Tutorial 2

for the Course Project of CS4182/CS5182 Computer Graphics

Outline

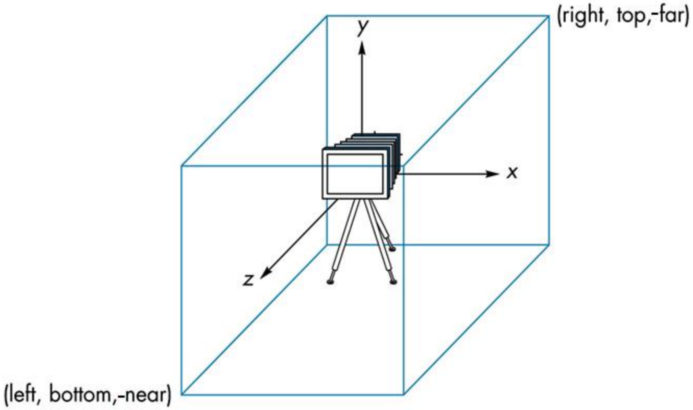
1. Introduction to OpenGL
2. Start OpenGL Program
3. OpenGL Functions
   - □ Primitives & Primitive Attributes
   - □ Light & Shading
   - □ **Viewing: Camera & Projection**
   - □ Transformations
   - □ Texture Mapping
   - □ Interaction: Keyboard, Mouse and Menus
4. Assignment

Viewing Functions

- Three aspects of viewing process:
  - Positioning the camera
    - Setting the model-view matrix
  - Specifying the type of projection
    - Setting the projection matrix
  - Clipping
    - Setting the view volume

Viewing

- **Default Setting of Viewing**
  - **Camera**: A virtual camera is placed at the **origin** in object space pointing the *negative z* direction
  - **Projection**: the default projection mode is **orthography**. Points are projected along z axis onto the plane z=0.
  - **Viewing volume**: the default viewing volume is a box centered at the origin with the size of **2×2×2**. Only objects in the viewing volume appears in the scene.
  - The **world and camera frames** are initially the same.



Positioning the camera

Moving the camera in OpenGL (two methods)

1) Transformation:

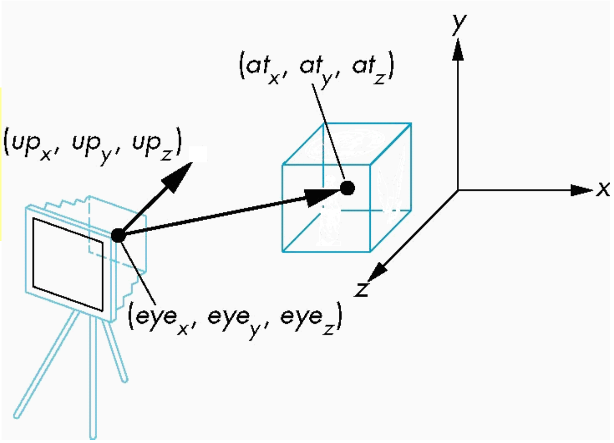
```
glMatrixModel (GL_MODELVIEW); //set the model_view matrix as current matrix
glLoadIdentity ();  //reset current matrix as identity matrix
    …   // transformations    (e.g. glTranslagef(0,0,0.3); )
```

2) Use gluLookAt function (we need to specify the followings):

- Eye point $[eye_x, eye_y, eye_z, 1]^T$: Location of the camera
- At point $[at_x, at_y, at_z, 1]^T$:A point that the camera is pointing at.
- Up direction $[up_x, up_y, up_z, 0]^T$: A vector which is used to determine the up direction.

Example:

```
glMatrixModel (GL_MODELVIEW);
glLoadIdentity ();
gluLookAt (eyex, eyey, eyez, atx, aty, atz, upx, upy, upz);
```

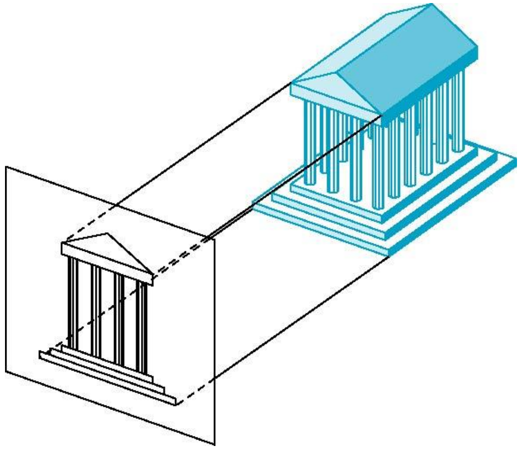


Projection

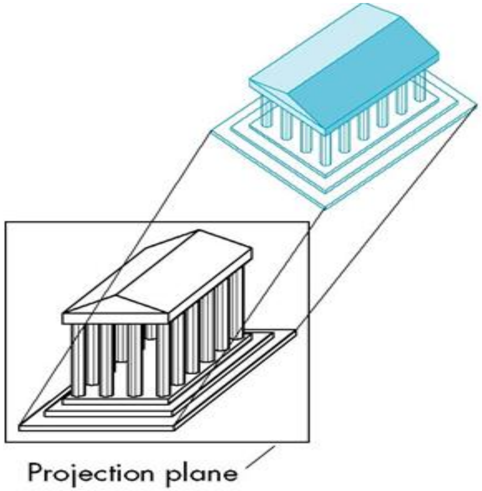
- Two types of projection:
  - Parallel projection
  - Perspective projection

Parallel projection

- Parallel projection includes:
  - ***Orthographic Projection***: projectors are orthogonal to the projection surface. It is a special case of parallel projection.
  - ***Oblique Projection***: projectors make an arbitrary angle with the projection plane



