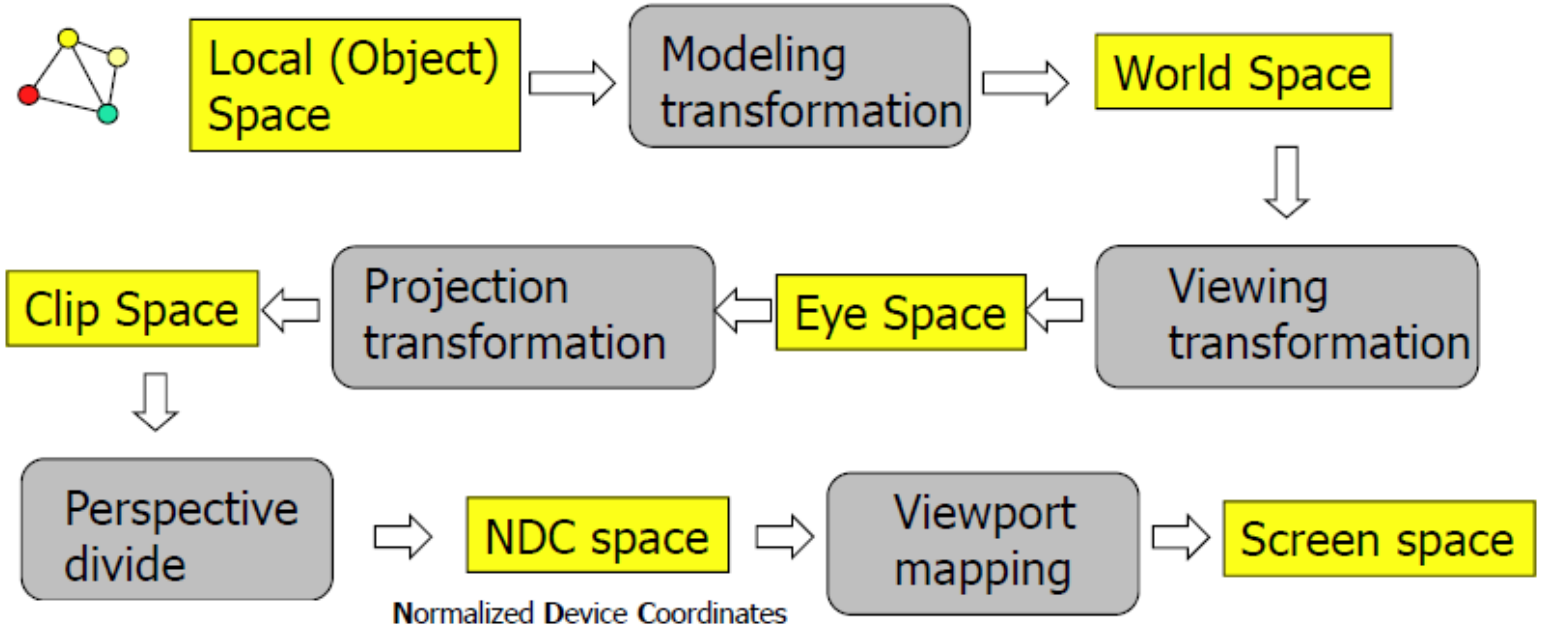
Computer Graphics
- Hierarchical Modeling

Junjie Cao @ DLUT

Spring 2017

http://jjcao.github.io/ComputerGraphics/

Transformation Pipeline - Recall



glMatrixMode(GL_MODELVIEW);
glMatrixMode(GL_PROJECTION);

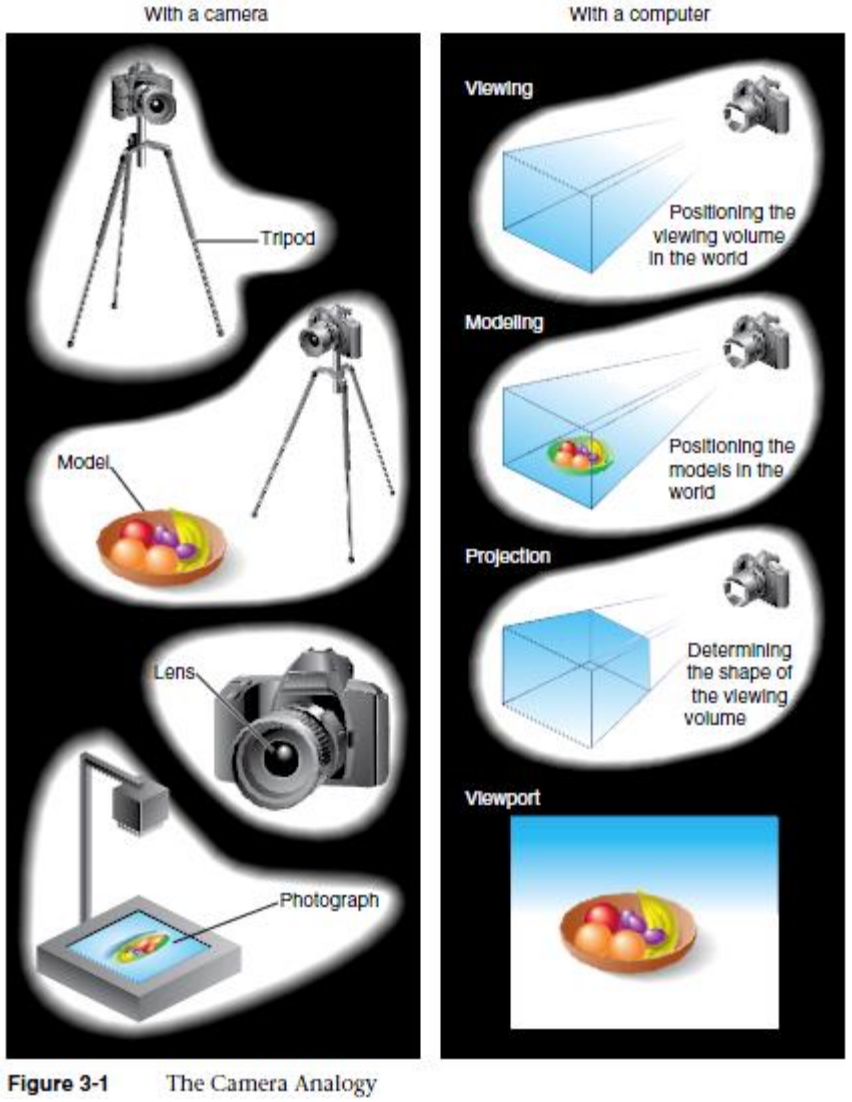


Figure 3-1 The Camera Analogy

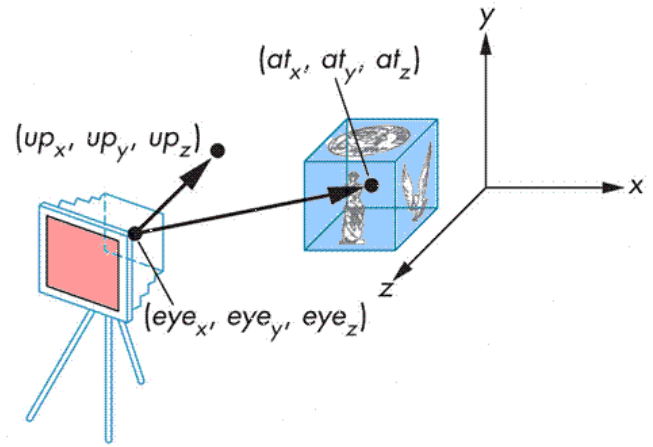
Recall

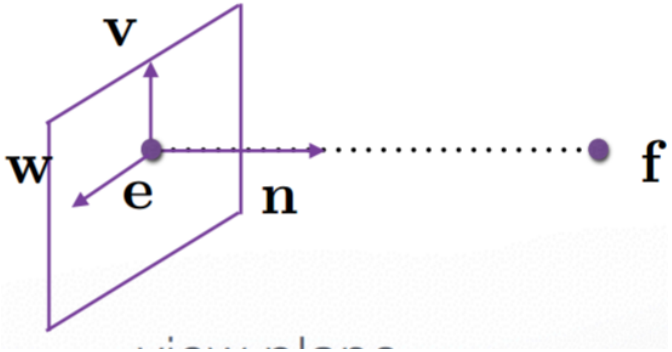
- Matrix representation
  - **column-major** matrices in GL
- Composing Transformations:
  - **postmultiplies** the *current matrix*
  - E.g., if current matrix is **C**, then **C=CS**
- Where are we drawing?

$$\begin{pmatrix} x_{eye} \\ y_{eye} \\ z_{eye} \\ w_{eye} \end{pmatrix} = M_{modelView} \cdot \begin{pmatrix} x_{obj} \\ y_{obj} \\ z_{obj} \\ w_{obj} \end{pmatrix} = M_{view} \cdot M_{model} \cdot \begin{pmatrix} x_{obj} \\ y_{obj} \\ z_{obj} \\ w_{obj} \end{pmatrix}$$

Recall – modeling & viewing transformation

- glMatrixMode(GL_MODELVIEW);
- gluLookAt(ex, ey, ez, fx, fy, fz, ux, uy, uz);
- Transform camera with W=TR;





- Calculate $\mathbf{V} = \mathbf{R}^{-1}\mathbf{T}^{-1}$

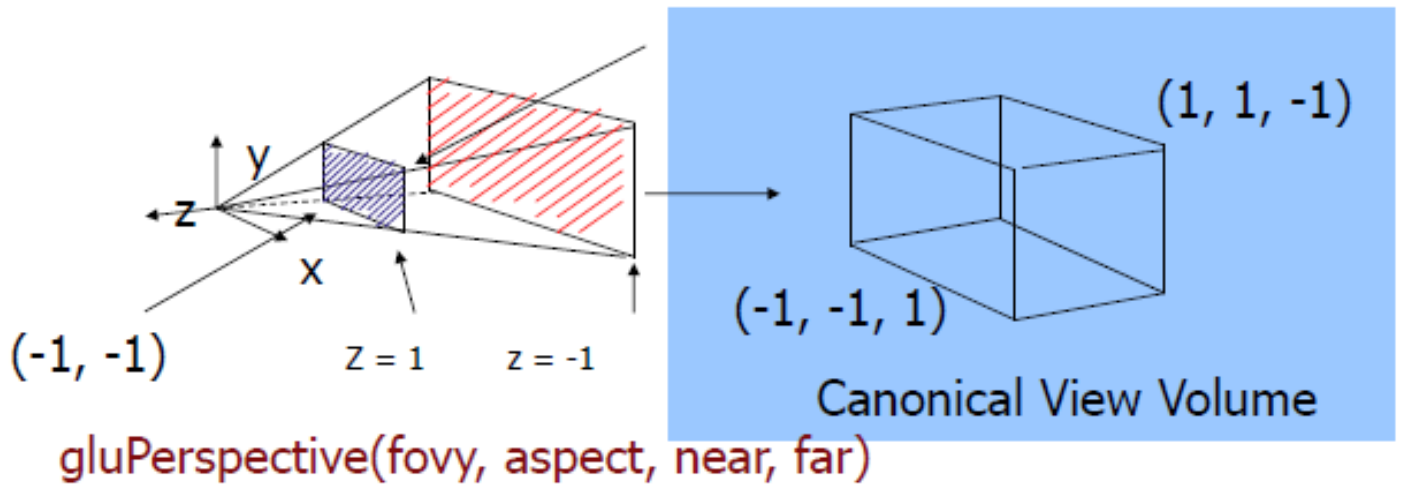
$$V = \begin{bmatrix} w_x & w_y & w_z & -w_x e_x - w_y e_y - w_z e_z \\ v_x & v_y & v_z & -v_x e_x - v_y e_y - v_z e_z \\ -n_x & -n_y & -n_z & n_x e_x + n_y e_y + n_z e_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Recall – perspective transformation

- **glMatrixMode(GL_PROJECTION);**
- viewing volume (frustum) to NDC
- Both clipping (frustum culling) and NDC transformations are integrated into **GL_PROJECTION** matrix.

$$\begin{bmatrix} x \\ y \\ z \\ \frac{z}{d} \end{bmatrix} \Rightarrow \begin{bmatrix} \frac{x}{z/d} \\ \frac{y}{z/d} \\ d \\ 1 \end{bmatrix}$$

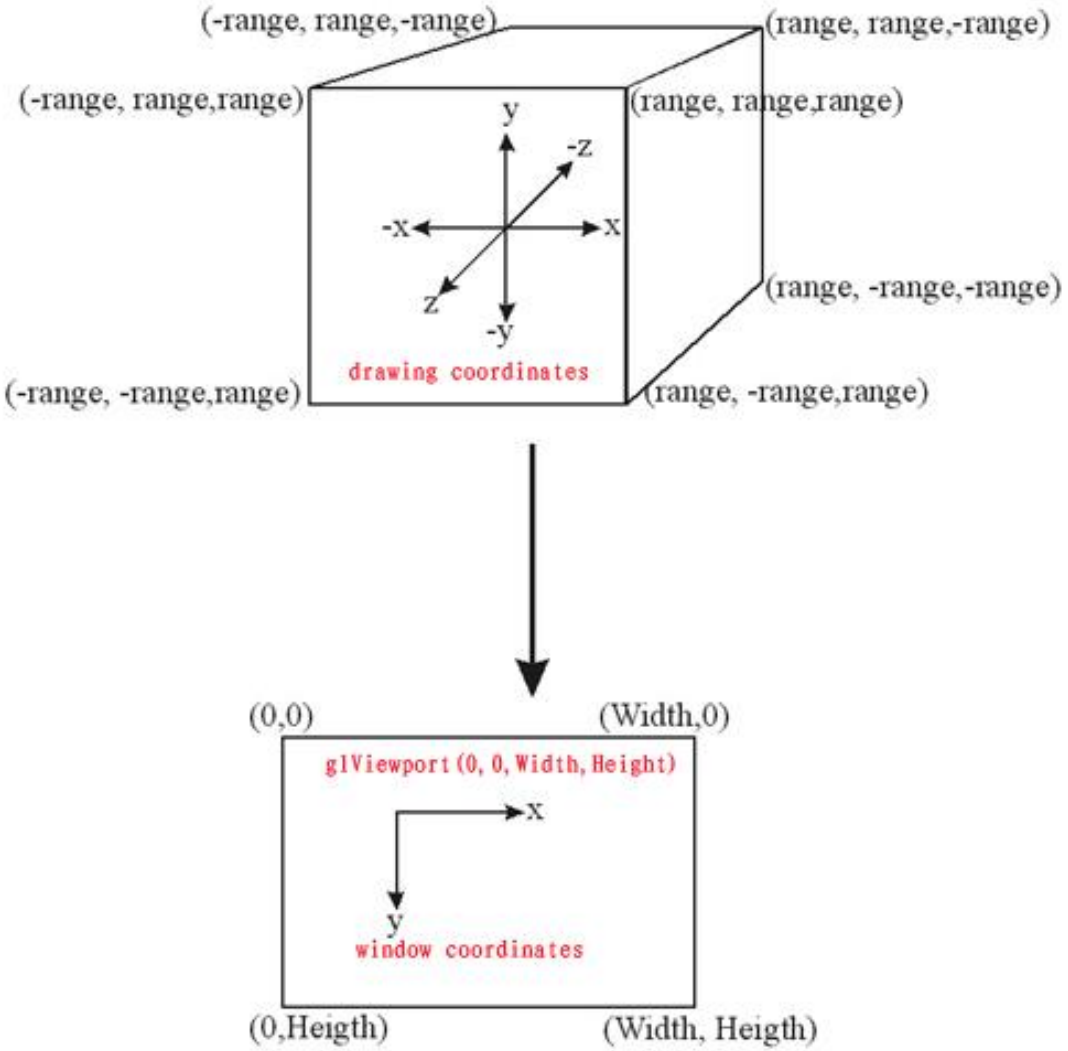
What we derived, but not really happened in GL



