Roll No: 1803062

Lab Performance Test [Lab Final]

Question1: Draw an OpenGL triangle, then if you press "C" it should turn the triangle upside down.

Solution (Bold your own written code):

```
#include "glad.h"
#include "glfw3.h"

#include <iostream>

void framebuffer_size_callback(GLFWwindow* window, int width, int height);
void processInput(GLFWwindow *window);

// settings
const unsigned int SCR_WIDTH = 800;
const unsigned int SCR_HEIGHT = 600;

const char *vertexShaderSource ="#version 330 core\n"
    "layout (location = 0) in vec3 aPos;\n"
    "layout (location = 1) in vec3 aColor;\n"
    "out vec3 ourColor;\n"
    "void main()\n"
    "{\n"
    "   gl_Position = vec4(aPos, 1.0);\n"
    "   ourColor = aColor;\n"
    "}\0";

const char *fragmentShaderSource = "#version 330 core\n"
    "out vec4 FragColor;\n"
    "in vec3 ourColor;\n"
    "void main()\n"
    "{\n"
    "   FragColor = vec4(ourColor, 1.0f);\n"
    "}\n\0";

int main()
{
    // glfw: initialize and configure
    // ------------------------------
    glfwInit();
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
```

```
#ifdef __APPLE__
    glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
#endif

    // glfw window creation
    // --------------------
    GLFWwindow* window = glfwCreateWindow(SCR_WIDTH, SCR_HEIGHT,
"LearnOpenGL", NULL, NULL);
    if (window == NULL)
    {
        std::cout << "Failed to create GLFW window" << std::endl;
        glfwTerminate();
        return -1;
    }
    glfwMakeContextCurrent(window);
    glfwSetFramebufferSizeCallback(window, framebuffer_size_callback);

    // glad: load all OpenGL function pointers
    // ---------------------------------------
    if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))
    {
        std::cout << "Failed to initialize GLAD" << std::endl;
        return -1;
    }

    // build and compile our shader program
    // ------------------------------------
    // vertex shader
    unsigned int vertexShader = glCreateShader(GL_VERTEX_SHADER);
    glShaderSource(vertexShader, 1, &vertexShaderSource, NULL);
    glCompileShader(vertexShader);
    // check for shader compile errors
    int success;
    char infoLog[512];
    glGetShaderiv(vertexShader, GL_COMPILE_STATUS, &success);
    if (!success)
    {
        glGetShaderInfoLog(vertexShader, 512, NULL, infoLog);
        std::cout << "ERROR::SHADER::VERTEX::COMPILATION_FAILED\n" <<
infoLog << std::endl;
    }
    // fragment shader
    unsigned int fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(fragmentShader, 1, &fragmentShaderSource, NULL);
    glCompileShader(fragmentShader);
    // check for shader compile errors
    glGetShaderiv(fragmentShader, GL_COMPILE_STATUS, &success);
```

```
    if (!success)
    {
        glGetShaderInfoLog(fragmentShader, 512, NULL, infoLog);
        std::cout << "ERROR::SHADER::FRAGMENT::COMPILATION_FAILED\n" <<
infoLog << std::endl;
    }
    // link shaders
    unsigned int shaderProgram = glCreateProgram();
    glAttachShader(shaderProgram, vertexShader);
    glAttachShader(shaderProgram, fragmentShader);
    glLinkProgram(shaderProgram);
    // check for linking errors
    glGetProgramiv(shaderProgram, GL_LINK_STATUS, &success);
    if (!success) {
        glGetProgramInfoLog(shaderProgram, 512, NULL, infoLog);
        std::cout << "ERROR::SHADER::PROGRAM::LINKING_FAILED\n" << infoLog
<< std::endl;
    }
    glDeleteShader(vertexShader);
    glDeleteShader(fragmentShader);

    // set up vertex data (and buffer(s)) and configure vertex attributes
    // ------------------------------------------------------------------
    float vertices[] = {
        // positions         // colors
         0.5f, -0.5f, 0.0f,  1.0f, 0.0f, 0.0f,  // bottom right
        -0.5f, -0.5f, 0.0f,  0.0f, 1.0f, 0.0f,  // bottom left
         0.0f,  0.5f, 0.0f,  0.0f, 0.0f, 1.0f   // top

    };

    unsigned int VBO, VAO;
    glGenVertexArrays(1, &VAO);
    glGenBuffers(1, &VBO);
    // bind the Vertex Array Object first, then bind and set vertex
buffer(s), and then configure vertex attributes(s).
    glBindVertexArray(VAO);

    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices,
GL_STATIC_DRAW);

    // position attribute
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float),
(void*)0);
    glEnableVertexAttribArray(0);
    // color attribute
```

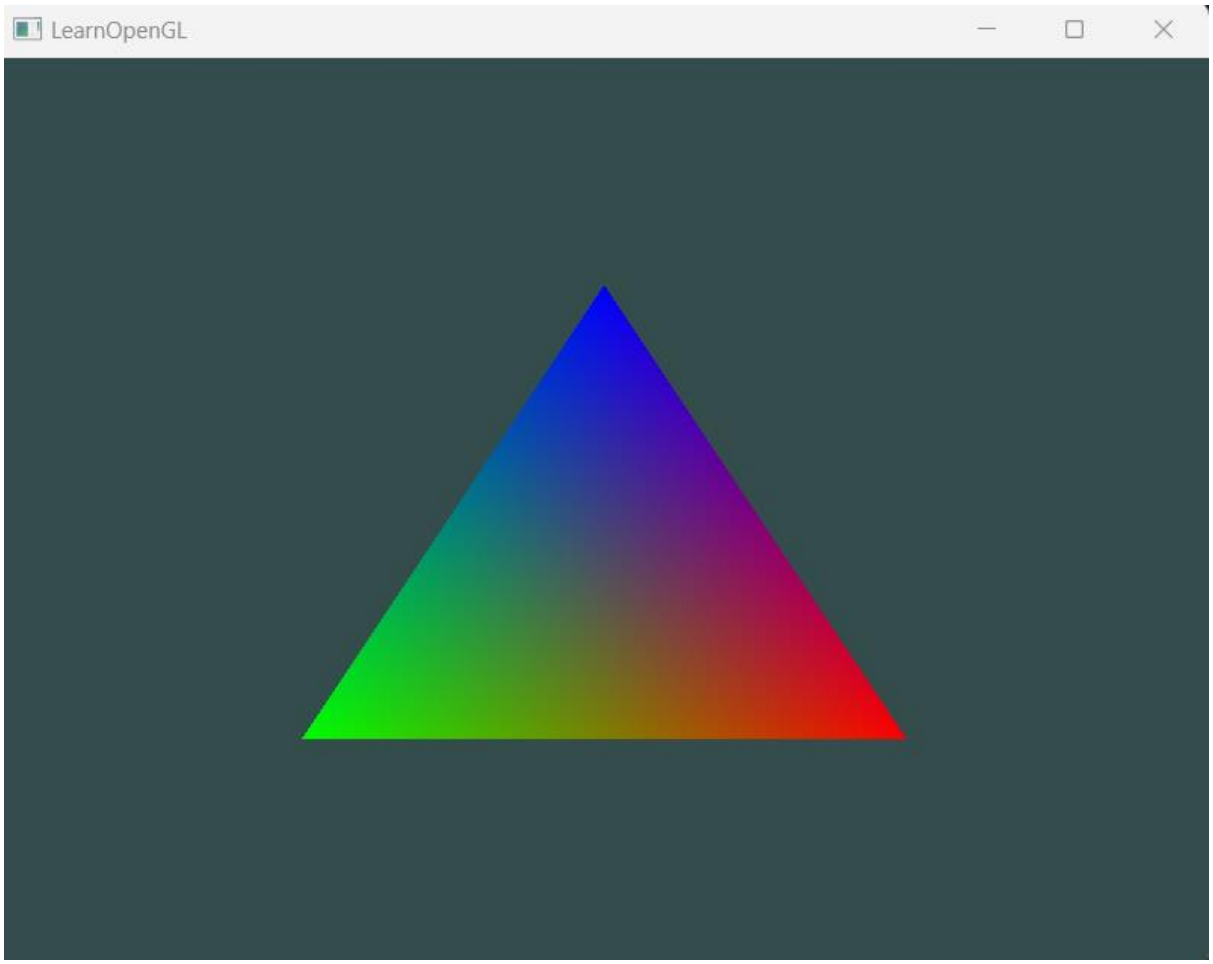
```
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float),
(void*)(3 * sizeof(float)));
    glEnableVertexAttribArray(1);


    glUseProgram(shaderProgram);

    // render loop
    // -----------
    while (!glfwWindowShouldClose(window))
    {
        // input
        // -----
        processInput(window);

        // render
        // ------
        glClearColor(0.2f, 0.3f, 0.3f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT);

        // render the triangle
        glBindVertexArray(VAO);
        glDrawArrays(GL_TRIANGLES, 0, 3);

        // glfw: swap buffers and poll IO events (keys pressed/released,
mouse moved etc.)
        // -------------------------------------------------------------------
--------------
        glfwSwapBuffers(window);
        glfwPollEvents();
    }

    // optional: de-allocate all resources once they've outlived their
purpose:
    // ------------------------------------------------------------------------
---
    glDeleteVertexArrays(1, &VAO);
    glDeleteBuffers(1, &VBO);
    glDeleteProgram(shaderProgram);

    // glfw: terminate, clearing all previously allocated GLFW resources.
    // ------------------------------------------------------------------
    glfwTerminate();
    return 0;
}

// process all input: query GLFW whether relevant keys are pressed/released
this frame and react accordingly
```

```
// ------------------------------------------------------------------------
-------------------------------
void processInput(GLFWwindow *window)
{
    if (glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS)
        glfwSetWindowShouldClose(window, true);
}

// glfw: whenever the window size changed (by OS or user resize) this
callback function executes
// ------------------------------------------------------------------------
--------------------
void framebuffer_size_callback(GLFWwindow* window, int width, int height)
{
    glViewport(0, 0, width, height);
}
```

```
#version 330 core
out vec4 FragColor;

in vec3 ourColor;

void main()
{
    FragColor = vec4(ourColor, 1.0f);
}
```