Orthographic projection

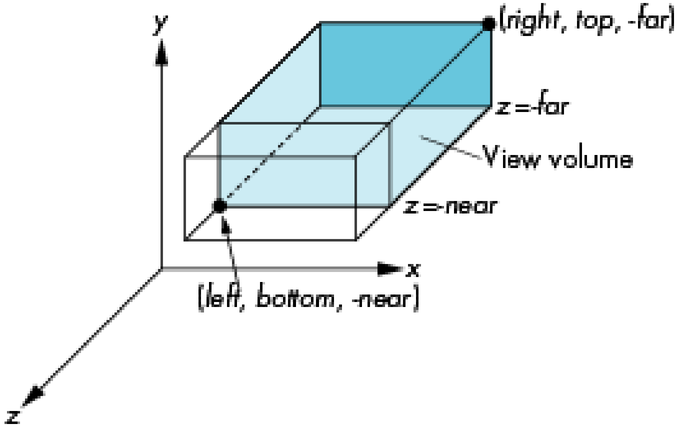


Oblique projection

Parallel Projections in OpenGL

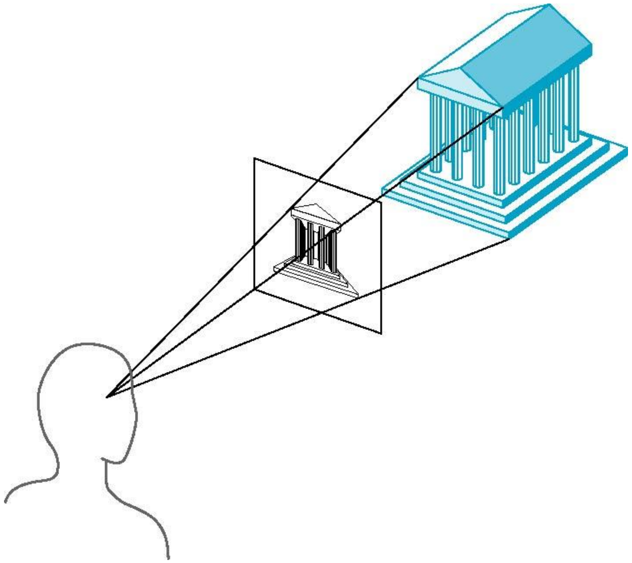
- ***Orthographic projection*** in OpenGL (default projection)
  - To set the projection mode as Orthographic Projection and to specify the camera view, we use:

```
glMatrixModel (GL_PROJECTION);
glLoadIdentity ();
glOrtho (left, right, bottom, top, near, far);
```



Perspective projection

- What is ***Perspective Projection***?
    - The COP (Center Of Projection) is at a finite distance.
    - The projectors are not parallel.



Perspective Projections in OpenGL

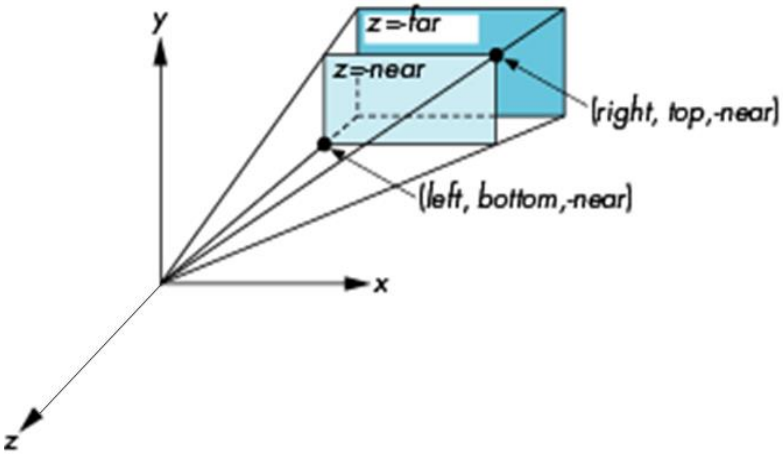
To set the projection mode as Perspective Projection and to specify the camera view, there are two methods:

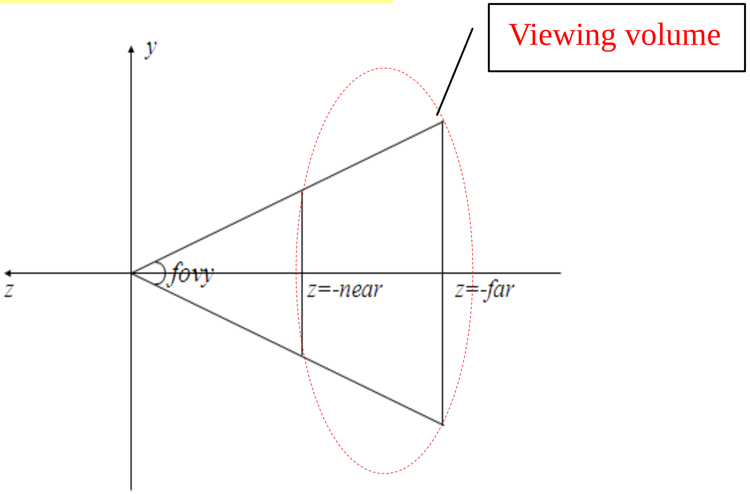
1)

```
glMatrixModel (GL_PROJECTION);
glLoadIdentity ();
glFrustum (left, right, bottom, top, near, far);
```

2)

```
glMatrixModel (GL_PROJECTION);
glLoadIdentity ();
gluPerspective (fovy, aspect, near, far);
```




Viewing volume

Outline

1. Introduction to OpenGL
2. Start OpenGL Program
3. OpenGL Functions
   - □ Primitives & Primitive Attributes
   - □ Light & Shading
   - □ Viewing: Camera & Projection
   - □ **Transformations**
   - □ Texture Mapping
   - □ Interaction: Keyboard, Mouse and Menus
4. Assignment

Transformation

- Types of transformations:
  - Translation
  - Rotation
  - Scale
  - Shear

Transformations in OpenGL

To perform transformations on the scene:

1) Load suitable matrix
2) Perform transformation on the matrix

- Load suitable matrix
  - Select matrix

```
glMatrixMode (GL_MODELVIEW);
glMatrixMode (GL_PROJECTION);
```

  - Load an identity matrix

```
glLoadIdentity ();
```

Transformations in OpenGL

- Three transformation functions
  - Translation

    ```
    glTranslatef (dx, dy, dz);
    ```

    - Function: move the objects across the 3 axes in 3D space
    - Parameters: dx, dy and dz are the components of a displacement vector
  - Rotation

    ```
    glRotatef (angle, vx, vy, vz);
    ```

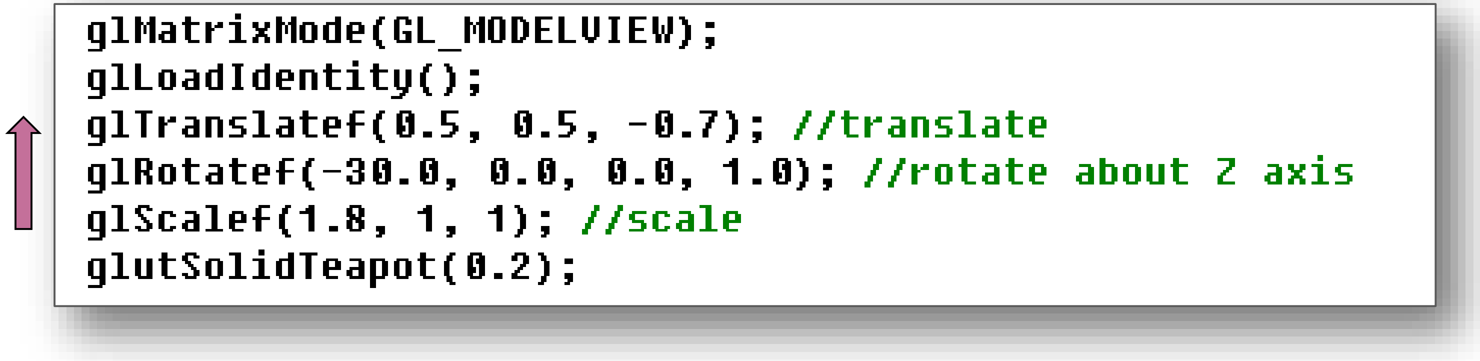
    - Function: rotate the objects about a vector with specific degree
    - Parameters: angle indicates the degrees; vx, vy and vx specify the components of the vector
  - Scaling

    ```
    glScalef (sx, sy, sz);
    ```

    - Function: enlarge or reduce the size of objects
    - Parameters: sx, sy and sz are the scale factors along the coordinate axes.

Transformations in OpenGL

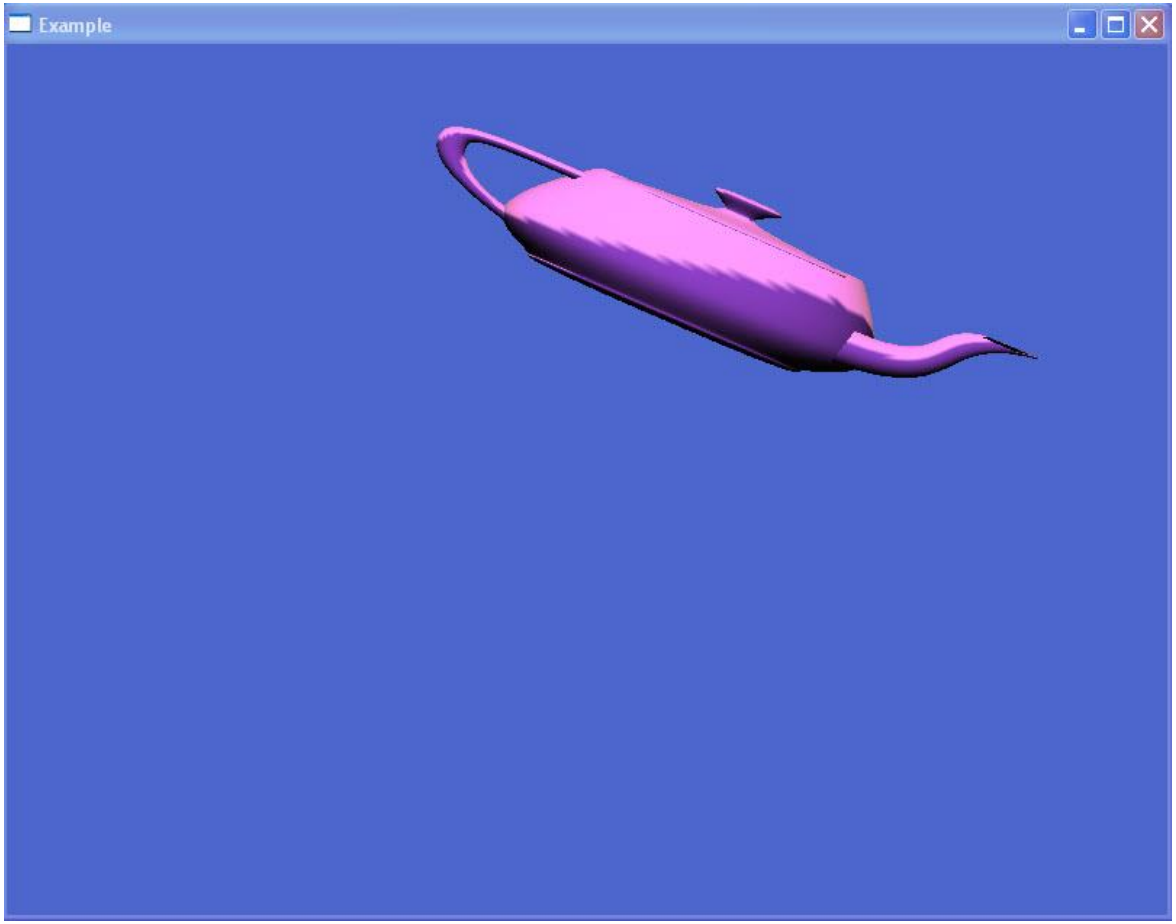
- Example



```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(0.5, 0.5, -0.7); //translate
glRotatef(-30.0, 0.0, 0.0, 1.0); //rotate about Z axis
glScalef(1.8, 1, 1); //scale
glutSolidTeapot(0.2);
```

- Order of transformations in OpenGL
  - The transformation functions are performed from down to up

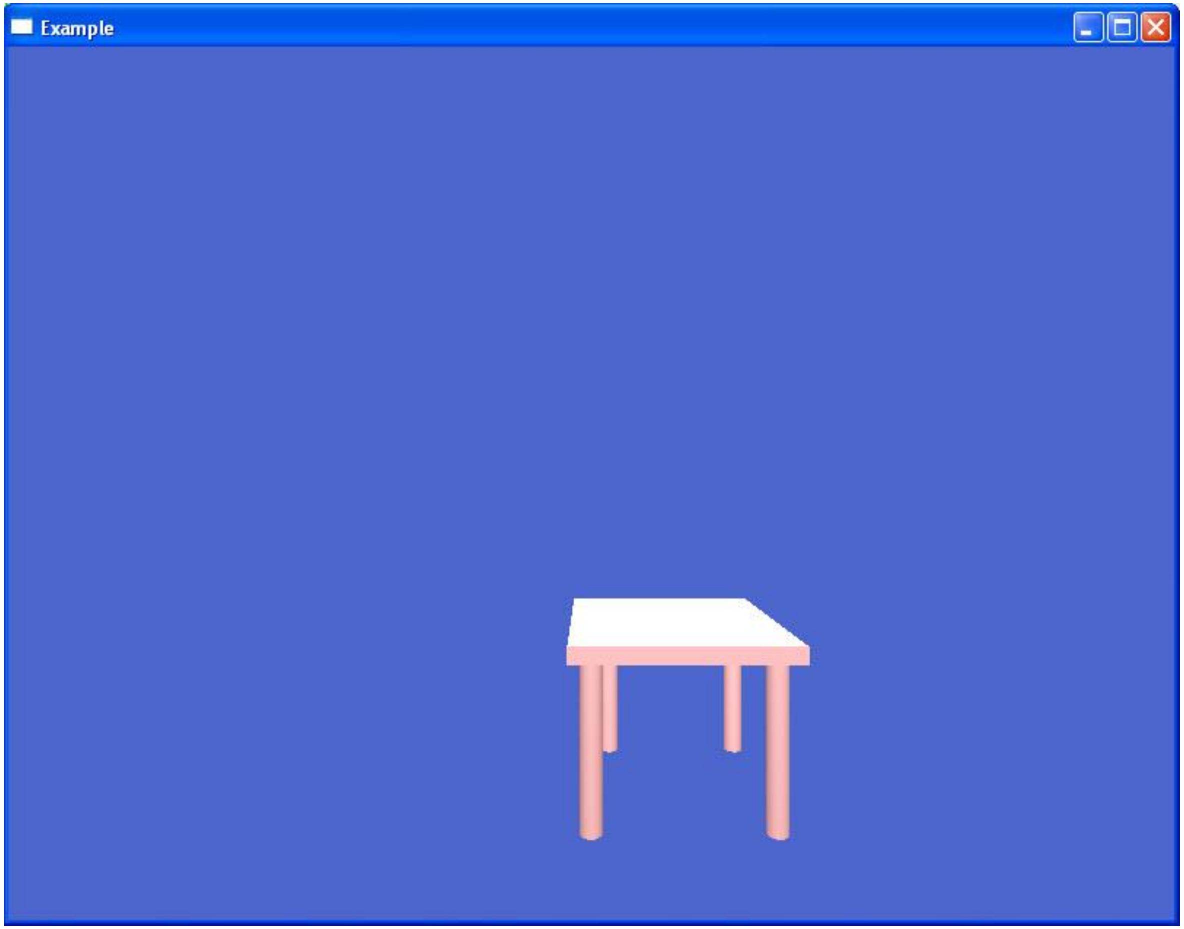
Example 3 - Rotated Teapot



Example 4 - Table

```
void Table() {
        //material property
        GLfloat tb_ambient[]={0.05,0.05,0.05,1};
        GLfloat tb_diffuse[]={0.8,0.8,0.8,1};
        GLfloat tb_specular[]={0.6,0.6,0.6,1};
        glMaterialfv(GL_FRONT_AND_BACK,GL_A
                MBIENT,tb_ambient);
        glMaterialfv(GL_FRONT_AND_BACK,GL_DI
                FFUSE,tb_diffuse);
        glMaterialfv(GL_FRONT_AND_BACK,GL_S
                PECULAR,tb_specular);

        glClear(GL_COLOR_BUFFER_BIT |
            GL_DEPTH_BUFFER_BIT);

        /******** 4 legs of the table *********/
        GLUquadricObj *ob=gluNewQuadric();

        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glPushMatrix();
        glTranslatef(0,-20,-45);
        glRotatef(-90,1,0,0);
        gluCylinder(ob,0.5,0.5,10,20,20);
        glPopMatrix();

        glPushMatrix();
        glTranslatef(1,-20,-53);
        glRotatef(-90,1,0,0);
        gluCylinder(ob,0.5,0.5,10,20,20);
        glPopMatrix();

        glPushMatrix();
        glTranslatef(8,-20,-53);
        glRotatef(-90,1,0,0);
        gluCylinder(ob,0.5,0.5,10,20,20);
        glPopMatrix();

        glPushMatrix();
        glTranslatef(8,-20,-45);
        glRotatef(-90,1,0,0);
        gluCylinder(ob,0.5,0.5,10,20,20);
        glPopMatrix();

        /********** surface of the table *******/
        glPushMatrix();
        glTranslatef(4,-9.5,-49);
        glScalef(1, 0.1, 1);
        glutSolidCube(10);
        glPopMatrix();

        glFlush();
}
```

Example 4 - Table

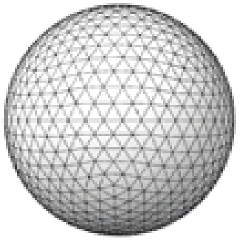


Outline

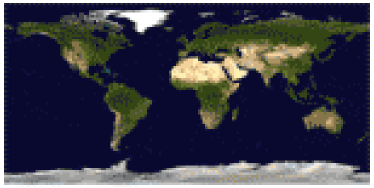
1. Introduction to OpenGL
2. Start OpenGL Program
3. OpenGL Functions
   - □ Primitives & Primitive Attributes
   - □ Light & Shading
   - □ Viewing: Camera & Projection
   - □ Transformations
   - □ **Texture Mapping**
   - □ Interaction: Keyboard, Mouse and Menus
4. Assignment

Texture Mapping

- Texture mapping: applies an image to a surface.



Sphere with no texture



Texture image

Sphere with texture

- Steps of texture mapping:
  1) Initialize and bind the texture
  2) Read the image file and produce texture
  3) Set up the texture parameters
  4) Apply the texture/ draw objects
  5) Clean the texture

Texture Mapping in OpenGL

Prior: Enable texture mapping

```
glEnable(GL_TEXTURE_2D);
```

1) Initialize the texture

```
GLuint myTexture;
glGenTextures(1, &myTexture);
```

//Allocate space for the texture

Number of textures, textures

Bind

```
glBindTexture(GL_TEXTURE_2D, myTexture);
```

// Set this texture as current/active

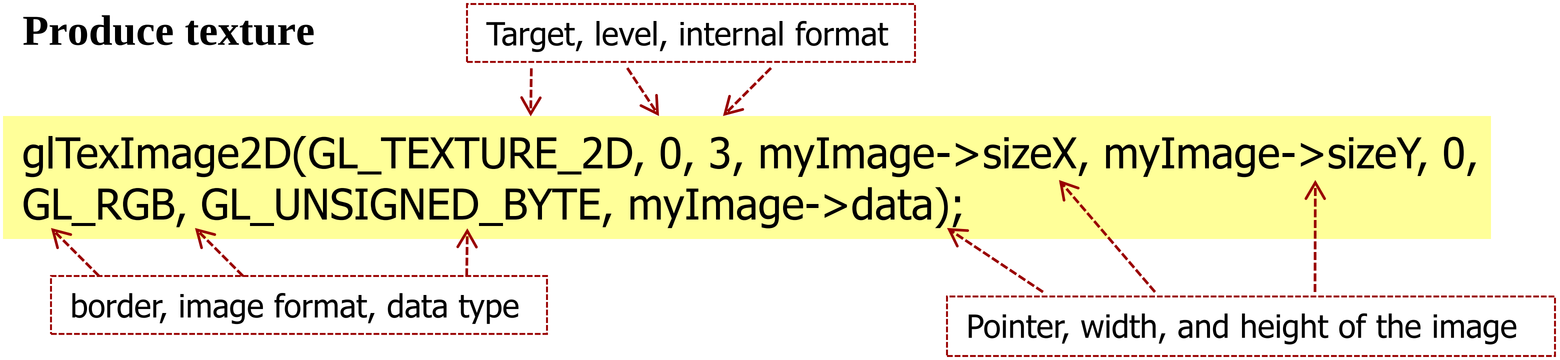
Target, texture

Texture Mapping in OpenGL

2) **Let OpenGL load the image**

```
char* filename=".\\wall.bmp";
AUX_RGBImageRec *myImage;
myImage=auxDIBImageLoad(filename);
```

