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: Verilen png görüntülerinden oluşan 2-B bir doğa sahnesi tarayınız. Farklı görüntüler için farklı konum ve ölçeklerle taramayı gerçekleştiriniz. Programı esneterek değişen sayıda görüntüyü aynı gölgelendirici program içerisinde farklı vertex dizi objeleriyle çizdiriniz.Renge alfa parametresini de ekleyiniz ve renk harmanlamayı aktifleştiriniz.filesystem, shader_s, stb_image, freetype ve glm kütüphanelerini ekleyiniz.Aynı sahneye metin taramak için diğer bir gölgelendirici kodunu geliştiriniz.Sahne taramadan sonra metin taramayı aktifleştiriniz. Metin için uygun bir konum ve renk ayarını gerçekleştiriniz.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 Lab3 ödevi arayüzüne yükleyiniz.

CEVAP KODUM:

```
#include <glad/glad.h>
#include <GLFW/glfw3.h>
#include <stb_image.h>
#include <learnopengl/filesystem.h>
#include <learnopengl/shader_s.h>
#include <iostream>
#include <vector>
#include <GL/freeglut.h>

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;

void renderText(float x, float y, const char* text) {
	glMatrixMode(GL_PROJECTION);
	glPushMatrix();
	glLoadIdentity();
	gluOrtho2D(0, 1, 0, 1);
	glMatrixMode(GL_MODELVIEW);
	glPushMatrix();
	glLoadIdentity();

	glColor3f(0.5f, 0.6f, 0.7f);
	glRasterPos2f(x + 0.002f, y - 0.002f);  // Hafif kaydırılmış gölge
	const char* shadow = text;
	while (*shadow) {
		glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, *shadow);
		shadow++;
	}

	glColor3f(0.7f, 0.8f, 0.9f);
	glRasterPos2f(x, y);
	while (*text) {
		glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, *text);
		text++;
	}

	glPopMatrix();
	glMatrixMode(GL_PROJECTION);
	glPopMatrix();
	glMatrixMode(GL_MODELVIEW);
}

struct Object {
	unsigned int VAO, VBO, EBO, texture;
};

Object createObject(float vertices[], unsigned int indices[], const char* texturePath) {
	Object obj;
	glGenVertexArrays(1, &obj.VAO);
	glGenBuffers(1, &obj.VBO);
	glGenBuffers(1, &obj.EBO);

	glBindVertexArray(obj.VAO);
	glBindBuffer(GL_ARRAY_BUFFER, obj.VBO);
	glBufferData(GL_ARRAY_BUFFER, 8 * 4 * sizeof(float), vertices, GL_STATIC_DRAW);
	glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, obj.EBO);
	glBufferData(GL_ELEMENT_ARRAY_BUFFER, 6 * sizeof(unsigned int), indices,
GL_STATIC_DRAW);

	glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)0);
	glEnableVertexAttribArray(0);
	glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(3 *
sizeof(float)));
	glEnableVertexAttribArray(1);
	glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(6 *
sizeof(float)));
	glEnableVertexAttribArray(2);

	glGenTextures(1, &obj.texture);
	glBindTexture(GL_TEXTURE_2D, obj.texture);

	if (std::string(texturePath).find("grass") != std::string::npos) {
		glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
		glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
	}
	else {
		glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
		glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
	}
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

	int width, height, nrChannels;
	unsigned char* data = stbi_load(FileSystem::getPath(texturePath).c_str(), &width,
&height, &nrChannels, 0);
	if (data) {
		GLenum format = (nrChannels == 4) ? GL_RGBA : GL_RGB;
		glTexImage2D(GL_TEXTURE_2D, 0, format, width, height, 0, format,
GL_UNSIGNED_BYTE, data);
		glGenerateMipmap(GL_TEXTURE_2D);
	}
	stbi_image_free(data);

	return obj;
}


int main() {
	int argc = 0;
	char* argv[] = { (char*)"" };
	glutInit(&argc, argv);
	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, "Photo", 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;