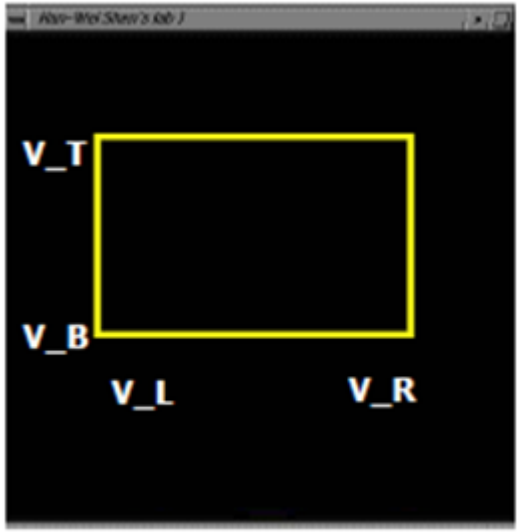
gluPerspective(fovy, aspect, near, far)

Recall – Viewpoint transformation

- In GL
  - NDC => Window Coordinates (Screen Coordinates)



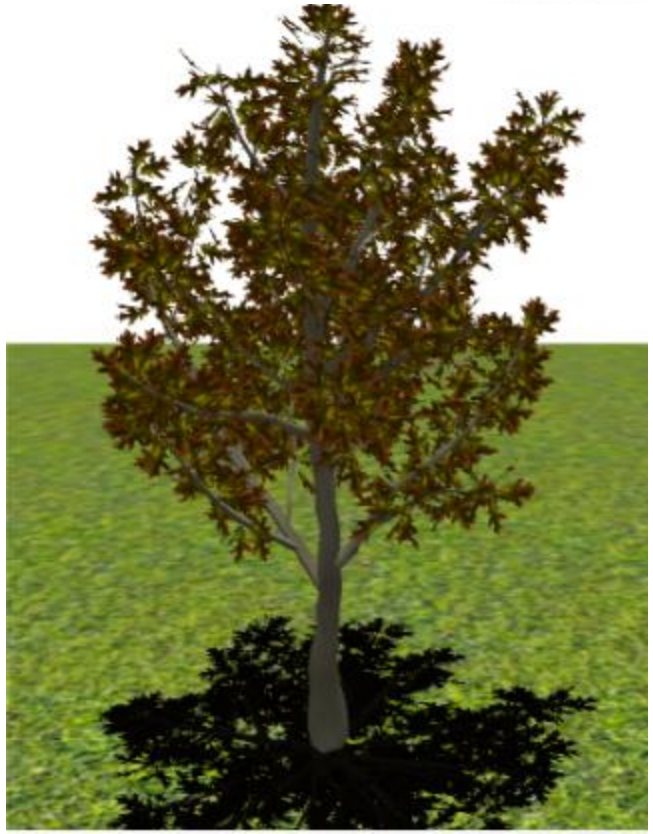
- Form user's perspective:
  - Near plane (projectived x & y) => screen



Hierarchical Models

- Many graphical objects are structured
- Structure often naturally hierarchical
  - Wheels of a car
  - Arms or legs of a figure
  - Chess pieces

- Exploit structure for
  - Efficient rendering
    - Example: tree leaves
  - Concise specification of model parameters
    - Example: joint angles

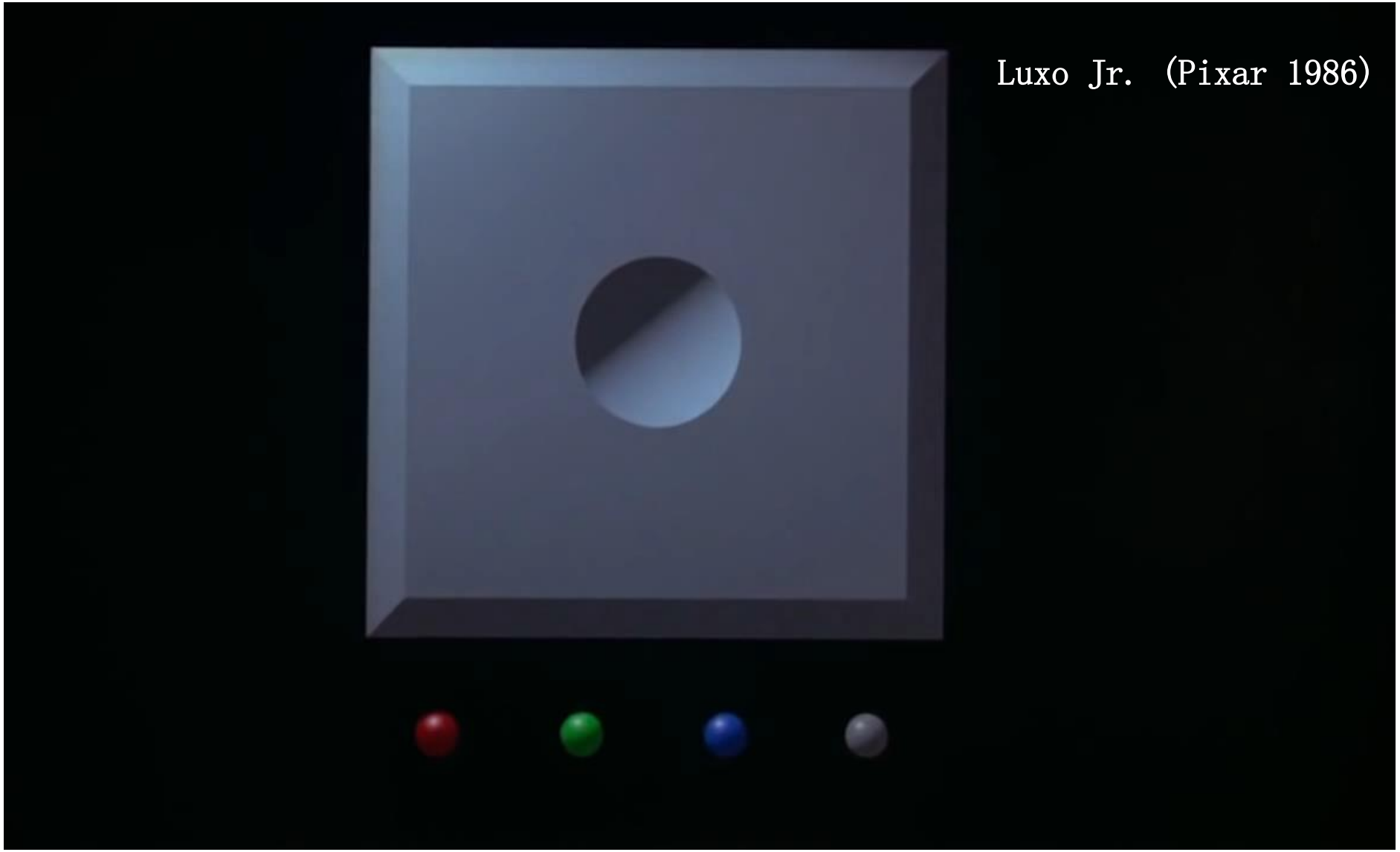


Instance Transformation

- Instances can be shared across space or time

- Write a function that renders the object in “standard” configuration
- Apply transformations to different instances
- Typical order: scaling, rotation, translation

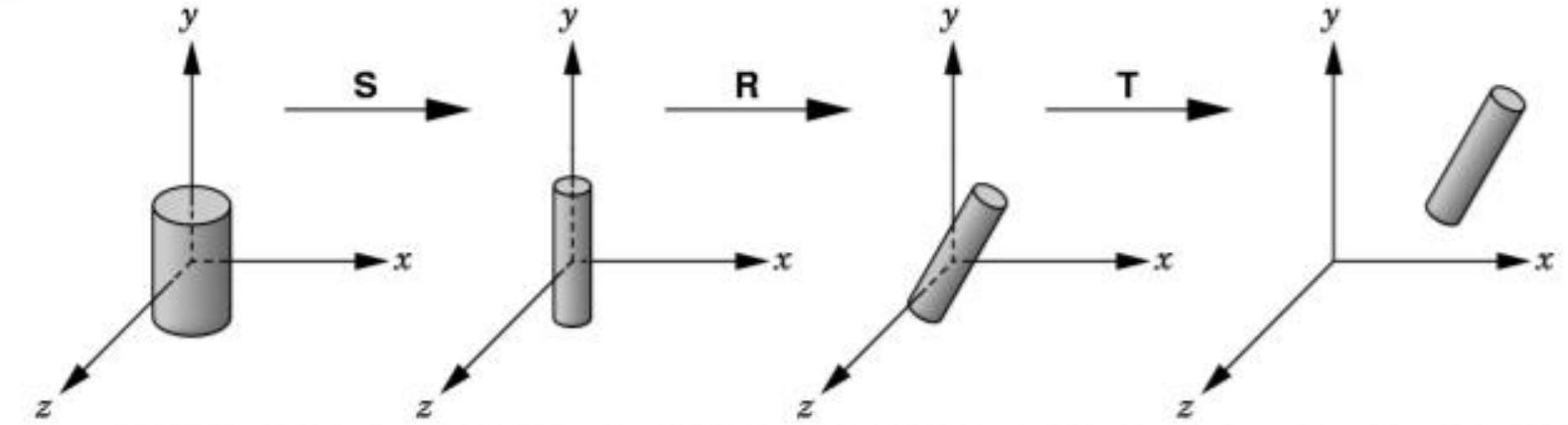
Animation: modeling motion



Pixar Shorts Collection Luxo Jr 1986

Sample Instance Transformation

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(...);
glRotatef(...);
glScalef(...);
gluCylinder(...);
```



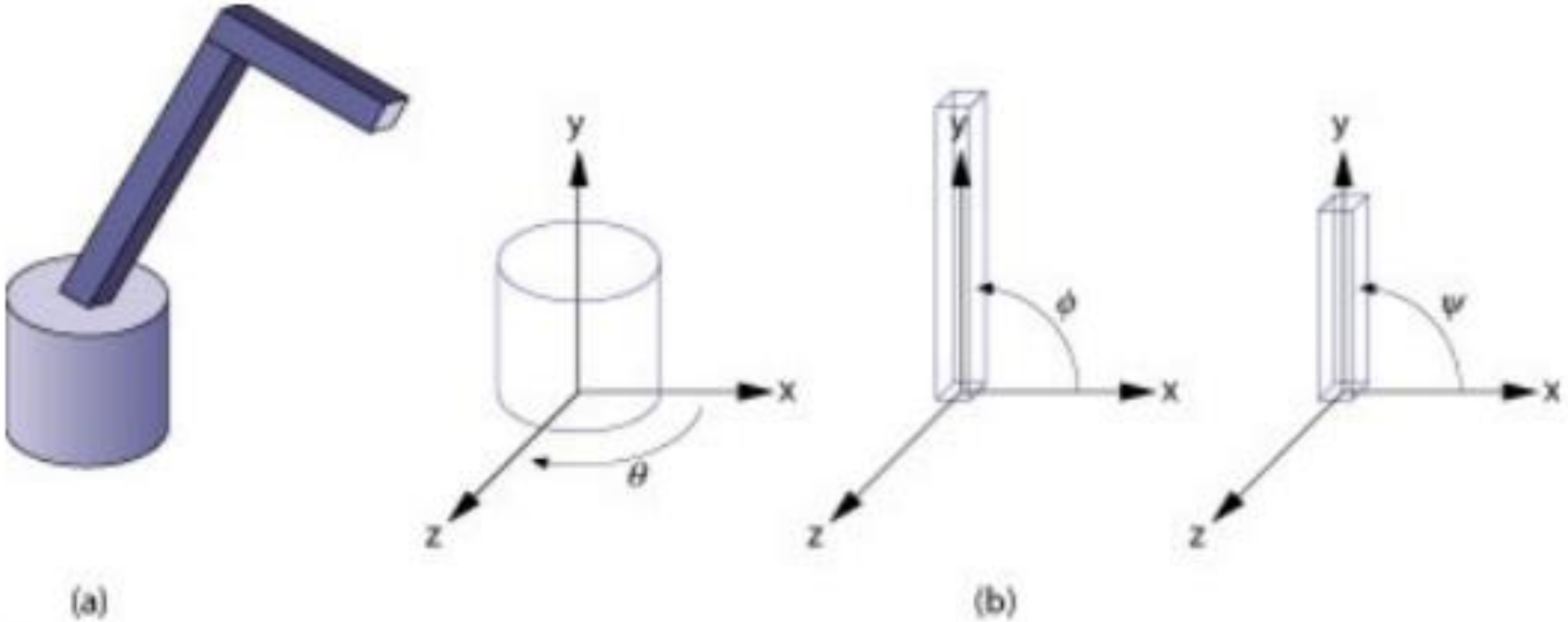
Display Lists

- Sharing display commands
- Display lists are stored on the GPU
- May contain drawing commands and transfns.
- Initialization:

```
GLuint torus = glGenLists(1);
glNewList(torus, GL_COMPILE);
  Torus(8, 25);
glEndList();
```

- Use: glCallList(torus);
- Can share both within each frame, and across different frames in time
- Can be hierarchical: a display list may call another

Drawing a Compound Object

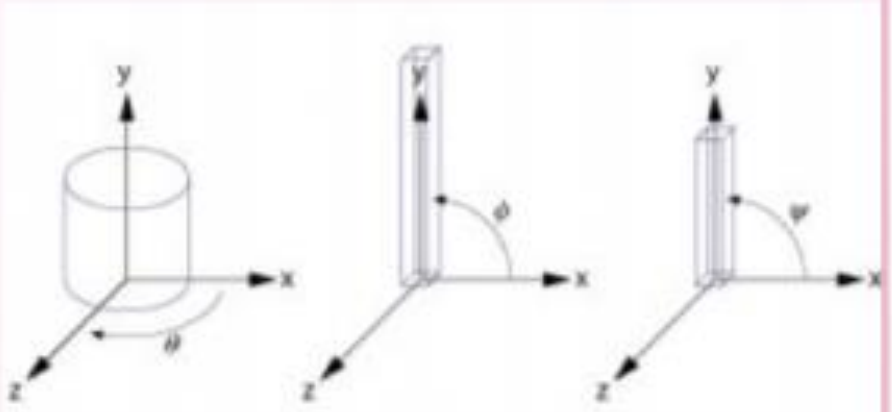


Base rotation θ, arm angle ϕ, joint angle ψ

Interleave Drawing & Transformation

h1 = height of base, h2 = length of lower arm

