

BURSA ULUDAĞ ÜNİVERSİTESİ

BİLGİSAYAR MÜHENDİSLİĞİ

2024-2025 EĞİTİM ÖĞRETİM YILI BAHAR DÖNEMİ

BİLGİSAYAR GRAFİKLERİ RAPORU

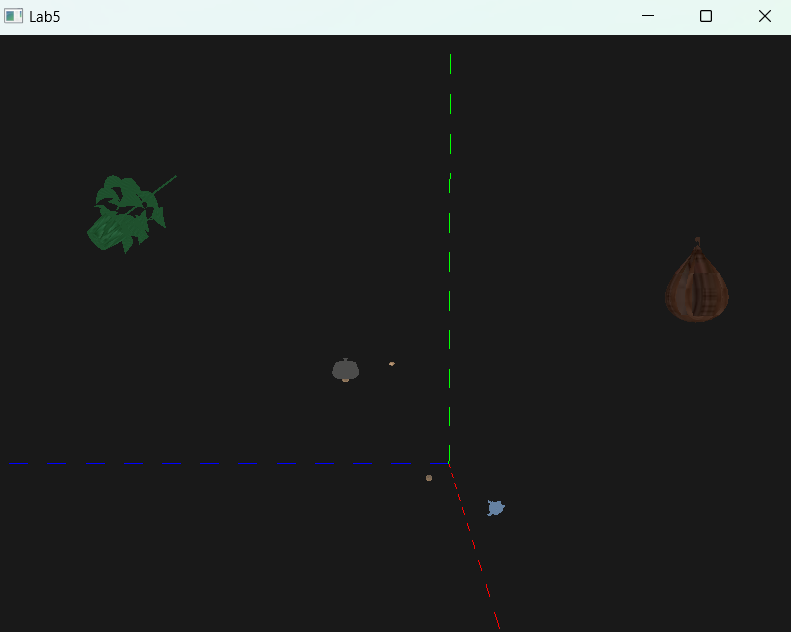
MURAT BERK YETİŞTİRİR

032290008

[032290008@ogr.uludag.edu.tr](mailto: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));

}