	}

	glEnable(GL_BLEND);
	glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
	Shader ourShader("4.1.texture.vs", "4.1.texture.fs");
```

```
stbi_set_flip_vertically_on_load(true);

float picnicVertices[] = {
        1.1f,  0.7f, 0.0f,   1.0f, 0.0f, 0.0f,   1.0f, 1.0f,
        1.1f, -1.1f, 0.0f,   0.0f, 1.0f, 0.0f,   1.0f, 0.0f,
      -0.45f, -1.1f, 0.0f,   0.0f, 0.0f, 1.0f,   0.0f, 0.0f,
      -0.45f,  0.7f, 0.0f,   1.0f, 1.0f, 0.0f,   0.0f, 1.0f
};
unsigned int picnicIndices[] = { 0, 1, 3, 1, 2, 3 };

float sunVertices[] = {
      -0.8f,  0.95f, 0.0f,   1.0f, 0.0f, 0.0f,   1.0f, 1.0f,
      -0.8f,  0.85f, 0.0f,   0.0f, 1.0f, 0.0f,   1.0f, 0.0f,
      -0.9f,  0.85f, 0.0f,   0.0f, 0.0f, 1.0f,   0.0f, 0.0f,
      -0.9f,  0.95f, 0.0f,   1.0f, 1.0f, 0.0f,   0.0f, 1.0f
};
unsigned int sunIndices[] = { 0, 1, 3, 1, 2, 3 };

float childVertices[] = {
      -0.40f,  0.0f, 0.0f,   1.0f, 0.0f, 0.0f,   1.0f, 1.0f,
      -0.40f, -1.0f, 0.0f,   0.0f, 1.0f, 0.0f,   1.0f, 0.0f,
      -0.70f, -1.0f, 0.0f,   0.0f, 0.0f, 1.0f,   0.0f, 0.0f,
      -0.70f,  0.0f, 0.0f,   1.0f, 1.0f, 0.0f,   0.0f, 1.0f
};
unsigned int childIndices[] = { 0, 1, 3, 1, 2, 3 };

float catVertices[] = {
      -0.05f, -0.3f, 0.0f,   1.0f, 0.0f, 0.0f,   1.0f, 1.0f,
      -0.05f, -0.7f, 0.0f,   0.0f, 1.0f, 0.0f,   1.0f, 0.0f,
      -0.25f, -0.7f, 0.0f,   0.0f, 0.0f, 1.0f,   0.0f, 0.0f,
      -0.25f, -0.3f, 0.0f,   1.0f, 1.0f, 0.0f,   0.0f, 1.0f
};
unsigned int catIndices[] = { 0, 1, 3, 1, 2, 3 };

float seaVertices[] = {
        1.0f,  1.5f, 0.0f,   1.0f, 0.0f, 0.0f,   1.0f, 1.0f,
        1.0f, -0.5f, 0.0f,   0.0f, 1.0f, 0.0f,   1.0f, 0.0f,
      -1.0f, -0.5f, 0.0f,   0.0f, 0.0f, 1.0f,   0.0f, 0.0f,
      -1.0f,  1.5f, 0.0f,   1.0f, 1.0f, 0.0f,   0.0f, 1.0f
};
unsigned int seaIndices[] = { 0, 1, 3, 1, 2, 3 };

float grassVertices[] = {
      -0.3f, -0.2f, 0.0f,   1.0f, 0.0f, 0.0f,   0.0f, 8.0f,
      -0.3f, -1.0f, 0.0f,   0.0f, 1.0f, 0.0f,   0.0f, 0.0f,
      -1.0f, -1.0f, 0.0f,   0.0f, 0.0f, 1.0f,   8.0f, 0.0f,
      -1.0f, -0.2f, 0.0f,   1.0f, 1.0f, 0.0f,   8.0f, 8.0f
};
unsigned int grassIndices[] = { 0, 1, 3, 1, 2, 3 };

std::vector<Object> objects = {
      createObject(seaVertices, seaIndices, "resources/textures/sea.png"),
      createObject(grassVertices, grassIndices, "resources/textures/grass.png"),
      createObject(picnicVertices, picnicIndices, "resources/textures/picnic.png"),
      createObject(sunVertices, sunIndices, "resources/textures/sun.png"),
      createObject(childVertices, childIndices, "resources/textures/man.png"),
      createObject(catVertices, catIndices, "resources/textures/cat.png"),
};

while (!glfwWindowShouldClose(window)) {
      glEnable(GL_BLEND);
      glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
      processInput(window);
      glClearColor(0.6f, 0.8f, 1.0f, 0.0f);
      glClear(GL_COLOR_BUFFER_BIT);
      ourShader.use();
      renderText(0.3f, 0.85f, "Picnic on Shore");  // Biraz daha aşağı kaydır
      for (const auto& obj : objects) {
            glBindTexture(GL_TEXTURE_2D, obj.texture);
```

```
                glBindVertexArray(obj.VAO);
                glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);
            }
            glfwSwapBuffers(window);
            glfwPollEvents();
        }

        for (const auto& obj : objects) {
            glDeleteVertexArrays(1, &obj.VAO);
            glDeleteBuffers(1, &obj.VBO);
            glDeleteBuffers(1, &obj.EBO);
        }

        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 ÇIKTISI: