```
// ------------------------------------------------------------------------
// Handout: rotate-cube-shading.cpp (Rotating Cube with shading)
//
// * Originally from Ed Angel's textbook "Interactive Computer Graphics" 6th Ed
//          sample code "example3.cpp" of Chapter 5.
// * Extensively modified by Yi-Jen Chiang for the program structure,
//   normal matrix, user interface, etc.
//  (See keyboard() and mouse() functions for user interactions.)
// * Display a rotating cube with shading.
//
// - Light and material properties & Normal Matrix are sent to the shader as
//   uniform variables.
// - Entire shading computation is done in the Eye Frame (in shader).
// ------------------------------------------------------------------------
#include "Angel-yjc.h"

typedef Angel::vec4  color4;
typedef Angel::vec4  point4;

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

// 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;

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

const int NumVertices = 36; //(6 faces)(2 triangles/face)(3 vertices/triangle)
point4 points[NumVertices];
vec3   normals[NumVertices];

// Vertices of a unit cube centered at origin, sides aligned with axes
point4 vertices[8] = {
    point4( -0.5, -0.5,  0.5, 1.0 ),
    point4( -0.5,  0.5,  0.5, 1.0 ),
    point4(  0.5,  0.5,  0.5, 1.0 ),
    point4(  0.5, -0.5,  0.5, 1.0 ),
    point4( -0.5, -0.5, -0.5, 1.0 ),
    point4( -0.5,  0.5, -0.5, 1.0 ),
    point4(  0.5,  0.5, -0.5, 1.0 ),
    point4(  0.5, -0.5, -0.5, 1.0 )
};

// Array of rotation angles (in degrees) for each coordinate axis
enum { Xaxis = 0, Yaxis = 1, Zaxis = 2, NumAxes = 3 };
int      Axis = Xaxis;
GLfloat  Theta[NumAxes] = { 0.0, 0.0, 0.0 };

// Model-view and projection matrices uniform location
GLuint  ModelView, Projection;

/*----- Shader Lighting Parameters -----*/
    color4 light_ambient( 0.2, 0.2, 0.2, 1.0 );
    color4 light_diffuse( 1.0, 1.0, 1.0, 1.0 );
    color4 light_specular( 1.0, 1.0, 1.0, 1.0 );
    float const_att = 1.0;
    float linear_att = 0.01;
    float quad_att = 0.01;
    point4 light_position(2.0, 2.0, 1.0, 1.0 );
          // In World frame.
          // Needs to transform it to Eye Frame
          // before sending it to the shader(s).

    color4 material_ambient( 1.0, 0.0, 1.0, 1.0 );
    color4 material_diffuse( 1.0, 0.8, 0.0, 1.0 );
    color4 material_specular( 1.0, 0.8, 0.0, 1.0 );
    float  material_shininess = 100.0;

    color4 ambient_product = light_ambient * material_ambient;
    color4 diffuse_product = light_diffuse * material_diffuse;
    color4 specular_product = light_specular * material_specular;

void SetUp_Lighting_Uniform_Vars(mat4 mv);

int Index = 0;

//----------------------------------------------------------------------------
// quad() generates two triangles for each face and assigns normals
//        to the vertices
void quad( int a, int b, int c, int d )
{
    // Initialize temporary vectors along the quad's edges to
    //   compute its face normal
    vec4 u = vertices[b] - vertices[a];
    vec4 v = vertices[d] - vertices[a];

    vec3 normal = normalize( cross(u, v) );

    normals[Index] = normal; points[Index] = vertices[a]; Index++;
    normals[Index] = normal; points[Index] = vertices[b]; Index++;
    normals[Index] = normal; points[Index] = vertices[c]; Index++;
    normals[Index] = normal; points[Index] = vertices[a]; Index++;
    normals[Index] = normal; points[Index] = vertices[c]; Index++;
    normals[Index] = normal; points[Index] = vertices[d]; Index++;
}
//----------------------------------------------------------------------------
// colorcube() generates 6 quad faces (12 triangles): 36 vertices & 36 normals
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 );
}
//----------------------------------------------------------------------------
// OpenGL initialization
void init()
{
    colorcube();

    // Create and initialize a vertex buffer object
    glGenBuffers( 1, &cube_buffer );
    glBindBuffer( GL_ARRAY_BUFFER, cube_buffer );
    glBufferData( GL_ARRAY_BUFFER, sizeof(points) + sizeof(normals),
                  NULL, GL_STATIC_DRAW );
    glBufferSubData( GL_ARRAY_BUFFER, 0, sizeof(points), points );
    glBufferSubData( GL_ARRAY_BUFFER, sizeof(points),
                     sizeof(normals), normals );

    // Load shaders and create a shader program (to be used in display())
    program = InitShader( "vshader53.glsl", "fshader53.glsl" );

    glEnable( GL_DEPTH_TEST );
    glClearColor( 1.0, 1.0, 1.0, 1.0 );
}

//----------------------------------------------------------------------------
// SetUp_Lighting_Uniform_Vars(mat4 mv):
```

```
// Set up lighting parameters that are uniform variables in shader.
//
// Note: "LightPosition" in shader must be in the Eye Frame.
//       So we use parameter "mv", the model-view matrix, to transform
//       light_position to the Eye Frame.
//----------------------------------------------------------------------
void SetUp_Lighting_Uniform_Vars(mat4 mv)
{
    glUniform4fv( glGetUniformLocation(program, "AmbientProduct"),
                  1, ambient_product );
    glUniform4fv( glGetUniformLocation(program, "DiffuseProduct"),
                  1, diffuse_product );
    glUniform4fv( glGetUniformLocation(program, "SpecularProduct"),
                  1, specular_product );

   // The Light Position in Eye Frame
    vec4 light_position_eyeFrame = mv * light_position;
    glUniform4fv( glGetUniformLocation(program, "LightPosition"),
                  1, light_position_eyeFrame);

    glUniform1f(glGetUniformLocation(program, "ConstAtt"),
                        const_att);
    glUniform1f(glGetUniformLocation(program, "LinearAtt"),
                        linear_att);
    glUniform1f(glGetUniformLocation(program, "QuadAtt"),
                        quad_att);

    glUniform1f(glGetUniformLocation(program, "Shininess"),
                        material_shininess );
}
//----------------------------------------------------------------------------
// 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, 4, GL_FLOAT, GL_FALSE, 0,
                           BUFFER_OFFSET(0) );

    GLuint vNormal = glGetAttribLocation( program, "vNormal" );
    glEnableVertexAttribArray( vNormal );
    glVertexAttribPointer( vNormal, 3, GL_FLOAT, GL_FALSE, 0,
                           BUFFER_OFFSET(sizeof(points)) );
    // 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(vNormal);
}
//----------------------------------------------------------------------------
void display( void )
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

//** Important: glUseProgram() must be called *before* any shader variable
//             locations can be retrieved. This is needed to pass on values to
//             uniform/attribute variables in shader ("variable binding" in
//             shader).
    glUseProgram( program );

    // Retrieve transformation uniform variable locations
    // ** Must be called *after* glUseProgram().
    ModelView = glGetUniformLocation( program, "ModelView" );
    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

    // Generate the model-view matrix
    const vec3 viewer_pos( 0.0, 0.0, 2.0 );
    const vec4 eye(3.0, 2.0, 0.0, 1.0);
          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); // model-view matrix using Correct LookAt()
               // model-view matrix for the light position.

    /*--- Set up lighting parameters that are uniform variables in shader ---*/
    // ** Must be called *after* glUseProgram().
    // ** Also, "LightPosition" in shader must be in the Eye Frame, so
    //    we need to use model-view matrix to transform light_position to Eye Frame.
    //    ==> Must be called *after* the model-view matrix mv for light position is
    //        set up.
    SetUp_Lighting_Uniform_Vars(mv);

    // The model-view matrix with all transformations for the cube
    mat4  model_view = mv *  Scale (1.4, 1.4, 1.4) *
                             RotateX( Theta[Xaxis] ) *
                             RotateY( Theta[Yaxis] ) *
                             RotateZ( Theta[Zaxis] );

#if 0
    mat4  model_view = ( Translate( -viewer_pos ) *
                             RotateX( Theta[Xaxis] ) *
                             RotateY( Theta[Yaxis] ) *
                             RotateZ( Theta[Zaxis] ) );
#endif

    glUniformMatrix4fv(ModelView, 1, GL_TRUE, model_view );

    // Set up the Normal Matrix from the model-view matrix
    mat3 normal_matrix = NormalMatrix(model_view, 1);
         // Flag in NormalMatrix():
         //    1: model_view involves non-uniform scaling
         //    0: otherwise.
         // Using 1 is always correct.
         // But if no non-uniform scaling,
         //     using 0 is faster (avoids matrix inverse computation).

    glUniformMatrix3fv(glGetUniformLocation(program, "Normal_Matrix"),
                       1, GL_TRUE, normal_matrix );

    drawObj(cube_buffer, NumVertices);  // draw the cube

    glutSwapBuffers();
}
//----------------------------------------------------------------------------
void mouse( int button, int state, int x, int y )
{
    if ( state == GLUT_DOWN ) {
        switch( button ) {
```

```
            case GLUT_LEFT_BUTTON:    Axis = Xaxis;  break;
            case GLUT_MIDDLE_BUTTON:  Axis = Yaxis;  break;
            case GLUT_RIGHT_BUTTON:   Axis = Zaxis;  break;
        }
    }
}
//----------------------------------------------------------------------------
void idle( void )
{
    // Theta[Axis] += 0.01;
    Theta[Axis] += 1.0;  //YJC: change this value to adjust the cube rotation speed.

    if ( Theta[Axis] > 360.0 ) {
        Theta[Axis] -= 360.0;  }

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

#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 );
    glutKeyboardFunc( keyboard );
    glutMouseFunc( mouse );
    glutIdleFunc( idle );

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

