BURSA ULUDAĞ ÜNİVERSİTESİ

BİLGİSAYAR MÜHENDİSLİĞİ

2024-2025 EĞİTİM ÖĞRETİM YILI BAHAR DÖNEMİ

BİLGİSAYAR GRAFİKLERİ RAPORU

MURAT BERK YETİŞTİRİR

032290008

032290008@ogr.uludag.edu.tr

SORU: Bir OpenGL penceresine istediğiniz birer arka plan ve dolgu rengi ile 2-B koordinat sisteminde içi boş veya dolu bir sekizgen tabanlı yıldız çiziniz. Ana profilde (Core profile) modern OpenGL ile kodu geliştiriniz. Görselleme sürecinde çizdirilecek nokta kümesini tercihe bağlı olarak çember denklemine göre oluşturunuz. Tam yorumlu kodunuzu ve OpenGL çıktısını içeren bir rapor hazırlayınız. Raporun içine ve dosya ismine adınızı, soyadınızı ve öğrenci numaranızı yazınız. Dosyayı pdf olarak kaydedip son teslim tarihinden önce UKEY'deki Lab1 ödevi arayüzüne yükleyiniz.

CEVAP KODUM:

```
#include <glad/glad.h>
#include <GLFW/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"
"void main()\n"
"{\n"
"   gl_Position = vec4(aPos.x, aPos.y, aPos.z, 1.0);\n"
"}\0";
const char* fragmentShaderSource = "#version 330 core\n"
"out vec4 FragColor;\n"
"void main()\n"
"{\n"
"   FragColor = vec4(0.2f, 0.4f, 0.2f, 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[] = {
    // 1. Üçgen
    -0.1f, 0.3f, 0.0f, // left
    0.1f, 0.3f, 0.0f, // right
    0.0f,  0.6f, 0.0f, // top
    // 2. Üçgen
    0.1f, 0.3f, 0.0f, // left
    0.3f, 0.1f, 0.0f, // right
    0.4f,  0.4f, 0.0f, // top
    // 3. Üçgen
    0.3f, 0.1f, 0.0f, // left
    0.3f, -0.1f, 0.0f, // right
    0.6f,  0.0f, 0.0f, // top
    // 4. Üçgen
    0.3f, -0.1f, 0.0f, // left
    0.1f, -0.3f, 0.0f, // right
    0.4f, -0.4f, 0.0f, // top
    // 5. Üçgen
    0.1f, -0.3f, 0.0f, // left
    -0.1f, -0.3f, 0.0f, // right
    -0.0f, -0.6f, 0.0f, // top
    // 6. Üçgen
    -0.1f, -0.3f, 0.0f, // left
    -0.3f, -0.1f, 0.0f, // right
    -0.4f, -0.4f, 0.0f, // top
    // 7. Üçgen
    -0.3f, -0.1f, 0.0f, // left
    -0.3f, 0.1f, 0.0f, // right
    -0.6f, -0.0f, 0.0f, // top
    // 8. Üçgen
```

```
        -0.3f, 0.1f, 0.0f, // left
        -0.1f, 0.3f, 0.0f, // right
        -0.4f, 0.4f, 0.0f, // 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);

    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0);
    glEnableVertexAttribArray(0);

    glBindBuffer(GL_ARRAY_BUFFER, 0);
    glBindVertexArray(0);

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

        // render
        // ------
        glClearColor(0.7f, 0.7f, 0.7f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT);

        // draw our first triangle
        glUseProgram(shaderProgram);
        glBindVertexArray(VAO);
        glDrawArrays(GL_TRIANGLES, 0, 24);
        // glBindVertexArray(0); // no need to unbind it every time

        // 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;
}

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

void framebuffer_size_callback(GLFWwindow* window, int width, int height)
{
    glViewport(0, 0, width, height);
}
```

CEVAP EKRAN GÖRÜNTÜSÜ: