yet another insignificant programming notes... | HOME

# 3D Graphics with OpenGL By Examples

#### TABLE OF CONTENTS (HIDE)

- 1. Example 1: 3D Shapes (0GL01!
- 2. Example 2: 3D Shape with Anin
- 3. Example 3: Orthographic Projec
- 4. Example 4: Vertex Array

I assume that you have some knowledge of OpenGL. Otherwise, read "Introduction to OpenGL with 2D Graphics".

# 1. Example 1: 3D Shapes (0GL01Shape3D.cpp)

This example is taken from Nehe OpenGL Tutorial Lesson # 5 (@ http://nehe.gamedev.net/), which displays a 3D color-cube and a pyramid. The cube is made of 6 quads, each having different colors. The hallow pyramid is made up of 4 triangle, with different colors on each of the vertices.

```
1  /*
2  * 0GL01Shape3D.cpp: 3D Shapes
3  */
4  #include <windows.h> // for MS Windows
5  #include <GL/glut.h> // GLUT, include glu.h and gl.h
6
7  /* Global variables */
8  char title[] = "3D Shapes";
9
10  /* Initialize OpenGL Graphics */
11  void initGL() {
12   glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Set background color to black and opaque
```

```
13
       glClearDepth(1.0f);
                                             // Set background depth to farthest
14
       glEnable(GL DEPTH TEST);
                                 // Enable depth testing for z-culling
15
       glDepthFunc(GL LEQUAL);
                                 // Set the type of depth-test
16
       glShadeModel(GL SM00TH);
                                 // Enable smooth shading
       qlHint(GL PERSPECTIVE CORRECTION HINT, GL NICEST); // Nice perspective corrections
17
18
    }
19
    /* Handler for window-repaint event. Called back when the window first appears and
20
       whenever the window needs to be re-painted. */
21
22
    void display() {
23
       glClear(GL COLOR BUFFER BIT | GL DEPTH BUFFER BIT); // Clear color and depth buffers
24
       qlMatrixMode(GL MODELVIEW); // To operate on model-view matrix
25
26
       // Render a color-cube consisting of 6 quads with different colors
27
       alLoadIdentitv():
                                         // Reset the model-view matrix
       qlTranslatef(1.5f, 0.0f, -7.0f); // Move right and into the screen
28
29
30
       glBegin(GL QUADS);
                                         // Begin drawing the color cube with 6 quads
31
          // Top face (y = 1.0f)
32
          // Define vertices in counter-clockwise (CCW) order with normal pointing out
33
          glColor3f(0.0f, 1.0f, 0.0f);
                                          // Green
34
          glVertex3f( 1.0f, 1.0f, -1.0f);
35
          glVertex3f(-1.0f, 1.0f, -1.0f);
36
          glVertex3f(-1.0f, 1.0f, 1.0f);
37
          glVertex3f( 1.0f, 1.0f, 1.0f);
38
39
          // Bottom face (y = -1.0f)
40
          glColor3f(1.0f, 0.5f, 0.0f);
                                           // Orange
          glVertex3f( 1.0f, -1.0f, 1.0f);
41
42
          glVertex3f(-1.0f, -1.0f, 1.0f);
          glVertex3f(-1.0f, -1.0f, -1.0f);
43
          glVertex3f( 1.0f, -1.0f, -1.0f);
44
45
46
          // Front face (z = 1.0f)
          qlColor3f(1.0f, 0.0f, 0.0f);
47
                                           // Red
48
          glVertex3f( 1.0f, 1.0f, 1.0f);
```

```
49
           glVertex3f(-1.0f, 1.0f, 1.0f);
50
           glVertex3f(-1.0f, -1.0f, 1.0f);
51
           glVertex3f( 1.0f, -1.0f, 1.0f);
52
53
           // Back face (z = -1.0f)
54
           glColor3f(1.0f, 1.0f, 0.0f);
                                            // Yellow
55
           glVertex3f( 1.0f, -1.0f, -1.0f);
56
           glVertex3f(-1.0f, -1.0f, -1.0f);
57
