#include <iostream>

#include <GL/glew.h>

#include <GLFW/glfw3.h>

#include <glm.hpp>

#include <gtc/matrix\_transform.hpp>

#include <gtc/type\_ptr.hpp>

#include <cstdlib>

#include <math.h>

#include <stdlib.h>

#include <vector>

#include <iostream>

#include <fstream>

using namespace std;

float positions[] = {

-0.5f, -0.5f, -0.5f, 1.0f,0.0f,0.0f,

0.5f, -0.5f, -0.5f, 1.0f,0.0f,0.0f,

0.5f, 0.5f, -0.5f, 1.0f,0.0f,0.0f, // BACK

0.5f, 0.5f, -0.5f, 1.0f,0.0f,0.0f,

-0.5f, 0.5f, -0.5f, 1.0f,0.0f,0.0f,

-0.5f, -0.5f, -0.5f, 1.0f,0.0f,0.0f,

-0.5f, -0.5f, 0.5f, 0.0f,1.0f,0.0f,

0.5f, -0.5f, 0.5f, 0.0f,1.0f,0.0f,

0.5f, 0.5f, 0.5f, 0.0f,1.0f,0.0f, // FRONT

0.5f, 0.5f, 0.5f, 0.0f,1.0f,0.0f,

-0.5f, 0.5f, 0.5f, 0.0f,1.0f,0.0f,

-0.5f, -0.5f, 0.5f, 0.0f,1.0f,0.0f,

-0.5f, 0.5f, 0.5f, 0.0f,0.0f,1.0f,

-0.5f, 0.5f, -0.5f, 0.0f,0.0f,1.0f,

-0.5f, -0.5f, -0.5f, 0.0f,0.0f,1.0f, //LEFT

-0.5f, -0.5f, -0.5f, 0.0f,0.0f,1.0f,

-0.5f, -0.5f, 0.5f, 0.0f,0.0f,1.0f,

-0.5f, 0.5f, 0.5f, 0.0f,0.0f,1.0f,

0.5f, 0.5f, 0.5f, 0.0f,0.5f,1.0f,

0.5f, 0.5f, -0.5f, 0.0f,0.5f,1.0f,

0.5f, -0.5f, -0.5f, 0.0f,0.5f,1.0f, // RIGHT

0.5f, -0.5f, -0.5f, 0.0f,0.5f,1.0f,

0.5f, -0.5f, 0.5f, 0.0f,0.5f,1.0f,

0.5f, 0.5f, 0.5f, 0.0f,0.5f,1.0f,

-0.5f, -0.5f, -0.5f, 1.0f,0.5f,0.0f,

0.5f, -0.5f, -0.5f, 1.0f,0.5f,0.0f,

0.5f, -0.5f, 0.5f, 1.0f,0.5f,0.0f, // BOTTOM

0.5f, -0.5f, 0.5f, 1.0f,0.5f,0.0f,

-0.5f, -0.5f, 0.5f, 1.0f,0.5f,0.0f,

-0.5f, -0.5f, -0.5f, 1.0f,0.5f,0.0f,

-0.5f, 0.5f, -0.5f, 1.0f,0.0f,0.5f,

0.5f, 0.5f, -0.5f, 1.0f,0.0f,0.5f,

0.5f, 0.5f, 0.5f, 1.0f,0.0f,0.5f, // TOP

0.5f, 0.5f, 0.5f, 1.0f,0.0f,0.5f,

-0.5f, 0.5f, 0.5f, 1.0f,0.0f,0.5f,

-0.5f, 0.5f, -0.5f, 1.0f,0.0f,0.5f,

};

glm::mat4 projectionTransform(1.0f);

int vertexCt = 0;

int verticesInCube = 6 \* 6;

unsigned int shader;

bool shouldGenerateChunks = true;

bool regenerateChunks = true;

struct {

float width = 800;

float height = 600;

float fov = 45.0;

} screen;

struct {

struct {

double x;

double y;

} mouse;

struct {

bool isKeyPressed[GLFW\_KEY\_MENU];

} keyboard;

} input;

struct {

struct {

float x = 0.0f;

float y = 2.0f;

float z = 0.0f;

} position;

struct {

float x = 0;

float y = 0;

float z = 0;

} looking;

struct {

int health = 5;

int hunger = 0;

} vitals;

float forwardSpeed = 1.0f;

float strafeSpeed = 1.0f;

} player;

struct {

struct {

struct {

int x;

int z;

} current;

int size = 16;

int totalCount = 0;

int toRender = 3;

int rendered = 0;

} chunks;

} world;

float cameraRotX = 0, cameraRotY = 0;

float cameraSpeed = 0.5f;

int \*chunkX = new int[10000];

int \*chunkZ = new int[10000];

intptr\_t \*chunkAddresses = new intptr\_t[10000];

void addChunk(int x, int z) {

chunkX[world.chunks.totalCount] = x;

chunkZ[world.chunks.totalCount] = z;

world.chunks.totalCount++;

}

void key\_callback(GLFWwindow\* window, int key, int scancode, int action, int mods)

{

input.keyboard.isKeyPressed[key] = action;

}

void handleInput() {

if (input.keyboard.isKeyPressed[GLFW\_KEY\_W]) {

player.position.z -= player.forwardSpeed \* cos(glm::radians(cameraRotY));

player.position.x += player.forwardSpeed \* sin(glm::radians(cameraRotY));

}

if (input.keyboard.isKeyPressed[GLFW\_KEY\_S]) {

player.position.z += player.forwardSpeed \* cos(glm::radians(cameraRotY));

player.position.x -= player.forwardSpeed \* sin(glm::radians(cameraRotY));

}

if (input.keyboard.isKeyPressed[GLFW\_KEY\_A]) {

player.position.x -= player.strafeSpeed \* cos(glm::radians(cameraRotY));

player.position.z -= player.strafeSpeed \* sin(glm::radians(cameraRotY));

}

if (input.keyboard.isKeyPressed[GLFW\_KEY\_D]) {

player.position.x += player.strafeSpeed \* cos(glm::radians(cameraRotY));

player.position.z += player.strafeSpeed \* sin(glm::radians(cameraRotY));

}

if (input.keyboard.isKeyPressed[GLFW\_KEY\_LEFT\_SHIFT]) {

player.position.y -= player.forwardSpeed;

}

if (input.keyboard.isKeyPressed[GLFW\_KEY\_SPACE]) {

player.position.y += player.forwardSpeed;

}

if (!(world.chunks.current.x == floor((player.position.x + world.chunks.size / 2) / world.chunks.size)) ||

!(world.chunks.current.z == floor((player.position.z + world.chunks.size / 2) / world.chunks.size))) {

world.chunks.current.x = floor((player.position.x + world.chunks.size / 2) / world.chunks.size);

world.chunks.current.z = floor((player.position.z + world.chunks.size / 2) / world.chunks.size);

regenerateChunks = true;

}

}

static int CompileShader(unsigned int type, const string& source) {

int id = glCreateShader(type);

const char\* src = source.c\_str();

glShaderSource(id, 1, &src, nullptr);

glCompileShader(id);

return id;

}

static unsigned int CreateShader(const string& vertexShader, const string& fragmentShader) {

unsigned int program = glCreateProgram();

unsigned int vs = CompileShader(GL\_VERTEX\_SHADER, vertexShader);

unsigned int fs = CompileShader(GL\_FRAGMENT\_SHADER, fragmentShader);

glAttachShader(program, vs);

glAttachShader(program, fs);

glLinkProgram(program);

glValidateProgram(program);

glDeleteShader(vs);

glDeleteShader(fs);

return program;

}

int blocks = 0;

void createBlock(glm::vec3 position, glm::vec3 rotation, glm::vec3 scale, glm::vec4 color) {

glm::mat4 modelTranslate(1.0f);

glm::mat4 modelRotate(1.0f);

glm::mat4 modelScale(1.0f);

modelTranslate = glm::translate(modelTranslate, glm::vec3(position.x - player.position.x, position.y - player.position.y, position.z - player.position.z));

modelRotate = glm::rotate(modelRotate, glm::radians(0.0f), glm::vec3(1.0f, 0.0f, 0.0f));

modelScale = glm::scale(modelScale, scale);

glm::mat4 modelTransform = modelRotate \* modelTranslate \* modelScale;

int modelUniform = glGetUniformLocation(shader, "model");

glUniformMatrix4fv(modelUniform, 1, GL\_FALSE, glm::value\_ptr(modelTransform));

int colorUniform = glGetUniformLocation(shader, "color");

glUniform4f(colorUniform, color.x, color.y, color.z, color.w);

blocks++;

/\*

int vertices

for (int i = 0; i < 36; i++) {

if (player.looking.z < -.5 && i >= 0 && i<6 && !(player.looking.y < -0.93)) {

vertices[i] = NULL;

continue;

}

if (player.looking.z > .5 && i >= 6 && i<12 && !(player.looking.y < -0.93)) {

vertices[i] = NULL;

continue;

}

if (player.looking.x < -.5 && i >= 12 && i<18 && !(player.looking.y < -0.93)) {

vertices[i] = NULL;

continue;

}

if (player.looking.x > .5 && i >= 18 && i<24 && !(player.looking.y < -0.93)) {

vertices[i] = NULL;

continue;

}

if (player.looking.y < -.5 && i >= 24 && i<30 && !(player.looking.z < -0.93)) {

vertices[i] = NULL;

continue;

}

if (player.looking.y > .5 && i >= 30 && i<36) {

vertices[i] = NULL;

continue;

}

vertices[i] = i;

}

//cout << "vertexCt: " << vertexCt << "/" << verticesInCube << endl;

\*/

/\*

unsigned int elemBuff;

glGenBuffers(1, &elemBuff);

glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, elemBuff);

glBufferData(GL\_ELEMENT\_ARRAY\_BUFFER, verticesInCube \* sizeof(int), vertices, GL\_DYNAMIC\_DRAW);

\*/

glDrawArrays(GL\_TRIANGLES, 0, 36);

//glDrawElements(GL\_TRIANGLES, verticesInCube, GL\_UNSIGNED\_INT, 0);

}

int main(void)

{

GLFWwindow\* window;

if (!glfwInit())

return -1;

window = glfwCreateWindow(screen.width, screen.height, "Hello World", NULL, NULL);

if (!window)

{

glfwTerminate();

return -1;

}

glfwMakeContextCurrent(window);

if (glewInit() != GLEW\_OK) {

//cout << "glew Error" << endl;

}

glEnable(GL\_DEPTH\_TEST);

glfwSetInputMode(window, GLFW\_CURSOR, GLFW\_CURSOR\_HIDDEN);

unsigned int buffer;

glGenBuffers(1, &buffer);

glBindBuffer(GL\_ARRAY\_BUFFER, buffer);

glBufferData(GL\_ARRAY\_BUFFER, sizeof(positions), positions, GL\_DYNAMIC\_DRAW);

glVertexAttribPointer(0, 3, GL\_FLOAT, GL\_FALSE, sizeof(float) \* 6, 0);

//glVertexAttribPointer(1, 3, GL\_FLOAT, GL\_FALSE, sizeof(float) \* 6, (void\*)(3 \* sizeof(float)));

glEnableVertexAttribArray(0);

//glEnableVertexAttribArray(1);

glfwSetKeyCallback(window, key\_callback);

string vertexShader =

"#version 330 core\n"

"\n"

"layout(location = 0) in vec4 position;"

"layout(location = 1) in vec3 color;"

"out vec4 fragColor;"

"uniform mat4 view;"

"uniform mat4 model;"

"uniform mat4 projection;"

"\n"

"void main(){\n"

"fragColor = vec4(color,1.0);"

"gl\_Position = projection \* view \* model \* position;"

"}\n";

string fragmentShader =

"#version 330 core\n"

"\n"

"in vec4 fragColor;"

"out vec4 outColor;"

"uniform vec4 color;"

"\n"

"void main(){\n"

"outColor = color;"

"}\n";

shader = CreateShader(vertexShader, fragmentShader);

glUseProgram(shader);

int projectionUniform = glGetUniformLocation(shader, "projection");

int viewUniform = glGetUniformLocation(shader, "view");

float a = 0;

/\* Loop until the user closes the window \*/

while (!glfwWindowShouldClose(window))

{

/\* Render here \*/

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

//cout << "playerPosition: " << player.position.x << ", " <<player.position.y << ", " << player.position.z << endl;

if (glfwGetWindowAttrib(window, GLFW\_FOCUSED))

{

handleInput();

glfwGetCursorPos(window, &input.mouse.x, &input.mouse.y);

if (input.mouse.x > 0 && input.mouse.y > 0 && input.mouse.x < screen.width && input.mouse.y < screen.height) {

//cout << "mouse: " << input.mouse.x <<", " << input.mouse.y << endl;

if (cameraRotX < 90 && cameraRotX > -90 ||

(cameraRotX >= 90 && (input.mouse.y - screen.height / 2) \* cameraSpeed < 0) ||

(cameraRotX <= -90 && (input.mouse.y - screen.height / 2) \* cameraSpeed > 0)) {

cameraRotX += (input.mouse.y - screen.height / 2) \* cameraSpeed;

}

cameraRotY += (input.mouse.x - screen.width / 2) \* cameraSpeed;

//cout << "cameraRot: " << cameraRotX << ", " << cameraRotY << endl;

player.looking.x = sin(glm::radians(cameraRotY));

player.looking.y = -sin(glm::radians(cameraRotX));

player.looking.z = -cos(glm::radians(cameraRotY));

//cout << "looking: " << player.looking.x << ", " << player.looking.y << ", " << player.looking.z << endl;

}

glfwSetCursorPos(window, screen.width / 2, screen.height / 2);

}

if (shouldGenerateChunks) {

if (regenerateChunks) {

world.chunks.totalCount = 0;

for (int x = 0; x < world.chunks.toRender \* 2 + 1; x++) {

for (int z = 0; z < world.chunks.toRender \* 2 + 1; z++) {

addChunk(world.chunks.current.x + x - world.chunks.toRender, world.chunks.current.z + z - world.chunks.toRender);

}

}

regenerateChunks = false;

}

}

if (world.chunks.totalCount > 0) {

blocks = 0;

for (int i = 0; i < world.chunks.totalCount; i++) {

cout << sizeof(blocks) << endl;

if ((player.looking.z < -.5 && chunkZ[i] <= world.chunks.current.z && chunkZ[i] > world.chunks.current.z - world.chunks.toRender && chunkX[i]> world.chunks.current.x - world.chunks.toRender && chunkX[i]<world.chunks.current.x + world.chunks.toRender) ||

(player.looking.z > .5 && chunkZ[i] >= world.chunks.current.z && chunkZ[i] < world.chunks.current.z + world.chunks.toRender && chunkX[i]> world.chunks.current.x - world.chunks.toRender && chunkX[i]<world.chunks.current.x + world.chunks.toRender) ||

(player.looking.x < -.5 && chunkX[i] <= world.chunks.current.x && chunkX[i] > world.chunks.current.x - world.chunks.toRender && chunkZ[i]> world.chunks.current.z - world.chunks.toRender && chunkZ[i]<world.chunks.current.z + world.chunks.toRender) ||

(player.looking.x > .5 && chunkX[i] >= world.chunks.current.x && chunkX[i] < world.chunks.current.x + world.chunks.toRender && chunkZ[i]> world.chunks.current.z - world.chunks.toRender && chunkZ[i]<world.chunks.current.z + world.chunks.toRender)) {

for (int x = -world.chunks.size / 2; x < world.chunks.size / 2; x++) {

for (int z = -world.chunks.size / 2; z < world.chunks.size / 2; z++) {

createBlock(glm::vec3(x + world.chunks.size \* (chunkX[i]), 0.0f, z + world.chunks.size \* (chunkZ[i])), glm::vec3(1.0f), glm::vec3(1.0f), glm::vec4(0.0f, (x % 2 == 0 && z % 2 == 0) ? (0.8f) : (0.5f), 0.0f, 1.0f));

}

}

world.chunks.rendered++;

}

}

}

else {

createBlock(glm::vec3(0.0f,0.0f,0.0f), glm::vec3(1.0f), glm::vec3(1.0f), glm::vec4(0.0f, 1.0f, 0.0f, 1.0f));

}

//cout << blocks << endl;

//cout << "world.chunks.rendered: " << world.chunks.rendered << "/" << world.chunks.totalCount << endl;

projectionTransform = glm::perspective(glm::radians(screen.fov), screen.width / screen.height, 0.1f, 255.0f);

glUniformMatrix4fv(projectionUniform, 1, GL\_FALSE, glm::value\_ptr(projectionTransform));

glm::mat4 viewTransform(1.0f);

glm::mat4 viewRotateX(1.0f);

glm::mat4 viewRotateY(1.0f);

viewRotateX = glm::rotate(viewRotateX, glm::radians(cameraRotX), glm::vec3(1.0f, 0.0f, 0.0f));

viewRotateY = glm::rotate(viewRotateY, glm::radians(cameraRotY), glm::vec3(0.0f, 1.0f, 0.0f));

viewTransform = viewRotateX \* viewRotateY;// \*viewRotate \* viewScale;

glUniformMatrix4fv(viewUniform, 1, GL\_FALSE, glm::value\_ptr(viewTransform));

a++;

world.chunks.rendered = 0;

//system("cls");

glfwSwapBuffers(window);

glfwPollEvents();

}

delete[] chunkX;

chunkX = NULL;

delete[] chunkZ;

chunkZ = NULL;

glDeleteProgram(shader);

glfwTerminate();

return 0;

}