ____________________
        ____________________
        ____________________
glPushMatrix();
        ____________________
        ____________________
        ____________________
        ____________________
glPopMatrix();
glPopMatrix();
```

Articulated Model Exercise

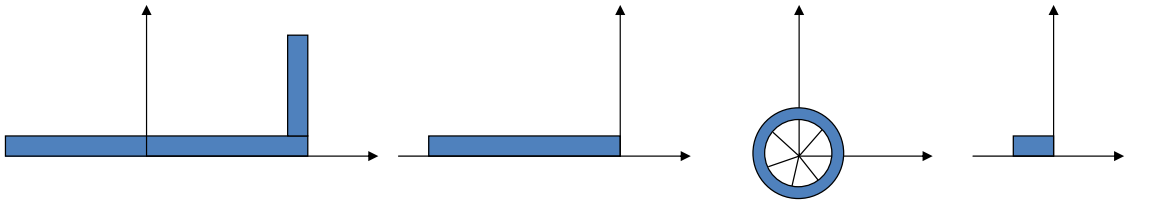
- Articulated model
  - rigid parts connected by joints



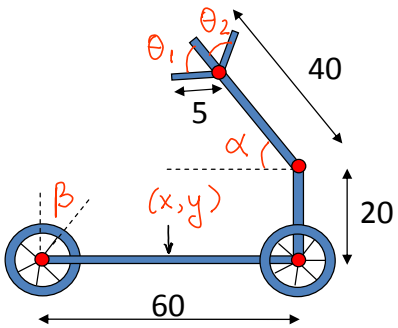
- Given x, y, α, β, θ_1 and θ_2, draw the scooter.

Step 1

- Write functions that draw parts in default location:

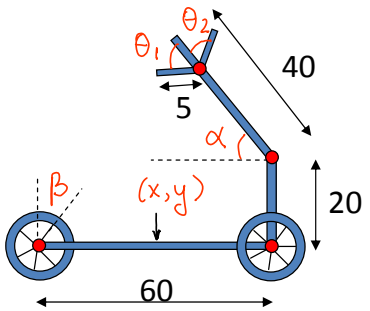


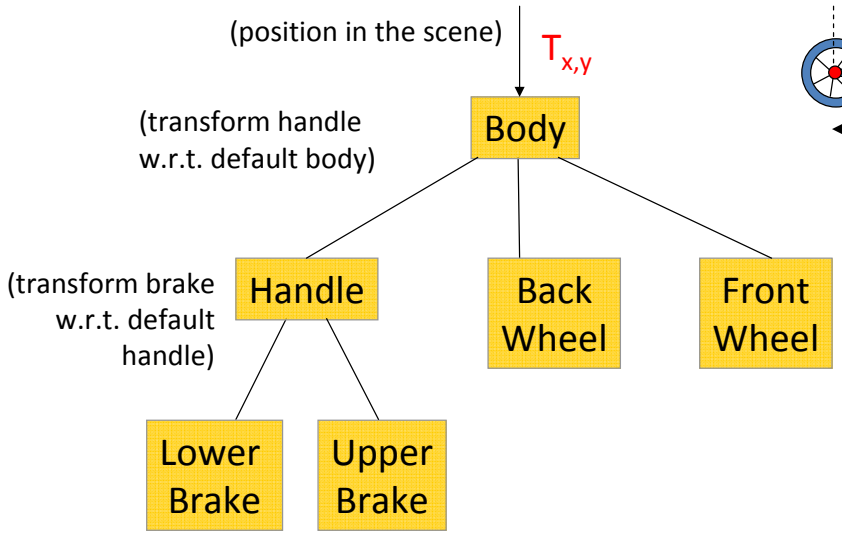
DrawBody() DrawHandle() DrawWheel() DrawBrake()

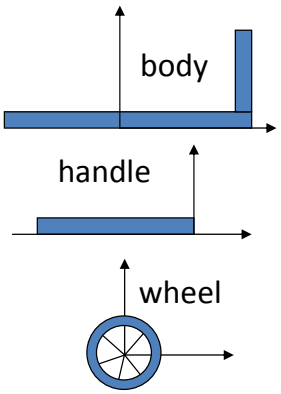


Step 2

- Construct the scene graph (model as a tree)





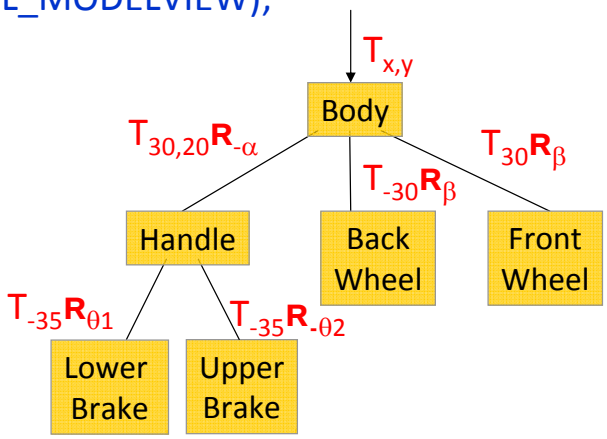


- Label each edge with transformation necessary to position child with respect to parent, in parent's default location

Step 3

- Code it up: glMatrixMode(GL_MODELVIEW);

```
glPushMatrix();
glTranslatef(x, y);
DrawBody();
```

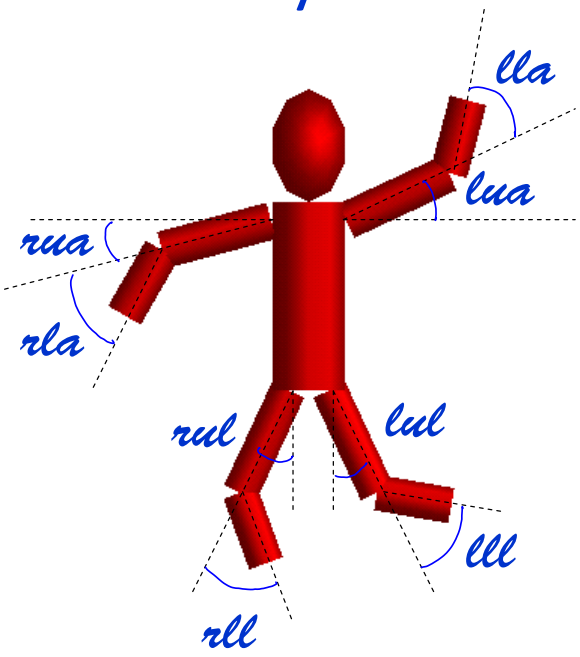


- Down edges: Push, transform, draw. Up edges: Pop.

Back to the Robot …

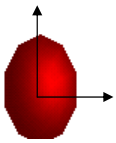
Robot Guy

l--	left
r--	right
-u-	upper
-l-	lower
--a	arm
--l	leg

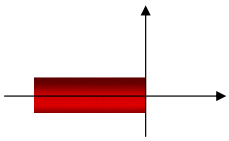


Step 1

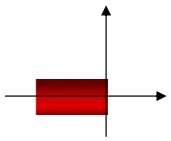
