Lecture 6

Hierarchical Modeling

Reading

Recommended:

- OGL Red Book Chapter 1-2

Further reading:

- Angel 8.1-8.6

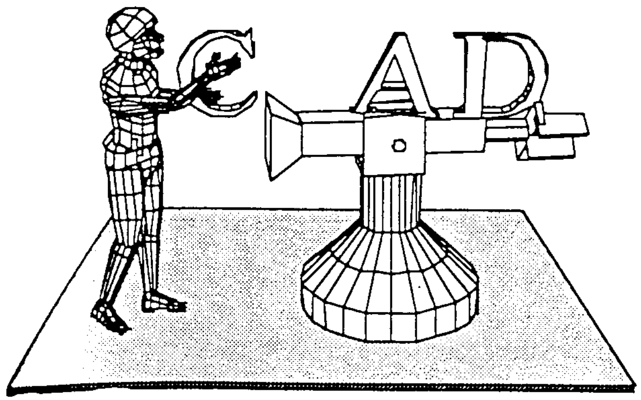
Lecture outline:

1. Hierarchical structure

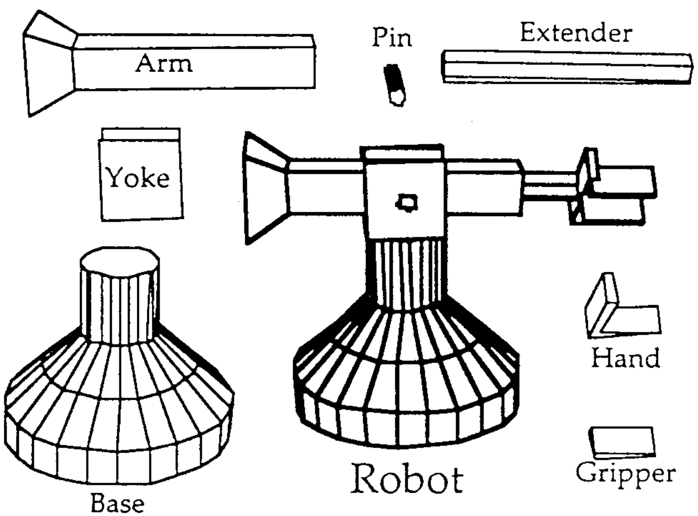
2. View-world or Modelview transformations

3. Basic scenegraph concept

Hierarchical Model



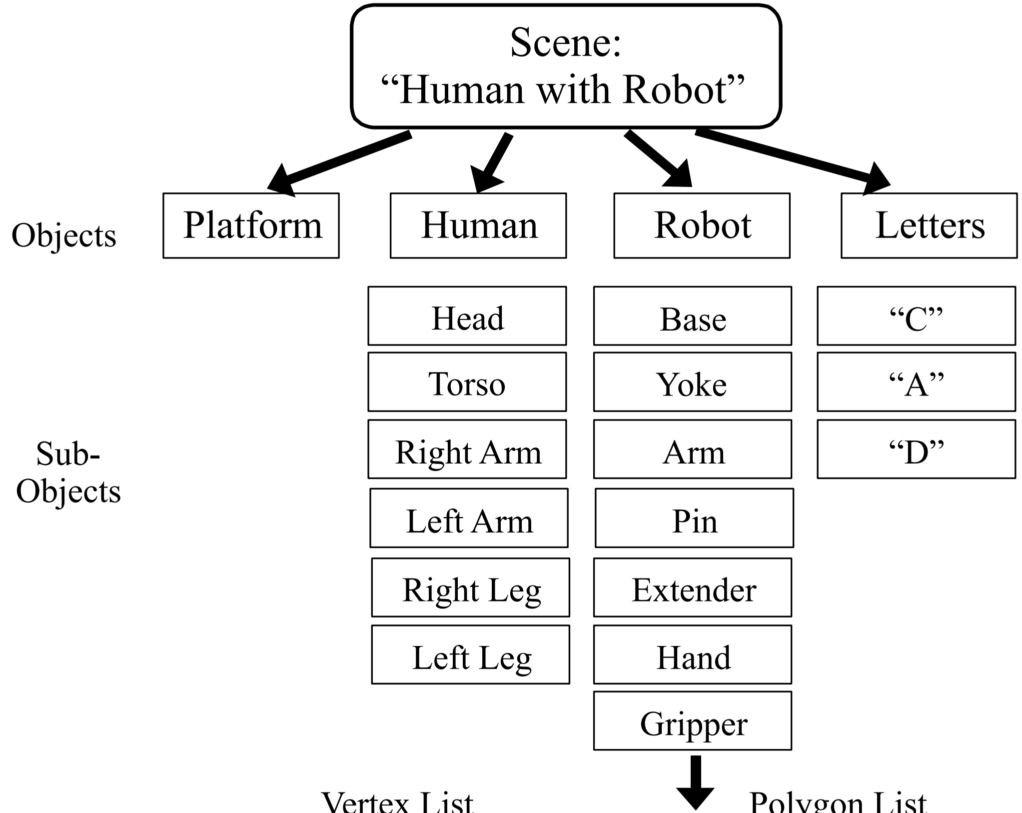
Human with Robot scene based on polyhedra. Note how the whole scene is composed of rectangles, trapezoids, or, in the case of the 3D letters, rectangles and n-sided polygons



Exploded view of hierarchical structure of robot. The robot main object (bold) is constructed by assembling graphical primitive subobjects which are easily generated by CAD systems

Hierarchical Model

Hierarchical structure of a polyhedral scene. Note that each subobject will have its own polygon list and associated vertex list. Also, subobjects such as right arm will have its own subobjects such as upper arm, lower arm, and hand. The hand may, in turn, have subobjects such as a fingers, and so on.



Data Structure

Vertex List

Vertex #	Coordinates
1	x1,y,1z1
2	x2,y2,z2
3	x3,y3,z3
4	x4,y4,z4
5	x5,y5,z5
6	x6,y6,z6
7	x7,y7,z7
8	x8,y8,z8

Polygon List

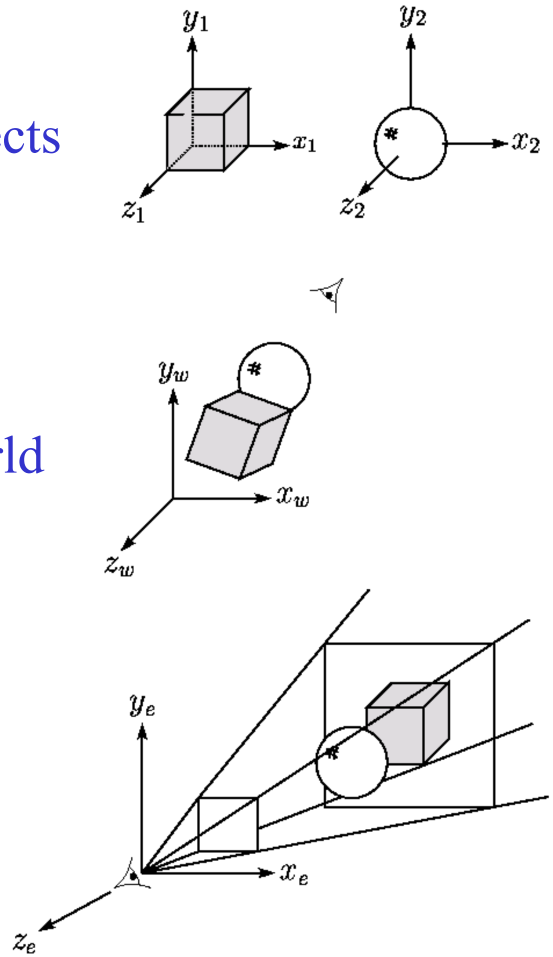
Gripper	
Polygon #	Vertex List
1	V1,V2,V3,V4
2	V1,V4,V8,V5
3	V5,V6,V2,V1
4	V2,V6,V7,V3
5	V4,V3,V7,V8
6	V5,V8,V7,V6

General Viewworld Transformation (2 Matrices)

Different objects

Common world

In front of the Eye



Model space
(Object space)

$\downarrow M_{obj2world}$

World space
(Object space)

$\downarrow M_{world2eye}$

Eye space
(View space)

$\downarrow$

……

Given object coordinate $P_{obj} = (x,y,z)^T$

World coord. = $M_{obj2world} \times P_{obj}$

Eye coord. = $M_{world2eye} \times M_{obj2world} \times P_{obj}$

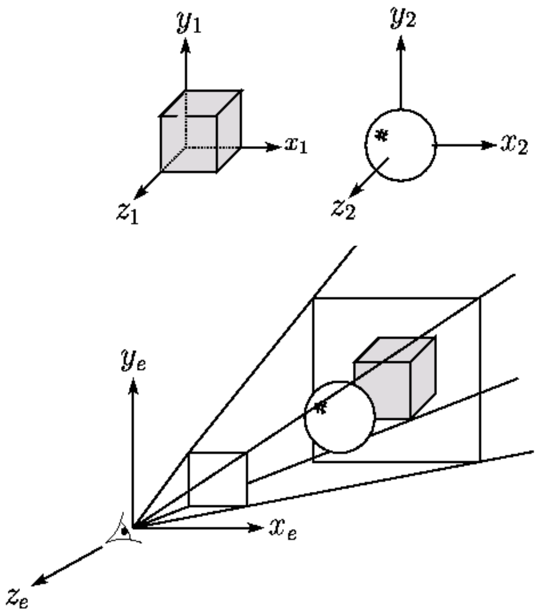
OpenGL Modelview Transformation (1 Matrix)

$M_{obj2world}$ and $M_{world2eye}$ are merged into one matrix,

called the modelview (or viewworld) matrix "$M_{modelview}$"

An object

In front of the Eye



Model space
(Object space)

$\downarrow M_{modelview}$

Eye space
(View space)

$\downarrow$

……

Given object coordinate $P_{obj} = (x,y,z)^T$

Eye coord. = $M_{modelview}$ x P_{obj}

Note: OpenGL puts the eye-point at the origin looking towards negative z-axis

OpenGL Modelview Transformation (1 Matrix)

About the modelview matrix "$M_{modelview}$"

Important Note:

1. OpenGL puts the eye-point at the origin looking towards negative z-axis
2. OpenGL is a state machine: it has an internal memory storage (4 x 4 floating point numbers) for the modelview matrix
3. When calling glTranslate/glRotate/…, the kernel will first construct a matrix for the T/R/S and right multiply it with its internal modelview matrix

OpenGL Modelview Transformation (1 Matrix)

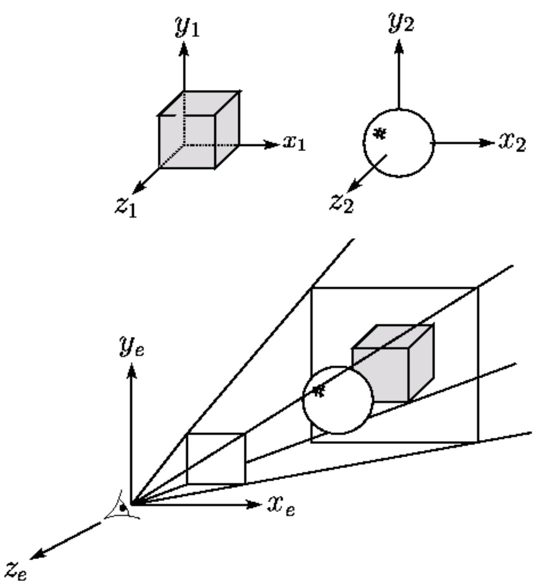
Illustration:

	Memory in Graphics hardware	OGL Commands	Kernel Operations	Comment
Step 0	$M_{modelview}$			Initial value: an identity
Step 1		glTranslate	Construct M_{tran}	Construct a matrix for T
Step 2			$M_{modelview} x M_{tran}$	Right multiply M_{tran} on $M_{modelview}$
Step 3	$M_{modelview} x M_{tran}$			Store the result

Note:

1. See Chapter 2 and Appendix F in the Red book for detail
2. OGL always uses right-multiplication whereas DirectX is flexible (?)

OpenGL Modelview Transformation (1 Matrix)



Model space
(Object space)

$\downarrow M_{\text{modelview}}$

Eye space
(View space)

$\downarrow$

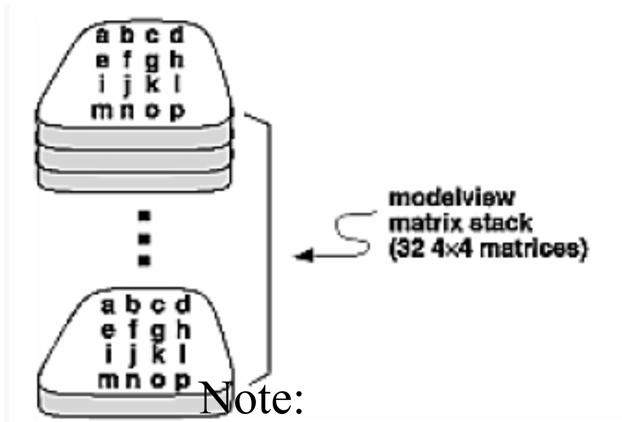
……

Example:

```
glMatrixMode ( GL_MODELVIEW ) ;
glLoadIdentity () ;
glTranslatef ( ball_X , ball_Y , ball_Z ) ;
glRotatef ( ball_ang , ball_dirX , ball_dirY , ball_dirZ ) ;
glScalef ( ball_Sx , ball_Sy , ball_Sz ) ;
Draw_ball() ;
glLoadIdentity () ;
glTranslatef ( cube_X , cube_Y , cube_Z ) ;
glRotatef ( cube_ang , cube_dirX , cube_dirY , cube_dirZ ) ;
glScalef ( cube_Sx , cube_Sy , cube_Sz ) ;
Draw_cube() ;
```

OpenGL Modelview Transformation (A Stack)

Modelview matrix has a stack (normally 32 levels, depending on the hardware)



Note:

1. OGL kernel apply the topmost matrix to the object coordinate (input of glVertex)
2. Beware: (i) glPushMatrix and glPopMatrix should be used in pair and (ii) stack underflow or overflow

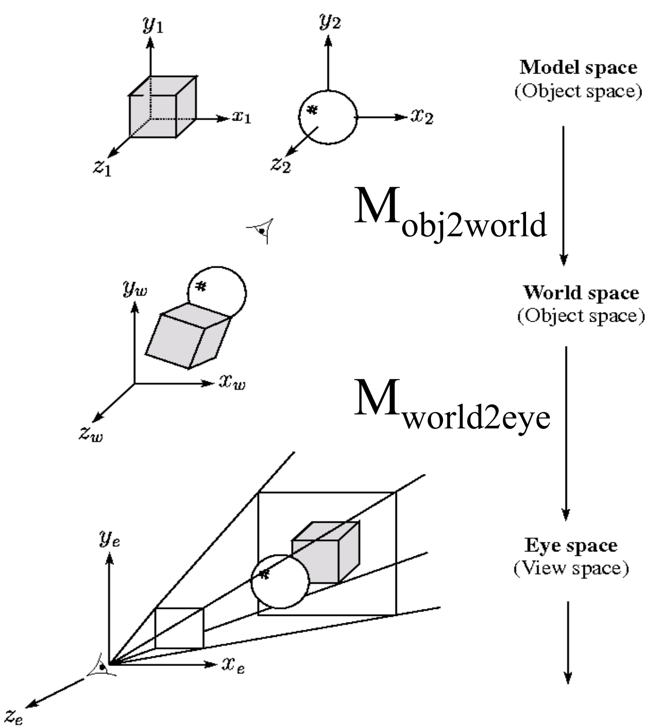
Example:

```
glMatrixMode ( GL_MODELVIEW ) ;
glLoadIdentity () ;
glPushMatrix() ;
glTranslatef ( ball_X , ball_Y , ball_Z ) ;
glRotatef ( ball_ang , ball_dirX , ball_dirY , ball_dirZ ) ;
glScalef ( ball_Sx , ball_Sy , ball_Sz ) ;
Draw_ball() ;
glPopMatrix() ;
glPushMatrix() ;
glTranslatef ( cube_X , cube_Y , cube_Z ) ;
glRotatef ( cube_ang , cube_dirX , cube_dirY , cube_dirZ ) ;
glScalef ( cube_Sx , cube_Sy , cube_Sz ) ;
Draw_cube() ;
glPopMatrix() ;
```

