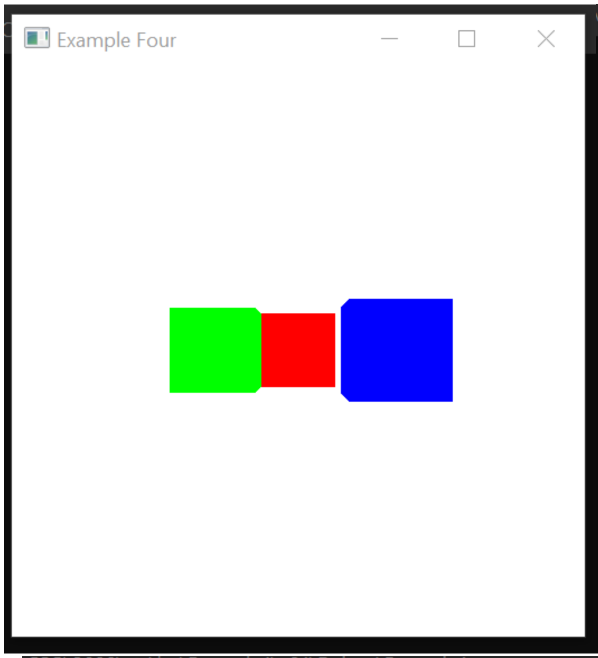
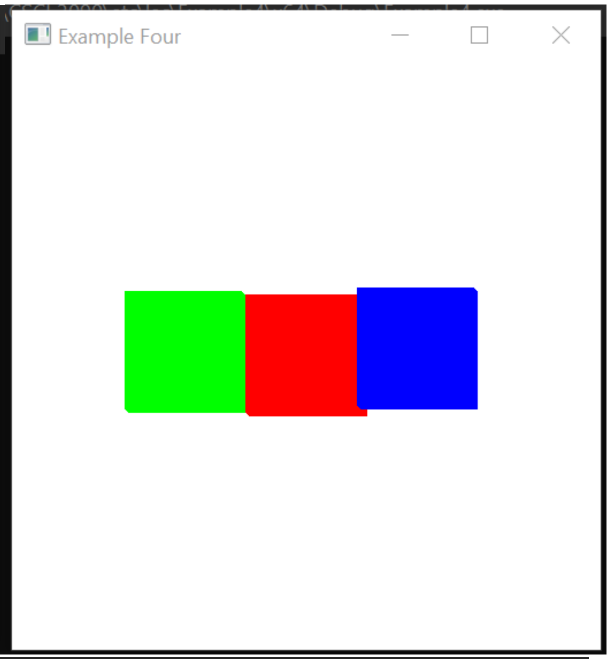
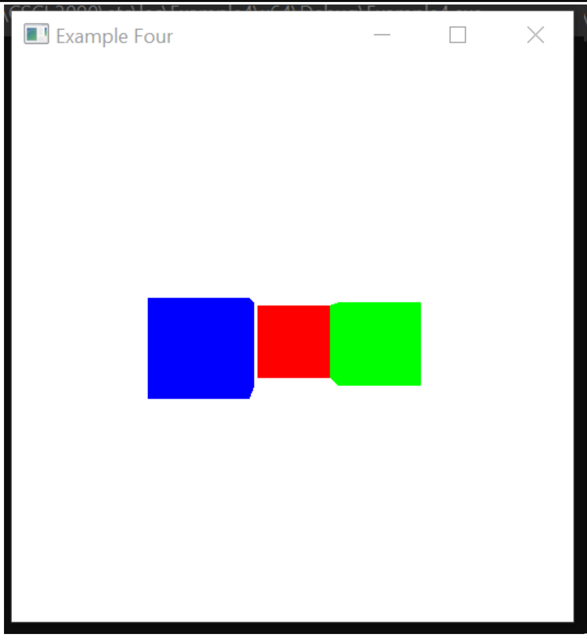
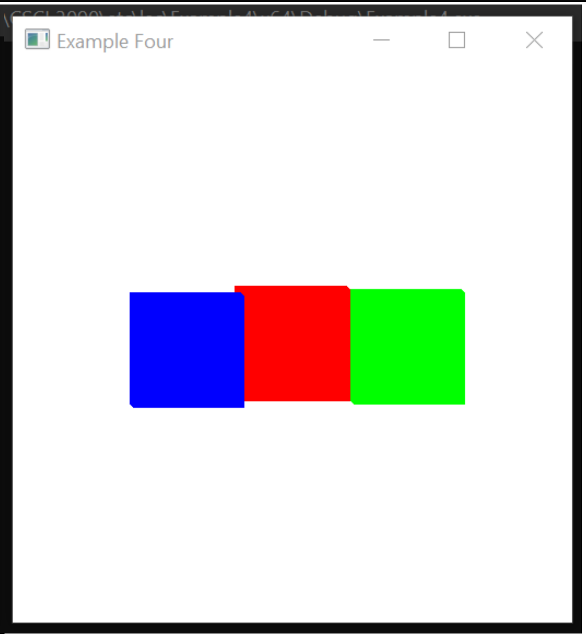
```
#include <Windows.h>
#include <gl/glew.h>
#define GLFW_DLL
#define GLFW_INCLUDE_NONE
#include <GLFW/glfw3.h>
#define GLM_FORCE_RADIANS
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>
#include "shaders.h"
#include <stdio.h>

GLuint program;         // shader programs
GLuint triangleVAO;     // the data to be displayed
float angle = 0.0;
double theta, phi;      // user's position  on a sphere centered on the object
double r;               // radius of the sphere
GLuint ibuffer;

glm::mat4 projection;   // projection matrix
float eyex, eyey, eyez; // eye position

/*
 *  The init procedure creates the OpenGL data structures
 *  that contain the triangle geometry, compiles our
 *  shader program and links the shader programs to
 *  the data.
