Chapter 8: Modeling and Hierarchy

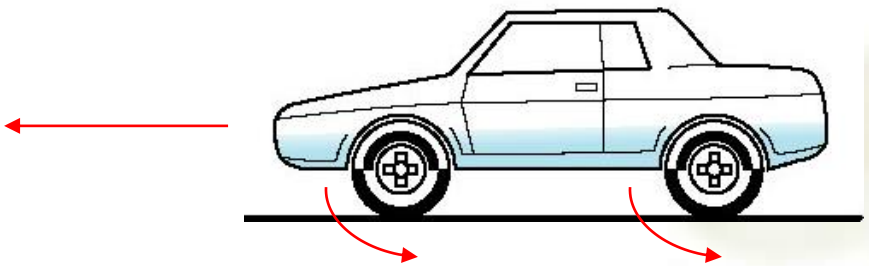
Scene Graph

Key Contents

1. Example I : Car Model
2. Example II : Robot Arm
3. Example III : Humanoid Figure
4. OpenGL and Object-oriented
5. Scene Graph
6. Events in OpenGL

1. Example I : Car Model

- Consider model of car
  - Chassis + 4 identical wheels



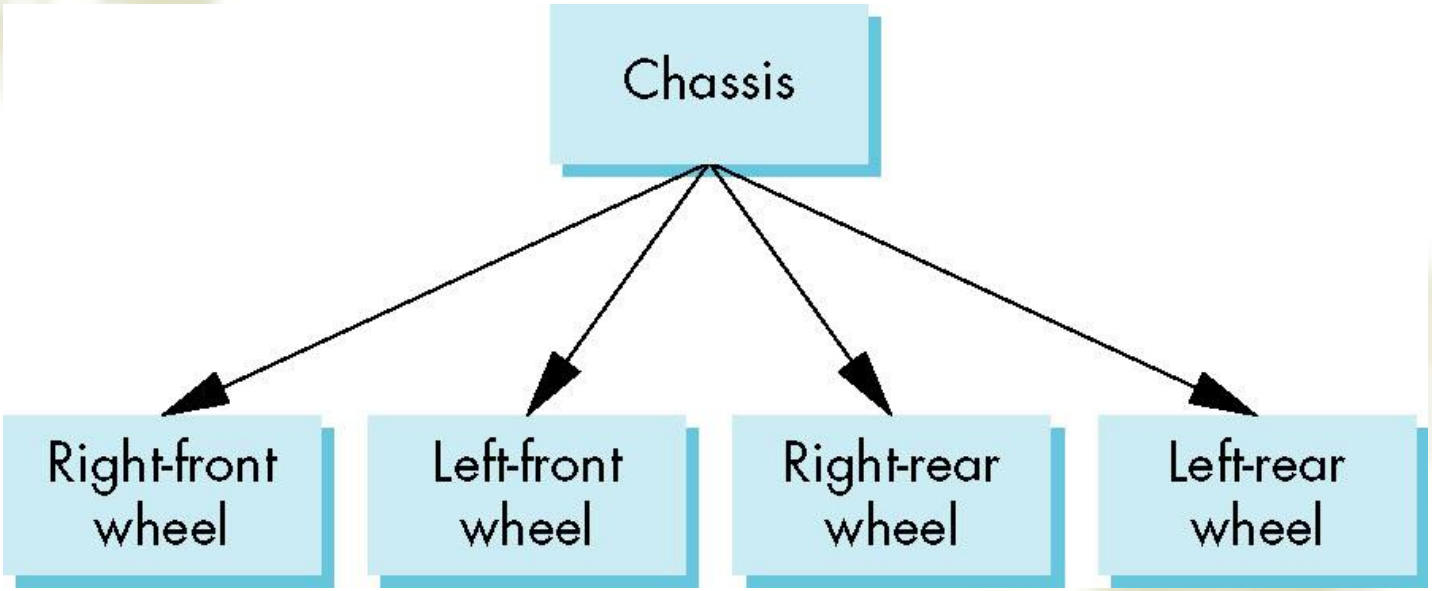
- Car is moving forward and wheels rotating
- Rate of forward motion determined by rotational speed of wheels: $d=2\pi r$

Structure Through Function Calls

```
car(speed)
{
      chassis()
      wheel(right_front);
      wheel(left_front);
      wheel(right_rear);
      wheel(left_rear);
}
```

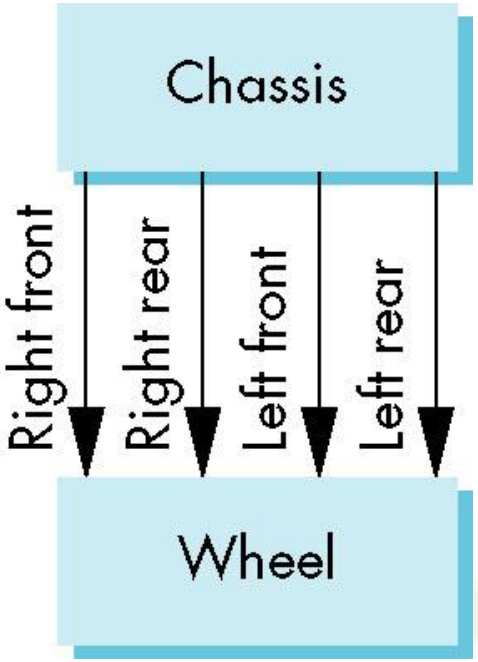
- draw four wheels
- Fails to show relationships well

Tree Model of Car



DAG Model

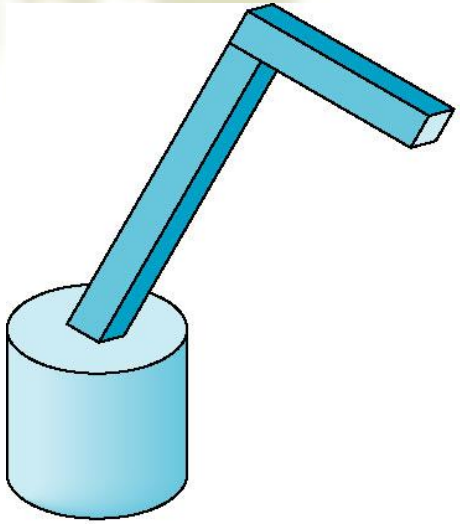
- If we use the fact that all the wheels are identical, we get a *directed acyclic graph*
  - Not much different than dealing with a tree



