```
/*****************************************************************
 * Handout: rotate-cube-new.cpp (A Sample Code for Shader-Based OpenGL ---
                                 for OpenGL version 3.1 and later)
 * Originally from Ed Angel's textbook "Interactive Computer Graphics" 6th Ed
             sample code "example3.cpp" of Chapter 4.
 * Moodified by Yi-Jen Chiang to include the use of a general rotation function
   Rotate(angle, x, y, z), where the vector (x, y, z) can have length != 1.0,
   and also to include the use of the function NormalMatrix(mv) to return the
   normal matrix (mat3) of a given model-view matrix mv (mat4).

   (The functions Rotate() and NormalMatrix() are added to the file "mat-yjc-new.h"
   by Yi-Jen Chiang, where a new and correct transpose function "transpose1()" and
   other related functions such as inverse(m) for the inverse of 3x3 matrix m are
   also added; see the file "mat-yjc-new.h".)

 * Extensively modified by Yi-Jen Chiang for the program structure and user
   interactions. See the function keyboard() for the keyboard actions.
   Also extensively re-structured by Yi-Jen Chiang to create and use the new
   function drawObj() so that it is easier to draw multiple objects. Now a floor
   and a rotating cube are drawn.

** Perspective view of a color cube using LookAt() and Perspective()

** Colors are assigned to each vertex and then the rasterizer interpolates
   those colors across the triangles.
*****************************************************************/
#include "Angel-yjc.h"

typedef Angel::vec3  color3;
typedef Angel::vec3  point3;

GLuint Angel::InitShader(const char* vShaderFile, const char* fShaderFile);

GLuint program;       /* shader program object id */
GLuint cube_buffer;   /* vertex buffer object id for cube */
GLuint floor_buffer;  /* vertex buffer object id for floor */

// Projection transformation parameters
GLfloat  fovy = 45.0;  // Field-of-view in Y direction angle (in degrees)
GLfloat  aspect;       // Viewport aspect ratio
GLfloat  zNear = 0.5, zFar = 3.0;

GLfloat angle = 0.0; // rotation angle in degrees
vec4 init_eye(3.0, 2.0, 0.0, 1.0); // initial viewer position
vec4 eye = init_eye;               // current viewer position

int animationFlag = 1; // 1: animation; 0: non-animation. Toggled by key 'a' or 'A'

int cubeFlag = 1;   // 1: solid cube; 0: wireframe cube. Toggled by key 'c' or 'C'
int floorFlag = 1;  // 1: solid floor; 0: wireframe floor. Toggled by key 'f' or 'F'

const int cube_NumVertices = 36; //(6 faces)*(2 triangles/face)*(3 vertices/triangle)
#if 0
point3 cube_points[cube_NumVertices]; // positions for all vertices
color3 cube_colors[cube_NumVertices]; // colors for all vertices
#endif
#if 1
point3 cube_points[100];
color3 cube_colors[100];
#endif

const int floor_NumVertices = 6; //(1 face)*(2 triangles/face)*(3 vertices/triangle)
point3 floor_points[floor_NumVertices]; // positions for all vertices
color3 floor_colors[floor_NumVertices]; // colors for all vertices

// Vertices of a unit cube centered at origin, sides aligned with axes
point3 vertices[8] = {
    point3( -0.5, -0.5,  0.5),
    point3( -0.5,  0.5,  0.5),
    point3(  0.5,  0.5,  0.5),
    point3(  0.5, -0.5,  0.5),
    point3( -0.5, -0.5, -0.5),
    point3( -0.5,  0.5, -0.5),
    point3(  0.5,  0.5, -0.5),
    point3(  0.5, -0.5, -0.5)
};
// RGBA colors
color3 vertex_colors[8] = {
    color3( 0.0, 0.0, 0.0),  // black
    color3( 1.0, 0.0, 0.0),  // red
    color3( 1.0, 1.0, 0.0),  // yellow
    color3( 0.0, 1.0, 0.0),  // green
    color3( 0.0, 0.0, 1.0),  // blue
    color3( 1.0, 0.0, 1.0),  // magenta
    color3( 1.0, 1.0, 1.0),  // white
    color3( 0.0, 1.0, 1.0)   // cyan
};
//----------------------------------------------------------------------------
int Index = 0; // YJC: This must be a global variable since quad() is called
               //      multiple times and Index should then go up to 36 for
               //      the 36 vertices and colors

// quad(): generate two triangles for each face and assign colors to the vertices
void quad( int a, int b, int c, int d )
{
    cube_colors[Index] = vertex_colors[a]; cube_points[Index] = vertices[a]; Index++;
    cube_colors[Index] = vertex_colors[b]; cube_points[Index] = vertices[b]; Index++;
    cube_colors[Index] = vertex_colors[c]; cube_points[Index] = vertices[c]; Index++;

    cube_colors[Index] = vertex_colors[c]; cube_points[Index] = vertices[c]; Index++;
    cube_colors[Index] = vertex_colors[d]; cube_points[Index] = vertices[d]; Index++;
    cube_colors[Index] = vertex_colors[a]; cube_points[Index] = vertices[a]; Index++;
}
//----------------------------------------------------------------------------
// generate 12 triangles: 36 vertices and 36 colors
void colorcube()
{
    quad( 1, 0, 3, 2 );
    quad( 2, 3, 7, 6 );
    quad( 3, 0, 4, 7 );
    quad( 6, 5, 1, 2 );
    quad( 4, 5, 6, 7 );
    quad( 5, 4, 0, 1 );
}
//-----------------------------
// generate 2 triangles: 6 vertices and 6 colors
void floor()
{
    floor_colors[0] = vertex_colors[3]; floor_points[0] = vertices[3];
    floor_colors[1] = vertex_colors[0]; floor_points[1] = vertices[0];
    floor_colors[2] = vertex_colors[4]; floor_points[2] = vertices[4];

    floor_colors[3] = vertex_colors[4]; floor_points[3] = vertices[4];
    floor_colors[4] = vertex_colors[7]; floor_points[4] = vertices[7];
    floor_colors[5] = vertex_colors[3]; floor_points[5] = vertices[3];
}
//----------------------------------------------------------------------------
// OpenGL initialization
void init()
{
    colorcube();
```

```
#if 0 //YJC: The following is not needed
    // Create a vertex array object
    GLuint vao;
    glGenVertexArrays( 1, &vao );
    glBindVertexArray( vao );
#endif

 // Create and initialize a vertex buffer object for cube, to be used in display()
    glGenBuffers(1, &cube_buffer);
    glBindBuffer(GL_ARRAY_BUFFER, cube_buffer);

#if 0
    glBufferData(GL_ARRAY_BUFFER, sizeof(cube_points) + sizeof(cube_colors),
                 NULL, GL_STATIC_DRAW);
    glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(cube_points), cube_points);
    glBufferSubData(GL_ARRAY_BUFFER, sizeof(cube_points), sizeof(cube_colors),
                    cube_colors);
#endif
#if 1
    glBufferData(GL_ARRAY_BUFFER,
                 sizeof(point3)*cube_NumVertices + sizeof(color3)*cube_NumVertices,
                 NULL, GL_STATIC_DRAW);
    glBufferSubData(GL_ARRAY_BUFFER, 0,
                    sizeof(point3) * cube_NumVertices, cube_points);
    glBufferSubData(GL_ARRAY_BUFFER,
                    sizeof(point3) * cube_NumVertices,
                    sizeof(color3) * cube_NumVertices,
                    cube_colors);
#endif

    floor();
 // Create and initialize a vertex buffer object for floor, to be used in display()
    glGenBuffers(1, &floor_buffer);
    glBindBuffer(GL_ARRAY_BUFFER, floor_buffer);
    glBufferData(GL_ARRAY_BUFFER, sizeof(floor_points) + sizeof(floor_colors),
                 NULL, GL_STATIC_DRAW);
    glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(floor_points), floor_points);
    glBufferSubData(GL_ARRAY_BUFFER, sizeof(floor_points), sizeof(floor_colors),
                    floor_colors);

 // Load shaders and create a shader program (to be used in display())
    program = InitShader("vshader42.glsl", "fshader42.glsl");

    glEnable( GL_DEPTH_TEST );
    glClearColor( 0.0, 0.0, 0.0, 1.0 );
    glLineWidth(2.0);
}
//----------------------------------------------------------------------------
// drawObj(buffer, num_vertices):
//   draw the object that is associated with the vertex buffer object "buffer"
//   and has "num_vertices" vertices.
//
void drawObj(GLuint buffer, int num_vertices)
{
    //--- Activate the vertex buffer object to be drawn ---//
    glBindBuffer(GL_ARRAY_BUFFER, buffer);

    /*----- Set up vertex attribute arrays for each vertex attribute -----*/
    GLuint vPosition = glGetAttribLocation(program, "vPosition");
    glEnableVertexAttribArray(vPosition);
    glVertexAttribPointer(vPosition, 3, GL_FLOAT, GL_FALSE, 0,
                          BUFFER_OFFSET(0) );

    GLuint vColor = glGetAttribLocation(program, "vColor");
    glEnableVertexAttribArray(vColor);
    glVertexAttribPointer(vColor, 3, GL_FLOAT, GL_FALSE, 0,
                          BUFFER_OFFSET(sizeof(point3) * num_vertices) );
      // the offset is the (total) size of the previous vertex attribute array(s)

    /* Draw a sequence of geometric objs (triangles) from the vertex buffer
       (using the attributes specified in each enabled vertex attribute array) */
    glDrawArrays(GL_TRIANGLES, 0, num_vertices);

    /*--- Disable each vertex attribute array being enabled ---*/
    glDisableVertexAttribArray(vPosition);
    glDisableVertexAttribArray(vColor);
}
//----------------------------------------------------------------------------
void display( void )
{
  GLuint  model_view;  // model-view matrix uniform shader variable location
  GLuint  projection;  // projection matrix uniform shader variable location

    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );

    glUseProgram(program); // Use the shader program

    model_view = glGetUniformLocation(program, "model_view" );
    projection = glGetUniformLocation(program, "projection" );

/*---  Set up and pass on Projection matrix to the shader ---*/
    mat4  p = Perspective(fovy, aspect, zNear, zFar);
    glUniformMatrix4fv(projection, 1, GL_TRUE, p); // GL_TRUE: matrix is row-major

/*---  Set up and pass on Model-View matrix to the shader ---*/
    // eye is a global variable of vec4 set to init_eye and updated by keyboard()
    vec4    at(0.0, 0.0, 0.0, 1.0);
    vec4    up(0.0, 1.0, 0.0, 0.0);

    mat4  mv = LookAt(eye, at, up);

/*----- Set Up the Model-View matrix for the cube -----*/
#if 0 // The following is to verify the correctness of the function NormalMatrix():
      // Commenting out Rotate() and un-commenting mat4WithUpperLeftMat3()
      // gives the same result.
      mv = mv * Translate(0.0, 0.5, 0.0) * Scale (1.4, 1.4, 1.4)
              * Rotate(angle, 0.0, 0.0, 2.0);
            // * mat4WithUpperLeftMat3(NormalMatrix(Rotate(angle, 0.0, 0.0, 2.0), 1));
#endif
#if 1 // The following is to verify that Rotate() about (0,2,0) is RotateY():
      // Commenting out Rotate() and un-commenting RotateY()
      // gives the same result.
  //
  // The set-up below gives a new scene (scene 2), using Correct LookAt().
      mv = mv * Translate(0.0, 0.5, 0.0) * Scale (1.4, 1.4, 1.4)
              * Rotate(angle, 0.0, 2.0, 0.0);
            // * RotateY(angle);
  //
  // The set-up below gives the original scene (scene 1), using Correct LookAt().
  //  mv = Translate(0.0, 0.5, 0.0) * mv * Scale (1.4, 1.4, 1.4)
  //              * Rotate(angle, 0.0, 2.0, 0.0);
  //            // * RotateY(angle);
  //
  // The set-up below gives the original scene (scene 1), when using previously
  //     Incorrect LookAt() (= Translate(1.0, 1.0, 0.0) * correct LookAt() )
  //  mv = Translate(-1.0, -0.5, 0.0) * mv * Scale (1.4, 1.4, 1.4)
  //              * Rotate(angle, 0.0, 2.0, 0.0);
  //            // * RotateY(angle);
  //
#endif
#if 0  // The following is to verify that Rotate() about (3,0,0) is RotateX():
       // Commenting out Rotate() and un-commenting RotateX()
```

```
      // gives the same result.
      mv = mv * Translate(0.0, 0.5, 0.0) * Scale (1.4, 1.4, 1.4)
                    * Rotate(angle, 3.0, 0.0, 0.0);
                  // * RotateX(angle);
#endif
    glUniformMatrix4fv(model_view, 1, GL_TRUE, mv); // GL_TRUE: matrix is row-major
    if (cubeFlag == 1) // Filled cube
       glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    else               // Wireframe cube
       glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    drawObj(cube_buffer, cube_NumVertices);  // draw the cube

/*----- Set up the Mode-View matrix for the floor -----*/
 // The set-up below gives a new scene (scene 2), using Correct LookAt() function
       mv = LookAt(eye, at, up) * Translate(0.3, 0.0, 0.0) * Scale (1.6, 1.5, 3.3);
 //
 // The set-up below gives the original scene (scene 1), using Correct LookAt()
 //    mv = Translate(0.0, 0.0, 0.3) * LookAt(eye, at, up) * Scale (1.6, 1.5, 3.3);
 //
 // The set-up below gives the original scene (scene 1), when using previously
 //       Incorrect LookAt() (= Translate(1.0, 1.0, 0.0) * correct LookAt() )
 //    mv = Translate(-1.0, -1.0, 0.3) * LookAt(eye, at, up) * Scale (1.6, 1.5, 3.3);
 //
    glUniformMatrix4fv(model_view, 1, GL_TRUE, mv); // GL_TRUE: matrix is row-major
    if (floorFlag == 1) // Filled floor
       glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    else               // Wireframe floor
       glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    drawObj(floor_buffer, floor_NumVertices);  // draw the floor

    glutSwapBuffers();
}
//----------------------------------------------------------------------------
void idle (void)
{
    //angle += 0.02;
    angle += 1.0;    //YJC: change this value to adjust the cube rotation speed.
    glutPostRedisplay();
}
//----------------------------------------------------------------------------
void keyboard(unsigned char key, int x, int y)
{
    switch(key) {
        case 033: // Escape Key
        case 'q': case 'Q':
            exit( EXIT_SUCCESS );
            break;

        case 'X': eye[0] += 1.0; break;
        case 'x': eye[0] -= 1.0; break;
        case 'Y': eye[1] += 1.0; break;
        case 'y': eye[1] -= 1.0; break;
        case 'Z': eye[2] += 1.0; break;
        case 'z': eye[2] -= 1.0; break;

        case 'a': case 'A': // Toggle between animation and non-animation
            animationFlag = 1 -  animationFlag;
            if (animationFlag == 1) glutIdleFunc(idle);
            else                    glutIdleFunc(NULL);
            break;

        case 'c': case 'C': // Toggle between filled and wireframe cube
            cubeFlag = 1 -  cubeFlag;
            break;

        case 'f': case 'F': // Toggle between filled and wireframe floor
            floorFlag = 1 -  floorFlag;
            break;

        case ' ':  // reset to initial viewer/eye position
            eye = init_eye;
            break;
    }
    glutPostRedisplay();
}
//----------------------------------------------------------------------------
void reshape(int width, int height)
{
    glViewport(0, 0, width, height);
    aspect = (GLfloat) width  / (GLfloat) height;
    glutPostRedisplay();
}
//----------------------------------------------------------------------------
int main( int argc, char **argv )
{
    glutInit(&argc, argv);
#ifdef __APPLE__ // Enable core profile of OpenGL 3.2 on macOS.
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH | GLUT_3_2_CORE_PROFILE);
#else
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH);
#endif
    glutInitWindowSize(512, 512);
    glutCreateWindow("Color Cube");

#ifdef __APPLE__ // on macOS
    // Core profile requires to create a Vertex Array Object (VAO).
    GLuint vao;
    glGenVertexArrays(1, &vao);
    glBindVertexArray(vao);
#else           // on Linux or Windows, we still need glew
    /* Call glewInit() and error checking */
    int err = glewInit();
    if (GLEW_OK != err)
    {
        printf("Error: glewInit failed: %s\n", (char*) glewGetErrorString(err));
        exit(1);
    }
#endif

    // Get info of GPU and supported OpenGL version
    printf("Renderer: %s\n", glGetString(GL_RENDERER));
    printf("OpenGL version supported %s\n", glGetString(GL_VERSION));

    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutIdleFunc(idle);
    glutKeyboardFunc(keyboard);

    init();
    glutMainLoop();
    return 0;
}
```

```
/*****************************
 * File: vshader42.glsl:
 *   A simple vertex shader.
 *
 * - Vertex attributes (positions & colors) for all vertices are sent
 *   to the GPU via a vertex buffer object created in the OpenGL program.
 *
 * - This vertex shader uses the Model-View and Projection matrices passed
 *   on from the OpenGL program as uniform variables of type mat4.
 *****************************/

// #version 150  // YJC: Comment/un-comment this line to resolve compilation errors
                 //      due to different settings of the default GLSL version

in  vec3 vPosition;
in  vec3 vColor;
out vec4 color;

uniform mat4 model_view;
uniform mat4 projection;

void main()
{
vec4 vPosition4 = vec4(vPosition.x, vPosition.y, vPosition.z, 1.0);
vec4 vColor4 = vec4(vColor.r, vColor.g, vColor.b, 1.0);

    // YJC: Original, incorrect below:
    //      gl_Position = projection*model_view*vPosition/vPosition.w;

    gl_Position = projection * model_view * vPosition4;

    color = vColor4;
}
```

```
/*****************************
 * File: fshader42.glsl
 *       A simple fragment shader
 *****************************/

// #version 150  // YJC: Comment/un-comment this line to resolve compilation errors
                 //      due to different settings of the default GLSL version

in  vec4 color;
out vec4 fColor;

void main()
{
    fColor = color;
}
```