           glVertex3f(-1.0f, 1.0f, -1.0f);
           glVertex3f( 1.0f, 1.0f, -1.0f);
58
59
60
           // Left face (x = -1.0f)
           glColor3f(0.0f, 0.0f, 1.0f);
61
                                            // Blue
           glVertex3f(-1.0f, 1.0f, 1.0f);
62
63
           glVertex3f(-1.0f, 1.0f, -1.0f);
64
           glVertex3f(-1.0f, -1.0f, -1.0f);
65
           glVertex3f(-1.0f, -1.0f, 1.0f);
66
67
           // Right face (x = 1.0f)
           qlColor3f(1.0f, 0.0f, 1.0f);
68
                                            // Magenta
69
           glVertex3f(1.0f, 1.0f, -1.0f);
70
           glVertex3f(1.0f, 1.0f, 1.0f);
           glVertex3f(1.0f, -1.0f, 1.0f);
71
72
           glVertex3f(1.0f, -1.0f, -1.0f);
73
        glEnd(); // End of drawing color-cube
74
75
        // Render a pyramid consists of 4 triangles
       glLoadIdentity();
76
                                           // Reset the model-view matrix
77
        qlTranslatef(-1.5f, 0.0f, -6.0f); // Move left and into the screen
78
79
        glBegin(GL TRIANGLES);
                                         // Begin drawing the pyramid with 4 triangles
80
          // Front
81
           glColor3f(1.0f, 0.0f, 0.0f);
                                            // Red
82
           glVertex3f( 0.0f, 1.0f, 0.0f);
           qlColor3f(0.0f, 1.0f, 0.0f);
83
                                            // Green
84
           glVertex3f(-1.0f, -1.0f, 1.0f);
```

```
85
            glColor3f(0.0f, 0.0f, 1.0f);
                                             // Blue
 86
            glVertex3f(1.0f, -1.0f, 1.0f);
 87
 88
            // Right
 89
            glColor3f(1.0f, 0.0f, 0.0f);
                                             // Red
 90
            glVertex3f(0.0f, 1.0f, 0.0f);
 91
            glColor3f(0.0f, 0.0f, 1.0f);
                                             // Blue
 92
            glVertex3f(1.0f, -1.0f, 1.0f);
 93
            glColor3f(0.0f, 1.0f, 0.0f);
                                             // Green
            qlVertex3f(1.0f, -1.0f, -1.0f);
 94
 95
 96
            // Back
 97
            glColor3f(1.0f, 0.0f, 0.0f);
                                             // Red
 98
            glVertex3f(0.0f, 1.0f, 0.0f);
                                             // Green
 99
            glColor3f(0.0f, 1.0f, 0.0f);
100
            glVertex3f(1.0f, -1.0f, -1.0f);
101
            glColor3f(0.0f, 0.0f, 1.0f);
                                             // Blue
102
            glVertex3f(-1.0f, -1.0f, -1.0f);
103
104
            // Left
105
            glColor3f(1.0f,0.0f,0.0f);
                                             // Red
106
            glVertex3f( 0.0f, 1.0f, 0.0f);
                                             // Blue
107
            glColor3f(0.0f,0.0f,1.0f);
108
            glVertex3f(-1.0f,-1.0f,-1.0f);
109
            glColor3f(0.0f,1.0f,0.0f);
                                             // Green
110
            glVertex3f(-1.0f,-1.0f, 1.0f);
111
         glEnd(); // Done drawing the pyramid
112
113
         qlutSwapBuffers(); // Swap the front and back frame buffers (double buffering)
114
     }
115
116
      /* Handler for window re-size event. Called back when the window first appears and
117
         whenever the window is re-sized with its new width and height */
118
      void reshape(GLsizei width, GLsizei height) {    // GLsizei for non-negative integer
119
         // Compute aspect ratio of the new window
120
         if (height == 0) height = 1;
                                                     // To prevent divide by 0
```

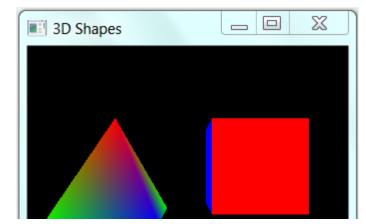
```
121
        GLfloat aspect = (GLfloat)width / (GLfloat)height;
122
123
        // Set the viewport to cover the new window
124
        glViewport(0, 0, width, height);
125
126
        // Set the aspect ratio of the clipping volume to match the viewport
127
        qlMatrixMode(GL PROJECTION); // To operate on the Projection matrix
128
        glLoadIdentity();
                                      // Reset
        // Enable perspective projection with fovy, aspect, zNear and zFar
129
130
        gluPerspective(45.0f, aspect, 0.1f, 100.0f);
131
132
133
      /* Main function: GLUT runs as a console application starting at main() */
134
      int main(int argc, char** argv) {
        glutInit(&argc, argv);
                                          // Initialize GLUT
135
136
        glutInitDisplayMode(GLUT DOUBLE); // Enable double buffered mode
        glutInitWindowSize(640, 480); // Set the window's initial width & height
137
        glutInitWindowPosition(50, 50); // Position the window's initial top-left corner
138
139
        glutCreateWindow(title);
                                   // Create window with the given title
                                        // Register callback handler for window re-paint event
140
        qlutDisplayFunc(display);
141
                                        // Register callback handler for window re-size event
        glutReshapeFunc(reshape);
142
        initGL();
                                        // Our own OpenGL initialization
143
        glutMainLoop();
                                        // Enter the infinite event-processing loop
144
        return 0;
145
    }
```

# GLUT Setup - main()

The program contains a initGL(), display() and reshape() functions.

The main() program:

```
1. glutInit(&argc, argv);
Initializes the GLUT.2. glutInitWindowSize(640, 480);
glutInitWindowPosition(50, 50);
```



## glutCreateWindow(title);

Creates a window with a title, initial width and height positioned at initial top-left corner.

# 3. glutDisplayFunc(display);

Registers display() as the re-paint event handler. That is, the graphics sub-system calls back display() when the window first appears and whenever there is a re-paint request.

# 4. glutReshapeFunc(reshape);

Registers reshape() as the re-sized event handler. That is, the graphics sub-system calls back reshape() when the window first appears and whenever the window is re-sized.

## 5. glutInitDisplayMode(GLUT DOUBLE);

Enables double buffering. In display(), we use glutSwapBuffers() to signal to the GPU to swap the front-buffer and back-buffer during the next VSync (Vertical Synchronization).

#### 6. initGL();

Invokes the initGL() once to perform all one-time initialization tasks.

#### 7. glutMainLoop();

Finally, enters the event-processing loop.

# One-Time Initialization Operations - initGL()

The initGL() function performs the one-time initialization tasks. It is invoked from main() once (and only once).

```
glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Set background color to black and opaque
glClearDepth(1.0f); // Set background depth to farthest
// In display()
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

Set the clearing (background) color to black (R=0, G=0, B=0) and opaque (A=1), and the clearing (background) depth to the farthest (Z=1). In display(), we invoke glClear() to clear the color and depth buffer, with the clearing color and depth, before rendering the graphics. (Besides the color buffer and depth buffer, OpenGL also maintains an accumulation buffer and a stencil buffer which shall be discussed later.)

```
glEnable(GL_DEPTH_TEST); // Enable depth testing for z-culling
glDepthFunc(GL LEQUAL); // Set the type of depth-test
```

We need to enable depth-test to remove the hidden surface, and set the function used for the depth test.

```
glShadeModel(GL SMOOTH); // Enable smooth shading
```

We enable smooth shading in color transition. The alternative is GL FLAT. Try it out and see the difference.

```
glHint(GL PERSPECTIVE CORRECTION HINT, GL NICEST); // Nice perspective corrections
```

In graphics rendering, there is often a trade-off between processing speed and visual quality. We can use glHint() to decide on the trade-off. In this case, we ask for the best perspective correction, which may involve more processing. The default is GL\_DONT\_CARE.

## **Defining the Color-cube and Pyramid**

OpenGL's object is made up of primitives (such as triangle, quad, polygon, point and line). A primitive is defined via one or more vertices. The color-cube is made up of 6 quads. Each quad is made up of 4 vertices, defined in counter-clockwise (CCW) order, such as the normal vector is pointing out, indicating the front face. All the 4 vertices have the same color. The color-cube is defined in its local space (called model space) with origin at the center of the cube with sides of 2 units.

Similarly, the pyramid is made up of 4 triangles (without the base). Each triangle is made up of 3 vertices, defined in CCW order. The 5 vertices of the pyramid are assigned different colors. The color of the triangles are interpolated (and blend smoothly) from its 3 vertices. Again, the pyramid is defined in its local space with origin at the center of the pyramid.

#### **Model Transform**

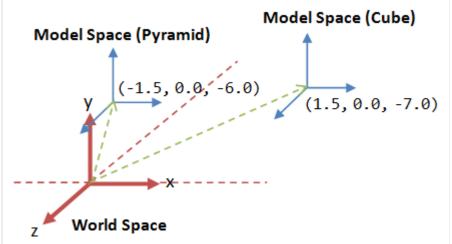
The objects are defined in their local spaces (model spaces). We need to transform them to the common world space, known as *model transform*.

To perform model transform, we need to operate on the so-called *model-view matrix* (OpenGL has a few transformation matrices), by setting the current matrix mode to model-view matrix:

```
glMatrixMode(GL_MODELVIEW); // To operate on model-
view matrix
```

We perform translations on cube and pyramid, respectively, to position them on the world space:

```
// Color-cube
glLoadIdentity(); // Reset model-view matrix
glTranslatef(1.5f, 0.0f, -7.0f); // Move right and into the screen
```



```
// Pyramid
glLoadIdentity();
glTranslatef(-1.5f, 0.0f, -6.0f); // Move left and into the screen
```

#### **View Transform**

The default camera position is:

```
gluLookAt(0.0, 0.0, 0.0, 0.0, -100.0, 0.0, 1.0, 0.0)
```

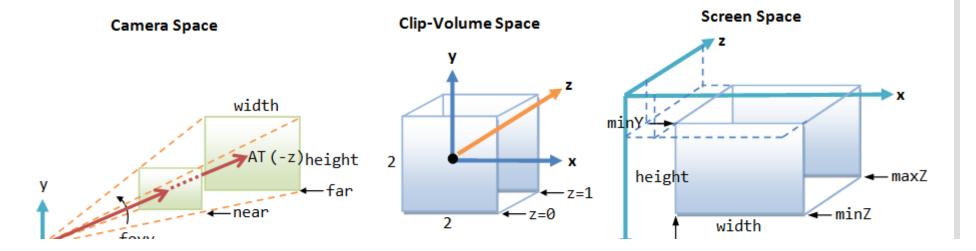
That is, EYE=(0,0,0) at the origin, AT=(0,0,-100) pointing at negative-z axis (into the screen), and UP=(0,1,0) corresponds to y-axis.

OpenGL graphics rendering pipeline performs so-called *view transform* to bring the *world space* to camera's *view space*. In the case of the default camera position, no transform is needed.

#### **Viewport Transform**

```
void reshape(GLsizei width, GLsizei height) {
   glViewport(0, 0, width, height);
```

The graphics sub-system calls back reshape() when the window first appears and whenever the window is resized, given the new window's width and height, in pixels. We set our application viewport to cover the entire window, top-left corner at (0, 0) of width and height, with default minZ of 0 and maxZ of 1. We also use the same aspect ratio of the viewport for the projection view frustum to prevent distortion. In the viewport, a pixel has (x, y) value as well as z-value for depth processing.



View Frustum

Clipping Volume (2x2x1 Cuboid)



Viewport

## **Projection Transform**

```
GLfloat aspect = (GLfloat)width / (GLfloat)height; // Compute aspect ratio of window
glMatrixMode(GL_PROJECTION); // To operate on the Projection matrix
glLoadIdentity(); // Reset
gluPerspective(45.0f, aspect, 0.1f, 100.0f); // Perspective projection: fovy, aspect, near, far
```

A camera has limited field of view. The projection models the view captured by the camera. There are two types of projection: perspective projection and orthographic projection. In perspective projection, object further to the camera appears smaller compared with object of the same size nearer to the camera. In orthographic projection, the objects appear the same regardless of the z-value. Orthographic projection is a special case of perspective projection where the camera is placed very far away. We shall discuss the orthographic projection in the later example.

To set the projection, we need to operate on the projection matrix. (Recall that we operated on the model-view matrix in model transform.)

We set the matrix mode to projection matrix and reset the matrix. We use the gluPerspective() to enable perspective projection, and set the fovy (view angle from the bottom-plane to the top-plane), aspect ratio (width/height), zNear and zFar of the *View Frustum* (truncated pyramid). In this example, we set the fovy to 45°. We use the same aspect ratio as the viewport to avoid distortion. We set the zNear to 0.1 and zFar to 100 (z=-100). Take that note the color-cube (1.5, 0, -7) and the pyramid (-1.5, 0, -6) are contained within the View Frustum.

The *projection transform* transforms the *view frustum* to a 2x2x1 cuboid *clipping-volume* centered on the near plane (z=0). The subsequent *viewport* transform transforms the *clipping-volume* to the *viewport* in screen space. The viewport is set earlier via the glViewport() function.

# 2. Example 2: 3D Shape with Animation (0GL02Animation.cpp)

Let's modify the previous example to carry out animation (rotating the cube and pyramid).

```
1  /*
2  * OGL02Animation.cpp: 3D Shapes with animation
3  */
4  #include <windows.h> // for MS Windows
5  #include <GL/glut.h> // GLUT, include glu.h and gl.h
6
```

```
/* Global variables */
    char title[] = "3D Shapes with animation";
    GLfloat anglePyramid = 0.0f; // Rotational angle for pyramid [NEW]
    GLfloat angleCube = 0.0f; // Rotational angle for cube [NEW]
10
    int refreshMills = 15;
                                  // refresh interval in milliseconds [NEW]
11
12
    /* Initialize OpenGL Graphics */
13
14
    void initGL() {
15
       glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Set background color to black and opaque
                                             // Set background depth to farthest
16
       glClearDepth(1.0f);
17
       glEnable(GL DEPTH TEST); // Enable depth testing for z-culling
18
       glDepthFunc(GL LEQUAL);
                                // Set the type of depth-test
19
       glShadeModel(GL SM00TH); // Enable smooth shading
20
       glHint(GL PERSPECTIVE CORRECTION HINT, GL NICEST); // Nice perspective corrections
21
    }
22
23
    /* Handler for window-repaint event. Called back when the window first appears and
       whenever the window needs to be re-painted. */
24
    void display() {
25
26
       glClear(GL COLOR BUFFER BIT | GL DEPTH BUFFER BIT); // Clear color and depth buffers
27
       qlMatrixMode(GL MODELVIEW); // To operate on model-view matrix
28
29
       // Render a color-cube consisting of 6 quads with different colors
30
       glLoadIdentity();
                                         // Reset the model-view matrix
31
       glTranslatef(1.5f, 0.0f, -7.0f); // Move right and into the screen
32
       glRotatef(angleCube, 1.0f, 1.0f, 1.0f); // Rotate about (1,1,1)-axis [NEW]
33
34
       glBegin(GL QUADS);
                                         // Begin drawing the color cube with 6 guads
          // Top face (v = 1.0f)
35
36
          // Define vertices in counter-clockwise (CCW) order with normal pointing out
37
          glColor3f(0.0f, 1.0f, 0.0f);
                                          // Green
38
          glVertex3f( 1.0f, 1.0f, -1.0f);
39
          glVertex3f(-1.0f, 1.0f, -1.0f);
40
          glVertex3f(-1.0f, 1.0f, 1.0f);
41
          glVertex3f( 1.0f, 1.0f, 1.0f);
42
```

```
43
           // Bottom face (y = -1.0f)
           glColor3f(1.0f, 0.5f, 0.0f);
44
                                            // Orange
45
           glVertex3f( 1.0f, -1.0f, 1.0f);
           glVertex3f(-1.0f, -1.0f, 1.0f);
46
           glVertex3f(-1.0f, -1.0f, -1.0f);
47
           glVertex3f( 1.0f, -1.0f, -1.0f);
48
49
50
           // Front face (z = 1.0f)
51
           glColor3f(1.0f, 0.0f, 0.0f);
                                            // Red
52
           glVertex3f( 1.0f, 1.0f, 1.0f);
53
           qlVertex3f(-1.0f, 1.0f, 1.0f);
54
           glVertex3f(-1.0f, -1.0f, 1.0f);
55
           glVertex3f( 1.0f, -1.0f, 1.0f);
56
57
           // Back face (z = -1.0f)
58
           glColor3f(1.0f, 1.0f, 0.0f);
                                            // Yellow
           glVertex3f( 1.0f, -1.0f, -1.0f);
59
60
           glVertex3f(-1.0f, -1.0f, -1.0f);
           glVertex3f(-1.0f, 1.0f, -1.0f);
61
62
           glVertex3f( 1.0f, 1.0f, -1.0f);
63
64
           // Left face (x = -1.0f)
65
           glColor3f(0.0f, 0.0f, 1.0f);
                                            // Blue
66
           glVertex3f(-1.0f, 1.0f, 1.0f);
67
           qlVertex3f(-1.0f, 1.0f, -1.0f);
68
           glVertex3f(-1.0f, -1.0f, -1.0f);
           glVertex3f(-1.0f, -1.0f, 1.0f);
69
70
71
           // Right face (x = 1.0f)
72
           glColor3f(1.0f, 0.0f, 1.0f);
                                            // Magenta
73
           glVertex3f(1.0f, 1.0f, -1.0f);
74
           glVertex3f(1.0f, 1.0f, 1.0f);
75
           glVertex3f(1.0f, -1.0f, 1.0f);
76
           glVertex3f(1.0f, -1.0f, -1.0f);
77
        glEnd(); // End of drawing color-cube
78
```

```
79
         // Render a pyramid consists of 4 triangles
 80
         glLoadIdentity();
                                            // Reset the model-view matrix
 81
         glTranslatef(-1.5f, 0.0f, -6.0f); // Move left and into the screen
 82
         glRotatef(anglePyramid, 1.0f, 1.0f, 0.0f); // Rotate about the (1,1,0)-axis [NEW]
 83
 84
         glBegin(GL TRIANGLES);
                                          // Begin drawing the pyramid with 4 triangles
 85
            // Front
 86
            glColor3f(1.0f, 0.0f, 0.0f);
                                             // Red
 87
            glVertex3f( 0.0f, 1.0f, 0.0f);
 88
            glColor3f(0.0f, 1.0f, 0.0f);
                                             // Green
 89
            glVertex3f(-1.0f, -1.0f, 1.0f);
 90
            glColor3f(0.0f, 0.0f, 1.0f);
                                             // Blue
 91
            qlVertex3f(1.0f, -1.0f, 1.0f);
 92
 93
            // Right
 94
            glColor3f(1.0f, 0.0f, 0.0f);
                                             // Red
 95
            glVertex3f(0.0f, 1.0f, 0.0f);
            glColor3f(0.0f, 0.0f, 1.0f);
 96
                                             // Blue
            glVertex3f(1.0f, -1.0f, 1.0f);
 97
 98
            glColor3f(0.0f, 1.0f, 0.0f);
                                             // Green
 99
            qlVertex3f(1.0f, -1.0f, -1.0f);
100
101
            // Back
            qlColor3f(1.0f, 0.0f, 0.0f);
                                             // Red
102
103
            glVertex3f(0.0f, 1.0f, 0.0f);
104
            glColor3f(0.0f, 1.0f, 0.0f);
                                             // Green
105
            glVertex3f(1.0f, -1.0f, -1.0f);
106
            glColor3f(0.0f, 0.0f, 1.0f);
                                             // Blue
107
            glVertex3f(-1.0f, -1.0f, -1.0f);
108
109
            // Left
            qlColor3f(1.0f,0.0f,0.0f);
                                             // Red
110
111
            glVertex3f( 0.0f, 1.0f, 0.0f);
112
            glColor3f(0.0f,0.0f,1.0f);
                                             // Blue
113
            qlVertex3f(-1.0f,-1.0f,-1.0f);
114
            glColor3f(0.0f,1.0f,0.0f);
                                             // Green
```

```
115
           glVertex3f(-1.0f,-1.0f, 1.0f);
116
        glEnd(); // Done drawing the pyramid
117
        glutSwapBuffers(); // Swap the front and back frame buffers (double buffering)
118
119
120
        // Update the rotational angle after each refresh [NEW]
121
        anglePyramid += 0.2f;
122
        angleCube -= 0.15f;
123
     }
124
125
      /* Called back when timer expired [NEW] */
126
      void timer(int value) {
127
        glutPostRedisplay();
                                  // Post re-paint request to activate display()
128
        glutTimerFunc(refreshMills, timer, 0); // next timer call milliseconds later
     }
129
130
131
      /* Handler for window re-size event. Called back when the window first appears and
        whenever the window is re-sized with its new width and height */
132
      void reshape(GLsizei width, GLsizei height) {    // GLsizei for non-negative integer
133
134
        // Compute aspect ratio of the new window
135
        if (height == 0) height = 1;
                                                    // To prevent divide by 0
136
        GLfloat aspect = (GLfloat)width / (GLfloat)height;
137
138
        // Set the viewport to cover the new window
139
        glViewport(0, 0, width, height);
140
141
        // Set the aspect ratio of the clipping volume to match the viewport
        qlMatrixMode(GL PROJECTION); // To operate on the Projection matrix
142
        alLoadIdentity():
                                      // Reset
143
144
        // Enable perspective projection with fovy, aspect, zNear and zFar
        gluPerspective(45.0f, aspect, 0.1f, 100.0f);
145
146
     }
147
     /* Main function: GLUT runs as a console application starting at main() */
148
     int main(int argc, char** argv) {
149
150
        glutInit(&argc, argv);
                                         // Initialize GLUT
```

```
151
        glutInitDisplayMode(GLUT DOUBLE); // Enable double buffered mode
152
        glutInitWindowSize(640, 480); // Set the window's initial width & height
153
        glutInitWindowPosition(50, 50); // Position the window's initial top-left corner
        glutCreateWindow(title);  // Create window with the given title
154
155
        glutDisplayFunc(display);
                                        // Register callback handler for window re-paint event
        glutReshapeFunc(reshape);
                                        // Register callback handler for window re-size event
156
157
        initGL();
                                        // Our own OpenGL initialization
                                       // First timer call immediately [NEW]
158
        glutTimerFunc(0, timer, 0);
        glutMainLoop();
                                        // Enter the infinite event-processing loop
159
        return 0;
160
161 }
```

The new codes are:

```
GLfloat anglePyramid = 0.0f; // Rotational angle for pyramid [NEW]
GLfloat angleCube = 0.0f; // Rotational angle for cube [NEW]
int refreshMills = 15; // refresh interval in milliseconds [NEW]
We define two global variables to keep track of the current rotational angles of the cube and pyramid. We also define the
```

We define two global variables to keep track of the current rotational angles of the cube and pyramid. We also define the refresh period as 15 msec (66 frames per second).

```
void timer(int value) {
   glutPostRedisplay(); // Post re-paint request to activate display()
   glutTimerFunc(refreshMills, timer, 0); // next timer call milliseconds later
}
```

To perform animation, we define a function called timer(), which posts a re-paint request to activate display() when the timer expired, and then run the timer again. In main(), we perform the first timer() call via glutTimerFunc(0, timer, 0).

```
glRotatef(angleCube, 1.0f, 1.0f, 1.0f); // Rotate the cube about (1,1,1)-axis [NEW]
.....
glRotatef(anglePyramid, 1.0f, 1.0f, 0.0f); // Rotate about the (1,1,0)-axis [NEW]
.....
anglePyramid += 0.2f; // update pyramid's angle
angleCube -= 0.15f; // update cube's angle
```

In display(), we rotate the cube and pyramid based on their rotational angles, and update the angles after each refresh.

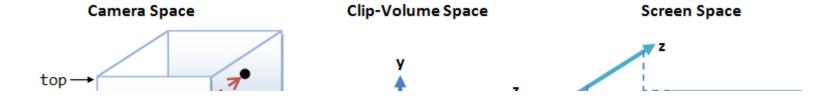
# 3. Example 3: Orthographic Projection (OGL030rthographic.cpp)

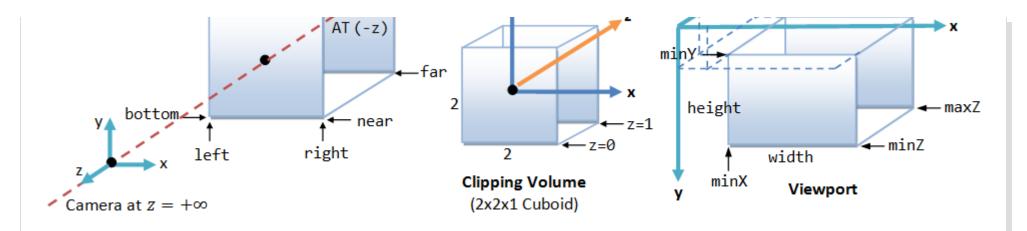
As mentioned, OpenGL support two type of projections: perspective and orthographic. In orthographic projection, an object appears to be the same size regardless of the depth. Orthographic is a special case of perspective projection, where the camera is placed very far away.

To use orthographic projection, change the reshape() function to invoke glortho().

```
void reshape(GLsizei width, GLsizei height) { // GLsizei for non-negative integer
   // Compute aspect ratio of the new window
   if (height == 0) height = 1;
                                               // To prevent divide by 0
   GLfloat aspect = (GLfloat)width / (GLfloat)height;
   // Set the viewport to cover the new window
   glViewport(0, 0, width, height);
   // Set the aspect ratio of the clipping volume to match the viewport
   qlMatrixMode(GL PROJECTION); // To operate on the Projection matrix
   glLoadIdentity();
                                 // Reset
   // Set up orthographic projection view [NEW]
   if (width >= height) {
     // aspect >= 1, set the height from -1 to 1, with larger width
      glOrtho(-3.0 * aspect, 3.0 * aspect, -3.0, 3.0, 0.1, 100);
   } else {
      // aspect < 1, set the width to -1 to 1, with larger height
     qlortho(-3.0, 3.0, -3.0 / aspect, 3.0 / aspect, 0.1, 100);
```

In this example, we set the cross-section of view-volume according to the aspect ratio of the viewport, and depth from 0.1 to 100, corresponding to z=-0.1 to z=-100. Take note that the cube and pyramid are contained within the view-volume.





# 4. Example 4: Vertex Array

In the earlier example, drawing a cube requires at least 24 glVertex functions and a pair of glBegin and glEnd. Function calls may involve high overhead and hinder the performance. Furthermore, each vertex is specified and processed three times.

Link to OpenGL/Computer Graphics References and Resources

Latest version tested: ??? Last modified: May, 2012

Feedback, comments, corrections, and errata can be sent to Chua Hock-Chuan (ehchua@ntu.edu.sg) | <u>HOME</u>