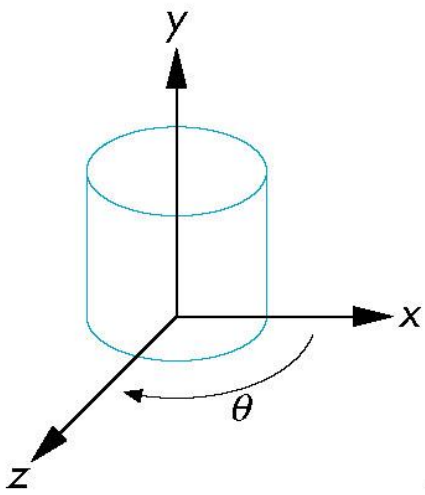
Modeling with Trees

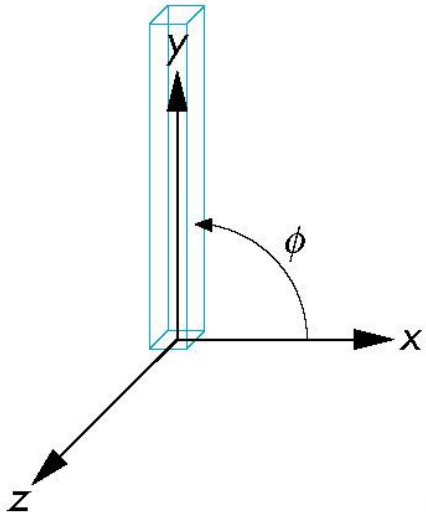
- Must decide what information to place in nodes and what to put in edges
- Nodes
  - What to draw
  - Pointers to children
- Edges
  - May have information on incremental changes to transformation matrices (can also store in nodes)

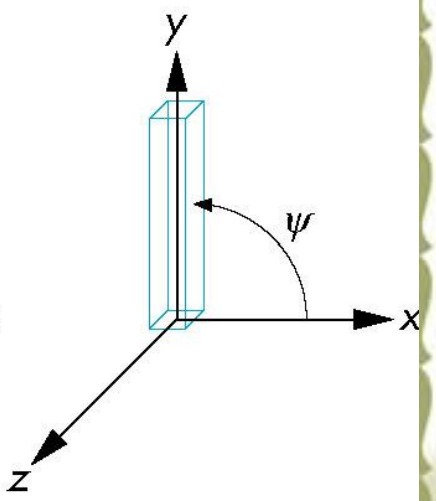
2. Example II : Robot Arm



robot arm



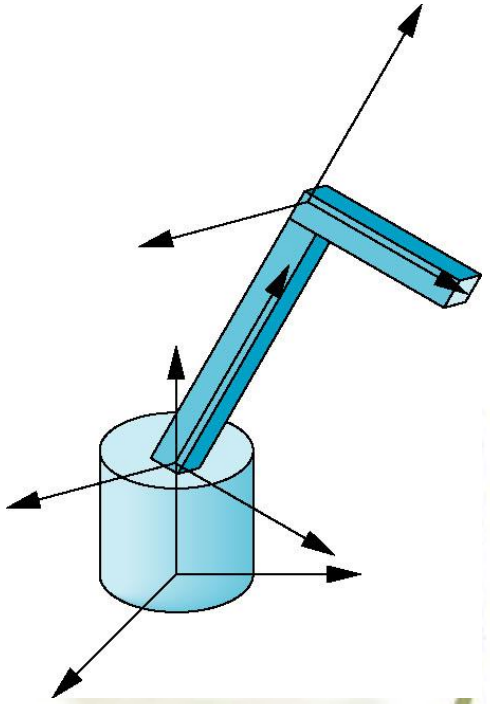




parts in their own
coodinate systems

also call Articulated Models

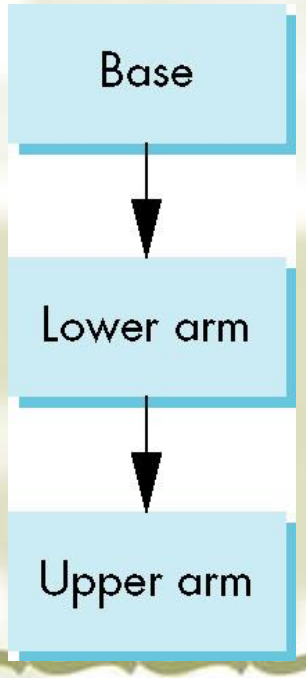
Required Matrices



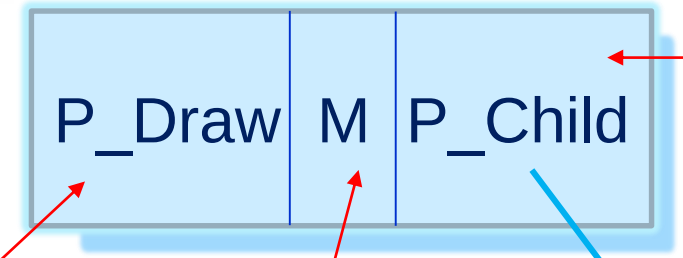
- Rotation of base: $\mathbf{R}_b$
  - Apply $\mathbf{M} = \mathbf{R}_b$ to base
- Translate lower arm relative to the base: $\mathbf{T}_{lu}$
- Rotate lower arm relative to the base: $\mathbf{R}_{lu}$
  - Apply $\mathbf{M} = \mathbf{R}_b \mathbf{T}_{lu} \mathbf{R}_{lu}$ to lower arm
- Translate upper arm relative to lower arm: $\mathbf{T}_{uu}$
- Rotate upper arm relative to the lower arm: $\mathbf{R}_{uu}$
  - Apply $\mathbf{M} = \mathbf{R}_b \mathbf{T}_{lu} \mathbf{R}_{lu} \mathbf{T}_{uu} \mathbf{R}_{uu}$ to upper arm

Tree Model of Robot

- Note code shows relationships between parts of model
  - Can change “look” of parts easily without altering relationships
- Simple example of tree model
- Want a general node structure for nodes



Possible Node Structure



linked list of pointers to children

Code for drawing part or pointer to drawing function; including material, texture

matrix relating node to parent

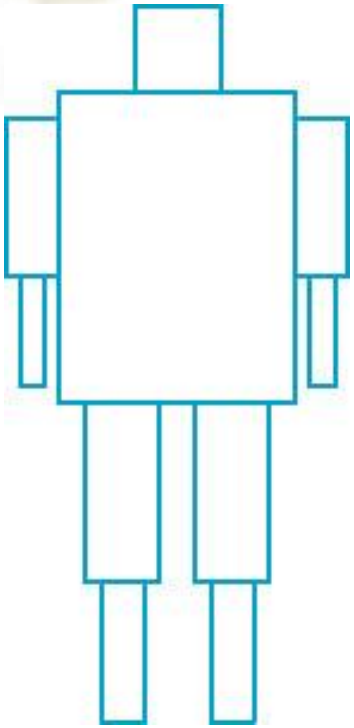
OpenGL Code for Robot

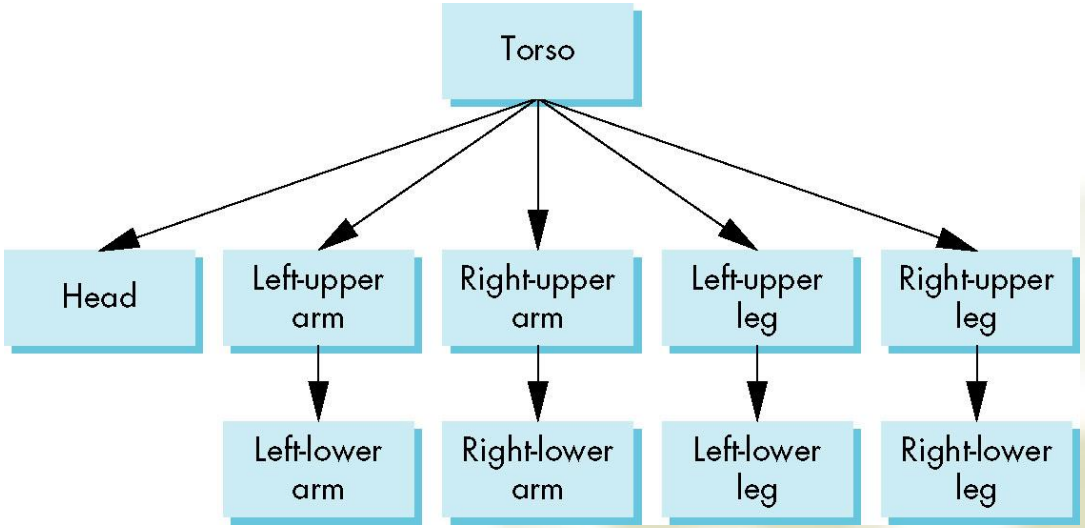
```
mat4 ctm;
robot_arm()
{
  ctm = RotateY(theta);
  base();
  ctm *= Translate(0.0, h1, 0.0) * RotateZ(phi);
  lower_arm();
  ctm *= Translate(0.0, h2, 0.0) * RotateZ(psi);
  upper_arm();
}
```

Generalizations

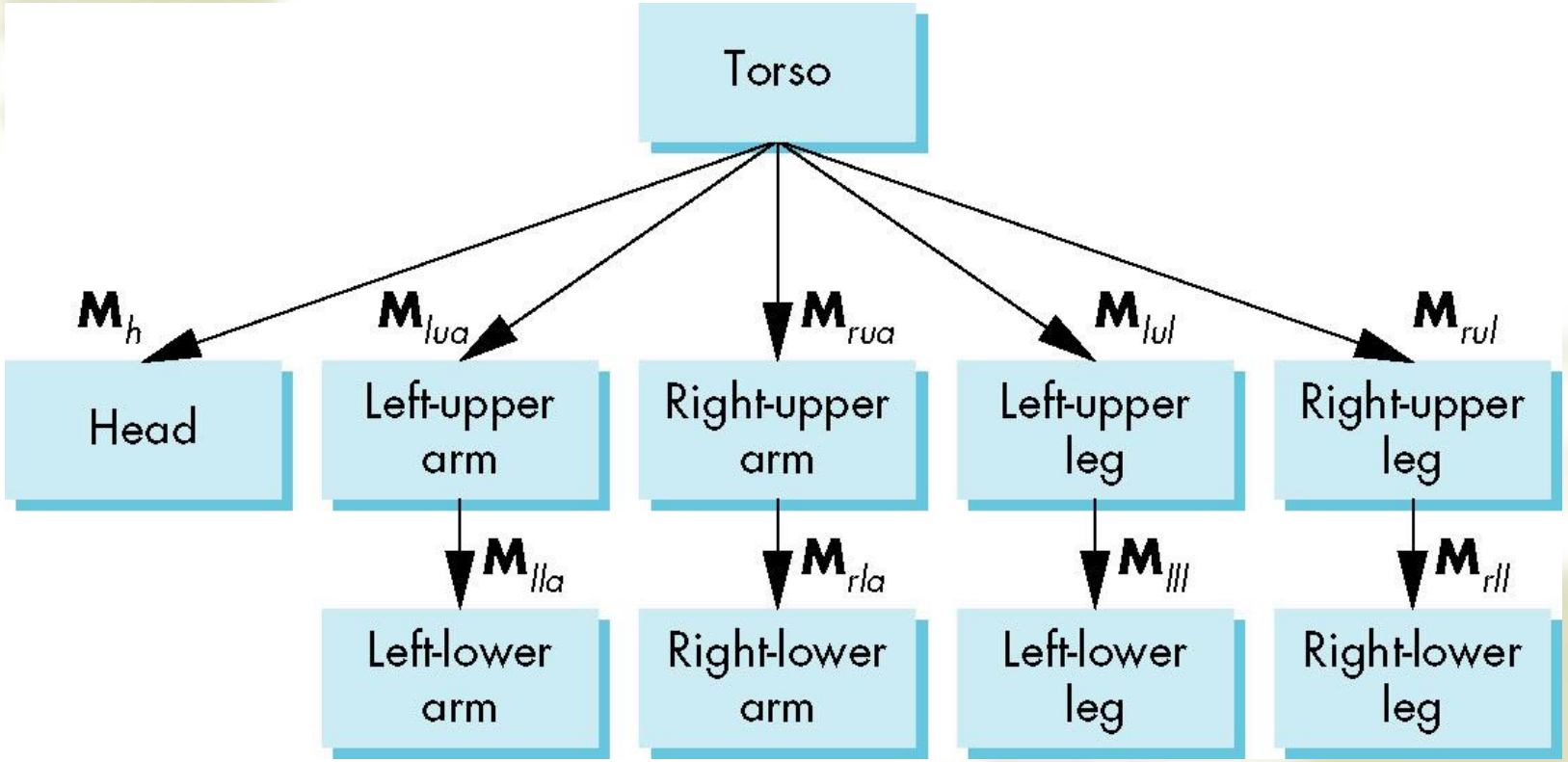
- Need to deal with multiple children
  - How do we represent a more general tree?
  - How do we traverse such a data structure?
- Animation
  - How to use dynamically?
  - Can we create and delete nodes during execution?