- Write functions that draw parts in default location



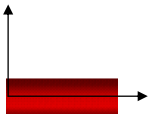
head();
h=3, w=2



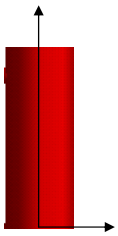
right_upper_arm();
h=1, w=3



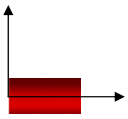
right_lower_arm();
h=1, w=2



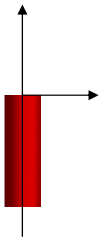
left_upper_arm();
h=1, w=3



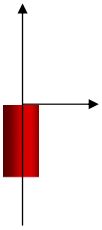
torso();
h=5, w=2



left_lower_arm();
h=1, w=2



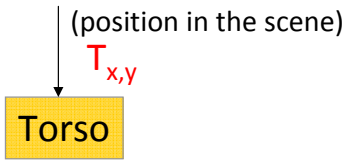
left_upper_leg();
right_upper_leg();
h=3, w=1

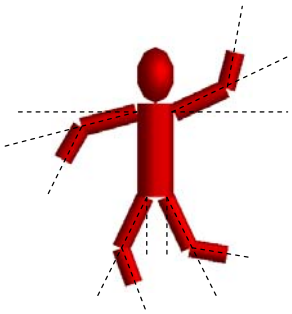


left_lower_leg();
right_lower_leg();
h=2, w=1

Step 2

- Scene Graph (Tree)





Step 3

- Code it up: glMatrixMode(GL_MODELVIEW);

```
glPushMatrix();
glTranslatef(x, y);
torso();
```

Hands-on Session

- Download robotSkeleton.cpp from class website
- Complete Steps 1, 2, 3 to draw the robot
- Add Keyboard events to make robot move and increase/decrease joint angles
- Add a menu that allows selection of individual joints and manipulation of joint angles (+, -)
- Add Idle or Timer events to make robot dance

Summary

- GLUT 3D Models
  - Cube, sphere, cone, torus, teapot
  - Tetr/oct/dodec/icos -- ahedron
- GLU Models
  - Cylinder, sphere, disk, partial disk
- Hierarchical Modeling
  - Transform each object relative to its parent
  - Transformation applies to parent and ALL children
  - Convenient for static models
  - Vital for animations