BURSA ULUDAĞ ÜNİVERSİTESİ

BİLGİSAYAR MÜHENDİSLİĞİ

2024-2025 EĞİTİM ÖĞRETİM YILI BAHAR DÖNEMİ

BİLGİSAYAR GRAFİKLERİ RAPORU

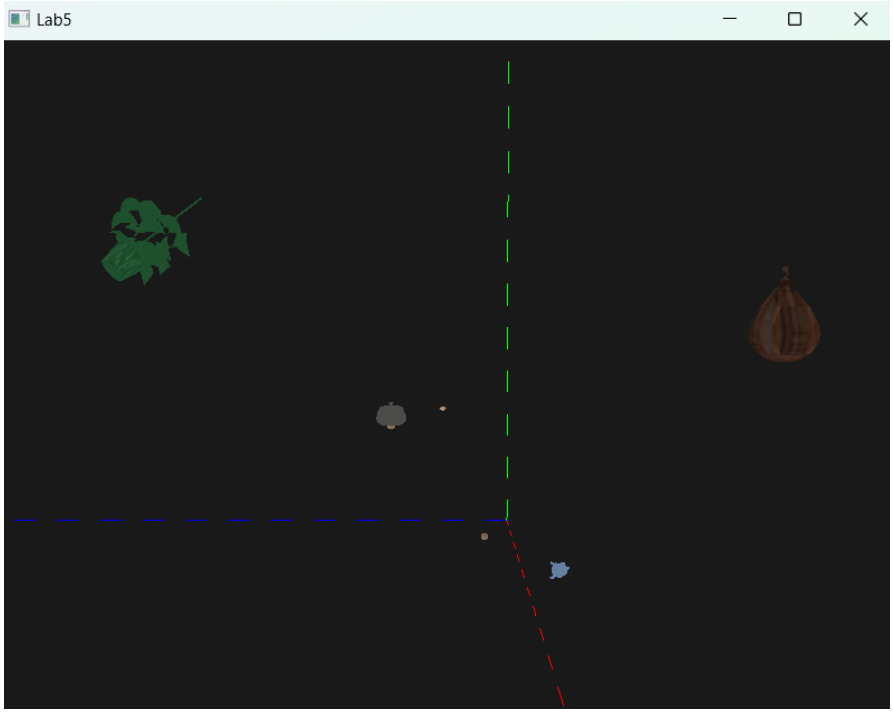
MURAT BERK YETİŞTİRİR

032290008

032290008@ogr.uludag.edu.tr

SORU:

- Çeşitli obje modellerine farklı doku ve geometrik dönüşümler uygulayarak 3-B bir sahne görünümünü görselleyiniz.
    - Dönüşümler için glm kütüphanesinden yararlanınız.
    - Programdaki modelleri okumada assimp kütüphanesinden yararlanınız.
    - 3-B sahnede gezinme için hazır klavye veya fare fonksiyonlarını ekleyiniz.
    - Bakış dönüşümleri için camera sınıfından yararlanınız.
    - model, camera, filesystem, shader ve stb_image kütüphanelerini ekleyiniz.
    - OFF ve OBJ dosya formatlarını inceleyip OBJ formatı için dokuyu harici ve materyal özelliği olarak tanımlama seçeneklerini inceleyiniz.
    - Sırasıyla yüklenecek modeller, dokular ve uygulanacak dönüşümler için bu slaytın notlar kısmına bakınız.
    - Konumunu belirlemek istediğimiz sabit noktaları ölçeklenmiş gezegen modeli veya kaya modeli şeklinde çizdiriniz.
    - x-y-z eksenlerini R-G-B renklerinde ayrı bir shader uygulaması üzerinden çizdiriniz.
    - Tam yorumlu kodunuzu ve OpenGL çıktısının kısa bir videosunu 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 Lab5 ödevi arayüzüne yükleyiniz.
    - 1-) x-y-z eksenlerinde (0.3, 0.3, -0.3) ölçekleme ve (5, 5, 5) kaydırma ile çaydanlık obj modelini metal dokuda çizdirme
    - 2-) x-y-z eksenlerinde (0.05, 0.05, 0.05) ölçekleme ve (5, 5, 5) kaydırma ile gezegen obj modelini orijinal mars dokusunda çizdirme
    - 3-) x-y-z eksenlerinde (4, 4, 4) ölçekleme ve iki noktası P0 = (-2, 5, 3) ve P1 = (-8, 9, 5) olan bir genel eksen etrafında 60° döndürme ile ev bitkisi off modelini yeşil mermer dokuda çizdirme
    - 4-) x-y-z eksenlerinde (0.1, 0.1, 0.1) ölçekleme ve (-2, 5, 3) kaydırma ile kaya obj modelini orijinal kaya dokusunda çizdirme
    - 5-) (10, 10, -10) sabit noktasına göre x-y-z eksenlerinde sırasıyla (0.1, -0.1, 0.1) ölçekleme ile hava balonu obj modelini ahşap dokuda çizdirme
    - 6-) (-3, -3, 3) orijinli, x'-y'-z' eksenleri sırasıyla z-x-y eksenleriyle hizalı ve ölçeği x'-y'-z' eksenlerinde (0.5, 0.5, 0.5) olan yerel koordinat sisteminde balık off modelini gökyüzü dokusunda çizdirme
    - 7-) x-y-z eksenlerinde (0.05, 0.05, 0.05) ölçekleme ve (-3, -3, 3) kaydırma ile gezegen obj modelini orijinal dokusunda çizdirme



CEVAP KODUM:

```
#include <glad/glad.h>
#include <GLFW/glfw3.h>
#include <stb_image.h>

#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>

#include <learnopengl/filesystem.h>
#include <learnopengl/shader.h>
#include <learnopengl/camera.h>
#include <learnopengl/model.h>

#include <iostream>
#define NUM_OF_POINTS 30


void framebuffer_size_callback(GLFWwindow* window, int width, int height);
void mouse_callback(GLFWwindow* window, double xpos, double ypos);
void scroll_callback(GLFWwindow* window, double xoffset, double yoffset);
void processInput(GLFWwindow* window);

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

// camera
Camera camera(glm::vec3(0.0f, 0.0f, 55.0f));
float lastX = (float)SCR_WIDTH / 2.0;
float lastY = (float)SCR_HEIGHT / 2.0;
bool firstMouse = true;

// timing
float deltaTime = 0.0f;
float lastFrame = 0.0f;

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, "Objects", NULL, NULL);
    if (window == NULL)
    {
        std::cout << "Failed to create GLFW window" << std::endl;
        glfwTerminate();
        return -1;
    }
    glfwMakeContextCurrent(window);
    glfwSetFramebufferSizeCallback(window, framebuffer_size_callback);
    glfwSetCursorPosCallback(window, mouse_callback);
    glfwSetScrollCallback(window, scroll_callback);

    // tell GLFW to capture our mouse
    glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_DISABLED);

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

```
    glEnable(GL_DEPTH_TEST);

    // build and compile shaders
    Shader shader("10.3.asteroids.vs", "10.3.asteroids.fs");

    // Teapot
    unsigned int teapotTexture = TextureFromFile("metal.png",
FileSystem::getPath("resources/objects/").c_str(), 0);
    Model teapot(FileSystem::getPath("resources/objects/teapot/teapot.obj"));

    // Mars
    Model mars(FileSystem::getPath("resources/objects/planet/planet.obj"));
    unsigned int marsTexture = TextureFromFile("mars.png",
FileSystem::getPath("resources/objects/planet").c_str(), 0);

    // Rock
    Model rock(FileSystem::getPath("resources/objects/rock/rock.obj"));

    // Air Balloon
    Model airBalloon(FileSystem::getPath("resources/objects/airballoon/Air_Balloon.obj"));
    unsigned int airBalloonTexture = TextureFromFile("wood.png",
FileSystem::getPath("resources/objects/").c_str(), 0);

    // Planet
    Model planet(FileSystem::getPath("resources/objects/planet/planet.obj"));

    // Plant
    Model plant(FileSystem::getPath("resources/objects/houseplant/houseplant.off"));
    unsigned int plantTexture = TextureFromFile("marble.png",
FileSystem::getPath("resources/objects/").c_str(), 0);

    // Fish
    Model fish(FileSystem::getPath("resources/objects/fish/fish.off"));
    unsigned int fishTexture = TextureFromFile("sky.png", FileSystem::getPath("resources/objects/").c_str(),
0);

    Shader shader1("10.3.planet.vs", "10.3.planet.fs");
    float vertices[NUM_OF_POINTS * 3];
    for (int i = 0; i < NUM_OF_POINTS; i++)
    {
        vertices[3 * i] = i;
        vertices[3 * i + 1] = 0;
        vertices[3 * i + 2] = 0;
    }

    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);

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

    glm::mat4 model_axis;
    while (!glfwWindowShouldClose(window))
    {
        // per-frame time logic
        float currentFrame = static_cast<float>(glfwGetTime());
        deltaTime = currentFrame - lastFrame;
        lastFrame = currentFrame;

        // input
        processInput(window);

        // render
        glClearColor(0.1f, 0.1f, 0.1f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        // configure transformation matrices
        glm::mat4 projection = glm::perspective(glm::radians(45.0f), (float)SCR_WIDTH / (float)SCR_HEIGHT,
0.1f, 1000.0f);
        glm::mat4 view = camera.GetViewMatrix();;
        shader.use();
        shader.setMat4("projection", projection);
```

```
shader.setMat4("view", view);

// draw teapot
glm::mat4 teapotModel = glm::mat4(1.0f);
teapotModel = glm::translate(teapotModel, glm::vec3(5, 5, 5));
teapotModel = glm::scale(teapotModel, glm::vec3(0.3, 0.3, -0.3));
glBindTexture(GL_TEXTURE_2D, teapotTexture);
shader.setMat4("model", teapotModel);
teapot.Draw(shader);

// draw mars
glm::mat4 marsModel = glm::mat4(1.0f);
marsModel = glm::translate(marsModel, glm::vec3(5, 5, 5));
marsModel = glm::scale(marsModel, glm::vec3(0.05, 0.05, 0.05));
glBindTexture(GL_TEXTURE_2D, marsTexture);
shader.setMat4("model", marsModel);
mars.Draw(shader);

// draw rock
glm::mat4 rockModel = glm::mat4(1.0f);
rockModel = glm::translate(rockModel, glm::vec3(-2, 5, 3));
rockModel = glm::scale(rockModel, glm::vec3(0.1, 0.1, 0.1));
shader.setMat4("model", rockModel);
rock.Draw(shader);

// draw air-balloon
glm::mat4 airBalloonModel = glm::mat4(1.0f);
airBalloonModel = glm::translate(glm::mat4(1.0f), glm::vec3(10, 10, -10));
airBalloonModel = glm::scale(airBalloonModel, glm::vec3(0.1, -0.1, 0.1));
glBindTexture(GL_TEXTURE_2D, airBalloonTexture);
shader.setMat4("model", airBalloonModel);
airBalloon.Draw(shader);

// draw planet
glm::mat4 planetModel = glm::mat4(1.0f);
rockModel = glm::translate(rockModel, glm::vec3(-3, -3, 3));
planetModel = glm::scale(planetModel, glm::vec3(0.05, 0.05, 0.05));
shader.setMat4("model", planetModel);
planet.Draw(shader);

// draw plant
glm::vec3 P0(-2.0f, 5.0f, 3.0f);
glm::vec3 P1(-8.0f, 9.0f, 5.0f);
glm::vec3 axis = glm::normalize(P1 - P0);
float angleRad = glm::radians(60.0f);

glm::mat4 plantModel = glm::mat4(1.0f);
plantModel = glm::translate(plantModel, -P0);
plantModel = glm::rotate(plantModel, angleRad, axis);
plantModel = glm::translate(plantModel, P0);
plantModel = glm::scale(plantModel, glm::vec3(4.0f, 4.0f, 4.0f));
glBindTexture(GL_TEXTURE_2D, plantTexture);
shader.setMat4("model", plantModel);
plant.Draw(shader);

// draw fish
glm::mat4 fishModel = glm::mat4(1.0f);
glm::mat4 rotation = glm::mat4(glm::mat3(
    glm::vec3(0, 0, 1),  // x' = z
    glm::vec3(1, 0, 0),  // y' = x
    glm::vec3(0, 1, 0)   // z' = y
));
glm::mat4 scale = glm::scale(glm::mat4(1.0f), glm::vec3(0.5f, 0.5f, 0.5f));
glm::mat4 translate = glm::translate(glm::mat4(1.0f), glm::vec3(-3.0f, -3.0f, 3.0f));
fishModel = translate * rotation * scale;
glBindTexture(GL_TEXTURE_2D, fishTexture);
shader.setMat4("model", fishModel);
fish.Draw(shader);

shader1.use();
shader1.setMat4("projection", projection);
shader1.setMat4("view", view);
glBindVertexArray(VAO);

shader1.setVec4("ourColor", 1, 0, 0, 1);
shader1.setMat4("model", glm::mat4(1.0f));
glDrawArrays(GL_LINES, 0, NUM_OF_POINTS);
```

```
        shader1.setVec4("ourColor", 0, 1, 0, 1);
        shader1.setMat4("model",glm::rotate(glm::mat4(1.0f),glm::radians(90.0f),glm::vec3(0, 0, 1)));
        glDrawArrays(GL_LINES, 0, NUM_OF_POINTS);

        shader1.setVec4("ourColor", 0, 0, 1, 1);
        shader1.setMat4("model",glm::rotate(glm::mat4(1.0f),glm::radians(-90.0f),glm::vec3(0, 1, 0)));
        glDrawArrays(GL_LINES, 0, NUM_OF_POINTS);

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

    if (glfwGetKey(window, GLFW_KEY_W) == GLFW_PRESS)
        camera.ProcessKeyboard(FORWARD, deltaTime);
    if (glfwGetKey(window, GLFW_KEY_S) == GLFW_PRESS)
        camera.ProcessKeyboard(BACKWARD, deltaTime);
    if (glfwGetKey(window, GLFW_KEY_A) == GLFW_PRESS)
        camera.ProcessKeyboard(LEFT, deltaTime);
    if (glfwGetKey(window, GLFW_KEY_D) == GLFW_PRESS)
        camera.ProcessKeyboard(RIGHT, deltaTime);
}

// 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);
}

void mouse_callback(GLFWwindow* window, double xposIn, double yposIn)
{
    float xpos = static_cast<float>(xposIn);
    float ypos = static_cast<float>(yposIn);

    if (firstMouse)
    {
        lastX = xpos;
        lastY = ypos;
        firstMouse = false;
    }

    float xoffset = xpos - lastX;
    float yoffset = lastY - ypos;

    lastX = xpos;
    lastY = ypos;

    camera.ProcessMouseMovement(xoffset, yoffset);
}

void scroll_callback(GLFWwindow* window, double xoffset, double yoffset)
{
    camera.ProcessMouseScroll(static_cast<float>(yoffset));
}
```