```
/******************************************************************
 * Handout: checker-new.cpp (modified by Yi-Jen Chiang)
 *
 * This program texture maps a checkerboard image onto two squares.
 ******************************************************************/

#include "Angel-yjc.h"
#include <stdio.h>

typedef Angel::vec3 point3;
typedef Angel::vec4 color4;

/*      Create checkerboard texture     */
#define checkImageWidth 64
#define checkImageHeight 64
static  GLubyte checkImage[checkImageHeight][checkImageWidth][4];

static GLuint texName;

/*--- Quad arrays: 6 vertices of 2 triangles, for the quad (a b c d).
      Triangles are abc, cda. --*/
point3 quad_vert[6] = {
  point3(-1.0, -1.0, 0.0),  // a
  point3(-1.0, 1.0, 0.0),   // b
  point3(1.0, 1.0, 0.0),    // c

  point3(1.0, 1.0, 0.0),    // c
  point3(1.0, -1.0, 0.0),   // d
  point3(-1.0, -1.0, 0.0),  // a
};
vec2 quad_texCoord[6] = {
  vec2(0.0, 0.0),  // for a
  vec2(0.0, 1.0),  // for b
  vec2(1.0, 1.0),  // for c

  vec2(1.0, 1.0),  // for c
  vec2(1.0, 0.0),  // for d
  vec2(0.0, 0.0),  // for a
};

GLuint program;
GLuint quad_buffer;

/*--- Parameters for Perspective() function ---*/
GLfloat fovy = 60.0;
GLfloat aspect;
GLfloat zNear = 1.0, zFar = 30.0;

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

vec4 quad_color(0.8, 0.8, 0.0, 1.0); // original quad color: yellowish

int texture_app_flag = 0;  // 0: no texture application: obj color
                           // 1: texutre color
                           // 2: (obj color) * (texture color)
//---------------------------
void makeCheckImage(void)
{
   int i, j, c;

   for (i = 0; i < checkImageHeight; i++) {
      for (j = 0; j < checkImageWidth; j++) {
        c = (((i & 0x8) == 0) ^ ((j & 0x8) ==0));

         /*-- c == 1: white, else brown --*/
        checkImage[i][j][0] = (GLubyte) ((c==1) ? 255 : 100);
        checkImage[i][j][1] = (GLubyte) ((c==1) ? 255 : 70);
        checkImage[i][j][2] = (GLubyte) ((c==1) ? 255 : 0);
        checkImage[i][j][3] = (GLubyte) 255; } }
}
//---------------------------
void init(void)
{
   glEnable(GL_DEPTH_TEST);
   glClearColor(0.529, 0.807, 0.92, 1.0);       /* sky blue */

   makeCheckImage();
   glPixelStorei(GL_UNPACK_ALIGNMENT, 1);

   /*--- Create and Initialize a texture object ---*/
   glGenTextures(1, &texName);      // Generate texture obj name(s)

   glActiveTexture( GL_TEXTURE0 );  // Set the active texture unit to be 0
   glBindTexture(GL_TEXTURE_2D, texName); // Bind the texture to this texture unit

   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

   glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, checkImageWidth, checkImageHeight,
                0, GL_RGBA, GL_UNSIGNED_BYTE, checkImage);

   /** Note: If using multiple textures, repeat the above process starting from
             glActiveTexture(), but each time use a *different texture unit*,
             so that each texture is bound to a *different texture unit*.    **/

   /*--- Create and initialize vertex buffer object for quad ---*/
    glGenBuffers(1, &quad_buffer);
    glBindBuffer(GL_ARRAY_BUFFER, quad_buffer);
    glBufferData(GL_ARRAY_BUFFER, sizeof(quad_vert)+sizeof(quad_texCoord),
                 NULL, GL_STATIC_DRAW);
    glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(quad_vert), quad_vert);
    glBufferSubData(GL_ARRAY_BUFFER, sizeof(quad_vert),
                    sizeof(quad_texCoord), quad_texCoord);

    // Load shaders and create a shader program (to be used in display())
    program = InitShader( "vTexture.glsl", "fTexture.glsl" );
}
//----------------------------------------------------------------------------
// 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 vTexCoord = glGetAttribLocation( program, "vTexCoord" );
    glEnableVertexAttribArray( vTexCoord );
    glVertexAttribPointer( vTexCoord, 2, GL_FLOAT, GL_FALSE, 0,
                           BUFFER_OFFSET(sizeof(quad_vert)) );
    // 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(vTexCoord);
}
//----------------------------------
void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

   glUseProgram( program );

   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

   // Set the value of the fragment shader texture sampler variable
   //   ("texture_2D") to the appropriate texture unit. In this case,
   //   0, for GL_TEXTURE0 which was previously set in init() by calling
   //   glActiveTexture( GL_TEXTURE0 ).
   glUniform1i( glGetUniformLocation(program, "texture_2D"), 0 );

   /** Note: If using multiple textures, each texture must be bound to a
             *different texture unit* (as commented in the "Note" in init()),
             and here each sampler variable must be set to the *corresponding
             texture unit*.                                              **/

   const vec4 eye(0.0, 0.0, 3.6, 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()

   // Pass on the quad_color to the uniform var "uColor" in vertex shader
   glUniform4fv( glGetUniformLocation(program, "uColor"), 1, quad_color);

   // Pass on the value of texture_app_flag to the fragment shader
   glUniform1i( glGetUniformLocation(program, "Texture_app_flag"),
                texture_app_flag);

   // Draw the first quad with translation only
   mat4 model_view = mv * Translate(0.8, 0.0, 0.0);
   glUniformMatrix4fv(ModelView, 1, GL_TRUE, model_view );
   drawObj(quad_buffer, 6);

   // Draw the 2nd quad with both rotation & translation
   model_view = mv * Translate(-1.4, 0.0, -0.6) * Rotate(-35, 0.0, 1.0, 0.0);
   glUniformMatrix4fv(ModelView, 1, GL_TRUE, model_view );
   drawObj(quad_buffer, 6);

   glutSwapBuffers();
}
//----------------------------------
void reshape(int w, int h)
{
   glViewport(0, 0, (GLsizei) w, (GLsizei) h);
   aspect = (GLfloat) w/(GLfloat) h;
   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 ' ':  // Toggle among No Texture (obj color), Texture Only,
                   //        and Modulate the two.
            texture_app_flag++;
            if (texture_app_flag > 2)
                texture_app_flag = 0;
            glutPostRedisplay();
            break;
    }
}
//----------------------------------
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(600, 600);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Checkerboard");

#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);

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

```
/*---------------
Vertex Shader: filename "vTexture.glsl"
---------------*/
// #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  vec2 vTexCoord;

uniform mat4 ModelView;
uniform mat4 Projection;

uniform vec4 uColor; // obj color (as a uniform variable)

out vec4 color;
out vec2 texCoord;

void main()
{
 vec4 vPosition4 = vec4(vPosition.x, vPosition.y, vPosition.z, 1.0);
 gl_Position = Projection * ModelView * vPosition4;

 color = uColor;
 texCoord = vTexCoord;

}
```

```
/*--------------
Fragment Shader: filename "fTexture.glsl"
---------------*/
// #version 150   // YJC: Comment/un-comment this line to resolve compilation errors
                  //      due to different settings of the default GLSL version

in  vec4 color;
in  vec2 texCoord;

uniform sampler2D texture_2D; /* Note: If using multiple textures,
                                       each texture must be bound to a
                                       *different texture unit*, with the
                                       sampler uniform var set accordingly.
                                 The (fragment) shader can access *all texture units*
                                 simultaneously.
                              */
uniform int Texture_app_flag; // 0: no texture application: obj color
                              // 1: texutre color
                              // 2: (obj color) * (texture color)
out vec4 fColor;

void main()
{
  if (Texture_app_flag == 0)
     fColor = color;
  else if (Texture_app_flag == 1)
     fColor = texture( texture_2D, texCoord );
  else // Texture_app_flag == 2
     fColor = color * texture( texture_2D, texCoord );
}
```