```
/*
File Name: "vshader53.glsl":
Vertex shader:
  - Per vertex shading for a single point light source;
    distance attenuation is Yet To Be Completed.
  - Entire shading computation is done in the Eye Frame.
*/

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

in  vec4 vPosition;
in  vec3 vNormal;
out vec4 color;

uniform vec4 AmbientProduct, DiffuseProduct, SpecularProduct;
uniform mat4 ModelView;
uniform mat4 Projection;
uniform mat3 Normal_Matrix;
uniform vec4 LightPosition;   // Must be in Eye Frame
uniform float Shininess;

uniform float ConstAtt;  // Constant Attenuation
uniform float LinearAtt; // Linear Attenuation
uniform float QuadAtt;   // Quadratic Attenuation

void main()
{
    // Transform vertex  position into eye coordinates
    vec3 pos = (ModelView * vPosition).xyz;

    vec3 L = normalize( LightPosition.xyz - pos );
    vec3 E = normalize( -pos );
    vec3 H = normalize( L + E );

    // Transform vertex normal into eye coordinates
      // vec3 N = normalize( ModelView*vec4(vNormal, 0.0) ).xyz;
    vec3 N = normalize(Normal_Matrix * vNormal);

// YJC Note: N must use the one pointing *toward* the viewer
//     ==> If (N dot E) < 0 then N must be changed to -N
//
   if ( dot(N, E) < 0 ) N = -N;


/*--- To Do: Compute attenuation ---*/
float attenuation = 1.0;

 // Compute terms in the illumination equation
    vec4 ambient = AmbientProduct;

    float d = max( dot(L, N), 0.0 );
    vec4  diffuse = d * DiffuseProduct;

    float s = pow( max(dot(N, H), 0.0), Shininess );
    vec4  specular = s * SpecularProduct;

    if( dot(L, N) < 0.0 ) {
        specular = vec4(0.0, 0.0, 0.0, 1.0);
    }

    gl_Position = Projection * ModelView * vPosition;

/*--- attenuation below must be computed properly ---*/
    color = attenuation * (ambient + diffuse + specular);
}
```

```
/*
File Name: "fshader53.glsl":
           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;
}
```