```
void drawRobot(GLfloat theta, GLfloat phi, GLfloat psi)
{
    glRotatef(theta, 0.0, 1.0, 0.0);
    drawBase();
    glTranslatef(0.0, h1, 0.0);
    glRotatef(phi, 0.0, 0.0, 1.0);
    drawLowerArm();
    glTranslatef(0.0, h2, 0.0);
    glRotatef(psi, 0.0, 0.0, 1.0);
    drawUpperArm();
}
```



Hierarchical Objects and Animation

- Drawing functions are time-invariant
  - drawBase(); drawLowerArm(); drawUpperArm();
- Can be easily stored in display list
- Change parameters of model with time
- Redraw when idle callback is invoked

A Bug to Watch

- GLfloat theta = 0.0; ...; /* update in idle callback */
- GLfloat phi = 0.0; ...; /* update in idle callback */
- GLuint arm = glGenLists(1);

- /* in init function */
- glNewList(arm, GL_COMPILE);
- glRotatef(theta, 0.0, 1.0, 0.0);
- drawBase();
- ...
- drawUpperArm();
- glEndList();

What is wrong?

- /* in display callback */
- glCallList(arm);

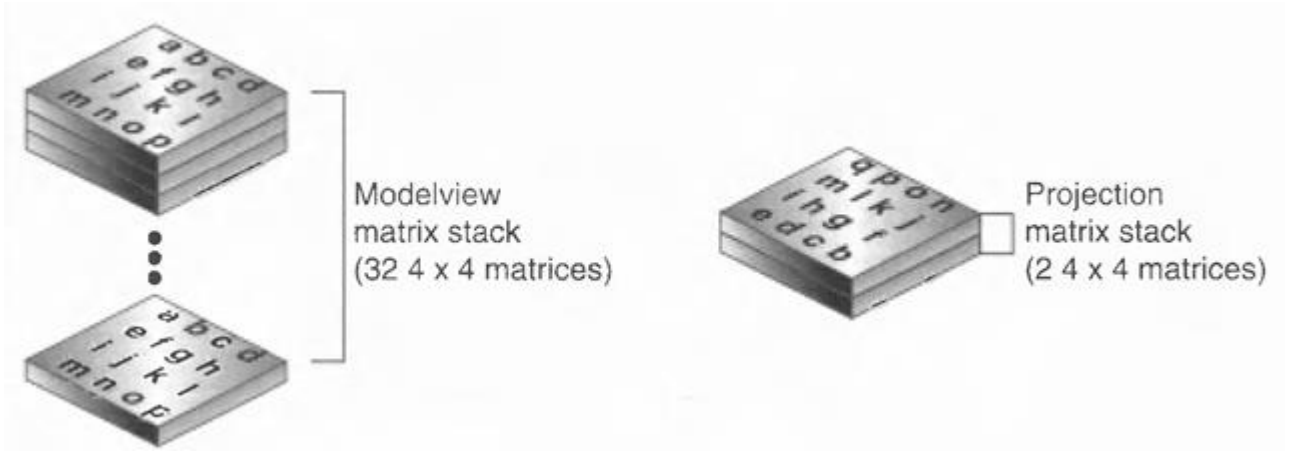
Example: modeling + viewing transformation

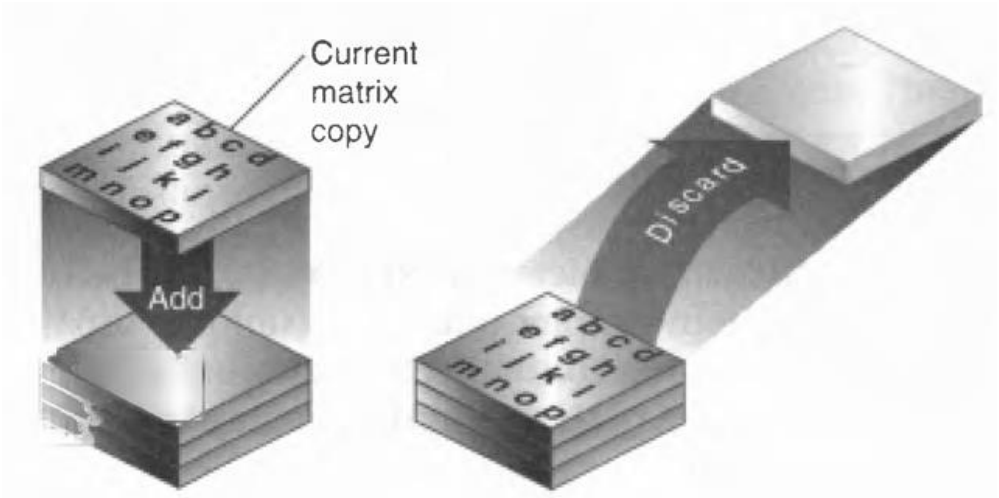
- With all this, we can give an outline for a typical display routine for drawing an image of a 3D scene with OpenGL 1.1:

```
// possibly set clear color here, if not set elsewhere
glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );

// possibly set up the projection here, if not done elsewhere
glMatrixMode( GL_MODELVIEW ); glLoadIdentity();
gluLookAt( eyeX,eyeY,eyeZ, refX,refY,refZ, upX,upY,upZ ); // Viewing transform

glPushMatrix();
. .. // apply modeling transform and draw an object
glPopMatrix();

glPushMatrix();
. .. // apply another modeling transform and draw another object
glPopMatrix()
…
```

pushes the current matrix stack down by one, duplicating the current matrix:
glPushMatrix()
pops the current matrix stack, replacing the current matrix with the one below it on the stack:
glPopMatrix()

Manipulating the Matrix Stacks





void glPushMatrix(void);

Pushes all matrices in the current stack down one level. The current stack is determined by **glMatrixMode()**. The topmost matrix is copied, so its contents are duplicated in both the top and second-from-the-top matrix. If too many matrices are pushed, an error is generated.

void glPopMatrix(void);

Pops the top matrix off the stack, destroying the contents of the popped matrix. What was the second-from-the-top matrix becomes the top matrix. The current stack is determined by **glMatrixMode()**. If the stack contains a single matrix, calling **glPopMatrix()** generates an error.

Assuming you are drawing a car with four wheels:
Draw the car body.

- Remember where you are, and translate to the right front wheel.
- Draw the wheel and throw away the last translation so your current position is back at the origin of the car body.
- Remember where you are, and translate to the left front wheel…

glPushMatrix() means “remember where you are” and **glPopMatrix()** means “go back to where you were.”

Current matrix and matrix stack

- glLoadIdentity — replace the current matrix with the identity matrix

  It is semantically equivalent to calling glLoadMatrix with the identity matrix

- glLoadMatrix — replace the current matrix with the specified matrix
- glMultMatrix
  - The current matrix is postmultiplied by the matrix

Column major

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

```
glLoadIdentity();
glMultMatrixf (M1);
glMultMatrixf (M2);
```

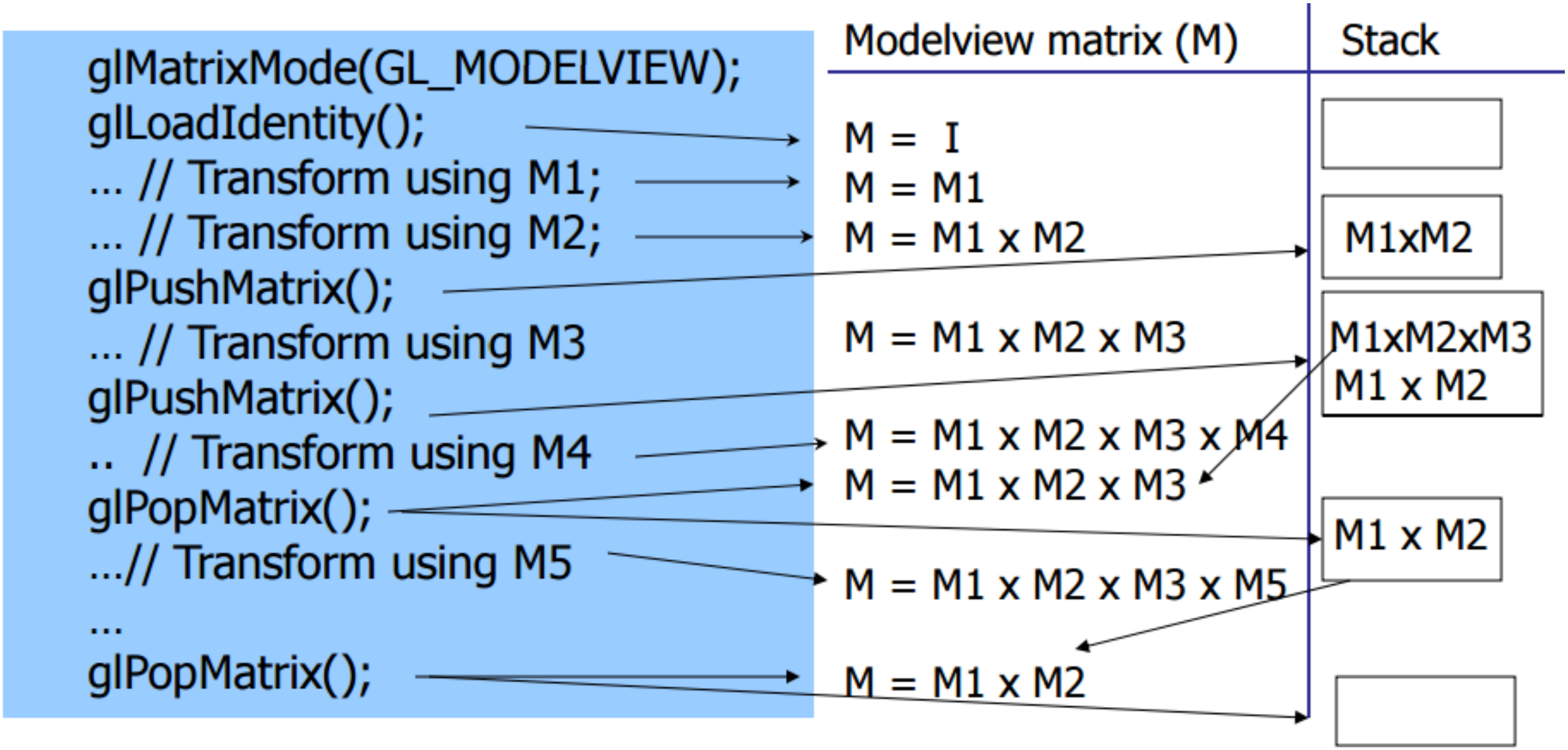
M = M1·M2

- matrix stack
  - glPushMatrix pushes the current matrix stack down by one, duplicating the current matrix. That is, after a glPushMatrix call, the matrix on top of the stack is identical to the one below it.
  - glPopMatrix pops the current matrix stack, replacing the current matrix with the one below it on the stack.
  - Initially, each of the stacks contains one matrix, an identity matrix.

- Stack query