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aColor;

out vec3 ourColor;

void main()
{
    gl_Position = vec4(aPos, 1.0);
    ourColor = aColor;
}
```

Output:



Question2: Show an OpenGL program which will a scale a green rectangle and blue triangle to half of its size and then rotate it by 90 degree in clockwise direction.

Solution (Bold your own written code):

```
#include "glad.h"
#include "glfw3.h"

#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"

#include "glm/glm.hpp"
#include "glm/gtc/matrix_transform.hpp"
#include "glm/gtc/type_ptr.hpp"
#include "learnopengl/shader_m.h"

#include <iostream>

void framebuffer_size_callback(GLFWwindow* window, int width, int height);
void processInput(GLFWwindow *window);

// settings
const unsigned int SCR_WIDTH = 800;
const unsigned int SCR_HEIGHT = 600;

int main()
{
    // glfw: initialize and configure
    // ------------------------------
    glfwInit();
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);

#ifdef __APPLE__
    glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
#endif

    // glfw window creation
    // --------------------
    GLFWwindow* window = glfwCreateWindow(SCR_WIDTH, SCR_HEIGHT,
"LearnOpenGL", NULL, NULL);
    if (window == NULL)
    {
        std::cout << "Failed to create GLFW window" << std::endl;
        glfwTerminate();
        return -1;
    }
```

```
    glfwMakeContextCurrent(window);
    glfwSetFramebufferSizeCallback(window, framebuffer_size_callback);

    // glad: load all OpenGL function pointers
    // ---------------------------------------
    if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))
    {
        std::cout << "Failed to initialize GLAD" << std::endl;
        return -1;
    }

    // build and compile our shader zprogram
    // ------------------------------------
    Shader ourShader("src/shader/6.1.coordinate_systems.vs",
"src/shader/6.1.coordinate_systems.fs");

    // set up vertex data (and buffer(s)) and configure vertex attributes
    // ------------------------------------------------------------------
    float vertices[] = {
        // positions          // texture coords
         0.5f,  0.5f, 0.0f,   1.0f, 1.0f, // top right
         0.5f, -0.5f, 0.0f,   1.0f, 0.0f, // bottom right
 //       -0.5f, -0.5f, 0.0f,   0.0f, 0.0f, // bottom left
 //       -0.5f,  0.5f, 0.0f,   0.0f, 1.0f  // top left
    };
    unsigned int indices[] = {
        0, 1, 3, // first triangle
        1, 2, 3  // second triangle
    };
    unsigned int VBO, VAO, EBO;
    glGenVertexArrays(1, &VAO);
    glGenBuffers(1, &VBO);
    glGenBuffers(1, &EBO);

    glBindVertexArray(VAO);

    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices,
GL_STATIC_DRAW);

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices,
GL_STATIC_DRAW);

    // position attribute
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 5 * sizeof(float),
(void*)0);
    glEnableVertexAttribArray(0);
```

```
    // texture coord attribute
    glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, 5 * sizeof(float),
(void*)(3 * sizeof(float)));
    glEnableVertexAttribArray(1);


    ourShader.use();
    ourShader.setInt("texture1", 0);
    ourShader.setInt("texture2", 1);


    // render loop
    // -----------
    while (!glfwWindowShouldClose(window))
    {
        // input
        // -----
        processInput(window);

        // render
        // ------
        glClearColor(0.2f, 0.3f, 0.3f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT);

        // // bind textures on corresponding texture units
        // glActiveTexture(GL_TEXTURE0);
        // glBindTexture(GL_TEXTURE_2D, texture1);
        // glActiveTexture(GL_TEXTURE1);
        // glBindTexture(GL_TEXTURE_2D, texture2);

        // activate shader
        ourShader.use();

        // create transformations
        glm::mat4 model         = glm::mat4(1.0f); // make sure to
initialize matrix to identity matrix first
        glm::mat4 view          = glm::mat4(1.0f);
        glm::mat4 projection    = glm::mat4(1.0f);
        model = glm::rotate(model, glm::radians(-55.0f), glm::vec3(1.0f,
0.0f, 0.0f));
        view  = glm::translate(view, glm::vec3(0.0f, 0.0f, -3.0f));
        projection = glm::perspective(glm::radians(45.0f), (float)SCR_WIDTH
/ (float)SCR_HEIGHT, 0.1f, 100.0f);
        // retrieve the matrix uniform locations
        unsigned int modelLoc = glGetUniformLocation(ourShader.ID, "model");
        unsigned int viewLoc  = glGetUniformLocation(ourShader.ID, "view");
        // pass them to the shaders (3 different ways)
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
```

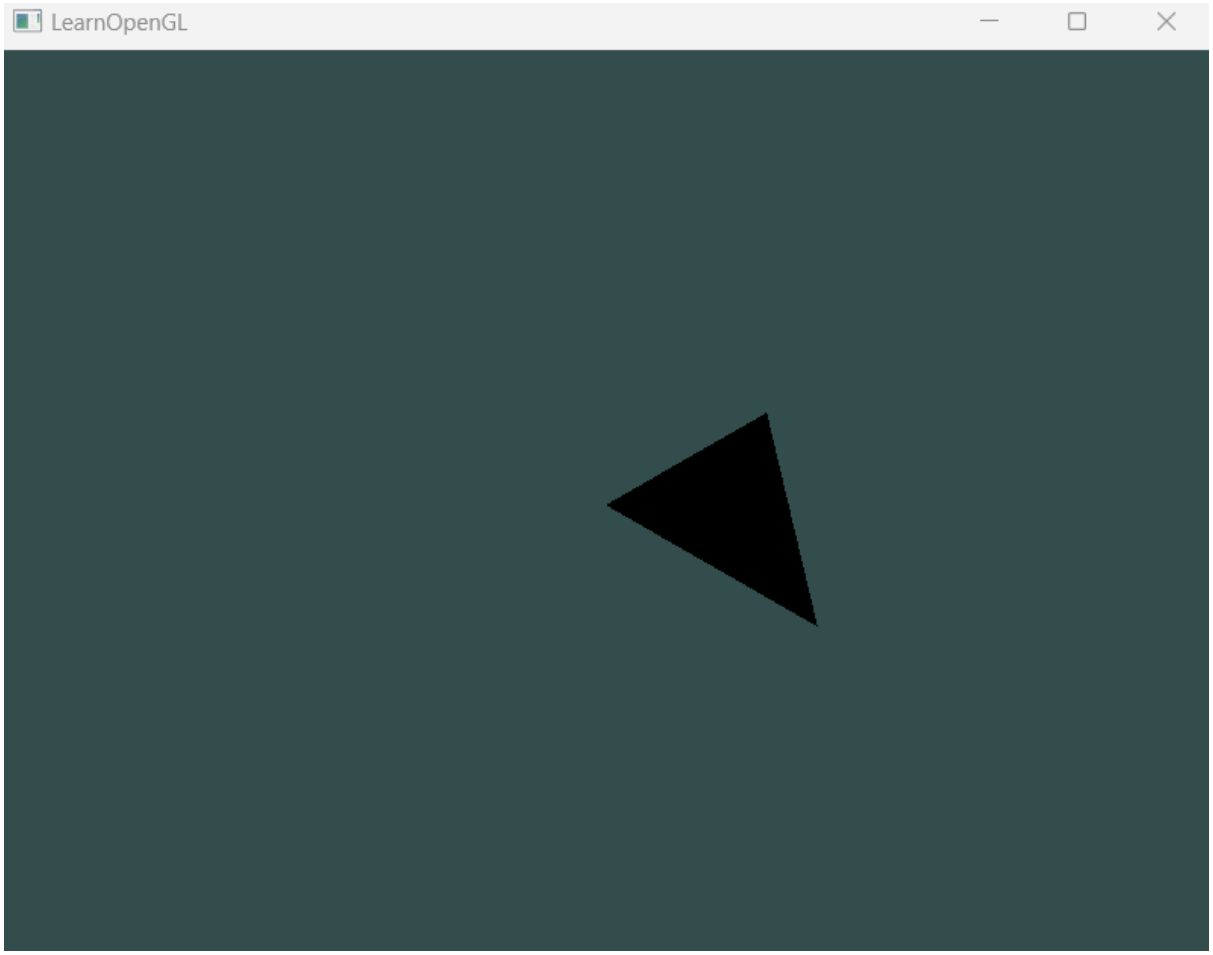
```
        glUniformMatrix4fv(viewLoc, 1, GL_FALSE, &view[0][0]);
        // note: currently we set the projection matrix each frame, but
since the projection matrix rarely changes it's often best practice to set
it outside the main loop only once.
        ourShader.setMat4("projection", projection);

        // render container
        glBindVertexArray(VAO);
        glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);

        // glfw: swap buffers and poll IO events (keys pressed/released,
mouse moved etc.)
        // -------------------------------------------------------------------
--------------
        glfwSwapBuffers(window);
        glfwPollEvents();
    }

    // optional: de-allocate all resources once they've outlived their
purpose:
    // ------------------------------------------------------------------------
---
    glDeleteVertexArrays(1, &VAO);
    glDeleteBuffers(1, &VBO);
    glDeleteBuffers(1, &EBO);

    // glfw: terminate, clearing all previously allocated GLFW resources.
    // ------------------------------------------------------------------
    glfwTerminate();
    return 0;
}

// process all input: query GLFW whether relevant keys are pressed/released
this frame and react accordingly
// ---------------------------------------------------------------------------
-------------------------------
void processInput(GLFWwindow *window)
{
    if (glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS)
        glfwSetWindowShouldClose(window, true);
}

// glfw: whenever the window size changed (by OS or user resize) this
callback function executes
// ---------------------------------------------------------------------------
--------------------
void framebuffer_size_callback(GLFWwindow* window, int width, int height)
{
```

```
    // make sure the viewport matches the new window dimensions; note that
width and
    // height will be significantly larger than specified on retina
displays.
    glViewport(0, 0, width, height);
}
```

```
#version 330 core
out vec4 FragColor;

in vec2 TexCoord;

// texture samplers
uniform sampler2D texture1;
uniform sampler2D texture2;

void main()
{
    // linearly interpolate between both textures (80% container, 20%
awesomeface)
    FragColor = mix(texture(texture1, TexCoord), texture(texture2,
TexCoord), 0.2);
}
```

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec2 aTexCoord;

out vec2 TexCoord;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

void main()
{
    gl_Position = projection * view * model *
vec4(aPos, 1.0);
    TexCoord = vec2(aTexCoord.x, aTexCoord.y);
}
```

Output:



Question3: Draw a Cube in OpenGL with 2 textures and control the mixture of texture using keyboard. You should be able to increase or decrease the opacity of second texture using keyboard. Use "w" for increase and "d" for decrease.

Solution (Bold your own written code):

```
#include "glad.h"
#include "glfw3.h"

#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"

#include "glm/glm.hpp"
#include "glm/gtc/matrix_transform.hpp"
#include "glm/gtc/type_ptr.hpp"
#include "learnopengl/shader_m.h"

#include <iostream>

void framebuffer_size_callback(GLFWwindow* window, int width, int height);
void processInput(GLFWwindow *window);

// settings
const unsigned int SCR_WIDTH = 800;
const unsigned int SCR_HEIGHT = 600;

int main()
{
    // glfw: initialize and configure
    // ------------------------------
    glfwInit();
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);

#ifdef __APPLE__
    glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
#endif

    // glfw window creation
    // --------------------
    GLFWwindow* window = glfwCreateWindow(SCR_WIDTH, SCR_HEIGHT,
"LearnOpenGL", NULL, NULL);
    if (window == NULL)
    {
        std::cout << "Failed to create GLFW window" << std::endl;
```

```
        glfwTerminate();
        return -1;
    }
    glfwMakeContextCurrent(window);
    glfwSetFramebufferSizeCallback(window, framebuffer_size_callback);

    // glad: load all OpenGL function pointers
    // ---------------------------------------
    if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))
    {
        std::cout << "Failed to initialize GLAD" << std::endl;
        return -1;
    }

    // configure global opengl state
    // -----------------------------
    glEnable(GL_DEPTH_TEST);

    // build and compile our shader zprogram
    // ------------------------------------
    Shader ourShader("src/shader/6.3.coordinate_systems.vs",
"src/shader/6.3.coordinate_systems.fs");

    // set up vertex data (and buffer(s)) and configure vertex attributes
    // ------------------------------------------------------------------
    float vertices[] = {
        -0.5f, -0.5f, -0.5f,  0.0f, 0.0f, //x,y,z,texCord
         0.5f, -0.5f, -0.5f,  1.0f, 0.0f,
         0.5f,  0.5f, -0.5f,  1.0f, 1.0f,
         0.5f,  0.5f, -0.5f,  1.0f, 1.0f,
        -0.5f,  0.5f, -0.5f,  0.0f, 1.0f,
        -0.5f, -0.5f, -0.5f,  0.0f, 0.0f,

        -0.5f, -0.5f,  0.5f,  0.0f, 0.0f,
         0.5f, -0.5f,  0.5f,  1.0f, 0.0f,
         0.5f,  0.5f,  0.5f,  1.0f, 1.0f,
         0.5f,  0.5f,  0.5f,  1.0f, 1.0f,
        -0.5f,  0.5f,  0.5f,  0.0f, 1.0f,
        -0.5f, -0.5f,  0.5f,  0.0f, 0.0f,

        -0.5f,  0.5f,  0.5f,  1.0f, 0.0f,
        -0.5f,  0.5f, -0.5f,  1.0f, 1.0f,
        -0.5f, -0.5f, -0.5f,  0.0f, 1.0f,
        -0.5f, -0.5f, -0.5f,  0.0f, 1.0f,
        -0.5f, -0.5f,  0.5f,  0.0f, 0.0f,
        -0.5f,  0.5f,  0.5f,  1.0f, 0.0f,

         0.5f,  0.5f,  0.5f,  1.0f, 0.0f,
```

```
         0.5f,  0.5f, -0.5f,  1.0f, 1.0f,
         0.5f, -0.5f, -0.5f,  0.0f, 1.0f,
         0.5f, -0.5f, -0.5f,  0.0f, 1.0f,
         0.5f, -0.5f,  0.5f,  0.0f, 0.0f,
         0.5f,  0.5f,  0.5f,  1.0f, 0.0f,

        -0.5f, -0.5f, -0.5f,  0.0f, 1.0f,
         0.5f, -0.5f, -0.5f,  1.0f, 1.0f,
         0.5f, -0.5f,  0.5f,  1.0f, 0.0f,
         0.5f, -0.5f,  0.5f,  1.0f, 0.0f,
        -0.5f, -0.5f,  0.5f,  0.0f, 0.0f,
        -0.5f, -0.5f, -0.5f,  0.0f, 1.0f,

        -0.5f,  0.5f, -0.5f,  0.0f, 1.0f,
         0.5f,  0.5f, -0.5f,  1.0f, 1.0f,
         0.5f,  0.5f,  0.5f,  1.0f, 0.0f,
         0.5f,  0.5f,  0.5f,  1.0f, 0.0f,
        -0.5f,  0.5f,  0.5f,  0.0f, 0.0f,
        -0.5f,  0.5f, -0.5f,  0.0f, 1.0f
    };
    // world space positions of our cubes
    // translation
    glm::vec3 cubePositions[] = {
        glm::vec3( 0.75f,  0.0f,  0.0f),
        glm::vec3( -0.75f,  0.0f, 0.0f)
    };
    unsigned int VBO, VAO;
    glGenVertexArrays(1, &VAO);
    glGenBuffers(1, &VBO);

    glBindVertexArray(VAO);

    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices,
GL_STATIC_DRAW);

    // position attribute
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 5 * sizeof(float),
(void*)0);
    glEnableVertexAttribArray(0);
    // texture coord attribute
    glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, 5 * sizeof(float),
(void*)(3 * sizeof(float)));
    glEnableVertexAttribArray(1);


    // load and create a texture
    // -------------------------
```

```
    unsigned int texture1, texture2;
    // texture 1
    // ---------
    glGenTextures(1, &texture1);
    glBindTexture(GL_TEXTURE_2D, texture1);
    // set the texture wrapping parameters
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    // set texture filtering parameters
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    // load image, create texture and generate mipmaps
    int width, height, nrChannels;
    stbi_set_flip_vertically_on_load(true); // tell stb_image.h to flip
loaded texture's on the y-axis.
    unsigned char *data = stbi_load("resources//textures//container.jpg",
&width, &height, &nrChannels, 0);
    if (data)
    {
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB,
GL_UNSIGNED_BYTE, data);
        glGenerateMipmap(GL_TEXTURE_2D);
    }
    else
    {
        std::cout << "Failed to load texture" << std::endl;
    }
    stbi_image_free(data);
    // texture 2
    // ---------
    glGenTextures(1, &texture2);
    glBindTexture(GL_TEXTURE_2D, texture2);
    // set the texture wrapping parameters
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    // set texture filtering parameters
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    // load image, create texture and generate mipmaps
    data = stbi_load("resources//textures//awesomeface.png", &width,
&height, &nrChannels, 0);
    if (data)
    {
        // note that the awesomeface.png has transparency and thus an alpha
channel, so make sure to tell OpenGL the data type is of GL_RGBA
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGBA,
GL_UNSIGNED_BYTE, data);
        glGenerateMipmap(GL_TEXTURE_2D);
```

```
    }
    else
    {
        std::cout << "Failed to load texture" << std::endl;
    }
    stbi_image_free(data);

    // tell opengl for each sampler to which texture unit it belongs to
(only has to be done once)
    // -------------------------------------------------------------------
----------------------
    ourShader.use();
    ourShader.setInt("texture1", 0);
    ourShader.setInt("texture2", 1);


    // render loop
    // -----------
    while (!glfwWindowShouldClose(window))
    {
        // input
        // -----
        processInput(window);

        // render
        // ------
        glClearColor(0.2f, 0.3f, 0.3f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // also clear
the depth buffer now!

         // bind textures on corresponding texture units
        glActiveTexture(GL_TEXTURE0);
        glBindTexture(GL_TEXTURE_2D, texture1);
        glActiveTexture(GL_TEXTURE1);
        glBindTexture(GL_TEXTURE_2D, texture2);

        // activate shader
        ourShader.use();

        // create transformations
        glm::mat4 view          = glm::mat4(1.0f); // make sure to
initialize matrix to identity matrix first
        glm::mat4 projection    = glm::mat4(1.0f);
        projection = glm::perspective(glm::radians(45.0f), (float)SCR_WIDTH
/ (float)SCR_HEIGHT, 0.1f, 100.0f);
        view       = glm::translate(view, glm::vec3(0.0f, 0.0f, -5.0f));
        // pass transformation matrices to the shader
```

```
        ourShader.setMat4("projection", projection); // note: currently we
set the projection matrix each frame, but since the projection matrix rarely
changes it's often best practice to set it outside the main loop only once.
        ourShader.setMat4("view", view);

        // render boxes
        glBindVertexArray(VAO);
        for (unsigned int i = 0; i < 2; i++)
        {
            // calculate the model matrix for each object and pass it to
shader before drawing
            glm::mat4 model = glm::mat4(1.0f);
            model = glm::translate(model, cubePositions[i]);
            float angle = 20.0f * i;
            float scaleAmount = static_cast<float>(sin(glfwGetTime()));
            if(i%2==0)
                model = glm::rotate(model, (float)glfwGetTime(),
glm::vec3(1.0f, 0.3f, 0.5f));
            // if(i%2==1)
            //     model = glm::scale(model, glm::vec3(scaleAmount,
scaleAmount, scaleAmount));
            ourShader.setMat4("model", model);


            glDrawArrays(GL_TRIANGLES, 0, 36);
        }

        // glfw: swap buffers and poll IO events (keys pressed/released,
mouse moved etc.)
        // -------------------------------------------------------------------
--------------
        glfwSwapBuffers(window);
        glfwPollEvents();
    }

    // optional: de-allocate all resources once they've outlived their
purpose:
    // ------------------------------------------------------------------------
---
    glDeleteVertexArrays(1, &VAO);
    glDeleteBuffers(1, &VBO);

    // glfw: terminate, clearing all previously allocated GLFW resources.
    // ------------------------------------------------------------------
    glfwTerminate();
    return 0;
}
```

```
// process all input: query GLFW whether relevant keys are pressed/released
this frame and react accordingly
// ---------------------------------------------------------------------------
-------------------------------
void processInput(GLFWwindow *window)
{
    if (glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS)
        glfwSetWindowShouldClose(window, true);
}

// glfw: whenever the window size changed (by OS or user resize) this
callback function executes
// ---------------------------------------------------------------------------
--------------------
void framebuffer_size_callback(GLFWwindow* window, int width, int height)
{
    // make sure the viewport matches the new window dimensions; note that
width and
    // height will be significantly larger than specified on retina
displays.
    glViewport(0, 0, width, height);
}
```

```
#version 330 core
out vec4 FragColor;

in vec2 TexCoord;

// texture samplers
uniform sampler2D texture1;
uniform sampler2D texture2;

void main()
{
    // linearly interpolate between both textures (80% container, 20%
awesomeface)
    FragColor = mix(texture(texture1, TexCoord), texture(texture2,
TexCoord), 0.2);
}
```

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec2 aTexCoord;

out vec2 TexCoord;
```

```
uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

void main()
{
     gl_Position = projection * view * model *
vec4(aPos, 1.0f);
     TexCoord = vec2(aTexCoord.x, aTexCoord.y);
}
```

Output: