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 yüzey rengi interpole edilmiş 3-B bir küp çizdiriniz. Çekirdek profilde (Core profile) modern OpenGL ile kodu geliştiriniz. Nokta indeksleme mantığını kullanınız. Renkleri nokta tampon listesinde tanımlayınız. Renge alfa parametresini de ekleyiniz ve renk harmanlamayı aktifleştiriniz. Kübün geometrisinin anlaşılması için noktaları uniform (düzgün) değişken kullanarak döndürünüz. glm (OpenGL Mathematics) kütüphanesinden yararlanınız. 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 Lab2 ödevi arayüzüne yükleyiniz.

CEVAP KODUM:

```
#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);

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"
"uniform mat4 transform;\n"
"void main()\n"
"{\n"
"   gl_Position = transform * 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, 0.5);\n"
"}\0";

int main()
{
    glfwInit();
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);

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

    if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))
    {
        std::cout << "Failed to initialize GLAD" << std::endl;
        return -1;
    }

    unsigned int vertexShader = glCreateShader(GL_VERTEX_SHADER);
```

```
glShaderSource(vertexShader, 1, &vertexShaderSource, NULL);
glCompileShader(vertexShader);

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;
}

unsigned int fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
glShaderSource(fragmentShader, 1, &fragmentShaderSource, NULL);
glCompileShader(fragmentShader);

glGetShaderiv(fragmentShader, GL_COMPILE_STATUS, &success);
if (!success)
{
    glGetShaderInfoLog(fragmentShader, 512, NULL, infoLog);
    std::cout << "ERROR::SHADER::FRAGMENT::COMPILATION_FAILED\n" << infoLog << std::endl;
}

unsigned int shaderProgram = glCreateProgram();
glAttachShader(shaderProgram, vertexShader);
glAttachShader(shaderProgram, fragmentShader);
glLinkProgram(shaderProgram);

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);

float vertices[] = {
    // Positions          // Colors
    -0.5f, -0.5f, -0.5f,  1.0f, 0.0f, 0.0f,
     0.5f, -0.5f, -0.5f,  0.0f, 1.0f, 0.0f,
     0.5f,  0.5f, -0.5f,  0.0f, 0.0f, 1.0f,
    -0.5f,  0.5f, -0.5f,  1.0f, 1.0f, 0.0f,
    -0.5f, -0.5f,  0.5f,  1.0f, 0.0f, 1.0f,
     0.5f, -0.5f,  0.5f,  0.0f, 1.0f, 1.0f,
     0.5f,  0.5f,  0.5f,  1.0f, 1.0f, 1.0f,
    -0.5f,  0.5f,  0.5f,  0.0f, 0.0f, 0.0f
};

unsigned int indices[] = {
    0, 1, 2, 2, 3, 0,
    1, 5, 6, 6, 2, 1,
    5, 4, 7, 7, 6, 5,
    4, 0, 3, 3, 7, 4,
    3, 2, 6, 6, 7, 3,
    4, 5, 1, 1, 0, 4
};

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);

glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)0);
```

```
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)(3 *
sizeof(float)));
    glEnableVertexAttribArray(1);

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

    while (!glfwWindowShouldClose(window))
    {
        processInput(window);
        glClearColor(0.2f, 0.3f, 0.3f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT);

        glm::mat4 transform = glm::rotate(glm::mat4(1.0f), (float)glfwGetTime(),
glm::normalize(glm::vec3(1.0f, 1.0f, 0.0f)));
        glUseProgram(shaderProgram);
        unsigned int transformLoc = glGetUniformLocation(shaderProgram, "transform");
        glUniformMatrix4fv(transformLoc, 1, GL_FALSE, glm::value_ptr(transform));

        glBindVertexArray(VAO);
        glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_INT, 0);
        glfwSwapBuffers(window);
        glfwPollEvents();
    }

    glDeleteVertexArrays(1, &VAO);
    glDeleteBuffers(1, &VBO);
    glDeleteBuffers(1, &EBO);
    glDeleteProgram(shaderProgram);

    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: