Отчёт к лабораторной работе #4

С дисциплины “Вычислительная геометрия и компьютерная графика”

Студента группы ПА-19-2

Ильяшенко Егора

Постановка задачи:

Изображение выглядит как текст

Автоматически созданное описание

#include <iostream>

#include <string>

#include <glew.h>

#include <glfw3.h>

#include "GLShader.h"

#include "Vertices.h"

#include "GLWindow.h"

#include "GLRenderSystem.h"

#include "stb\_image.h"

#include "MyModel.h"

#include <glm/glm.hpp>

#include <glm/gtc/matrix\_transform.hpp>

#include <glm/gtc/type\_ptr.hpp>

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);

bool firstMouse = true;

float yaw = -90.0f;

float pitch = 0.0f;

float lastX = 800.0f / 2.0;

float lastY = 600.0 / 2.0;

float fov = 45.0f;

const unsigned int SCR\_WIDTH = 800;

const unsigned int SCR\_HEIGHT = 600;

float deltaTime = 0.0f;

float lastFrame = 0.0f;

glm::vec3 cameraPos = glm::vec3(0.0f, 0.0f, 3.0f);

glm::vec3 cameraFront = glm::vec3(0.0f, 0.0f, -1.0f);

glm::vec3 cameraUp = glm::vec3(0.0f, 1.0f, 0.0f);

Vintall::GLWindow\* my\_window;

Shader\* shader;

Model\*dirt, \*grass, \*stone;

void InitWindow()

{

my\_window = new Vintall::GLWindow("Lab\_4\_Vintall", 640, 480);

glfwMakeContextCurrent(my\_window->getGLFWHandle());

glewExperimental = true;

if (glewInit() != GLEW\_OK)

{

fprintf(stderr, "Невозможно инициализировать GLEW\n");

exit(0);

}

}

void InitModel()

{

shader = new Shader("BrightAndDim\_VertexShader.vs", "BrightAndDim\_FragmentShader.fs");

shader->use();

dirt = new Model("dirt/cube.fbx");

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_NEAREST);

grass = new Model("grass/cube.fbx");

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_NEAREST);

stone = new Model("stone/cube.fbx");

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_NEAREST);

}

int map\_size = 10;

int MapSinX(int coord, int max\_value)

{

int result;

double sin\_res = sin(coord) \* 5;

return sin\_res < 0 ? -1 \* (int)sin\_res : (int)sin\_res;

}

int MapSinY(int coord, int max\_value)

{

int result;

double sin\_res = sin(coord+0.5f) \* 5;

return sin\_res < 0 ? -1 \* (int)sin\_res : (int)sin\_res;

}

int\*\*\* InitMap() //0-air 1-stone 2-dirt 3-grass

{

int\*\*\* map\_ = new int\*\* [map\_size];

for (int i = 0; i < map\_size; i++)

{

map\_[i] = new int\* [map\_size];

for (int j = 0; j < map\_size; j++)

map\_[i][j] = new int [map\_size];

}

for (int i = 0; i < map\_size; i++)

for (int j = 0; j < map\_size; j++)

for (int k = 0; k < map\_size; k++)

{

map\_[i][j][k] = rand() % 2;

}

for (int i = 0; i < map\_size; i++)

for (int j = 0; j < map\_size; j++)

for (int k = 0; k < map\_size; k++)

{

if (map\_[i][j][k] == 1)

{

for (int o = 0; o < map\_size; o++)

{

if (map\_[i][j][o] == 0)

{

map\_[i][j][o] = 1;

map\_[i][j][k] = 0;

break;

}

}

}

}

int all\_count;

all\_count = 0;

for (int i = 0; i < map\_size; i++)

for (int j = 0; j < map\_size; j++)

{

int count = 0;

for (int k = 0; k < map\_size; k++)

{

if (map\_[i][j][k] != 0)

count++;

else

break;

}

all\_count += count;

}

all\_count /= (map\_size \* map\_size);

for (int i = 0; i < map\_size; i++)

for (int j = 0; j < map\_size; j++)

{

int row\_count = 0;

for (int k = 0; k < map\_size; k++)

if (map\_[i][j][k] != 0)

row\_count++;

else

break;

row\_count = (row\_count + all\_count) / 2;

for (int k = 0; k < map\_size; k++)

{

if (k < row\_count)

map\_[i][j][k] = 1;

else

map\_[i][j][k] = 0;

}

}

for (int i = 0; i < map\_size; i++)

for (int j = 0; j < map\_size; j++)

for (int k = map\_size-1; k >=0 ; k--)

{

if (map\_[i][j][k] != 0)

{

map\_[i][j][k] = 3;

if (k - 1 >= 0)

map\_[i][j][k - 1] = 2;

if (k - 1 >= 0)

map\_[i][j][k - 2] = 2;

break;

}

}

int\*\*\* map\_buff = new int\*\* [map\_size];

for (int i = 0; i < map\_size; i++)

{

map\_buff[i] = new int\* [map\_size];

for (int j = 0; j < map\_size; j++)

