BURSA ULUDAĞ ÜNİVERSİTESİ

BİLGİSAYAR MÜHENDİSLİĞİ

2023-2024 EĞİTİM ÖĞRETİM YILI BAHAR DÖNEMİ

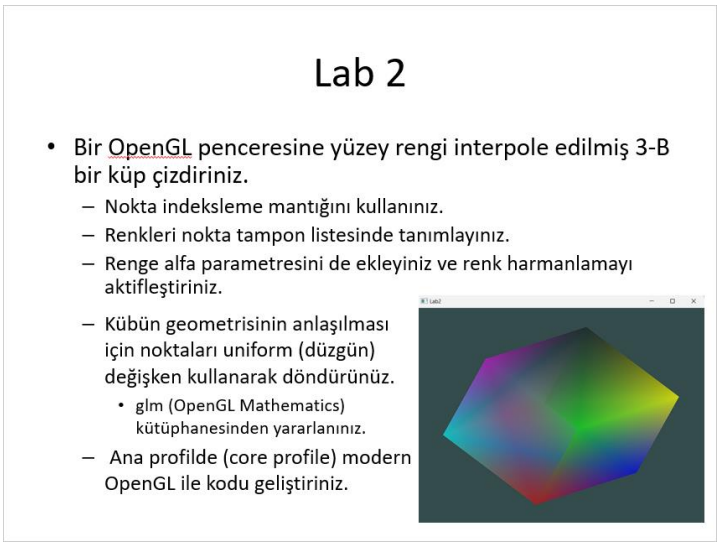
BİLGİSAYAR GRAFİKLERİ RAPORU

MURAT BERK YETİŞTİRİR

032290008

032290008@ogr.uludag.edu.tr

SORU: 03_Renklendirme ve indeksleme ders notunun son slaytına bakınız.



CEVAP KODU:

```
#include <glad/glad.h>
#include <GLFW/glfw3.h>
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>

#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 vec4 aColor;\n"
"out vec4 ourColor;\n"
"uniform mat4 model;"
"void main()\n"
"{\n"
"   gl_Position = model * vec4(aPos,1.0);\n"
"   ourColor = aColor;\n"
"}\0";

const char* fragmentShaderSource = "#version 330 core\n"
"out vec4 FragColor;\n"
"in vec4 ourColor;\n"
"void main()\n"
"{\n"
"   FragColor = ourColor;\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
    // ------------------------------------------------------------------

/*
    typedef GLint vertex3[3];
    vertex3 pt[8] = {{0, 0, 0}, {0, 0.5f, 0}, {0.5f, 0, 0}, {0.5f, 0.5f, 0}, {0, 0, 0.5f}, {0,
0.5f, 0.5f}, {0.5f, 0, 0.5f}, {0.5f, 0.5f, 0.5f}};
```

```
*/

    float vertices[] = {
        -0.5f,-0.5f,-0.5f,     1.0f, 0.0f, 0.0f, 0.3f,
        -0.5f,0.5f,-0.5f,      0.0f, 1.0f, 0.0f, 0.6f,
        0.5f,-0.5f,-0.5f,      0.0f, 0.0f, 1.0f, 0.4f,
        0.5f,0.5f,-0.5f,       1.0f, 1.0f, 0.0f, 0.5f,
        -0.5f,-0.5f,0.5f,      0.0f, 1.0f, 1.0f, 0.4f,
        -0.5f,0.5f,0.5f,       1.0f, 0.0f, 1.0f, 0.3f,
        0.5f,-0.5f,0.5f,       0.9f, 0.8f, 0.9f, 0.1f,
        0.5f,0.5f,0.5f,        0.0f, 0.0f, 0.0f, 0.2f
    };

    unsigned int indices[] = {
        6,2,3,
        3,7,6,
        5,1,0,
        0,4,5,
        7,3,1,
        1,5,7,
        4,0,2,
        2,6,4,
        2,0,1,
        1,3,2,
        7,5,4,
        4,6,7
    };
    unsigned int VBO, VAO, EBO;
    glGenVertexArrays(1, &VAO);
    glGenBuffers(1, &VBO);
    glGenBuffers(1, &EBO);
    // 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);

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

    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 7 * sizeof(float), (void*)0);
    glEnableVertexAttribArray(0);

    glVertexAttribPointer(1, 4, GL_FLOAT, GL_FALSE, 7 * sizeof(float), (void*)(3 *
sizeof(float)));
    glEnableVertexAttribArray(1);

    // note that this is allowed, the call to glVertexAttribPointer registered VBO as the
vertex attribute's bound vertex buffer object so afterwards we can safely unbind
    glBindBuffer(GL_ARRAY_BUFFER, 0);

    // remember: do NOT unbind the EBO while a VAO is active as the bound element buffer
object IS stored in the VAO; keep the EBO bound.
    //glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);

    // You can unbind the VAO afterwards so other VAO calls won't accidentally modify this
VAO, but this rarely happens. Modifying other
    // VAOs requires a call to glBindVertexArray anyways so we generally don't unbind VAOs
(nor VBOs) when it's not directly necessary.
    glBindVertexArray(0);


    // uncomment this call to draw in wireframe polygons.
    //glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);

    //glEnable(GL_BLEND);
    //glBlendFunc();
```

```
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

    // render loop
    // -----------
    int angle = 0.0f;
    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);

        // draw our first triangle
        glUseProgram(shaderProgram);
        glBindVertexArray(VAO); // seeing as we only have a single VAO there's no need to bind
it every time, but we'll do so to keep things a bit more organized
        glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
        //glDrawArrays(GL_TRIANGLES, 0, 6);
        // glBindVertexArray(0); // no need to unbind it every time

        glm::mat4 model = glm::mat4(1.0f);

        float rotationSpeed = 0.1f; // Adjust the speed by changing this value
        float rotationValue = angle * rotationSpeed;;

        model = glm::rotate(model, glm::radians(rotationValue), glm::vec3(0.5f, 0.5f, 0.5f));

        angle++;

        unsigned int modelLoc = glGetUniformLocation(shaderProgram, "model");
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, &model[0][0]);

        // glfw: swap buffers and poll IO events (keys pressed/released, mouse moved etc.)
        glDrawElements(GL_TRIANGLES, sizeof(indices) / sizeof(unsigned int), GL_UNSIGNED_INT,
0);
        // -------------------------------------------------------------------------------
        glfwSwapBuffers(window);
        glfwPollEvents();
    }

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

CEVAP EKRAN ÇIKTISI: