**Министерство науки и высшего образования РФ**

**ФГБОУ ВО «БРЯНСКИЙ ГОСУДАРСТВЕННЫЙ  
ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**Кафедра «Информатика и программное обеспечение»**

**Курсовая работа**

**по дисциплине «Компьютерная графика»**

Тема: «Интерактивная трехмерная сцена»

Всего листов 24

Студент гр. О-19-ИВТ2-ПО-Б

зач. кн.№ 19.0320

\_\_\_\_\_\_\_\_\_\_\_\_\_Филюткин М. А.

«20» июня 2022 г.

Преподаватель

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Радченко А. О.

«20» июня 2022 г.

**Брянск** **2022 г.**

Содержание

[ВВЕДЕНИЕ 4](#_Toc106626987)

[1. АНАЛИТИЧЕСКАЯ ЧАСТЬ 5](#_Toc106626988)

[1.1. Основные определения 5](#_Toc106626989)

[1.2. Методы выбора объектов 5](#_Toc106626990)

[1.2.1. Буфер выбора 5](#_Toc106626991)

[1.2.2. Выбор цветом 6](#_Toc106626992)

[1.2.3. Выбор лучом 7](#_Toc106626993)

[1.3. Вывод 7](#_Toc106626994)

[2. КОНСТРУКТОРСКАЯ ЧАСТЬ 8](#_Toc106626995)

[2.1. Класс объекта 8](#_Toc106626996)

[2.2. Класс шейдера 9](#_Toc106626997)

[2.3. Обработка освещения 10](#_Toc106626998)

[2.3.1. Вершинный шейдер 10](#_Toc106626999)

[2.3.2. Фрагментный шейдер 11](#_Toc106627000)

[2.4. Создание объектов 13](#_Toc106627001)

[2.5. Выбор цветом 14](#_Toc106627002)

[2.5.1. Определение цвета объекта 14](#_Toc106627003)

[2.5.2. Написание шейдера 14](#_Toc106627004)

[2.5.3. Передача данных в шейдер 15](#_Toc106627005)

[2.5.4. Выбор объекта 15](#_Toc106627006)

[2.6. Взаимодействие с выбранным объектом 17](#_Toc106627007)

[2.7. Отрисовка сцены 17](#_Toc106627008)

[3. ЭКСПЕРЕМЕНТАЛЬНАЯ ЧАСТЬ 19](#_Toc106627009)

[3.1. Тестирование трехмерной сцены 19](#_Toc106627010)

[3.2. Тестирование шейдера цвета 19](#_Toc106627011)

[3.3. Тестирование выбора объекта 20](#_Toc106627012)

[3.4. Взаимодействие с выбранным объектом 22](#_Toc106627013)

[ЗАКЛЮЧЕНИЕ 23](#_Toc106627014)

[СПИСОК ЛИТЕРАТУРЫ 24](#_Toc106627015)

# ВВЕДЕНИЕ

В данной курсовой работе был рассмотрен программный интерфейс OpenGL (Open Graphics Library). Также с его помощью была создана интерактивная трехмерная сцена, которая представляет из себя небольшую карту с различными объектами.

Библиотека OpenGL разработана в качестве низкоуровневого, аппаратно-независимого интерфейса, допускающего реализацию на множестве различных аппаратных платформ. Для того чтобы достичь этих качеств, в состав библиотеки OpenGL не включены никакие команды для выполнения задач работы с окнами или для получения пользовательского ввода; вместо этого необходимо работать через любую систему управления окнами, которая работает с конкретными аппаратными средствами. Точно так же библиотека OpenGL не предоставляет команды высокого уровня для описания моделей трехмерных объектов. При использовании библиотеки OpenGL необходимо создавать нужную модель из ограниченного набора геометрических примитивов — точек, линий и многоугольников.

# АНАЛИТИЧЕСКАЯ ЧАСТЬ

## Основные определения

Полигон – многоугольник, минимальная поверхность для визуализации в трёхмерной графике.

Объект – множество полигонов, создающих определенный трёхмерный предмет.

Буфер – область памяти, используемая для временного хранения данных ввода-вывода.

## Методы выбора объектов

Само понятие выбора объектов больше подходит графическим редакторам. Необходимость выбрать примитив и делать над ним дальнейшие операции одно из первых условий нормальной работы, например, графического редактора. Нет таких программ, где идет работа с графикой и при этом нет возможности выбирать объекты. Также выбор можно делать и в трехмерных графических интерфейсах. С помощью графической библиотеки OpenGL, можно сделать выбор объектов довольно качественно и быстро.

### Буфер выбора

Его достоинства в том, что это, пожалуй, единственный способ выбора объектов, где есть хоть какая-то надежда, что он более аппаратный. Поэтому и работает, возможно, быстрее других. Для OpenGL это также является неким стандартом выбора объектов. При этом минусов у этого способа достаточно. Чего только стоит отсутствие теста на пересечение с другими, не выделяющимися объектами. То есть для OpenGL нужно создать такие условия, при которых на сцене все объекты должны выделяться, тогда все будет работать корректно.

### Выбор цветом

Достоинства в том, что это уже чисто аппаратный способ выбора. Хотя и не всегда это может быть достоинством, особенно если видео карта слабенькая. Такой способ рекомендуется для всего и вся. То есть, и количество объектов может быть огромным и их детализация тоже, и все будет работать быстро и без проблем. И, разумеется, нет никаких глюков при проверке пересечения с другими объектами, которых нельзя выбрать.

Основная его идея в том, что все объекты рисуются своим, уникальным цветом, не обращая внимания на их реальный цвет, после чего уже по цвету определяется сам объект.

Пример исходного изображения (рис. 1).



Рис. 1. Исходное изображение

После обработки с использованием выбора цветом изображение приобретает следующий вид (рис. 2).

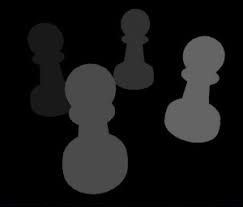


Рис. 2. Изображение после обработки

### Выбор лучом

Идея данного метода так же проста: выпускается луч, после чего проверяется, с каким полигоном он пересёкся, далее уже проверяется к какому объекту относится данный полигон.

## Вывод

Проанализировав имеющиеся варианты методов выбора, было принято решение реализовать выборку объектов методом выбора цветом, поскольку данный вариант является достаточно простым в реализации, а так же не требует большого числа ресурсов для обработки и по сравнению с остальными методами в большинстве случаев работает быстрее, при этом не требуя особых условий для объектов.

# КОНСТРУКТОРСКАЯ ЧАСТЬ

Интерактивная трехмерная сцена создана на языке C++ c использованием библиотек: OpenGL, GLAD, GLFW, GLM и Assimp.

## Класс объекта

Для загрузки объекта был создан класс «Model». Методы класса позволяют загрузить объект, задать ему шейдер, нарисовать объект, а также разместить его на сцене. Так реализованы методы для его перемещения в трехмерном пространстве (Листинг 1).

*Листинг 1. Класс Model*

#ifndef MODEL\_H

#define MODEL\_H

#include "Mesh.h"

#include "Shader.h"

#include "Light.h"

#include <string>

#include <vector>

using namespace std;

class Model

{

public:

vector<Texture> textures\_loaded;

vector<Mesh> meshes;

string directory;

Shader\* shader;

Shader\* pickShader;

glm::vec3 translate;

glm::vec3 rotate;

glm::vec3 scale;

glm::mat4 modelMat4;

unsigned int id;

bool gammaCorrection;

Model(string const& path, Shader\* shader, glm::vec3 translate, glm::vec3 rotate, glm::vec3 scale, bool isUV\_flipped, bool gamma, bool canPick);

void Draw();

void pickDraw();

void updateShader(glm::vec3 cameraPos, glm::mat4 pvMatrix, vector<Light\*> &lights);

void updatePickShader(glm::vec3 cameraPos, glm::mat4 pvMatrix, float alpha = 1.0f, int r = -1, int g = -1, int b = -1);

int r();

int g();

int b();

void setPosition(float x, float y, float z);

void setPosition(glm::vec3 pos);

void setRotate(float x, float y, float z);

void setScale(float scale);

void updateModel();

private:

void loadModel(string const& path, bool isUV\_flipped);

void processNode(aiNode\* node, const aiScene\* scene);

Mesh processMesh(aiMesh\* mesh, const aiScene\* scene);

vector<Texture> loadMaterialTextures(aiMaterial\* mat, aiTextureType type, string typeName);

unsigned int getID();

};

unsigned int TextureFromFile(const char\* path, const string& directory, bool gamma = false);

#endif

## Класс шейдера

Для реализации взаимодействия с шейдером был разработан класс «Shader». С его помощью можно передавать различные параметры в шейдер (Листинг 2).

*Листинг 2. Класс Shader*

#pragma once

#include <glad/glad.h>

#include <glm/glm.hpp>

#include <string>

#include <fstream>

#include <sstream>

#include <iostream>

class Shader

{

public:

Shader(const char\* vertexPath, const char\* fragmentPath);

~Shader();

void use();

void setBool(const std::string& name, bool value) const;

void setInt(const std::string& name, int value) const;

void setFloat(const std::string& name, float value) const;

void setFloatVec(const std::string& name, float\* vec, int vec\_size) const;

void setVec3(const std::string& name, glm::vec3 vec) const;

void setMatrix4F(const std::string& name, glm::mat4& m);

unsigned int ID();

private:

unsigned int programID;

void checkCompileErrors(unsigned int shader, std::string type);

};

## Обработка освещения

Для обработки освещения от различных источников света были разработаны 2 шейдера: вершинный и фрагментный.

### Вершинный шейдер

Данный шейдер принимает в себя параметры вершины (позиция, нормаль, текстурные координаты, тангент и битангент). Далее происходит расчёт позиции и вычисляется матрица TBN (матрица касательного пространства). На выход данный шейдер отправляет текстурные координаты, матрицу TBN и позицию пикселя (Листинг 3).

*Листинг 3. Вершинный шейдер*

#version 330 core

layout (location = 0) in vec3 inPos;

layout (location = 1) in vec3 inNormal;

layout (location = 2) in vec2 inTexCoords;

layout (location = 3) in vec3 inTangent;

layout (location = 4) in vec3 inBiTangent;

out vec2 texCoords;

out vec3 vertNormal;

out mat3 TBN;

out vec3 FragPos;

uniform mat4 pv;

uniform mat4 model;

void main()

{

vec4 vertPos = model \* vec4(inPos, 1.0);

gl\_Position = pv \* vertPos;

texCoords = inTexCoords;

vertNormal = mat3(model)\*inNormal;

FragPos = vertPos.xyz;

vec3 T = normalize((model\*vec4(inTangent, 0.0f)).xyz);

vec3 B = normalize((model\*vec4(inBiTangent, 0.0f)).xyz);

vec3 N = normalize((model\*vec4(inNormal, 0.0f)).xyz);

TBN = mat3(T,B,N);

}

### Фрагментный шейдер

Данный шейдер принимает значения из вершинного шейдера, а также дополнительные данные о текстурах и источниках света в сцене, например, позиция, количество, характеристики света и тому подобные.

После чего происходит расчёт освещения от всех источников света. Результатом работы шейдера является цвет пикселя (Листинг 4).

*Листинг 4. Фрагментный шейдер*

#version 330 core

struct Light {

int type;

vec3 position;

vec3 direction;

float cutOff;

vec3 ambient;

vec3 diffuse;

vec3 specular;

float constant;

float linear;

float quadratic;

};

in vec3 vertColor;

in vec2 texCoords;

in vec3 vertNormal;

in vec3 FragPos;

in mat3 TBN;

out vec4 OutColor;

uniform sampler2D texture\_diffuse1;

uniform sampler2D texture\_specular1;

uniform sampler2D texture\_normal1;

uniform sampler2D texture\_opacity1;

uniform float shininess = 64.0f;

uniform bool haveNormal;

uniform vec3 viewPos;

#define MAX\_LIGHTS 4

uniform int lights\_count;

uniform Light light[MAX\_LIGHTS];

float getAtten(int i)

{

float dist = distance(light[i].position, FragPos);

float attenuation = 1.0 / (light[i].constant + light[i].linear\*dist + light[i].quadratic \* dist \* dist);

return attenuation;

}

vec3 CalcDiffusePlusSpecular(int i, vec3 lightDir)

{

vec3 norm;

if(haveNormal){

norm = texture(texture\_normal1, texCoords).rgb;

norm = normalize(norm \* 2.0f - 1.0f);

norm = normalize(TBN \* norm);

} else {

norm = normalize(vertNormal);

}

float diff\_koef = max(dot(norm, lightDir), 0.0);

vec3 diffuse = light[i].diffuse \* diff\_koef \* vec3(texture(texture\_diffuse1, texCoords));

// specular

vec3 reflectDir = reflect(lightDir, norm);

vec3 viewDir = normalize(FragPos-viewPos);

float spec\_koef = pow(max(dot(viewDir, reflectDir), 0.0f), shininess);

vec3 specular = light[i].specular \* spec\_koef \* vec3(texture(texture\_specular1, texCoords));

return diffuse + specular;

}

void main()

{

vec3 lresult;

for (int i = 0; i<lights\_count; i++)

{

if (light[i].type == 1) // Directional Light

{

vec3 lightDir = -light[i].direction;

vec3 ambient = light[i].ambient \* texture(texture\_diffuse1, texCoords).rgb;

vec3 diffspec = CalcDiffusePlusSpecular(i, lightDir);

lresult = ambient + diffspec;

}

else

{

vec3 lightDir = normalize(light[i].position - FragPos);

if (light[i].type == 2) // Point Light

{

float attenuation = getAtten(i);

vec3 ambient = light[i].ambient \* vec3(texture(texture\_diffuse1, texCoords));

vec3 diffspec = CalcDiffusePlusSpecular(i, lightDir);

lresult = (ambient + diffspec) \* attenuation;

}

else if (light[i].type == 3) // SpotLight

{

float angle = acos(dot(lightDir, normalize(-light[i].direction)));

if (angle <= light[i].cutOff\*2.0f)

{

float koef = 1.0f;

if (angle >= light[i].cutOff)

{

koef = (light[i].cutOff\*2.0f - angle) / light[i].cutOff;

}

float attenuation = getAtten(i);

vec3 ambient = light[i].ambient \* vec3(texture(texture\_diffuse1, texCoords));

vec3 diffspec = CalcDiffusePlusSpecular(i, lightDir) \* koef;

lresult = (ambient + diffspec) \* attenuation;

}

else

{

lresult = vec3(texture(texture\_diffuse1, texCoords)) \* light[i].ambient;

}

}

}

OutColor += vec4(lresult, 1.0f);

}

}

## Создание объектов

Объект создается при помощи конструктора класса «Model», после чего каждый объект необходимо добавить в вектор объектов для дальнейшей работы с ними. Объект принимает на вход путь к объекту на диске, шейдер, его координаты и размеры, местоположение, а также дополнительную информацию для его отрисовки. Ниже предоставлен пример загрузки одного из объектов (Листинг 5).

Листинг 5. Загрузка объекта и добавление в массив

Model dust2("models/dust2/source/de\_dust2.obj", dust2\_shader,

glm::vec3(0.0f, 0.0f, 0.0f), // pos

glm::vec3(270.0f, 0.0f, 0.0f),

glm::vec3(0.01f, 0.01f, 0.01f),

true, false, false);

objects.push\_back(&dust2);

## Выбор цветом

### Определение цвета объекта

Для реализации выбора цветом каждому объекту необходимо было выдать уникальный, не повторяющийся номер.

Каждый объект получает свой уникальный номер, на 1 больший от предыдущего объекта. Притом объекты, которые выбрать нельзя, получали отдельный номер. В данном случае это 0 (Листинг 6, Листинг 7).

*Листинг 6. Выбор ID для объекта*

if (canPick)

this->id = getID();

else this->id = 0;

Листинг 7. Получение уникального ID

unsigned int Model::getID()

{

static int id = 1;

return id++;

}

### Написание шейдера

Далее необходимо был реализован фрагментный шейдер, который отрисовывает объект указанным цветами (Листинг 8).

Листинг 8. Фрагментный шейдер выбора цветом

#version 330 core

out vec4 outColor;

uniform int rID;

uniform int gID;

uniform int bID;

uniform float aID;

void main()

{

outColor = vec4(rID/255.f, gID/255.f,bID/255.f, aID);

}

### Передача данных в шейдер

В целях оптимизации шейдеры для выбора цветом принимают в себя только матрицу проекции, позицию камеры, а также цвета объекта в формате RGB. Ниже предоставлена функция для передачи данных (Листинг 9).

Листинг 9. Функция передачи данных в шейдер

void Model::updatePickShader(glm::vec3 cameraPos, glm::mat4 pvMatrix, float alpha, int r, int g, int b)

{

this->pickShader->use();

this->pickShader->setMatrix4F("pv", pvMatrix);

this->pickShader->setMatrix4F("model", modelMat4);

this->pickShader->setVec3("viewPos", cameraPos);

int Blue = id & 255;

int Green = (id >> 8) & 255;

int Red = (id >> 16) & 255;

if (r >= 0 && g >= 0 && b >= 0)

{

Blue = b;

Green = g;

Red = r;

}

this->pickShader->setInt("rID", Red);

this->pickShader->setInt("gID", Green);

this->pickShader->setInt("bID", Blue);

this->pickShader->setFloat("aID", alpha);

}

### Выбор объекта

Для выбора объекта необходимо понять, когда его вообще нужно выбирать. В данном случае выбор объекта начинается при нажатии на клавишу. Если в данный момент объект уже выбран – считается, что его больше не выбирают, если же объект выбран не был – идёт проверка, какой объект нужно выбрать.

Для начала отрисовывается сцена с объектами, где каждый объект имеет свой, уникальный цвет. Далее идёт процесс получение информации об объекте.

Для получения информации о том, какой именно объект будет выбран разработана функция getPickedObject(). В ней происходит чтение цвета центрального пикселя на экране, после чего перебираются все объекты и если цвет совпадает с цветом одного из объектов – функция возвращает указатель на данный объект, если же объектов не было – возвращает 0. Функция предоставлена ниже (Листинг 10).

Листинг 10. Получение объекта

Model\* getPickedObject()

{

vector< unsigned char > pixels(1 \* 1 \* 4);

glReadPixels

(

1280 / 2, 720 / 2,

1, 1,

GL\_RGBA, GL\_UNSIGNED\_BYTE, &pixels[0]

);

for (auto var : objects)

{

if (var->id == 0) continue;

if (pixels[0] == var->r() && pixels[1] == var->g() && pixels[2] == var->b())

return var;

}

return 0;

}

В самом конце экран очищается от нарисованного до этого картинки. Ниже представлен полный листинг данной функции (Листинг 11).

Листинг 11. Процесс выбора объекта

if (pickClick)

{

if (pickObject == 0)

{

for (auto var : objects)

{

var->updatePickShader(camera.Position, pv);

var->pickDraw();

}

pickObject = getPickedObject();

}

else

pickObject = 0;

pickClick = false;

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

}

## Взаимодействие с выбранным объектом

После выбора объекта имеется возможность перемещать его по сцене. Для этого проверяется имеется ли выбранный объект, если да – он двигается в сторону взгляда камеры на указанное расстояние (Листинг 12).

Листинг 12. Передвижение объекта

if (pickObject != 0)

{

glm::vec3 dist = glm::vec3(camera.Front.x \* distP, camera.Front.y \* distP, camera.Front.z \* distP);

pickObject->setPosition(camera.Position + dist);

}

## Отрисовка сцены

После обработки выбора объекта начинается этап основной отрисовки. Происходит он аналогично отрисовке цветом, но при этом используется обычный шейдер для объектов. Так же, если имеется выбранный объект, он отрисовывается в другом цвете, для наглядности выбора. Фрагмент кода предоставлен ниже (Листинг 13).

Листинг 13. Отрисовка сцены

for (auto var : objects)

{

if (colorPickingShow)

{

var->updatePickShader(camera.Position, pv);

var->pickDraw();

}

else

{

if (pickObject != 0 && pickObject->id == var->id)

{

var->updatePickShader(camera.Position, pv, 0.2f, 255, 255, 255);

var->pickDraw();

}

else

{

var->updateShader(camera.Position, pv, lights);

var->Draw();

}

}

}

glfwSwapBuffers(win);

# ЭКСПЕРЕМЕНТАЛЬНАЯ ЧАСТЬ

## Тестирование трехмерной сцены

При запуске приложения мы можем управлять камерой. Так же свет, который должен быть на объектах, отображается нормально (рис. 3).

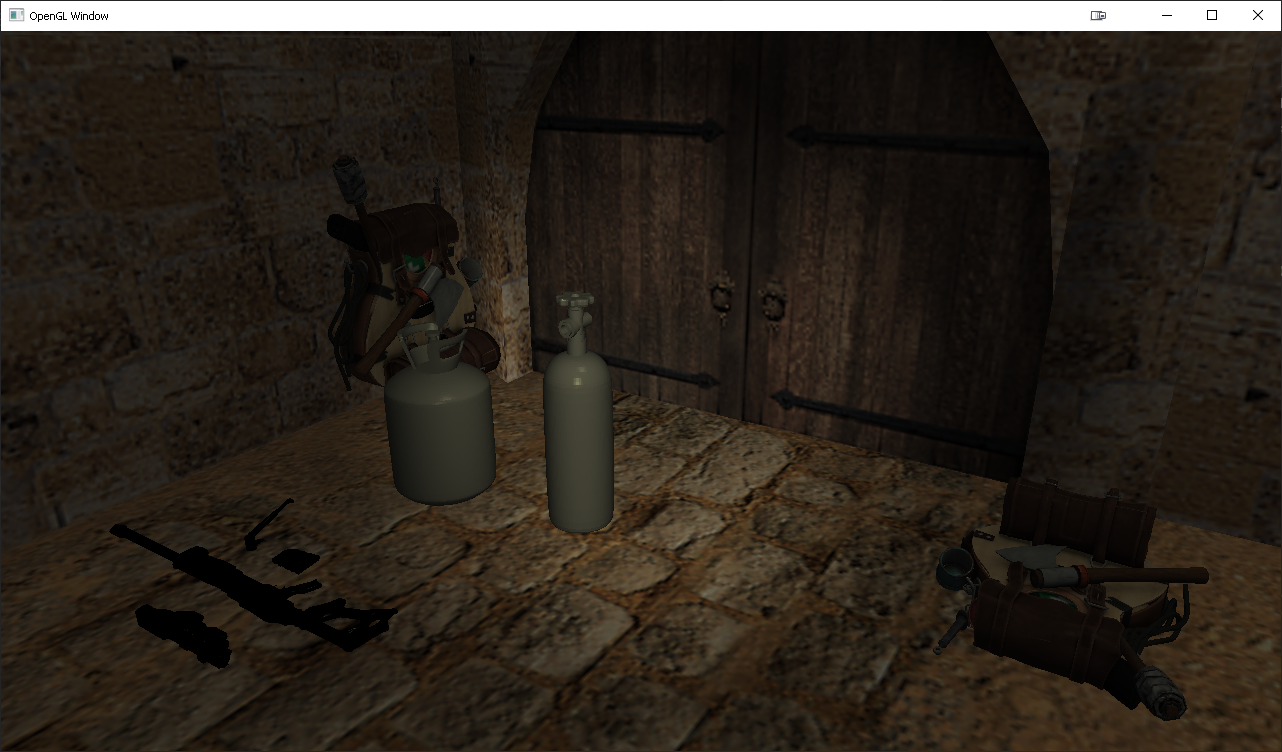


Рис. 3. Трехмерная сцена

## Тестирование шейдера цвета

Попробуем включить отображение объектов в шейдере цвета. У каждого объекта должен появиться свой, уникальный цвет, что и было замечено (рис. 4).

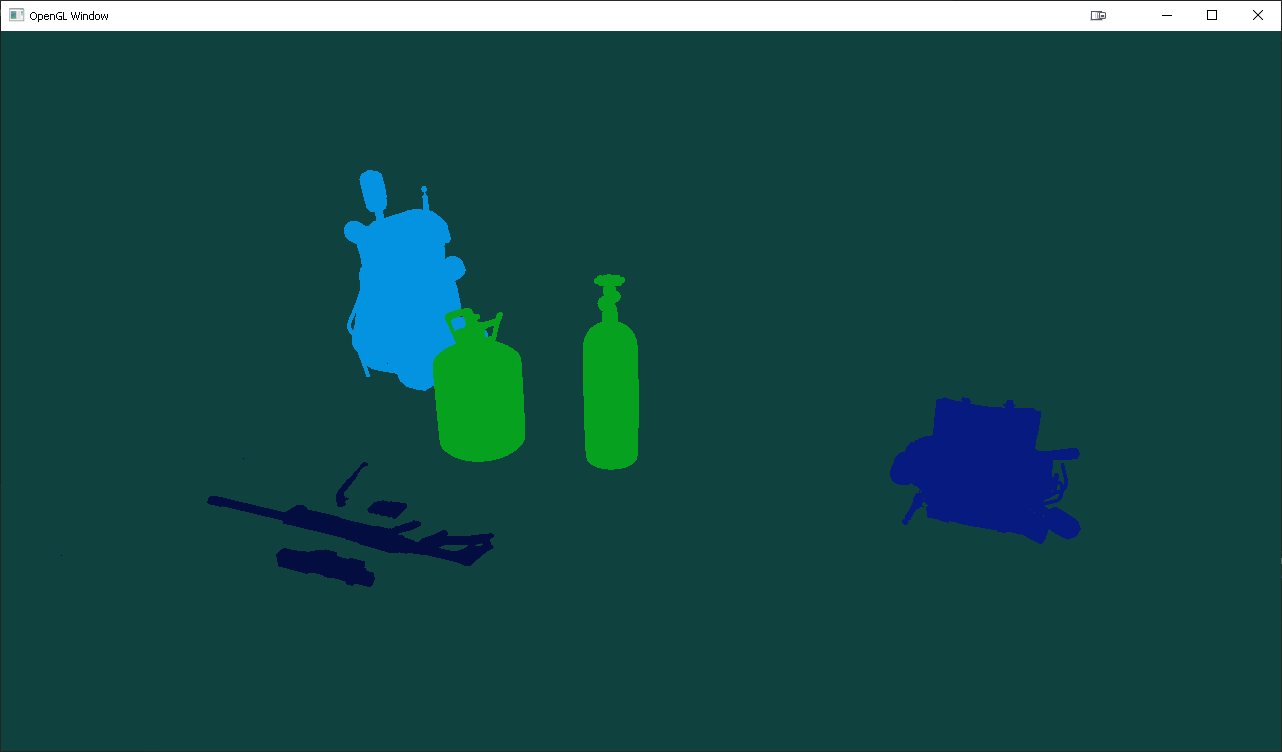


Рис. 4. Режим в шейдере цвета

## Тестирование выбора объекта

Попробуем выбрать какой-либо объект, например, баллоны. Для этого нужно переместить центр экрана на баллон и нажать кнопку для выбора (рис. 5).



Рис. 5. Выбор объекта

Попробуем переместить объект за стену и снова взять его. В данный момент объект находится за стеной (рис. 6).

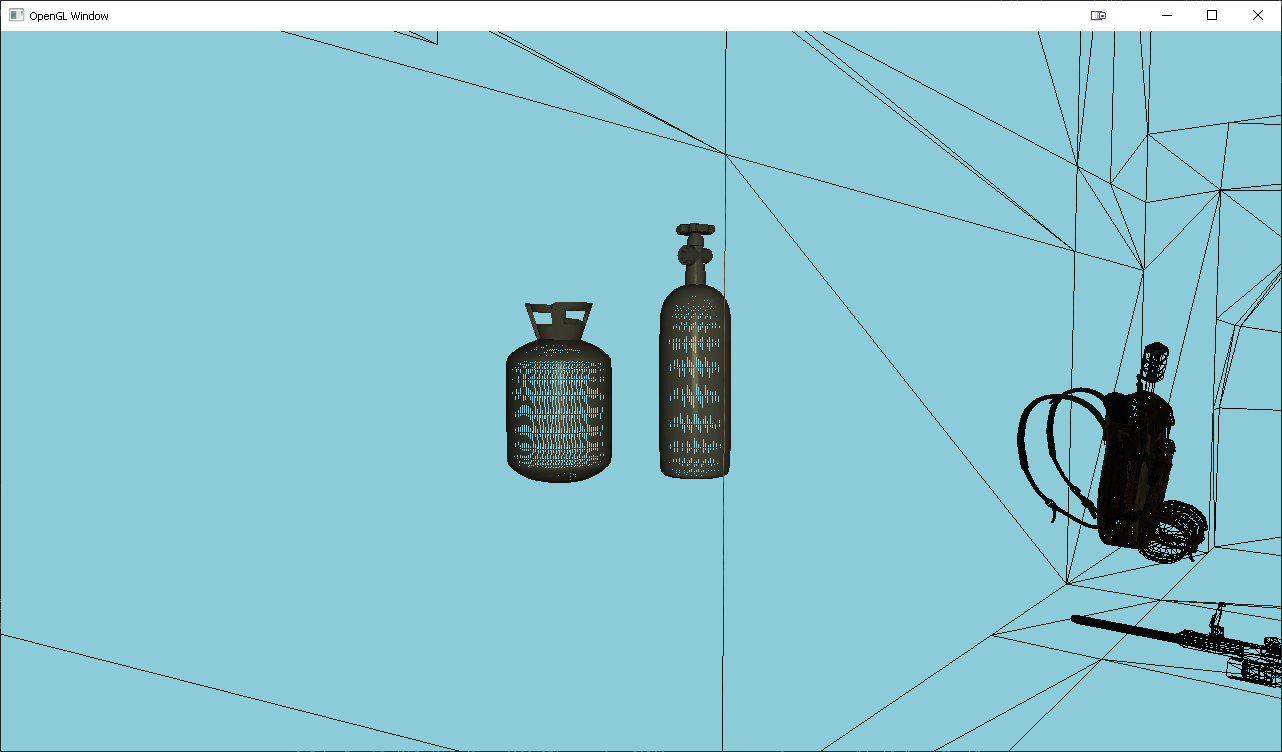


Рис. 6. Объект за стеной

При этом при включении отрисовке цветами можно заметить, что объект становится не видно за стеной (рис. 7).

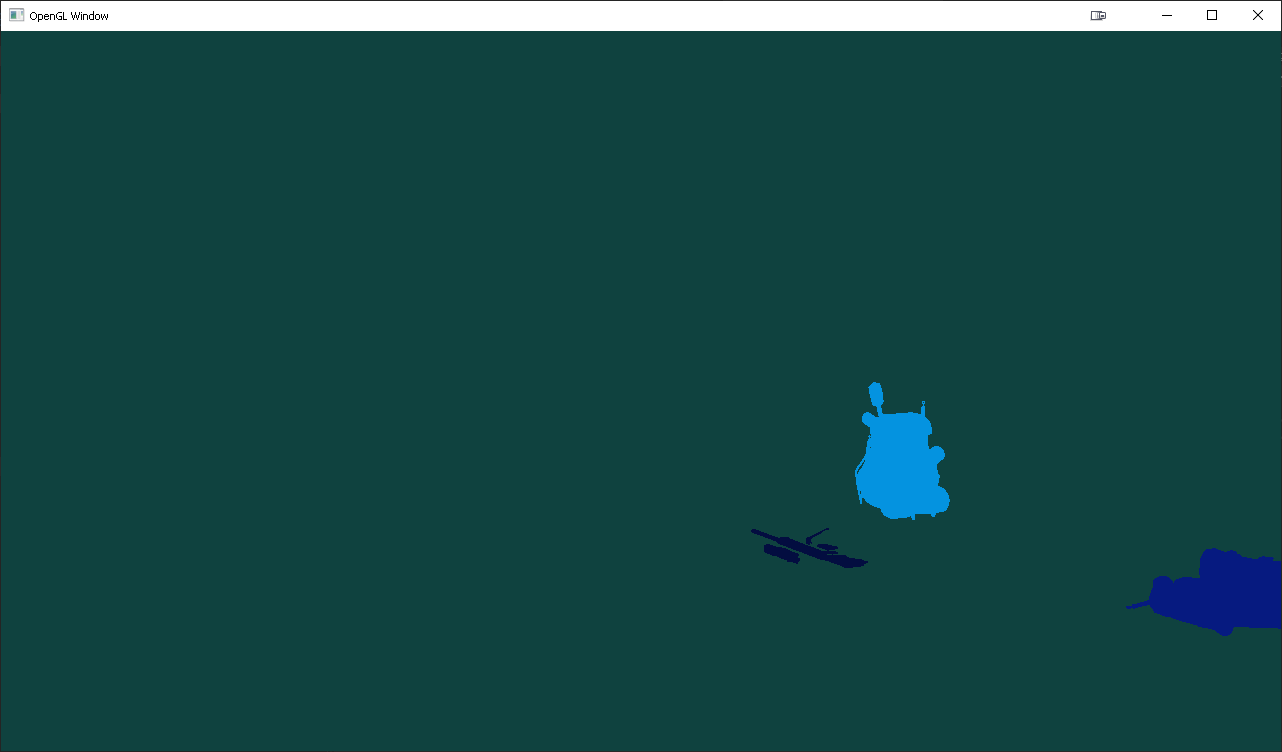


Рис. 7. Объект за стеной в режиме шейдера цвета

При попытке нажатия кнопки для выбора объекта ничего не происходит и объект не выбирается, всё работает корректно (рис. 8).

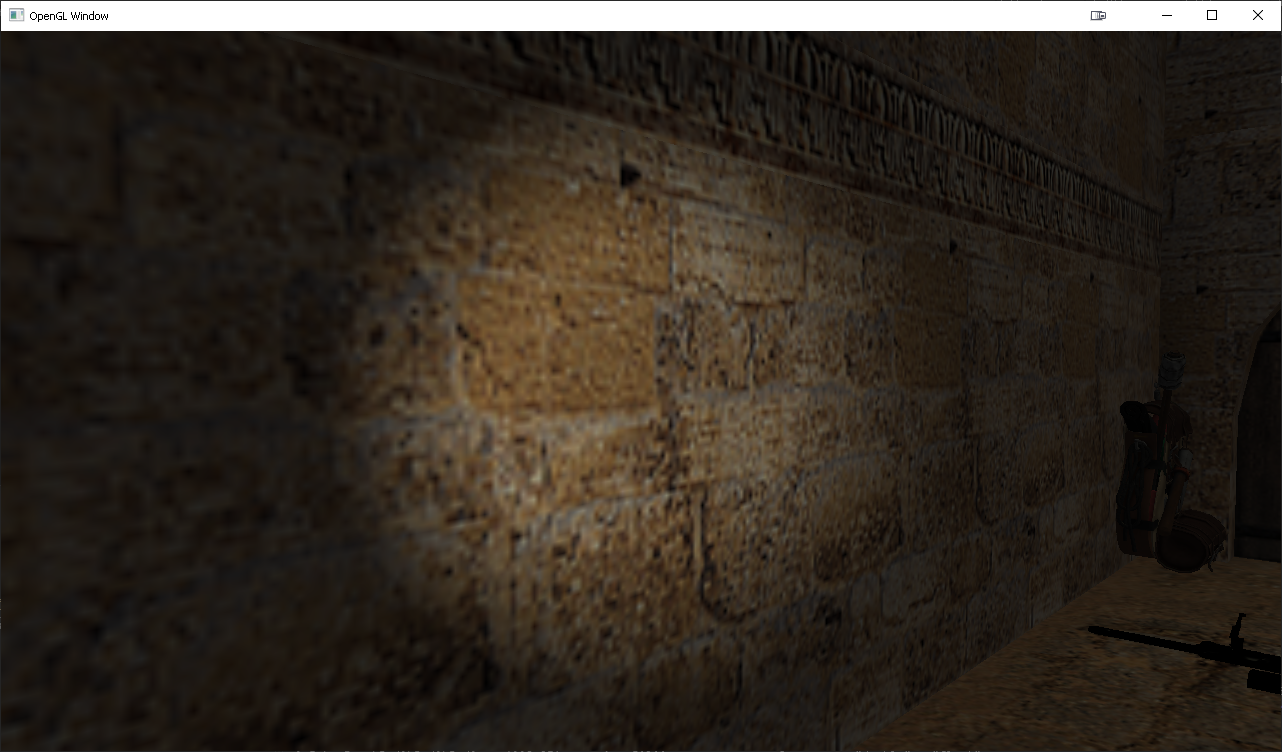


Рис. 8. Попытка выбора объекта за стеной

## Взаимодействие с выбранным объектом

Выбранный объект можно перемещать, вращать, изменять размеры. Протестируем данные возможности. Для этого выберем объект рюкзак и попробуем его переместить, увеличить и повернуть. После выполнения этих операций был получен следующий результат (рис. 9).

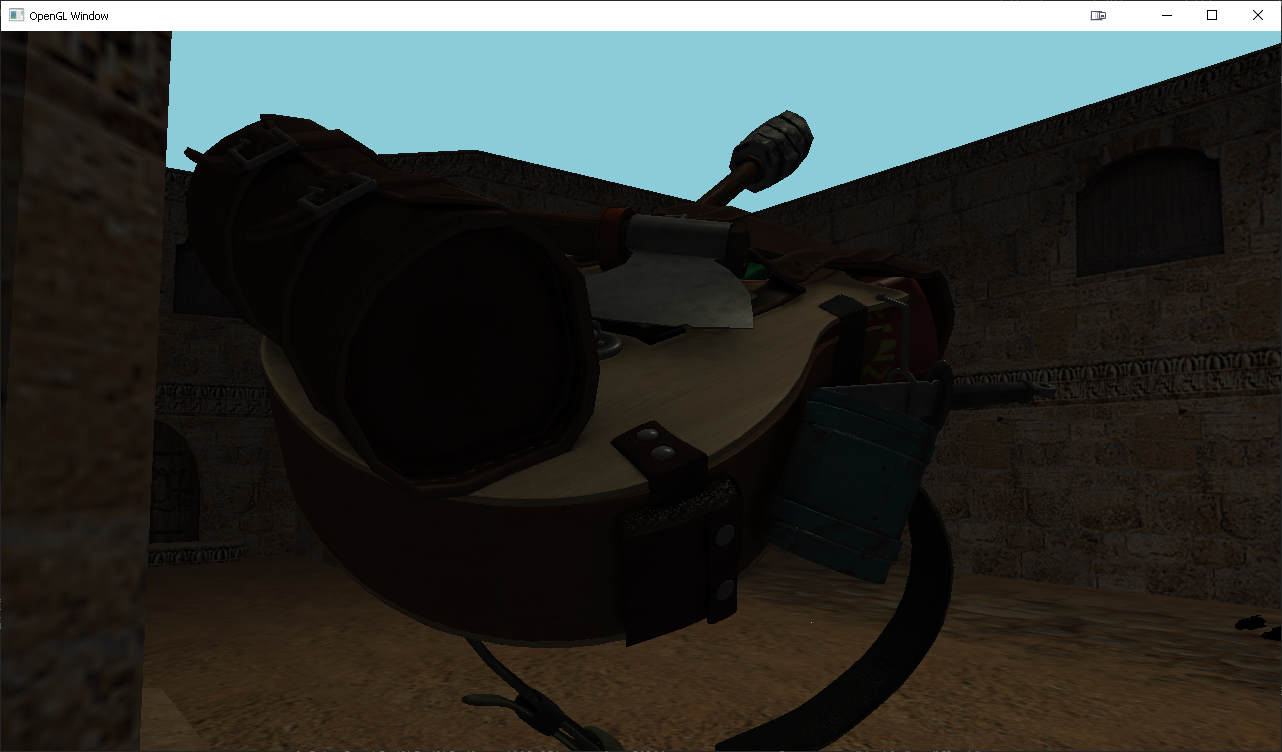


Рис. 9. Взаимодействие с объектом

# ЗАКЛЮЧЕНИЕ

В ходе создания трехмерной сцены были проанализированы различные варианты выбора объекта в трехмерном пространстве.

Были реализованы необходимые классы для моделей, их загрузка, перемещение, изменение и иные возможности для их редактирования.

Так же реализованы различные шейдеры, в том числе шейдер цвета, шейдер для обработки освещения и другие вершинные и фрагментные шейдеры.

В дальнейшем планируется реализовать возможность выбора объекта мышкой, более удобное меню для его редактирования, возможность сохранения сцены с объектами, а также реализация дополнительных спецэффектов.

# СПИСОК ЛИТЕРАТУРЫ

1. Боресков, А.В. Разработка и отладка шейдеров / А.В. Боресков. - М.: БХВ-Петербург, 2017. - 360 c.
2. Поляков, А. Методы и алгоритмы компьютерной графики в примерах на Visual C++ / А. Поляков. - М.: БХВ-Петербург, 2018. - 416 c.
3. Майкл Ласло. Вычислительная геометрия и компьютерная графика на С++. /М.: БИНОМ, 1997.
4. Дональд Херн, М. Паулин Бейкер. Компьютерная графика и стандарт OpenGL//Спб.: Вильямс, 2004
5. Дэвид Вольф. OpenGL. Язык шейдеров. Книга рецептов. Москв, ДМК Пресс, 2015. 368 с
6. Кэмпбелл, М. Компьютерная графика / М. Кэмпбелл. - М.: АСТ, 2014. - 714 c.
7. Эдвард Энджел. Интерактивная компьютерная графика. Вводный курс на базе OpenGL — 2-е изд. — М.: Вильямс, 2001. — 592 с.