 */

void init() {
    GLuint vbuffer;
    GLint vPosition;
    GLint vNormal;
    int vs;
    int fs;

    glGenVertexArrays(1, &triangleVAO);
    glBindVertexArray(triangleVAO);

    GLfloat vertices[8][4] = {
        {-1.0, -1.0, -1.0, 1.0 },  //0
        {-1.0, -1.0,  1.0, 1.0},   //1
        {-1.0,  1.0, -1.0, 1.0},   //2
        {-1.0,  1.0,  1.0, 1.0},   //3
        { 1.0, -1.0, -1.0, 1.0},   //4
        { 1.0, -1.0,  1.0, 1.0},   //5
        { 1.0,  1.0, -1.0, 1.0},   //6
        { 1.0,  1.0,  1.0, 1.0}    //7
    };

    GLfloat normals[8][3] = {
        {-1.0, -1.0, -1.0},  //0
        {-1.0, -1.0,  1.0},  //1
        {-1.0,  1.0, -1.0},  //2
        {-1.0,  1.0,  1.0},  //3
        { 1.0, -1.0, -1.0},  //4
        { 1.0, -1.0,  1.0},  //5
        { 1.0,  1.0, -1.0},  //6
        { 1.0,  1.0,  1.0}   //7
    };

    GLushort indexes[36] = { 0, 1, 3, 0, 2, 3,
                             0, 4, 5, 0, 1, 5,
                             2, 6, 7, 2, 3, 7,
                             0, 4, 6, 0, 2, 6,
                             1, 5, 7, 1, 3, 7,
                             4, 5, 7, 4, 6, 7 };

    /*
     *  load the vertex coordinate data
     */
    glGenBuffers(1, &vbuffer);
    glBindBuffer(GL_ARRAY_BUFFER, vbuffer);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices) + sizeof(normals), NULL,
      GL_STATIC_DRAW);
    glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(vertices), vertices);
    glBufferSubData(GL_ARRAY_BUFFER, sizeof(vertices), sizeof(normals),
      normals);

    /*
     *  load the vertex indexes
     */
    glGenBuffers(1, &ibuffer);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, ibuffer);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indexes), indexes,
      GL_STATIC_DRAW);

    /*
     *  compile and build the shader program
     */
    vs = buildShader(GL_VERTEX_SHADER, "example4.vs");
    fs = buildShader(GL_FRAGMENT_SHADER, "example4.fs");
    program = buildProgram(vs, fs, 0);

    /*
     *  link the vertex coordinates to the vPosition
     *  variable in the vertex program
     */
    glUseProgram(program);
    vPosition = glGetAttribLocation(program, "vPosition");
    glVertexAttribPointer(vPosition, 4, GL_FLOAT, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(vPosition);
    vNormal = glGetAttribLocation(program, "vNormal");
    glVertexAttribPointer(vNormal, 3, GL_FLOAT, GL_FALSE, 0, (void*)sizeof
      (vertices));
    glEnableVertexAttribArray(vNormal);

}

void framebufferSizeCallback(GLFWwindow* window, int w, int h) {
    // Prevent a divide by zero, when window is too short (you cant make a
      window of zero width).
    if (h == 0) h = 1;

    float ratio = 1.0f * w / h;

    glfwMakeContextCurrent(window);
    glViewport(0, 0, w, h);

    projection = glm::perspective(45.0f, 1.0f, 1.0f, 100.0f);

}

/*
 *  This procedure is called each time the screen needs
 *  to be redisplayed
 */
void display() {
    glm::mat4 model;
    glm::mat4 view;
    glm::mat4 viewPerspective;
    int modelLoc;
    int normalLoc;
    int viewLoc;
    int colourLoc;

    model = glm::mat4(1.0);

    view = glm::lookAt(glm::vec3(eyex, eyey, eyez),
        glm::vec3(0.0f, 0.0f, 0.0f),
        glm::vec3(0.0f, 0.0f, 1.0f));

    glm::mat3 normal = glm::transpose(glm::inverse(glm::mat3(view * model)));

    viewPerspective = projection * view;

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glUseProgram(program);
    modelLoc = glGetUniformLocation(program, "model");
    viewLoc = glGetUniformLocation(program, "viewPerspective");
    glUniformMatrix4fv(viewLoc, 1, 0, glm::value_ptr(viewPerspective));
    normalLoc = glGetUniformLocation(program, "normalMat");
    glUniformMatrix3fv(normalLoc, 1, 0, glm::value_ptr(normal));
    colourLoc = glGetUniformLocation(program, "colour");

    glBindVertexArray(triangleVAO);

    glUniform4f(colourLoc, 1.0, 0.0, 0.0, 1.0);
    glUniformMatrix4fv(modelLoc, 1, 0, glm::value_ptr(model));
    glUniformMatrix3fv(normalLoc, 1, 0, glm::value_ptr(normal));
    glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_SHORT, NULL);

    model = glm::translate(model, glm::vec3(2.0, 2.0, 0.0));
    normal = glm::transpose(glm::inverse(glm::mat3(view * model)));
    glUniform4f(colourLoc, 0.0, 1.0, 0.0, 1.0);
    glUniformMatrix4fv(modelLoc, 1, 0, glm::value_ptr(model));
    glUniformMatrix3fv(normalLoc, 1, 0, glm::value_ptr(normal));
    glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_SHORT, NULL);

    model = glm::translate(model, glm::vec3(-4.0, 2.0, 0.0));
    normal = glm::transpose(glm::inverse(glm::mat3(view * model)));
    glUniform4f(colourLoc, 0.0, 0.0, 1.0, 1.0);
    glUniformMatrix4fv(modelLoc, 1, 0, glm::value_ptr(model));
    glUniformMatrix3fv(normalLoc, 1, 0, glm::value_ptr(normal));
    glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_SHORT, NULL);
}


/*
 *  Called each time a key is pressed on
 *  the keyboard.
 */
static void key_callback(GLFWwindow* window, int key, int scancode, int
  action, int mods)
{
    if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS)
      glfwSetWindowShouldClose(window, GLFW_TRUE);

    // change camera position
    if (key == GLFW_KEY_A && action == GLFW_PRESS) phi -= 0.1;
    if (key == GLFW_KEY_D && action == GLFW_PRESS) phi += 0.1;
    if (key == GLFW_KEY_W && action == GLFW_PRESS) theta += 0.1;
    if (key == GLFW_KEY_S && action == GLFW_PRESS) theta -= 0.1;

    // change projection
    if (key == GLFW_KEY_P && action == GLFW_PRESS) projection =
```

```cpp
        glm::perspective(45.0f, 1.0f, 1.0f, 100.0f);
    if (key == GLFW_KEY_O && action == GLFW_PRESS) projection = glm::ortho
      (-5.0f, 5.0f, -5.0f, 5.0f, 1.0f, 100.0f);

    // change fov
    if (key == GLFW_KEY_I && action == GLFW_PRESS) projection =
      glm::perspective(-45.0f, 1.0f, 1.0f, 100.0f);
    if (key == GLFW_KEY_U && action == GLFW_PRESS) projection = glm::ortho
      (5.0f, -5.0f, 5.0f, -5.0f, 1.0f, 100.0f);

    eyex = (float)(r * sin(theta) * cos(phi));
    eyey = (float)(r * sin(theta) * sin(phi));
    eyez = (float)(r * cos(theta));

}

void error_callback(int error, const char* description)
{
    fprintf(stderr, "Error: %s\n", description);
}


int main(int argc, char** argv) {
    GLFWwindow* window;

    // start by setting error callback in case something goes wrong
    glfwSetErrorCallback(error_callback);

    // initialize glfw
    if (!glfwInit()) fprintf(stderr, "can't initialize GLFW\n");

    // create the window used by our application
    window = glfwCreateWindow(512, 512, "Example Four", NULL, NULL);

    if (!window) {
        glfwTerminate();
        exit(EXIT_FAILURE);
    }

    // establish framebuffer size change and input callbacks
    glfwSetFramebufferSizeCallback(window, framebufferSizeCallback);
    glfwSetKeyCallback(window, key_callback);

    /*
     *  initialize glew
     */
    glfwMakeContextCurrent(window);
    GLenum error = glewInit();
    if (error != GLEW_OK) {
        printf("Error starting GLEW: %s\n", glewGetErrorString(error));
        exit(0);
    }

    glEnable(GL_DEPTH_TEST);
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glViewport(0, 0, 512, 512);

    projection = glm::perspective(45.0f, 1.0f, 1.0f, 100.0f);

    init();

    eyex = 0.0;
    eyey = 15.0;
    eyez = 0.0;

    theta = 1.5;
    phi = 1.5;
    r = 15.0;

    glfwSwapInterval(1);

    // GLFW main loop, display model, swapbuffer and check for input
    while (!glfwWindowShouldClose(window)) {
        display();
        glfwSwapBuffers(window);
        glfwPollEvents();
    }

    glfwTerminate();

}
```


Example Four


Example Four


Example Four


Example Four