```
float mat[16];// get the modelview matrix
glGetFloatv(GL_MODELVIEW_MATRIX, mat);

Int depth;

glGetIntegerv( GL_MODELVIEW_STACK_DEPTH, &depth);
```

Push and Pop Matrix Stack



Assessment of Interleaving

- Compact
- Correct “by construction”
- Efficient
- Inefficient alternative:

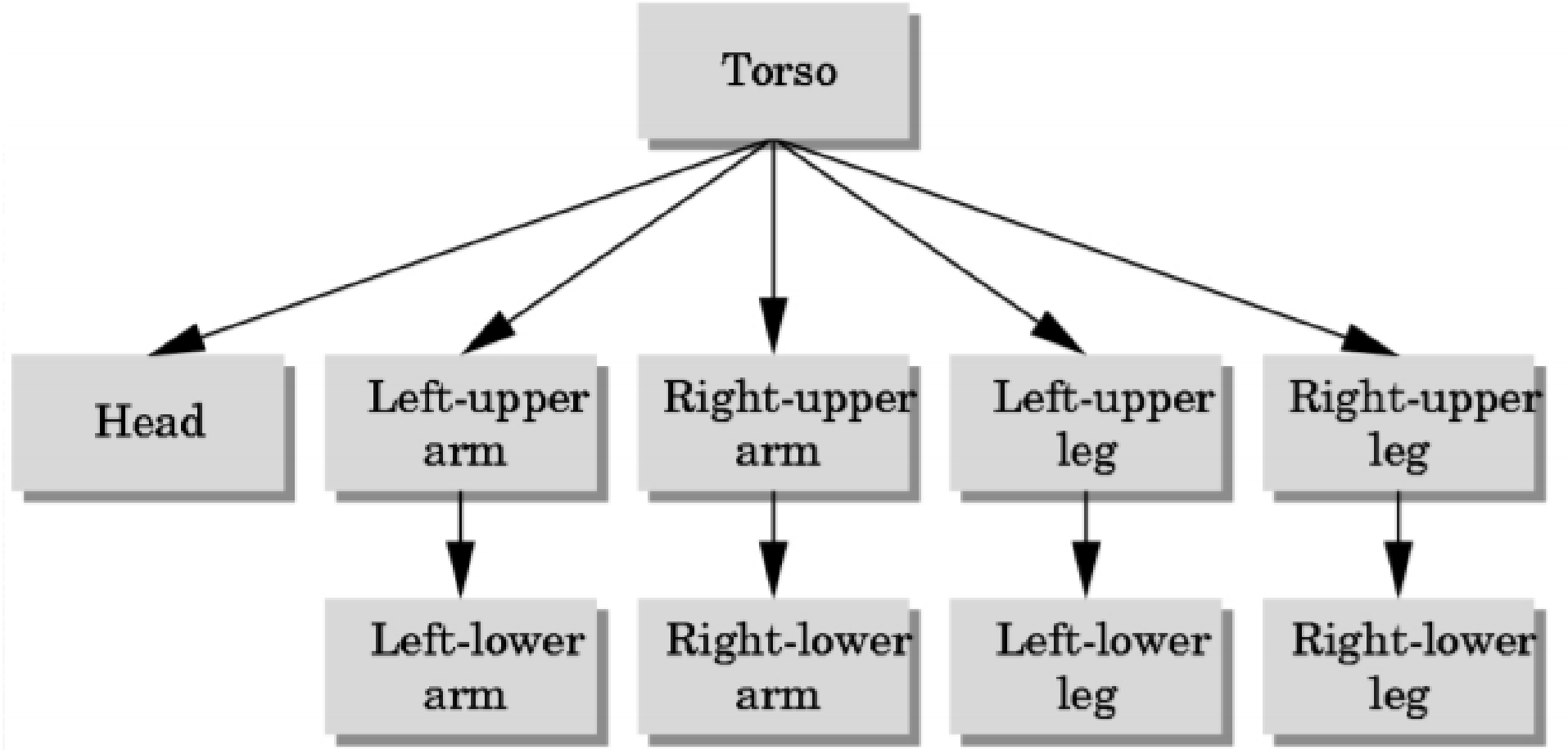
```
glPushMatrix();
glRotatef(theta, ...);
drawBase();
glPopMatrix();
```

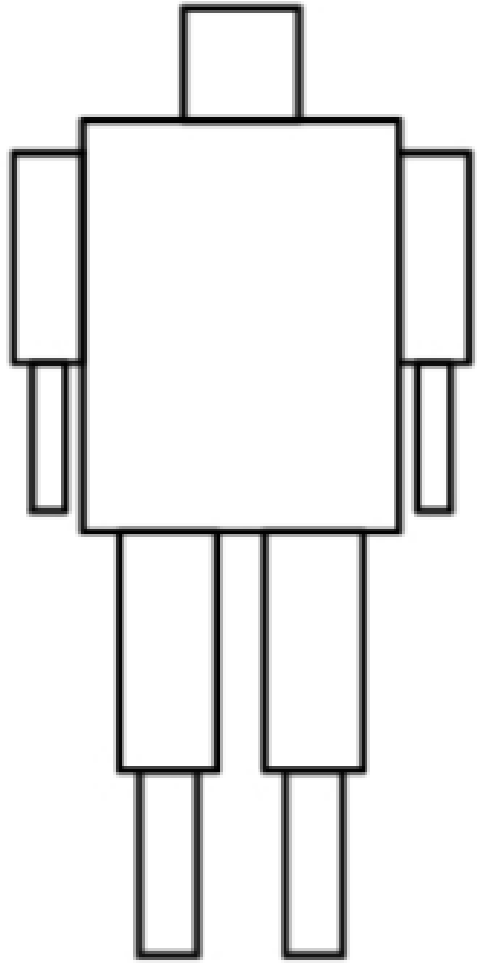
```
glPushMatrix();
glRotatef(theta, ...);
glTranslatef(...);
glRotatef(phi, ...);
drawLowerArm();
glPopMatrix();
```

```
...etc...
```

More Complex Objects

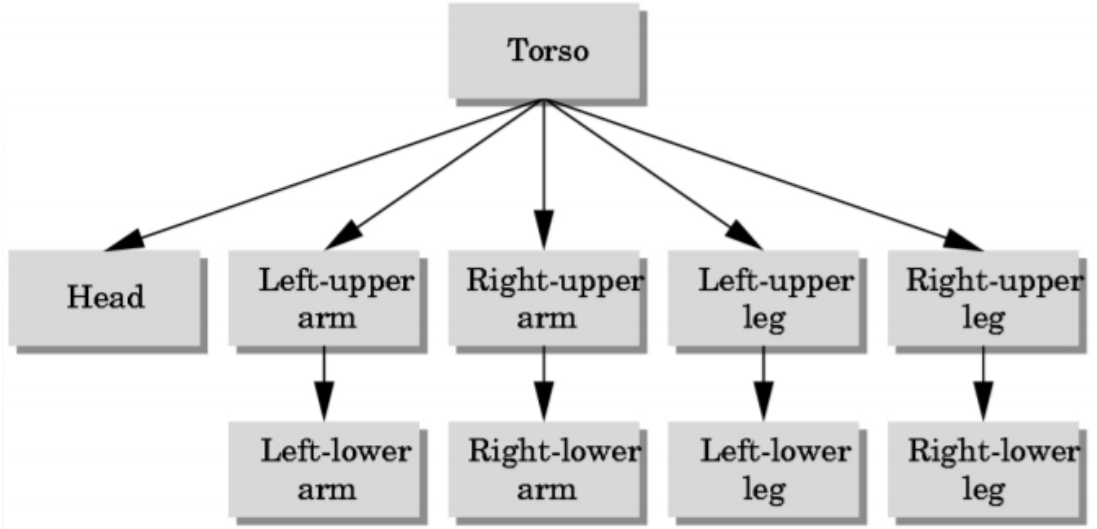
- **Tree rather than linear structure**
- **Interleave** along each branch
- Use push and pop to save state





Hierarchical Tree Traversal

- Order not necessarily fixed
- Example:



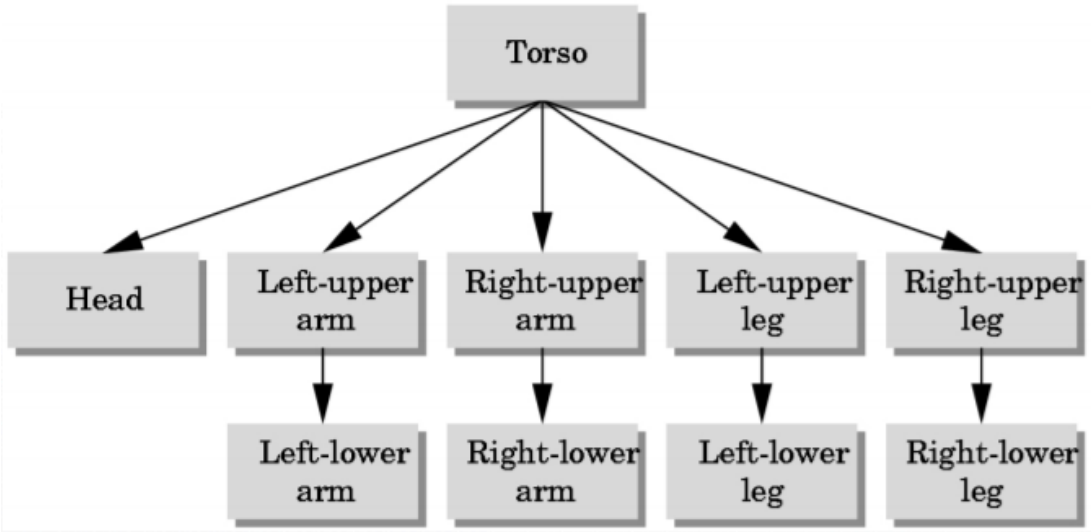

```
void drawFigure()
{
  glPushMatrix();  /* save */
  drawTorso();
  glTranslatef(...); /* move head */
  glRotatef(...);     /* rotate head */
  drawHead();
  glPopMatrix();   /* restore */
```

```
  glPushMatrix();
  glTranslatef(...);
  glRotatef(...);
  drawLeftUpperArm();
  glTranslatef(...)
  glRotatef(...)
  drawLeftLowerArm();
  glPopMatrix();
  ... }
```

Using Tree Data Structures

- Can make tree form explicit in data structure

```
typedef struct treenode
{
  GLfloat m[16];
  void (*f) ( );
  struct treenode *sibling;
  struct treenode *child;
} treenode;
```



Initializing Tree Data Structure

- Initializing transformation matrix for node

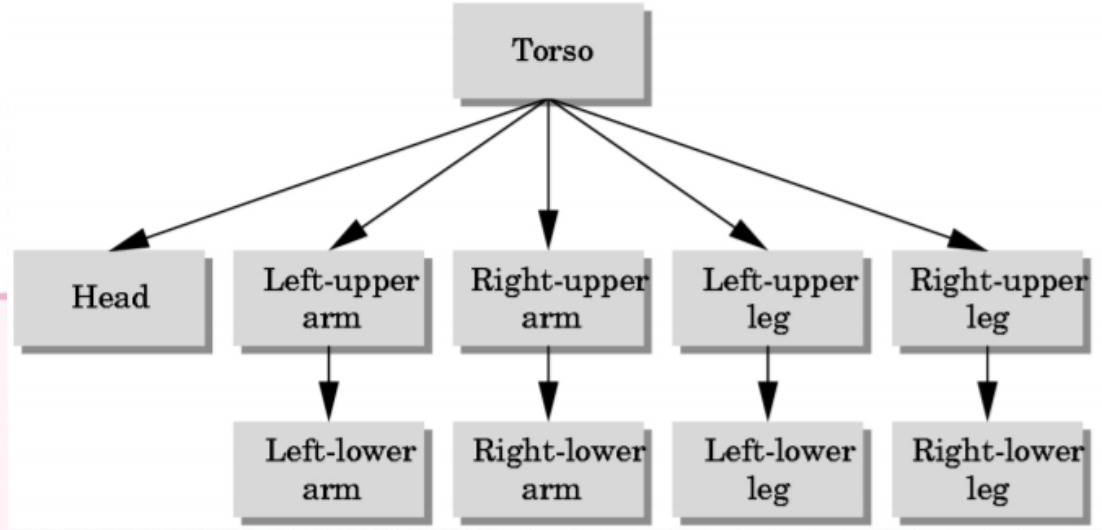
```
treenode torso, head, ...;
/* in init function */
glLoadIdentity();
glRotatef(...);
glGetFloatv(GL_MODELVIEW_MATRIX, torso.m);
```

- Initializing pointers

```
torso.f = drawTorso;
torso.sibling = NULL;
torso.child = &head;
```

Generic Traversal

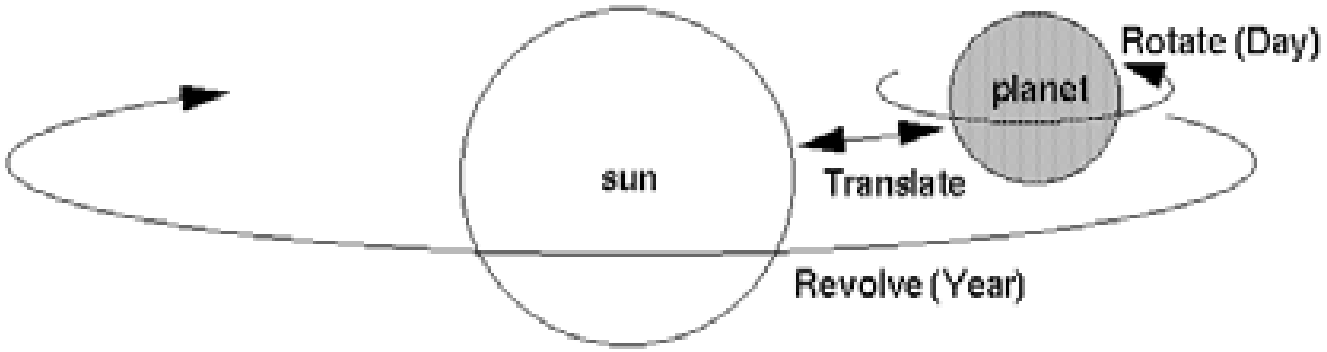
- Recursive definition




```
void traverse (treenode *root)
{
  if (root == NULL) return;
  glPushMatrix();
  glMultMatrixf(root->m);
  root->f();
  if (root->child != NULL) traverse(root->child);
  glPopMatrix();
  if (root->sibling != NULL) traverse(root->sibling);
}
```

See demo code

Assignment 1： Building the solar system



- You will need to write from scratch a complete OpenGL programme that renders a Sun with an orbiting planet and a moon orbiting the planet

Assignment Basic Implementation

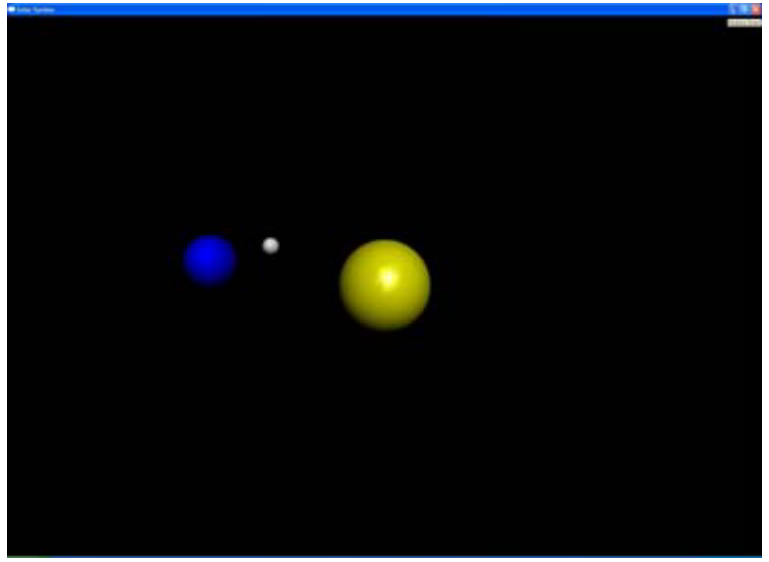
The basic implementation includes the following:

- Add a sphere representing the sun planet
- Make the sun planet to rotate around itself
- Add another sphere representing the earth
- Make the earth planet to rotate around itself
- Make the earth planet to rotate around sun
- Add another sphere representing the moon
- Make the moon planet to rotate around itself
- Make the moon planet to rotate around the earth
- Control the camera position using the keyboard
- Control the camera position using widget menus
- Add a light source
- Add shading to the planets
- Add material properties to the planets (you have to check this out
- yourselves)

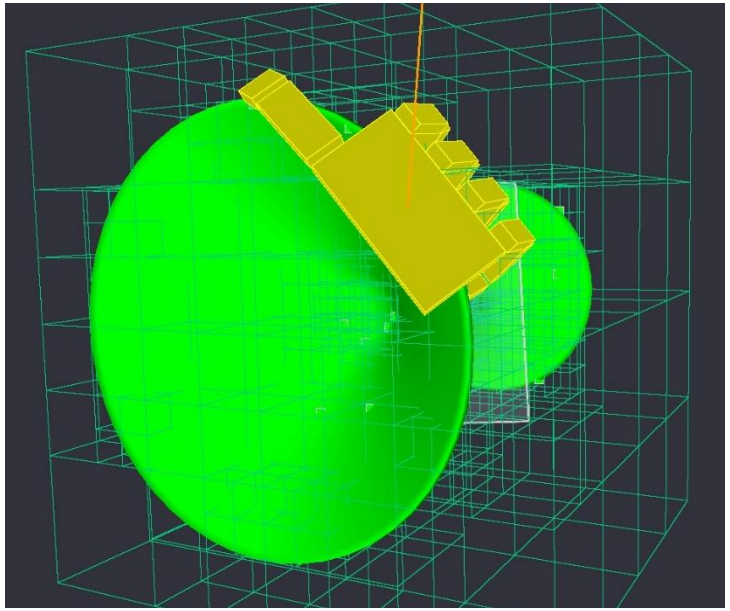
Assignment Advanced Implementation

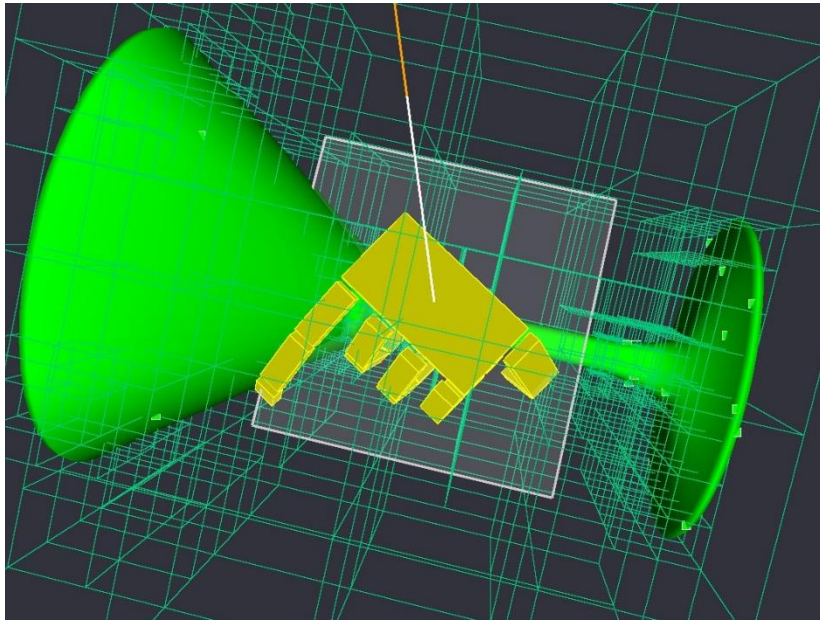
Recommended Implementation

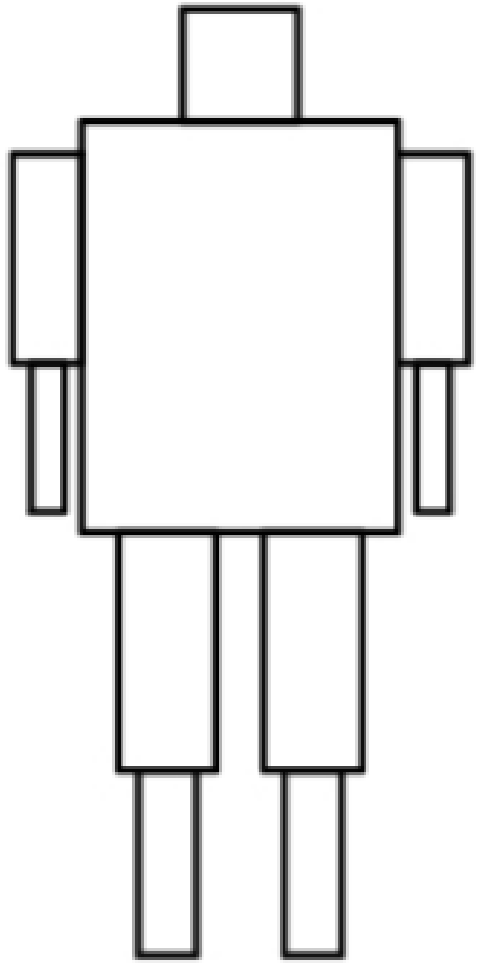
- Add more planets, e.g. if you are quick enough you could create the complete solar system
- Add more light sources (OpenGL supports up to 8 lights)
- Have planets counter rotating
- Add more moons to planets
- Add stars to the planetary system
- Add spaceships



Assignment 2: Building a robot arm or a robot







Thanks