Produce texture



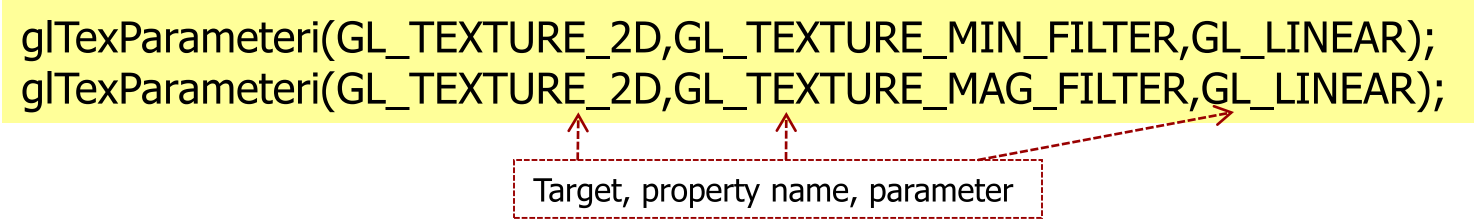
```
glTexImage2D(GL_TEXTURE_2D, 0, 3, myImage->sizeX, myImage->sizeY, 0,
GL_RGB, GL_UNSIGNED_BYTE, myImage->data);
```

Target, level, internal format

border, image format, data type

Pointer, width, and height of the image

3) **Set up the texture parameters**



```
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
```

Target, property name, parameter

Function: specify a filter to use on the texture when it is scaled

Texture Mapping in OpenGL

4) Apply the texture

```
glBegin(GL_POLYGON);
        glTexCoord2f(0,0);
        glVertex3f(-0.5,0.5,0);
        glTexCoord2f(1.0,0);
        glVertex3f(-0.5,-0.5,0);
        glTexCoord2f(1.0,1.0);
        glVertex3f(0.5,-0.5,0);
        glTexCoord2f(0,1.0);
        glVertex3f(0.5,0.5,0);
glEnd();
```

//Create object and specify the texture

5) Clean the texture

```
glDeleteTextures(1, &myTexture);
```

// Free the allocated space

Number of textures, textures

Outline

1. Introduction to OpenGL
2. Start OpenGL Program
3. OpenGL Functions
   - □ Primitives & Primitive Attributes
   - □ Light & Shading
   - □ Viewing: Camera & Projection
   - □ Transformations
   - □ Texture Mapping
   - □ **Interaction: Keyboard, Mouse and Menus**
4. Assignment

Add Interaction to Your Program

Three types of interaction:

- ***Keyboard Interaction***
- ***Mouse Interaction***
- ***Pop-up menus***

Two steps:

1) Specify the function to be called for specific type of interaction

   e.g. void myKeyboard (……)

2) To tell the 'main' function to call the function, when specific action occurs

   e.g. void main (int argc, char argv[]) {*

   ```
   e.g.   void main (int argc, char* argv[]) {
                  …
                  glutKeyboardFunc(myKeyboard);
          }
   ```

Keyboard Interaction – General Keys

- Add keyboard interaction for ***general keys***

1) Create a function to specify the keyboard control:

Function name can be changed

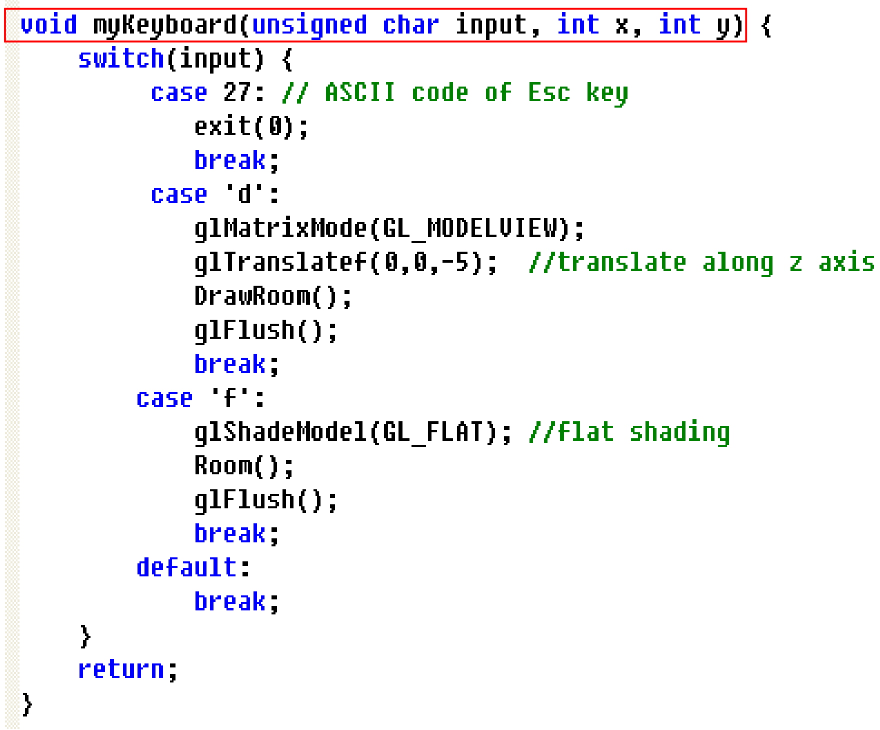
Three parameters can not be changed

```
void myKeyboard (unsigned char input, int x, int y) {}
```

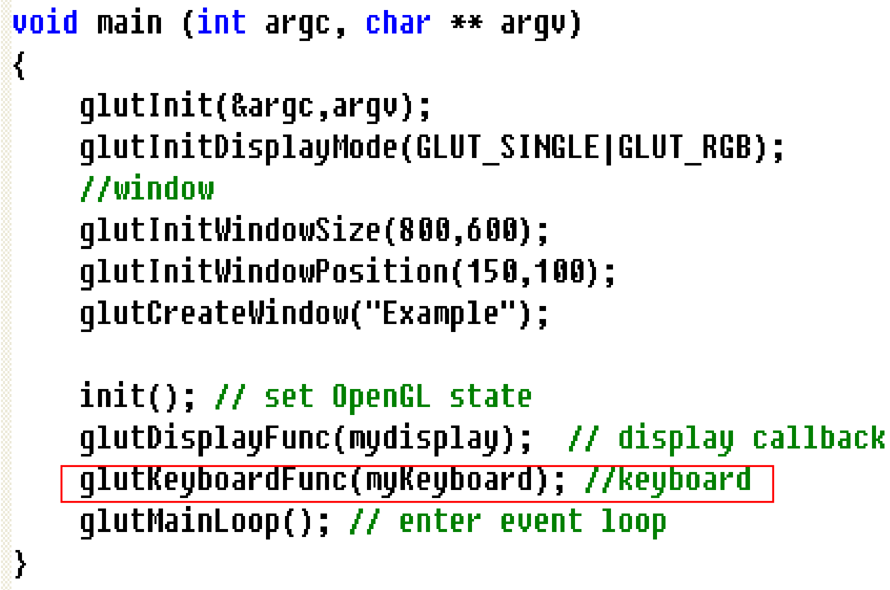
2) Tell the 'main' function to call the function, when keyboard interaction occurs:

```
void glutKeyboardFunc ( function_name );
```

Keyboard Interaction - General Keys -Example



```
void myKeyboard(unsigned char input, int x, int y) {
    switch(input) {
         case 27: // ASCII code of Esc key
            exit(0);
            break;
         case 'd':
            glMatrixMode(GL_MODELVIEW);
            glTranslatef(0,0,-5);  //translate along z axis
            DrawRoom();
            glFlush();
            break;
        case 'f':
            glShadeModel(GL_FLAT); //flat shading
            Room();
            glFlush();
            break;
        default:
            break;
    }
    return;
}
```



```
void main (int argc, char ** argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    //window
    glutInitWindowSize(800,600);
    glutInitWindowPosition(150,100);
    glutCreateWindow("Example");

    init(); // set OpenGL state
    glutDisplayFunc(mydisplay);  // display callback
    glutKeyboardFunc(myKeyboard); //keyboard
    glutMainLoop(); // enter event loop
}
```

Keyboard Interaction – Special Keys

- Add keyboard interaction for ***special keys*** *(←, F1, page_up, home, end, insert)*

1) Create a function to specify the keyboard control:

Function name can be changed

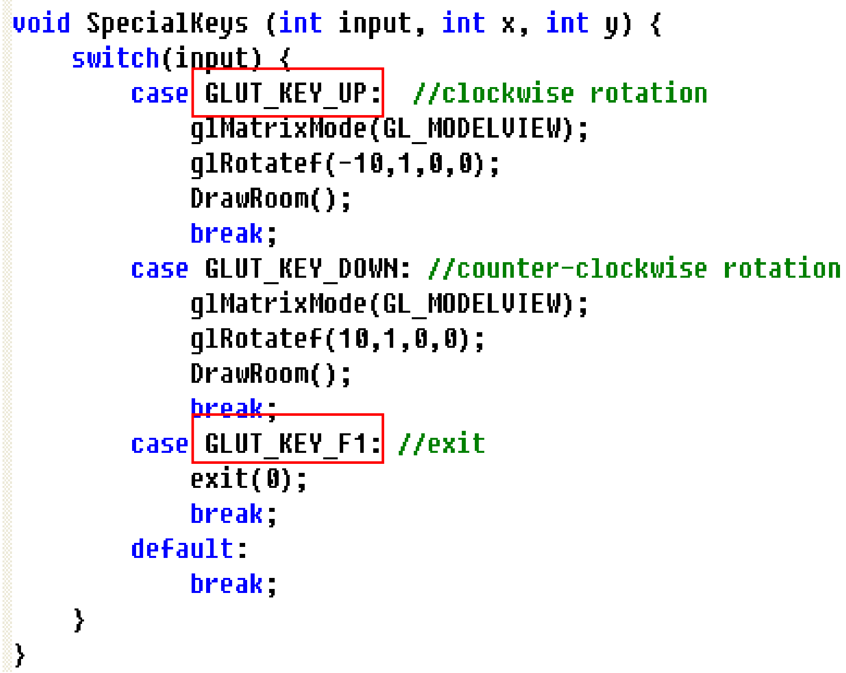
Three parameters can not be changed

```
void mySpecialKeys (int input, int x, int y) {}
```

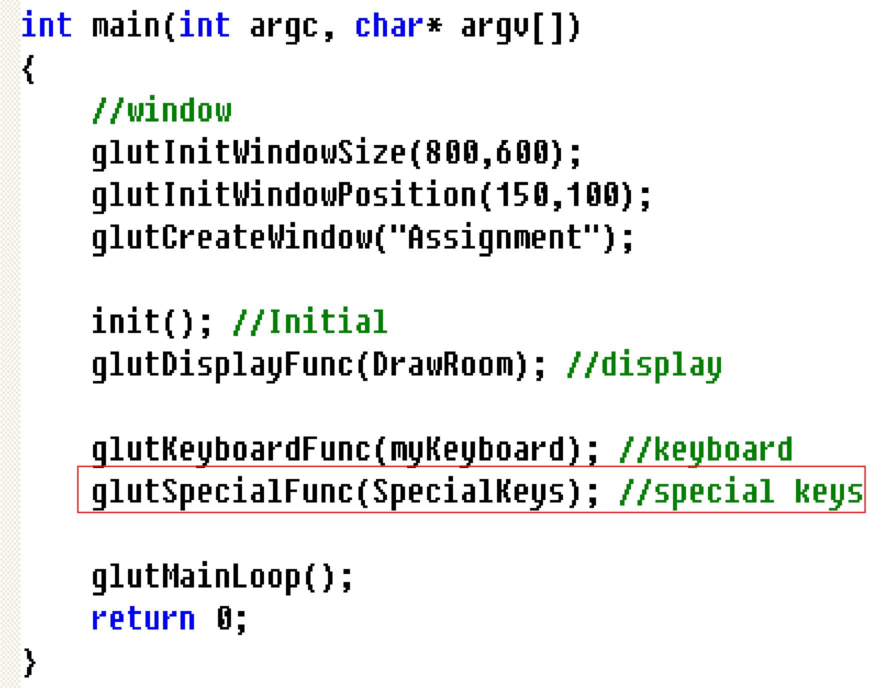
2) Tell the ‘main’ function to call the function, when special keys are entered:

```
void glutSpecialFunc ( function_name );
```

Keyboard Interaction – Special Keys - Example



```
void SpecialKeys (int input, int x, int y) {
    switch(input) {
        case GLUT_KEY_UP:  //clockwise rotation
            glMatrixMode(GL_MODELVIEW);
            glRotatef(-10,1,0,0);
            DrawRoom();
            break;
        case GLUT_KEY_DOWN: //counter-clockwise rotation
            glMatrixMode(GL_MODELVIEW);
            glRotatef(10,1,0,0);
            DrawRoom();
            break;
        case GLUT_KEY_F1: //exit
            exit(0);
            break;
        default:
            break;
    }
}
```

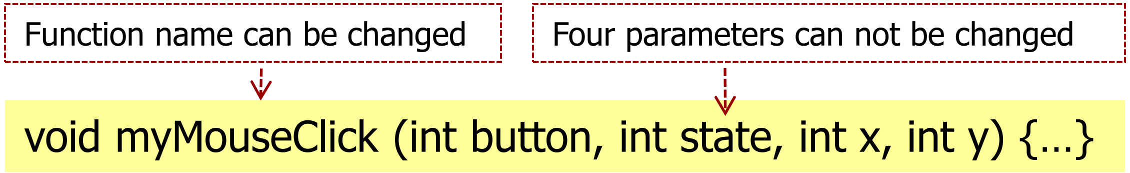


```
int main(int argc, char* argv[])
{
    //window
    glutInitWindowSize(800,600);
    glutInitWindowPosition(150,100);
    glutCreateWindow("Assignment");

    init(); //Initial
    glutDisplayFunc(DrawRoom); //display

    glutKeyboardFunc(myKeyboard); //keyboard
    glutSpecialFunc(SpecialKeys); //special keys

    glutMainLoop();
    return 0;
}
```

Mouse Interaction - Clicks

- **Mouse Clicks**

1) Specify a function to process mouse click event



Function name can be changed

Four parameters can not be changed

```
void myMouseClick (int button, int state, int x, int y) {…}
```

- Function: process the mouse click events
- Parameters:
  - First: the button that are pressed. (GLUT_LEFT_BUTTON / GLUT_MIDDLE_BUTTON / GLUT_RIGHT_BUTTON)
  - Second: the state of the button when the callback was generated, pressed or released. (GLUT_DOWN / GLUT_UP)
  - Third & forth: the coordinates (x,y) of the mouse relatively to the upper left corner of the client area of the window.

2) To tell the 'main' function to detect mouse click event

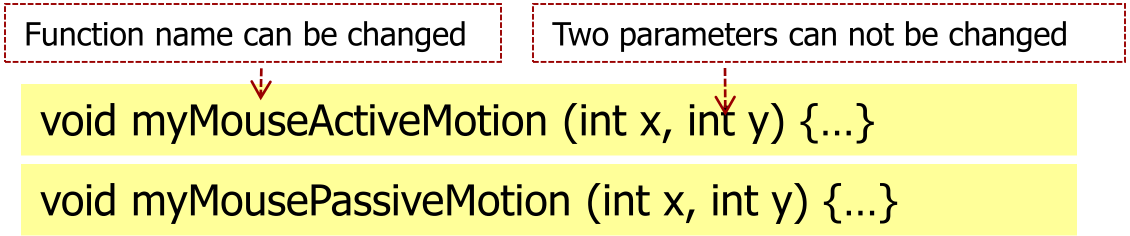
```
void glutMouseFunc ( function_name );
```

Reference: http://www.lighthouse3d.com/opengl/glut/index.php?9

Mouse Interaction - Motion

- **Mouse Motion**
  - Active motion: mouse is moving and a button is pressed
  - Passive motion: mouse is moving and no button is pressed
- **Detect and Process of Mouse Motion**

1) Specify a function to process mouse active/passive motion event



Function name can be changed

Two parameters can not be changed

```
void myMouseActiveMotion (int x, int y) {…}
void myMousePassiveMotion (int x, int y) {…}
```

- Parameters: the coordinates (x,y) of the mouse relatively to the upper left corner of the client area of the window.

2) To tell the ‘main’ function to detect mouse active/passive motion

```
void glutMotionFunc ( function_name );          // for active motion

void glutPassiveMotionFunc ( function_name );   // for passive motion
```

Mouse Interaction – Enter/Leave

- **Mouse Enters or Leaves the Window**

1) Specify a function to process mouse entry/leave event

Function name can be changed

the parameter can not be changed

```
void myMouseEntry (int state) {…}
```

- Function: process the mouse entry/leave events
- Parameters:
    - State: the mouse has entered or left the window. (GLUT_LEFT / GLUT_ENTERED)

2) To tell the ‘main’ function to detect mouse entry/leave event

```
void glutEntryFunc ( function_name );
```

Reference: http://www.lighthouse3d.com/opengl/glut/index.php?9

Mouse Interaction - Example

- Example

```
#include <GL/glut.h>
void Init () { //setting up the environment
        …
}

void Display () { //create objects
        …
}

// Mouse click
void myMouseClick (int button, int state, int x, int y) {
    …
}

// Mouse active motion
void myMouseActiveMotion (int x, int y) {
     …
}
```

```
void main (int argc, char** argv) {
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        Init();
         glutInitWindowSize (800,600);
         glutInitWindowPosition (150,100);
         glutCreateWindow ("Example");
         glutDisplayFunc(Display);
         glutMouseFunc(myMouseClick); //mouse click
         glutMotionFunc(myMouseActiveMotion); //active motion
       glutMainLoop();
}
```

Pop-up Menus

- Steps to add pop-up menus:
  1) Create a menu
  2) Add entries to the menu
  3) Attach the menu to a mouse button
  4) Specify a function to process the menu events

Pop-up Menus

1) Create a menu:

```
void glutCreateMenu ( function_name );
```

- Function: create pop-up menu when menu events occurs
- Parameters: function name (The callback function will be called to process the menu events)

2) Add entries to the menu:

```
void glutAddMenuEntry ( char* name, int value);
```

- Function: add an entry to the menu
- Parameters
  - First: name of the entry
  - Value: the value that will be returned to the callback function when this entry is selected.

3) Attach the menu to a mouse button:

```
void glutAttachMenu ( int button );
```

- Function: specify which mouse click activity will trigger the menu event
- Parameter: an integer to specify the button (GLUT_LEFT_BUTTON / GLUT_MIDDLE_BUTTON . BLUT_RIGHT_BUTTON)

