

BURSA ULUDAĞ ÜNİVERSİTESİ

BİLGİSAYAR MÜHENDİSLİĞİ

2023-2024 EĞİTİM ÖĞRETİM YILI BAHAR DÖNEMİ

BİLGİSAYAR GRAFİKLERİ RAPORU

MURAT BERK YETİŞTİRİR

032290008

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

**SORU:**

**CEVAP KODU:**

#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>

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, "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);

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;

}

// configure global opengl state

// -----------------------------

glEnable(GL\_DEPTH\_TEST);

// build and compile shaders

// -------------------------

Shader shader("10.2.instancing.vs", "10.2.instancing.fs");

// load models

// -----------

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

unsigned int texture1 = TextureFromFile("top.jpg", FileSystem::getPath("resources/objects/lab4/").c\_str(),0);

Model fish(FileSystem::getPath("resources/objects/lab4/fish/fish.off"));

unsigned int texture2 = TextureFromFile("green.png", FileSystem::getPath("resources/objects/lab4/").c\_str(),0);

Model teapot(FileSystem::getPath("resources/objects/lab4/teapot/teapot.obj"));

// render loop

// -----------

while (!glfwWindowShouldClose(window))

{

// per-frame time logic

// --------------------

float currentFrame = static\_cast<float>(glfwGetTime());

deltaTime = (currentFrame - lastFrame) \* 10;

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 planet

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

model = glm::translate(model, glm::vec3(0.0f, -3.0f, 0.0f));

model = glm::scale(model, glm::vec3(3.0f, 3.0f, 3.0f));

shader.setMat4("model", model);

planet.Draw(shader);

// draw fish

glm::mat4 model2 = glm::mat4(1.0f);

model2 = glm::translate(model2, glm::vec3(20.0f, 0.0f, 0.0f));

model2 = glm::scale(model2, glm::vec3(5.0f, 5.0f, 5.0f));

glBindTexture(GL\_TEXTURE\_2D, texture1);

shader.setMat4("model", model2);

fish.Draw(shader);

// draw teapot

glm::mat4 model3 = glm::mat4(1.0f);

model3 = glm::translate(model3, glm::vec3(30.0f, 0.0f, 0.0f));

model3 = glm::scale(model3, glm::vec3(2.0f, 2.0f, 2.0f));

glBindTexture(GL\_TEXTURE\_2D, texture2);

shader.setMat4("model", model3);

teapot.Draw(shader);

// 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)

{

// 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);

}

// glfw: whenever the mouse moves, this callback is called

// -------------------------------------------------------

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; // reversed since y-coordinates go from bottom to top

lastX = xpos;

lastY = ypos;

camera.ProcessMouseMovement(xoffset, yoffset);

}

// glfw: whenever the mouse scroll wheel scrolls, this callback is called

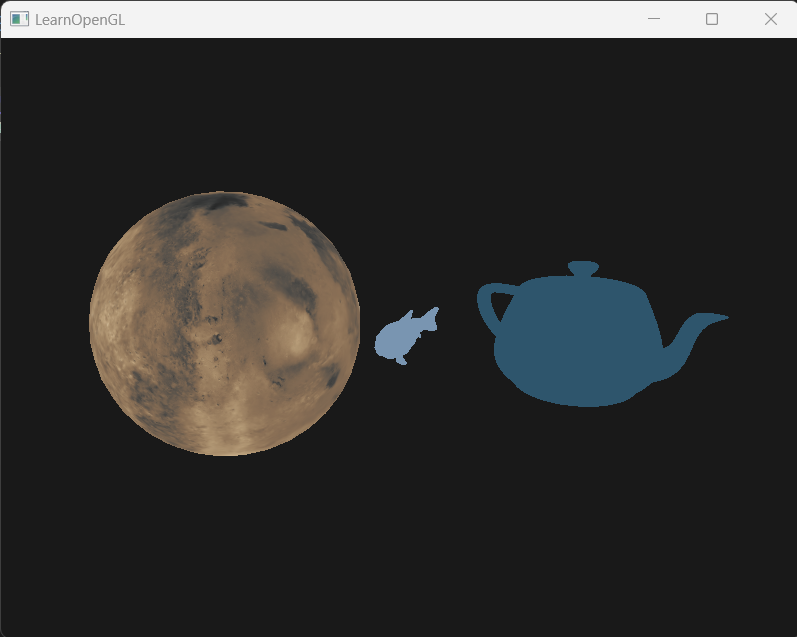
// ----------------------------------------------------------------------

void scroll\_callback(GLFWwindow\* window, double xoffset, double yoffset)

{

camera.ProcessMouseScroll(static\_cast<float>(yoffset));

}

**CEVAP EKRAN GÖRÜNTÜSÜ: **