5. Translation & Rotation

In the above sample, we positioned each of the shapes by defining their vertices with respective to the *same* origin (called *world space*). It took me quite a while to figure out the absolute coordinates of these vertices.

Instead, we could position each of the shapes by defining their vertices with respective to their own center (called *model space* or *local space*). We can then use translation and/or rotation to position the shapes at the desired locations in the world space, as shown in the following revised `display()` function.

5.1 Example 4: Translation and Rotation (GL04ModelTransform.cpp)

```
/*
 * GL04ModelTransform.cpp: Model Transform - Translation and Rotation
 * Transform primitives from their model spaces to world space.
 */
#include <windows.h>  // for MS Windows
#include <GL/glut.h>  // GLUT, include glu.h and gl.h

/* Initialize OpenGL Graphics */
void initGL() {
   // Set "clearing" or background color
   glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Black and opaque
}

/* Handler for window-repaint event. Call back when the window first appears and
   whenever the window needs to be re-painted. */
void display() {
   glClear(GL_COLOR_BUFFER_BIT);    // Clear the color buffer
   glMatrixMode(GL_MODELVIEW);      // To operate on Model-View matrix
   glLoadIdentity();                // Reset the model-view matrix

   glTranslatef(-0.5f, 0.4f, 0.0f); // Translate left and up
   glBegin(GL_QUADS);               // Each set of 4 vertices form a quad
      glColor3f(1.0f, 0.0f, 0.0f);  // Red
      glVertex2f(-0.3f, -0.3f);     // Define vertices in counter-clockwise (CCW) order
      glVertex2f( 0.3f, -0.3f);     //  so that the normal (front-face) is facing you
      glVertex2f( 0.3f,  0.3f);
      glVertex2f(-0.3f,  0.3f);
   glEnd();

   glTranslatef(0.1f, -0.7f, 0.0f); // Translate right and down
   glBegin(GL_QUADS);               // Each set of 4 vertices form a quad
      glColor3f(0.0f, 1.0f, 0.0f); // Green
      glVertex2f(-0.3f, -0.3f);
      glVertex2f( 0.3f, -0.3f);
      glVertex2f( 0.3f,  0.3f);
      glVertex2f(-0.3f,  0.3f);
   glEnd();

   glTranslatef(-0.3f, -0.2f, 0.0f); // Translate left and down
   glBegin(GL_QUADS);                // Each set of 4 vertices form a quad
      glColor3f(0.2f, 0.2f, 0.2f); // Dark Gray
      glVertex2f(-0.2f, -0.2f);
      glColor3f(1.0f, 1.0f, 1.0f); // White
      glVertex2f( 0.2f, -0.2f);
      glColor3f(0.2f, 0.2f, 0.2f); // Dark Gray
      glVertex2f( 0.2f,  0.2f);
      glColor3f(1.0f, 1.0f, 1.0f); // White
      glVertex2f(-0.2f,  0.2f);
   glEnd();

   glTranslatef(1.1f, 0.2f, 0.0f); // Translate right and up
   glBegin(GL_TRIANGLES);          // Each set of 3 vertices form a triangle
      glColor3f(0.0f, 0.0f, 1.0f); // Blue
      glVertex2f(-0.3f, -0.2f);
      glVertex2f( 0.3f, -0.2f);
      glVertex2f( 0.0f,  0.3f);
   glEnd();

   glTranslatef(0.2f, -0.3f, 0.0f);     // Translate right and down
   glRotatef(180.0f, 0.0f, 0.0f, 1.0f); // Rotate 180 degree
      glBegin(GL_TRIANGLES);               // Each set of 3 vertices form a triangle
      glColor3f(1.0f, 0.0f, 0.0f); // Red
      glVertex2f(-0.3f, -0.2f);
      glColor3f(0.0f, 1.0f, 0.0f); // Green
      glVertex2f( 0.3f, -0.2f);
      glColor3f(0.0f, 0.0f, 1.0f); // Blue
      glVertex2f( 0.0f,  0.3f);
   glEnd();

   glRotatef(-180.0f, 0.0f, 0.0f, 1.0f); // Undo previous rotate
   glTranslatef(-0.1f, 1.0f, 0.0f);      // Translate right and down
   glBegin(GL_POLYGON);                  // The vertices form one closed polygon
      glColor3f(1.0f, 1.0f, 0.0f); // Yellow
      glVertex2f(-0.1f, -0.2f);
      glVertex2f( 0.1f, -0.2f);
      glVertex2f( 0.2f,  0.0f);
      glVertex2f( 0.1f,  0.2f);
      glVertex2f(-0.1f,  0.2f);
      glVertex2f(-0.2f,  0.0f);
   glEnd();

   glFlush();   // Render now
}

/* 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 */
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 area to match the viewport
   glMatrixMode(GL_PROJECTION);  // To operate on the Projection matrix
   glLoadIdentity();
   if (width >= height) {
     // aspect >= 1, set the height from -1 to 1, with larger width
      gluOrtho2D(-1.0 * aspect, 1.0 * aspect, -1.0, 1.0);
   } else {
      // aspect < 1, set the width to -1 to 1, with larger height
     gluOrtho2D(-1.0, 1.0, -1.0 / aspect, 1.0 / aspect);
   }
}

/* Main function: GLUT runs as a console application starting at main() */
int main(int argc, char** argv) {
   glutInit(&argc, argv);          // Initialize GLUT
   glutInitWindowSize(640, 480);   // Set the window's initial width & height - non-squa
   glutInitWindowPosition(50, 50); // Position the window's initial top-left corner
   glutCreateWindow("Model Transform");  // Create window with the given title
   glutDisplayFunc(display);       // Register callback handler for window re-paint even
   glutReshapeFunc(reshape);       // Register callback handler for window re-size event
   initGL();                       // Our own OpenGL initialization
   glutMainLoop();                 // Enter the infinite event-processing loop
   return 0;
}
```

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

Translation and rotation are parts of so-called *model transform*, which transform from the objects from the local space (or model space) to the common world space. To carry out model transform, we set the matrix mode to mode-view matrix (`GL_MODELVIEW`) and reset the matrix. (Recall that in the previous example, we set the matrix mode to projection matrix (`GL_PROJECTION`) to set the clipping area.)

OpenGL is operating as a state machine. That is, once a state is set, the value of the state persists until it is changed. In other words, once the coordinates are translated or rotated, all the subsequent operations will be based on this coordinates.

Translation is done via `glTranslate` function:

```
void gltranslatef (GLfloat x, GLfloat y, GLfloat z)
   // where (x, y, z) is the translational vector
```

Take note that `glTranslatef` function must be placed outside the `glBegin`/`glEnd`, where as `glColor` can be placed inside `glBegin`/`glEnd`.
Rotation is done via `glRotatef` function:

```
void glRotatef (GLfloat angle, GLfloat x, GLfloat y, GLfloat z)
   // where angle specifies the rotation in degree, (x, y, z) forms the axis of
rotation.
```

Take note that the rotational angle is measured in degrees (instead of radians) in OpenGL.

In the above example, we translate within the x-y plane (z=0) and rotate about the z-axis (which is normal to the x-y plane).