Topmost modelview matrix reverts to an Identity

OpenGL Modelview Transformation (A Stack)

Example:



Construct $M_{world2eye}$ at the beginning

```
glMatrixMode ( GL_MODELVIEW ) ;
glLoadIdentity () ;
glPushMatrix() ;
glTranslatef ( ball_X , ball_Y , ball_Z ) ;
glRotatef ( ball_ang , ball_dirX , ball_dirY , ball_dirZ ) ;
glScalef ( ball_Sx , ball_Sy , ball_Sz ) ;
Draw_ball() ;
glPopMatrix() ;
glPushMatrix() ;
glTranslatef ( cube_X , cube_Y , cube_Z ) ;
glRotatef ( cube_ang , cube_dirX , cube_dirY , cube_dirZ ) ;
glScalef ( cube_Sx , cube_Sy , cube_Sz ) ;
Draw_cube() ;
glPopMatrix() ;
```

Construct $M_{obj2world}$ (ball)

Construct $M_{obj2world}$ (cube)

$M_{world2eye} \times M_{obj2world} \times P_{obj}$

OpenGL Modelview Transformation (A Stack)

Example:

Sometimes, it is efficient to look at the code in this reverse order (because of right multiplication)

```
glMatrixMode ( GL_MODELVIEW ) ;
glLoadIdentity () ;
glPushMatrix() ;
    glTranslatef ( ball_X , ball_Y , ball_Z ) ;
    glRotatef ( ball_ang , ball_dirX , ball_dirY , ball_dirZ ) ;
    glScalef ( ball_Sx , ball_Sy , ball_Sz ) ;
    Draw_ball() ;
glPopMatrix() ;
glPushMatrix() ;
    glTranslatef ( cube_X , cube_Y , cube_Z ) ;
    glRotatef ( cube_ang , cube_dirX , cube_dirY , cube_dirZ ) ;
    glScalef ( cube_Sx , cube_Sy , cube_Sz ) ;
    Draw_cube() ;
glPopMatrix() ;
```

Indentation makes it clearer

OpenGL Modelview Transformation

Example: A robot arm

Consider this robot arm with 3 degrees of freedom:

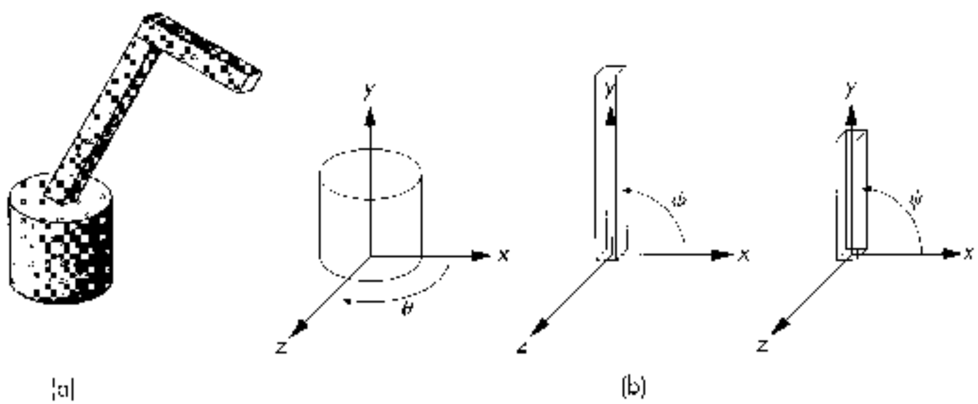
1. Base rotates about its vertical axis by θ
2. Lower arm rotates in its xy-plane by ϕ
3. Upper arm rotates in its xy-plane by ψ

Q: What is the tree structure of the robot?

Q: What matrix do we use to transform the base?

Q: What matrix for the lower arm?

Q: What matrix for the upper arm?



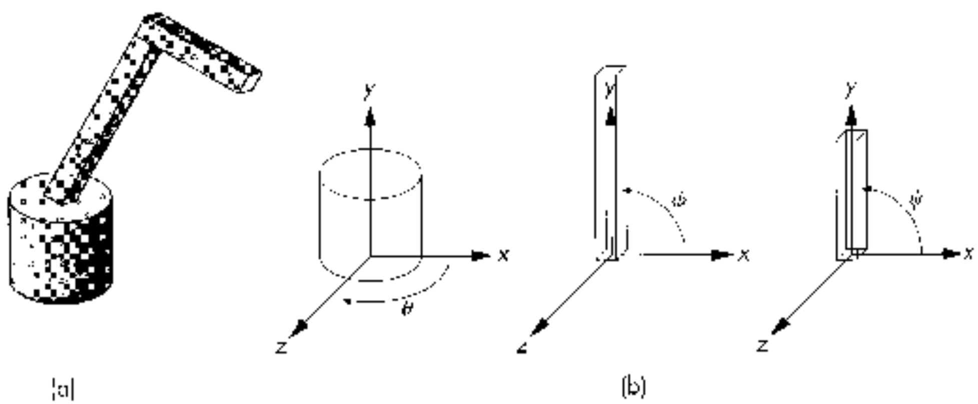
A robot arm (Angel, fig. 8.8)

OpenGL Modelview Transformation

Example: A robot arm

Consider this robot arm with 3 degrees of freedom:

1. Base rotates about its vertical axis by θ
2. Lower arm rotates in its xy-plane by ϕ
3. Upper arm rotates in its xy-plane by ψ



A robot arm (Angel, fig. 8.8)

```
display()
{
    glRotatef(theta, 0., 1., 0.);
    base();
    glTranslatef(0., h1, 0.);
    glRotatef(phi, 0., 0., 1.);
    lower_arm();
    glTranslatef(0., h2, 0.);
    glRotatef(psi, 0., 0., 1.);
    upper_arm();
}
```

Any error here?

OpenGL Modelview Transformation

Error 1

glPushMatrix

```
display()
{
    glRotatef(theta, 0., 1., 0.);
    base();
    glTranslatef(0., h1, 0.);
    glRotatef(phi, 0., 0., 1.);
    lower_arm();
    glTranslatef(0., h2, 0.);
    glRotatef(psi, 0., 0., 1.);
    upper_arm();
}
```

glPopMatrix

Note:

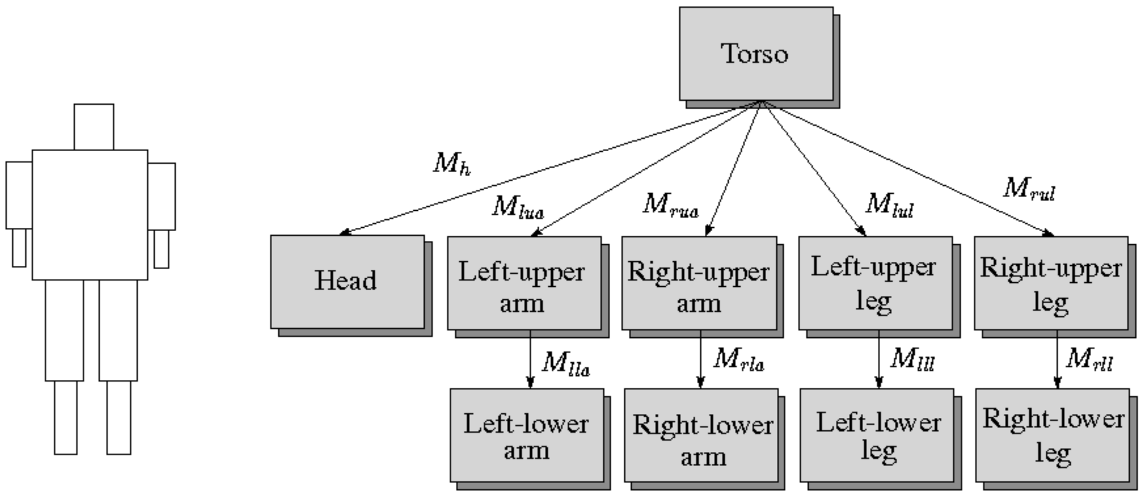
- A good practice: we should properly enclose *glTranslate*, *glRotate*, etc. with *glPushMatrix* and *glPopMatrix*
- Because OGL is a state machine, changes to the modelview matrix is permanent and if we call *display*() twice, we will end up with a strange transformation the next time.

Error 2

Since eye is at the origin, we cannot see the object

OpenGL Modelview Transformation

A more complex example: Human figure

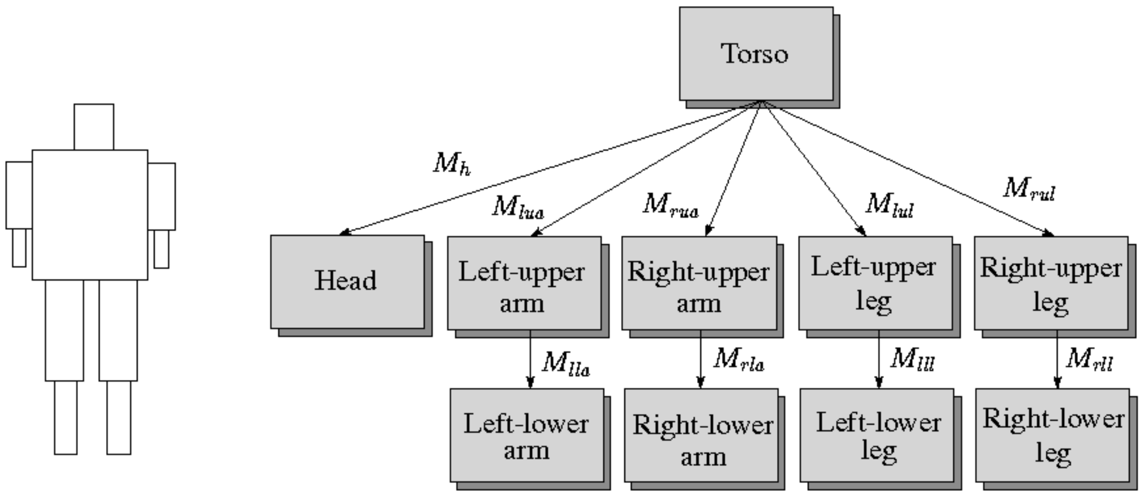


Human figure

Q: What's the most sensible way to traverse this tree?

OpenGL Modelview Transformation

A more complex example: Human figure



Human figure

Q: What's the most sensible way to traverse this tree?

```
figure()
{
   torso();
   glPushMatrix();
      glTranslate
      glRotate
      head();
   glPopMatrix();
   glPushMatrix();
      glTranslate
      glRotate
      left_upper_leg();
      glTranslate
      glRotate
      left_lower_leg();
   glPopMatrix();
   glPushMatrix();
      .
      .
}
```

Basic Scenegraph Concept

- Organize the whole model hierarchy (the geometry as well as the lighting, material, camera, etc.) as a tree structure
- Examples (API/Language): VRML, OpenGL Inventor, OpenGL Performer, OpenSG (open scenegraph), etc.

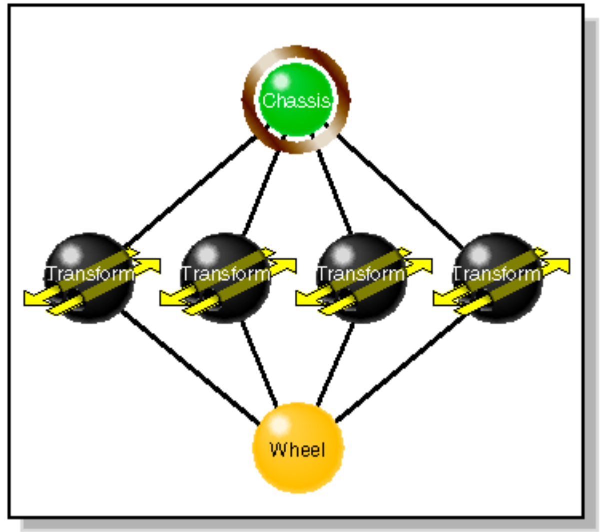
For example:

Two most general classifications of node functionality are

Group nodes - associate nodes into hierarchies.

Leaf nodes - contain all the descriptive data of objects in tl

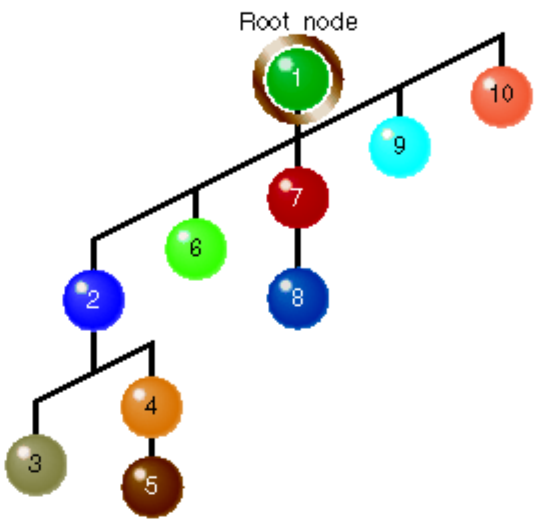
virtual world used to render them.



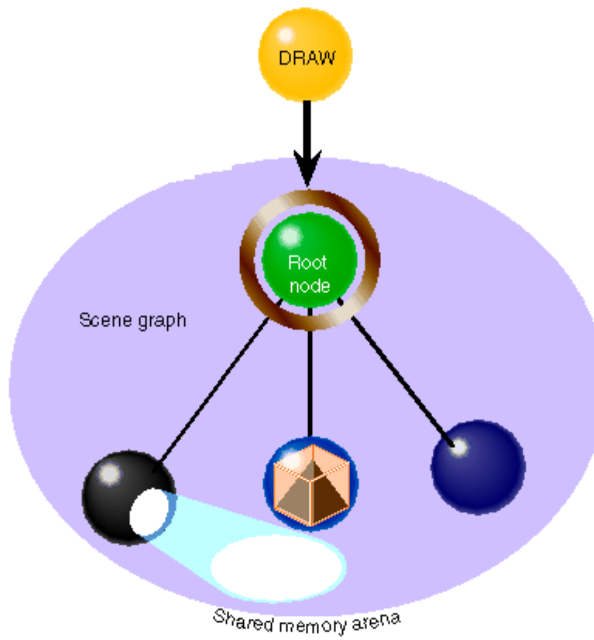
* Reuse the wheel geometry

Basic Scenegraph Concept

The renderer (VRML / OpenGL Performer / Inventor) traverses the tree: add light source, apply material attributes, render geometry etc.



Traversal order



[OGL Performer Programming Guide]