{

map\_buff[i][j] = new int[map\_size];

for (int k = 0; k < map\_size; k++)

{

map\_buff[i][j][k] = map\_[i][j][k];

}

}

}

for (int i = 1; i < map\_size-1; i++)

for (int j = 1; j < map\_size - 1; j++)

for (int k = 1; k < map\_size - 1; k++)

{

if (map\_[i][j][k] != 0

&& map\_[i - 1][j][k] != 0

&& map\_[i][j - 1][k] != 0

&& map\_[i][j][k - 1] != 0

&& map\_[i + 1][j][k] != 0

&& map\_[i][j + 1][k] != 0

&& map\_[i][j][k + 1] != 0)

{

map\_buff[i][j][k] = 0;

}

}

for (int i = 1; i < map\_size - 1; i++)

for (int j = 1; j < map\_size - 1; j++)

for (int k = 1; k <= map\_size - 1; k++)

{

map\_[i][j][k] = map\_buff[i][j][k];

}

return map\_;

}

int main(int argc, char\*\* argv)

{

if (argc == 2)

map\_size = atoi(argv[1]);

fprintf(stdout, "Made by Vintall");

InitWindow();

InitModel();

int\*\*\* map = InitMap();

glfwMakeContextCurrent(my\_window->getGLFWHandle());

glfwSetFramebufferSizeCallback(my\_window->getGLFWHandle(), framebuffer\_size\_callback);

glfwSetCursorPosCallback(my\_window->getGLFWHandle(), mouse\_callback);

glfwSetScrollCallback(my\_window->getGLFWHandle(), scroll\_callback);

glfwSetInputMode(my\_window->getGLFWHandle(), GLFW\_CURSOR, GLFW\_CURSOR\_DISABLED);

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_CULL\_FACE);

glCullFace(GL\_FRONT\_FACE);

shader->use();

glm::mat4 projection = glm::perspective(glm::radians(fov), (float)SCR\_WIDTH / (float)SCR\_HEIGHT, 0.1f, 100.0f);

shader->setMat4("projection", projection);

glfwMakeContextCurrent(my\_window->getGLFWHandle());

glClearColor(0.8f, 0.8f, 1, 1.0f);

while (glfwGetKey(my\_window->getGLFWHandle(), GLFW\_KEY\_ESCAPE) != GLFW\_PRESS && glfwWindowShouldClose(my\_window->getGLFWHandle()) == 0)

{

float currentFrame = glfwGetTime();

deltaTime = currentFrame - lastFrame;

lastFrame = currentFrame;

processInput(my\_window->getGLFWHandle());

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glm::mat4 view = glm::lookAt(cameraPos, cameraPos + cameraFront, cameraUp);

shader->setMat4("view", view);

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

model = glm::rotate(model, glm::radians(-90.f), glm::vec3(1.0f, 0.f, 0.f));

for (int i = 0; i < map\_size; i++)

for (int j = 0; j < map\_size; j++)

for (int k = 0; k < map\_size; k++)

{

model = glm::translate(model, glm::vec3(i \* 2, j \* 2, k \* 2));

shader->setMat4("model", model);

switch (map[i][j][k])

{

case 1:

stone->Draw(\*shader);

break;

case 2:

dirt->Draw(\*shader);

break;

case 3:

grass->Draw(\*shader);

break;

}

model = glm::translate(model, glm::vec3(-i \* 2, -j \* 2, -k \* 2));

}

glfwSwapBuffers(my\_window->getGLFWHandle());

glfwPollEvents();

}

glfwDestroyWindow(my\_window->getGLFWHandle());

glfwTerminate();

return 0;

}

void processInput(GLFWwindow\* window)

{

if (glfwGetKey(window, GLFW\_KEY\_ESCAPE) == GLFW\_PRESS)

glfwSetWindowShouldClose(window, true);

float cameraSpeed = 20 \* deltaTime;

if (glfwGetKey(window, GLFW\_KEY\_W) == GLFW\_PRESS)

cameraPos += cameraSpeed \* cameraFront;

if (glfwGetKey(window, GLFW\_KEY\_S) == GLFW\_PRESS)

cameraPos -= cameraSpeed \* cameraFront;

if (glfwGetKey(window, GLFW\_KEY\_A) == GLFW\_PRESS)

cameraPos -= glm::normalize(glm::cross(cameraFront, cameraUp)) \* cameraSpeed;

if (glfwGetKey(window, GLFW\_KEY\_D) == GLFW\_PRESS)

cameraPos += glm::normalize(glm::cross(cameraFront, cameraUp)) \* cameraSpeed;

}

void framebuffer\_size\_callback(GLFWwindow\* window, int width, int height)

{

glViewport(0, 0, width, height);

}

void mouse\_callback(GLFWwindow\* window, double xpos, double ypos)

{

if (firstMouse)

{

lastX = xpos;

lastY = ypos;

firstMouse = false;

}

float xoffset = xpos - lastX;

float yoffset = lastY - ypos;

lastX = xpos;

lastY = ypos;

float sensitivity = 0.1f;

xoffset \*= sensitivity;

yoffset \*= sensitivity;

yaw += xoffset;

pitch += yoffset;

if (pitch > 89.0f)

pitch = 89.0f;

if (pitch < -89.0f)

pitch = -89.0f;

glm::vec3 front;

front.x = cos(glm::radians(yaw)) \* cos(glm::radians(pitch));

front.y = sin(glm::radians(pitch));

front.z = sin(glm::radians(yaw)) \* cos(glm::radians(pitch));

cameraFront = glm::normalize(front);

}

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

{

fov -= (float)yoffset;

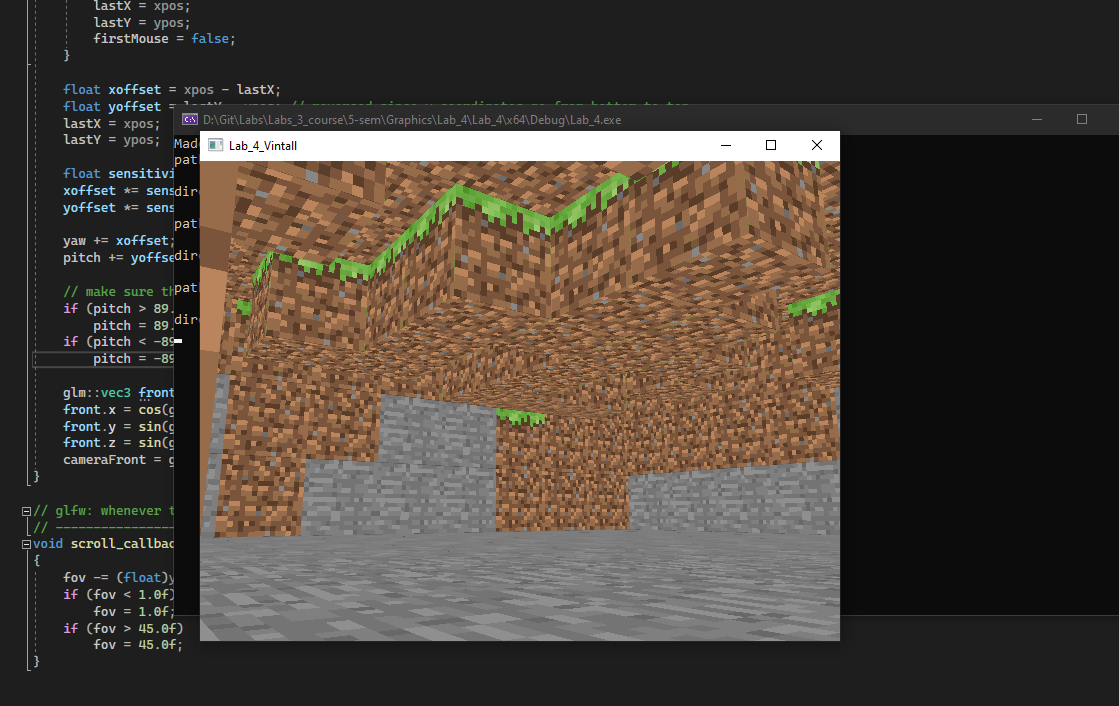
if (fov < 1.0f)

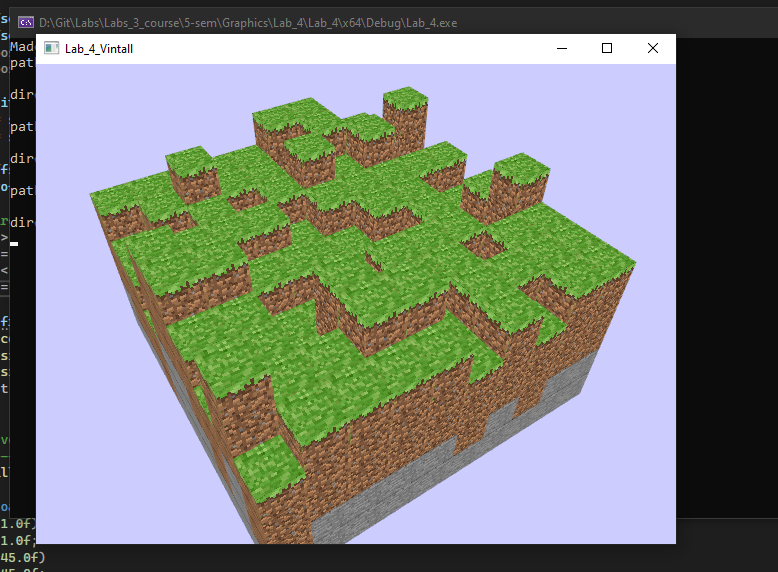
fov = 1.0f;

if (fov > 45.0f)

fov = 45.0f;

}





Заключение: сделал Minecraft :D

Разобрался с шейдерами, моделями, текстурами, камерой.