3. Example III : Humanoid Figure





Tree with Matrices



Display and Traversal

- The position of the figure is determined by 11 joint angles (two for the head and one for each other part)
- Display of the tree requires a *graph traversal*
  - Visit each node once
  - Display function at each node that describes the part associated with the node, applying the correct transformation matrix for position and orientation
- see also class define in page 464

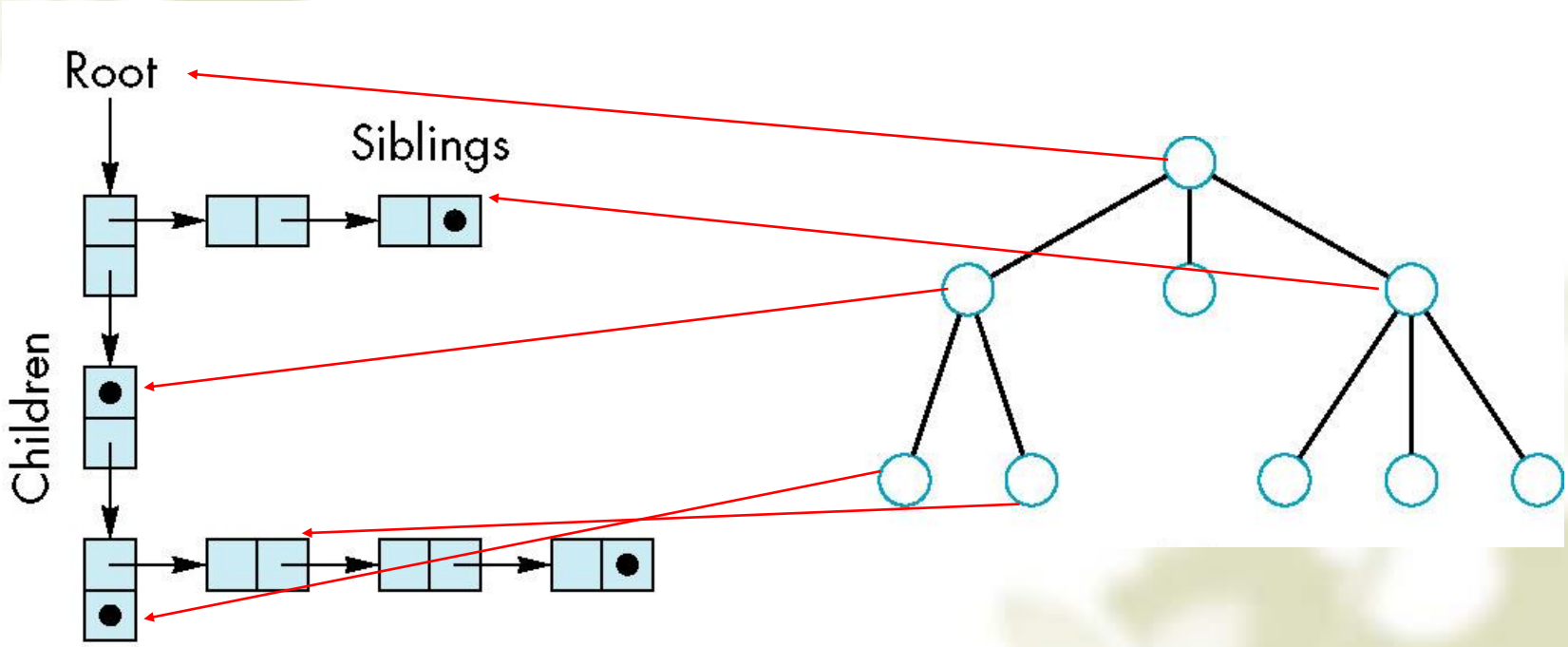
Stack-based Traversal Code

```
figure() {
 mvstack.push( model_view );//save present model-view matrix
 torso();
 //update model-view matrix for head
 model_view = model_view*Translate(…)*Rotate(…);
 head();

 model_view = mvstack.pop(); //recover original model-view matrix
 mvstack.push( model_view );//save it again
 //update model-view matrix for left upper arm
 model_view= model_view*Translate(…)*Rotate(…);
 left_upper_arm();

 model_view = mvstack.pop(); //recover original model-view matrix again
    ……
}
```

General Tree Data Structure

- Need a data structure to represent tree and an algorithm to traverse the tree by a binary tree
- We will use a *left-child right sibling* structure
  - Uses linked lists
  - Each node in data structure is two pointers
  - Left: next node
  - Right: linked list of children

Left-Child Right-Sibling Tree



Tree node Structure

- At each node we need to store
  - Pointer to sibling
  - Pointer to child
  - Pointer to a function that draws the object represented by the node
  - Homogeneous coordinate matrix to multiply on the right of the current model-view matrix
    - Represents changes going from parent to node
    - In OpenGL this matrix is a 1D array storing matrix by columns

C Definition of treenode

```
typedef struct treenode
{
    mat4 m;
    void (*f)();
    struct treenode *sibling;
    struct treenode *child;
} treenode;
```

torso and head nodes

```
treenode torso_node, head_node, lua_node, … ;

torso_node.m = RotateY(theta[0]);
torso_node.f = torso;
torso_node.sibling = NULL;
torso_node.child =  &head_node;

head_node.m=translate(0.0,TORSO_HEIGHT+0.5*HEAD_HEIGHT,0.0)
            *RotateX(theta[1])*RotateY(theta[2]);
 head_node.f = head;
 head_node.sibling = &lua_node;
 head_node.child = NULL;
```

Preorder Traversal

```
void traverse(treenode* root)
{
    if(root==NULL) return;
    mvstack.push(model_view);
    model_view = model_view*root->m;
    root->f();
    if(root->child!=NULL)
        traverse(root->child);
    model_view = mvstack.pop();
    if(root->sibling!=NULL)
        traverse(root->sibling);
}
```

Dynamic Trees

- If we use pointers, the structure can be dynamic

```
typedef treenode *tree_ptr;
tree_ptr torso_ptr;
torso_ptr = malloc(sizeof(treenode));
```

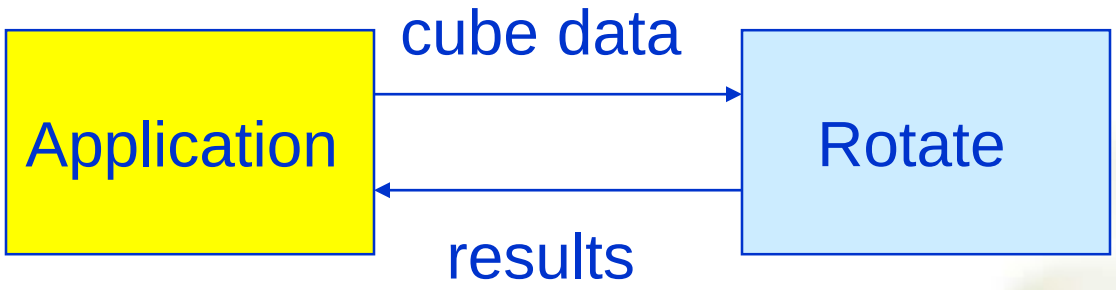
- Definition of nodes and traversal are essentially the same as before but we can **add and delete** nodes during execution

4. OpenGL and Objects

- OpenGL lacks an object orientation
- Consider, for example, a green sphere
  - We can model the sphere with polygons or use OpenGL quadrics
  - Its color is determined by the OpenGL state and is not a property of the object
- We can try to build better objects in code using object-oriented languages/techniques

Imperative Programming Model

- Example: rotate a cube



- The rotation function must know how the cube is represented
  - Vertex list
  - Edge list

Object-Oriented Programming Model

- In this model, the representation is stored with the object

- The application sends a *message* to the object
- The object contains functions (*methods*) which allow it to transform itself

Cube Object

- Suppose that we want to create a simple cube object that we can scale, orient, position and set its color directly through code such as

```
cube mycube;
mycube.color[0]=1.0;
mycube.color[1]= mycube.color[2]=0.0;
mycube.matrix[0][0]=……
```

Cube Object Functions

- We would also like to have functions that act on the cube such as
  - `mycube.translate(1.0, 0.0,0.0);`
  - `mycube.rotate(theta, 1.0, 0.0, 0.0);`
  - `setcolor(mycube, 1.0, 0.0, 0.0);`
- We also need a way of displaying the cube
  - `mycube.render();`

Building the Cube Object

```
class cube {
    public:
        float color[3];
        float matrix[4][4];
    // public methods

    private:
    // implementation
    // define a vertex list
}
```

Other Objects

- Other objects have geometric aspects
  - Cameras
  - Light sources
- But we should be able to have nongeometric objects too
  - Materials
  - Colors
  - Transformations (matrices)

Light Object

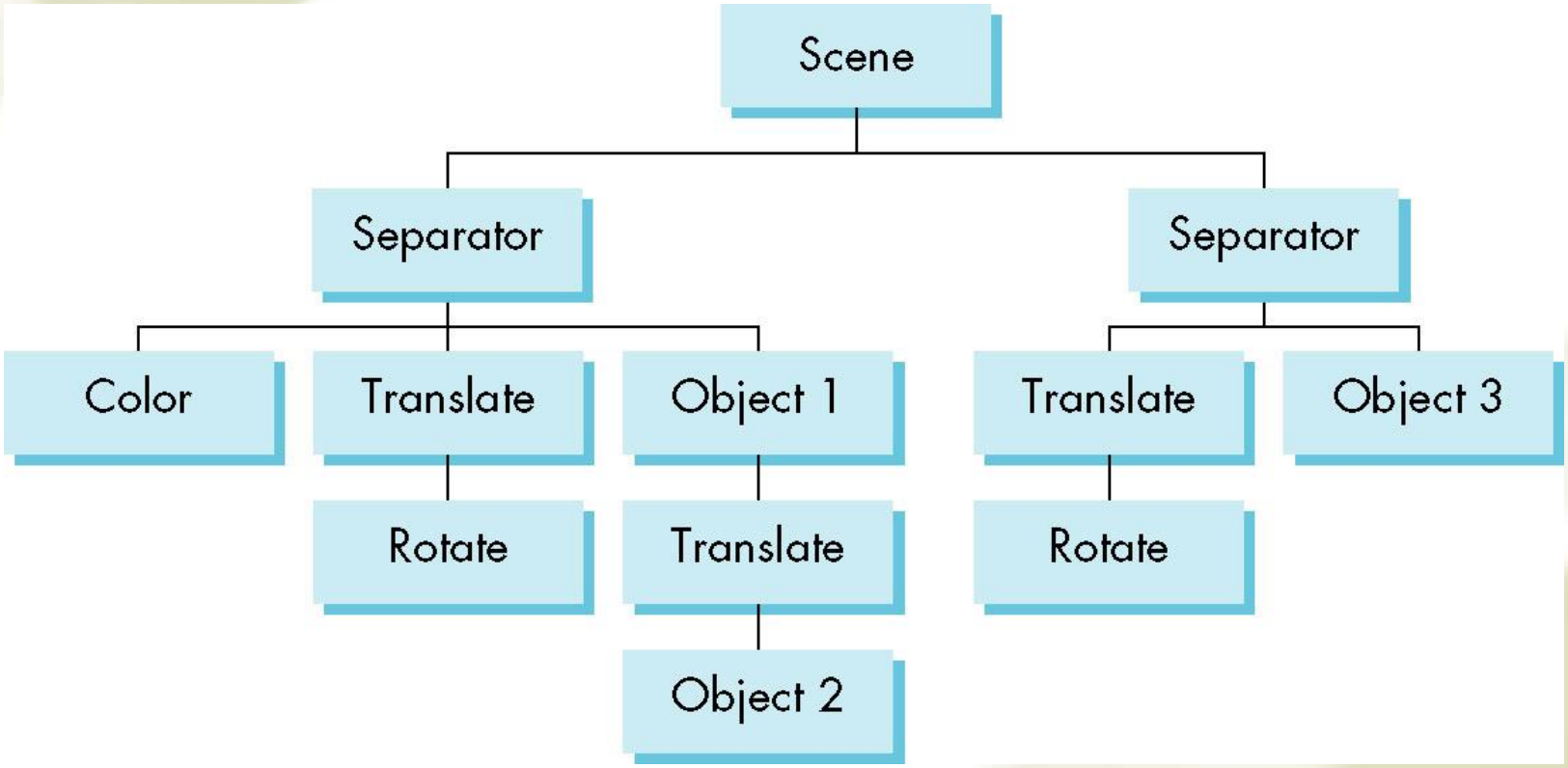
```
class light {     // match Phong model
  public:
   boolean type; //ortho or perspective
   boolean near;
   float position[3];
   float orientation[3];
   float specular[3];
   float diffuse[3];
   float ambient[3];
}
```

Application Code

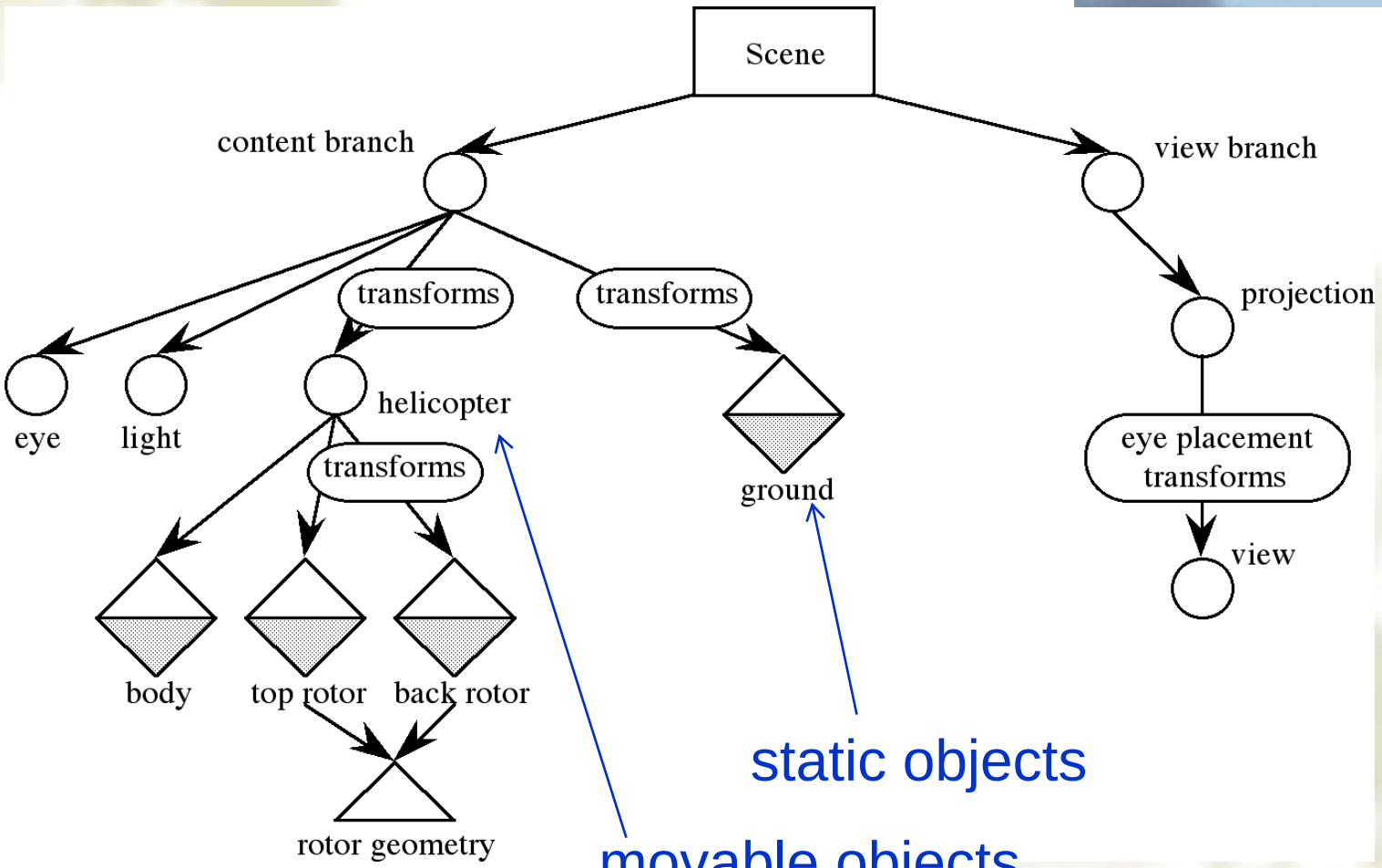
```
cube mycube;
material plastic;
mycube.setMaterial(plastic);

camera frontView;
frontView.position(x ,y, z);
```

We will learn more object-oriented graphics programming in the next course.

5. Scene Graph



Scene Graph


Scene
content branch
view branch
transforms
transforms
projection
eye
light
helicopter
ground
eye placement transforms
transforms
view
body
top rotor
back rotor
rotor geometry
static objects
movable objects

Group Nodes

- Necessary to isolate state chages
  - Equivalent to Push/Pop
- Note that as with the figure model
  - We can write a universal traversal algorithm
  - The order of traversal can matter
    - If we do not use the group node, state changes can persist

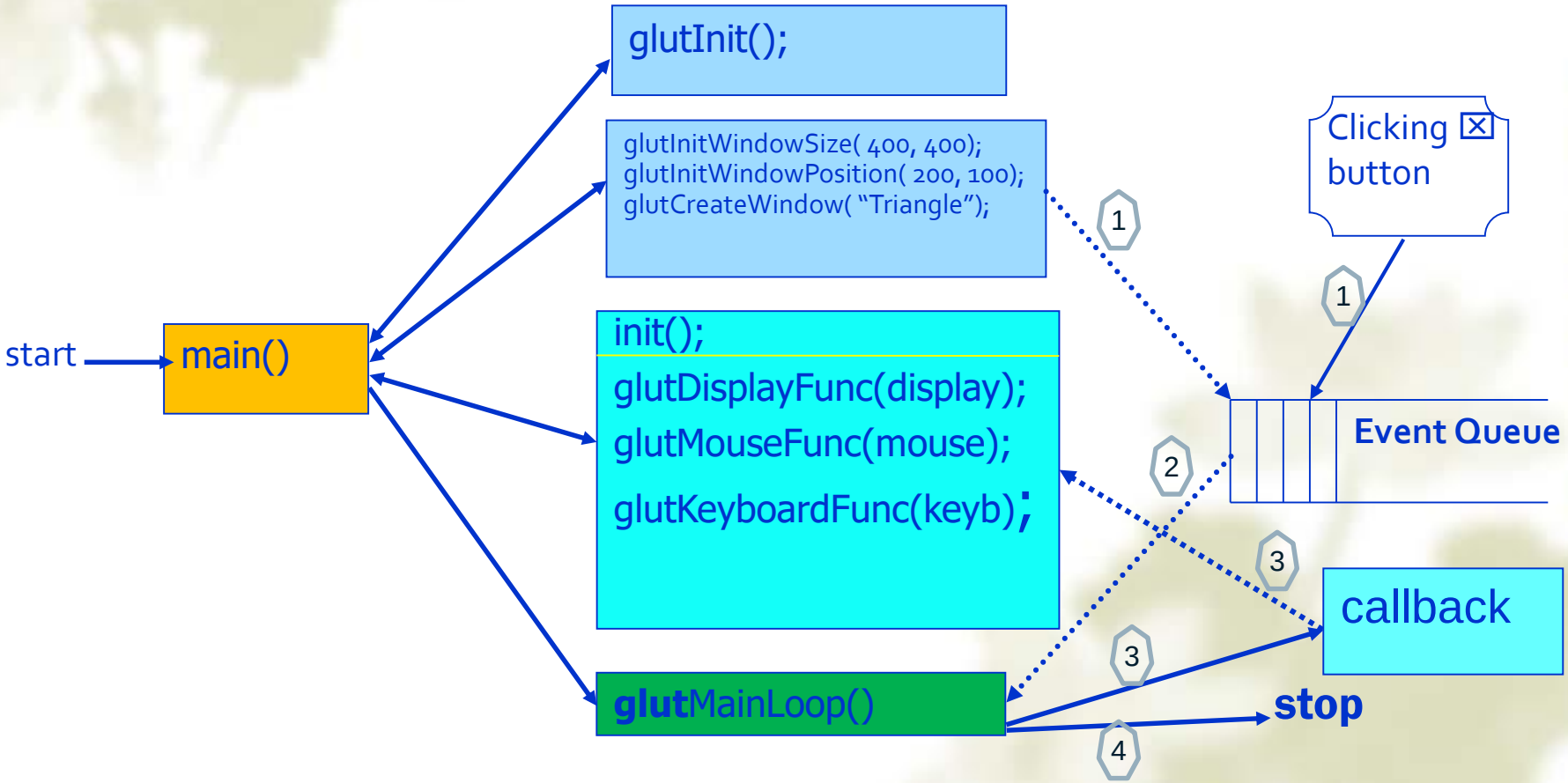
Events and the Scene Graph

- It is not difficult to let any part of the scene graph vary
- Events could be used to identify the part of the scene graph and how it is to change
  - Change geometry
  - change transformations
  - change appearance

6. Events in OpenGL

- Display
- Keyboard
- Special
- Mouse active motion
- Mouse passive motion
- Reshape
- Idle
- Timer
- Menu
- PostRedisplay

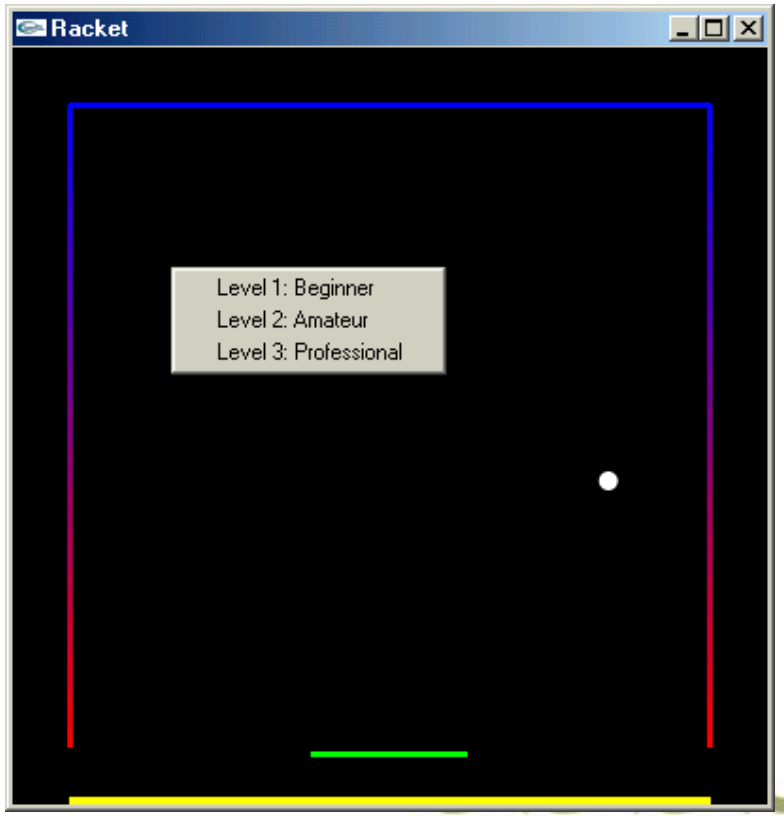
Event Processing in OpenGL



```
int  main( int  argc,  char  **argv)
{   glutInit( &argc, argv);                    // Initialize GLUT function callings
    glutInitWindowSize( 400, 400);
    glutInitWindowPosition( 200, 100);

    glutCreateWindow( "Sample");          // Schedule a window creation event

    //specify callback functions
    glutDisplayFunc( display);            // call display() if creating or redrawing the window
    glutIdleFunc( animation);             // call animation() when CPU idle
    glutKeyboardFunc( kboard);            // call kboard() if pressing keyboard
    glutSpecialFunc(special_key);         // call special_key() if special key is pressed
    glutMouseFunc( mousef);               // call mousef() if mouse button is pressed
    glutMotionFunc( mousePos);            // call mousePos() if mouse is moving
    glutCreateMenu( menuf );              //call menuf() if menu is triggered
    glutTimerFunc(300, timer, -1);        //call timer() again in 300 ms
    glutReshapeFunc(reshape);             //call reshape() if varying window size

    init();                               // Invoke this function for initialization
    glutMainLoop();                       // Enter the event processing loop

    return 0;
}                          timer() {…  ;   glutPostRedisplay();    // call display() right now}
```

```
void mousef(int button, int state, int x, int y) {
   if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
        drag = true;
    }
    if (button == GLUT_LEFT_BUTTON && state == GLUT_UP) {
         xx = x/100.-2.;   yy = 2.-y/100.;
         drag = false;
         glutPostRedisplay();
    }
}  // mousef()

void mousePos( int x, int y) {
    if ( drag ) {
        xx = x/100.-2.;   yy = 2.-y/100.;
        glutPostRedisplay();
    }
} //mousePos()
```

```
glutCreateMenu( menuf ); //callback menuf( int i) is called if triggered
glutAddMenuEntry("Level 1: Beginner", 1);
glutAddMenuEntry("Level 2: amateur", 2);
glutAddMenuEntry("Level 3: Professional", 3);
glutAttachMenu( GLUT_RIGHT_BUTTON);
```



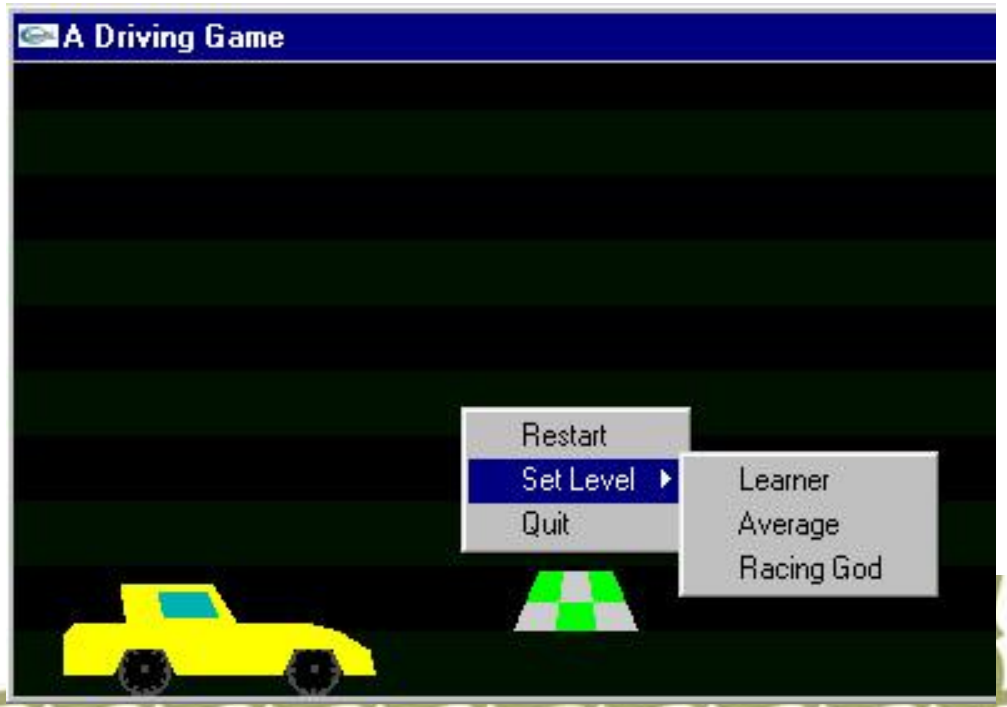
```
void menuf(  int i)  {
    if  (i == 1)  {                    //Beginner level
        ispeed = 0.002;                //Global variables
        rw = .125;
    }
    else  if  (i == 2)  {              //Amateur
        . . .
    }
   else  if  (i == 3)  {               //Professional
        . . .
    }

   . . .
}  //menuf()
```

Pop-up Menus with Sub-Menu Interaction Event

```
//Create the main menu
glutCreateMenu( mainf );
glutAddMenuEntry("Restart", 1);
//Link the sub-menu to the main menu
glutAddSubMenu("Set Level", levelMenu);
glutAddMenuEntry("Quit", 3);
glutAttachMenu( GLUT_RIGHT_BUTTON);

//Create a sub-menu
int  levelMenu = glutCreateMenu( levelf);
glutAddMenuEntry("Learner", 1);
glutAddMenuEntry("Average", 2);
glutAddMenuEntry("Racing God", 3);
```

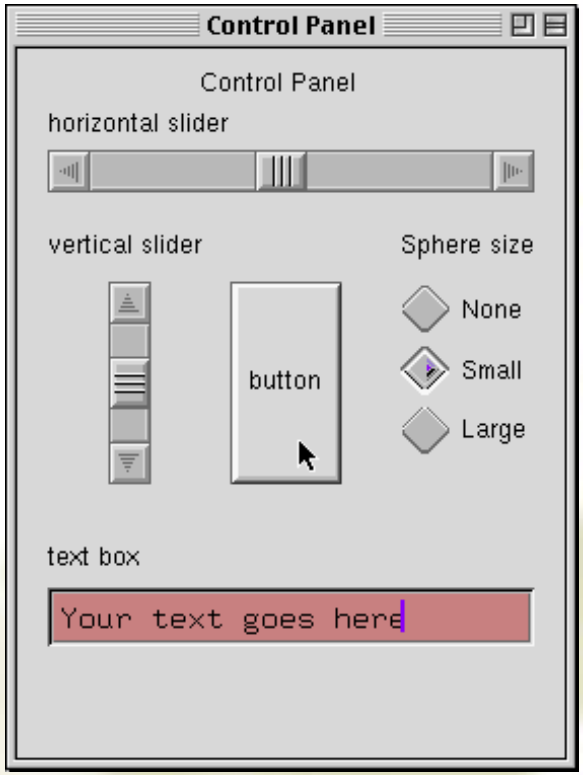
Callbacks

```
//define Callback function for the main menu
void mainf( int i) {
        if (i == 1)  { restart();  . . .  }
        else if (i == 3) { . . . }
        //If i == 2, levelf() will be called automatically
}

//define Callback for
//the sub-menu
void levelf( int i) {
        if ( i == 1)  . . .
        . . .
}
```



MUI(1)

- MUI is a particular user interaction toolkit
- It's very simple, but very easy to use
- This figure shows the look of a MUI control panel



MUI (2)

- An example of a simple MUI application