Pop-up Menus

4) Specify a callback function to process the menu events

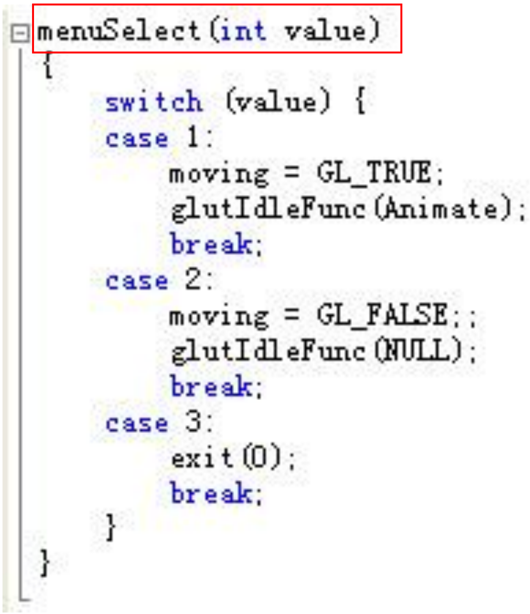
Function name can be changed

the parameter can not be changed

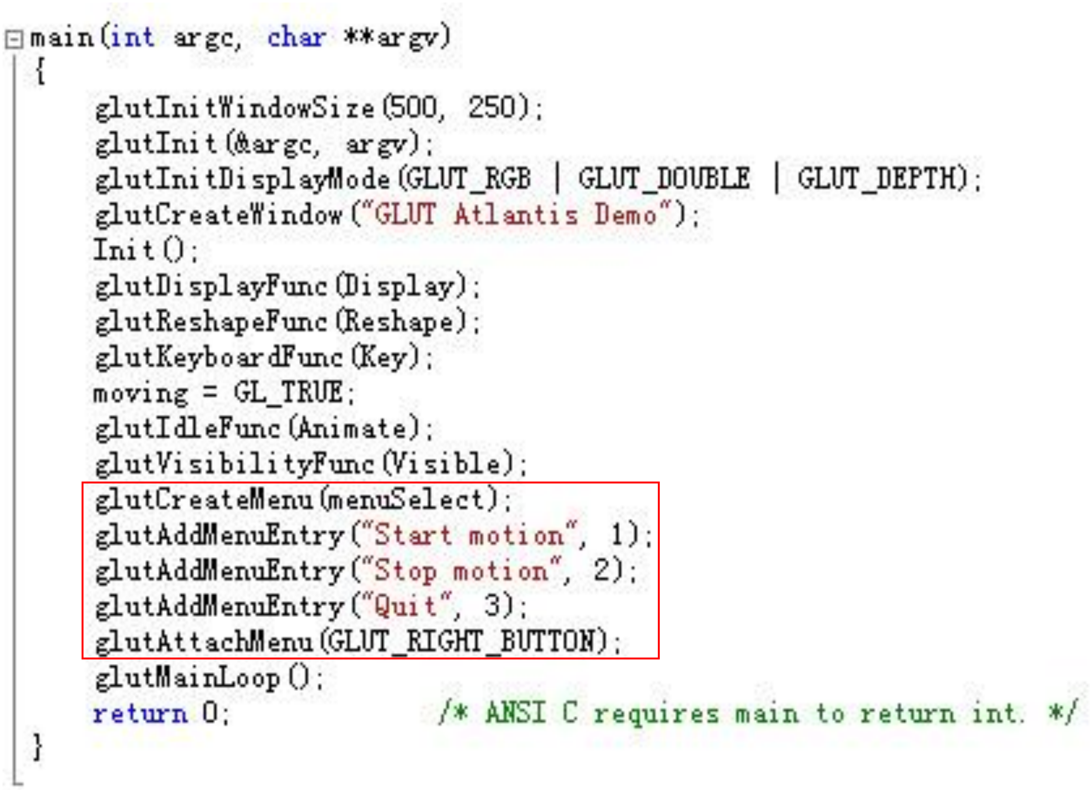
```
void myMenu ( int value );
```

- Function: specify the commands to be executed, when an entry of the menu is selected
- Parameter: the value indicates which entry is selected

Example 5 - Fishes

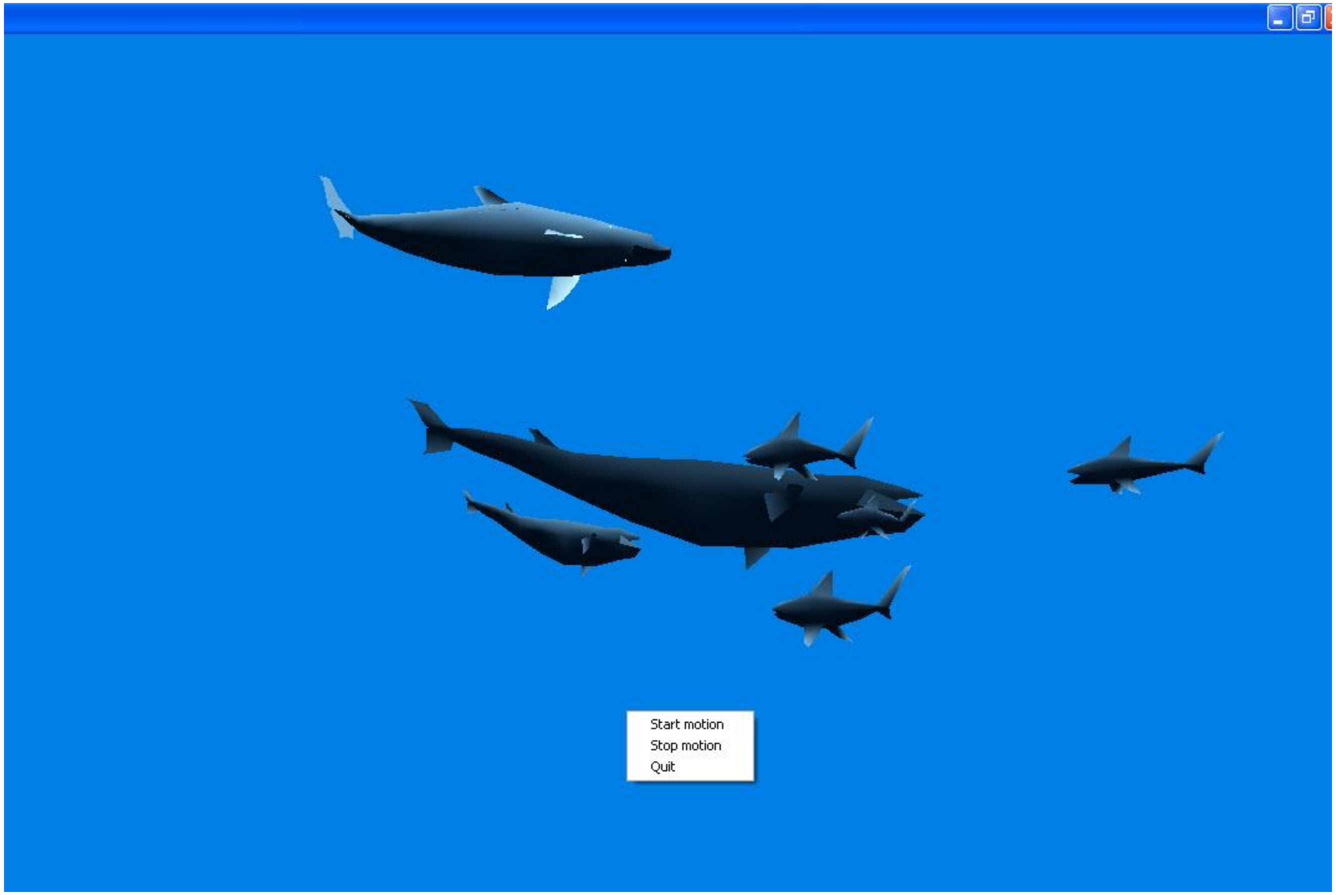


```
menuSelect(int value)
{
    switch (value) {
    case 1:
        moving = GL_TRUE;
        glutIdleFunc(Animate);
        break;
    case 2:
        moving = GL_FALSE;;
        glutIdleFunc(NULL);
        break;
    case 3:
        exit(0);
        break;
    }
}
```



```
main(int argc, char **argv)
{
        glutInitWindowSize(500, 250);
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
        glutCreateWindow(“GLUT Atlantis Demo”);
        Init();
        glutDisplayFunc(Display);
        glutReshapeFunc(Reshape);
        glutKeyboardFunc(Key);
        moving = GL_TRUE;
        glutIdleFunc(Animate);
        glutVisibilityFunc(Visible);
        glutCreateMenu(menuSelect);
        glutAddMenuEntry(“Start motion”, 1);
        glutAddMenuEntry(“Stop motion”, 2);
        glutAddMenuEntry(“Quit”, 3);
        glutAttachMenu(GLUT_RIGHT_BUTTON);
        glutMainLoop();
        return 0;           /* ANSI C requires main to return int. */
}
```

Example 5 - Fishes


Start motion
Stop motion
Quit