**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

**отчет**

**по курсовой работе**

**по дисциплине «Компьютерная графика»**

**Тема: «Реализация сцены с визуализацией 3D-сцены»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 8383 |  | Киреев К.А. |
| Студент гр. 8383 |  | Муковский Д.В. |
| Преподаватель |  | Герасимова Т.В. |

Санкт-Петербург

2021

**Цель работы.**

Реализовать сцену с визуализацией 3D-сцены.

**Задачи.**

1. Подбор материала по теме для обзора (1-2 страницы), материал должен быть творчески переработан, дополнен примерами вашей реализации. Обязательны ссылки на литературу.

2. Создать описание генерации вашей модели (не создавать в средствах типа Blender, 3D MAX).

3. Разработка демонстрационной сцены.

Сцена управляема – можно облететь вокруг, изменить положение источников света.

Для выполнения задания необходимо создать сцену (фотореалистичность желательна). Оценка, выставленная за задание, зависит от исполнения сцены, и использованных в ней средств.

****

Рисунок 1 – Задание

**Ход работы.**

Стул собран из следующих *3D* примитивов:

* Плоский параллелепипед, который выступает в качестве сиденья, части спинки и пола;
* Ножки будут представлять собой длинные и тонкие цилиндры, а часть спинки, где присутствуют пробелы, короткие цилиндры;

За генерацию параллелепипеда для симуляции пола отвечает класс *Sitting*, который принимает параметры ширины, длины и высоты, также он принимает координату центра и смещение по оси *z*, на основе этих данных генерируется *8* точек параллелепипеда, а далее задаются грани, которые представляют из себя полигон из 4 точек.

На рис. 1 представлен пример отрисовки пола-параллелепипеда.

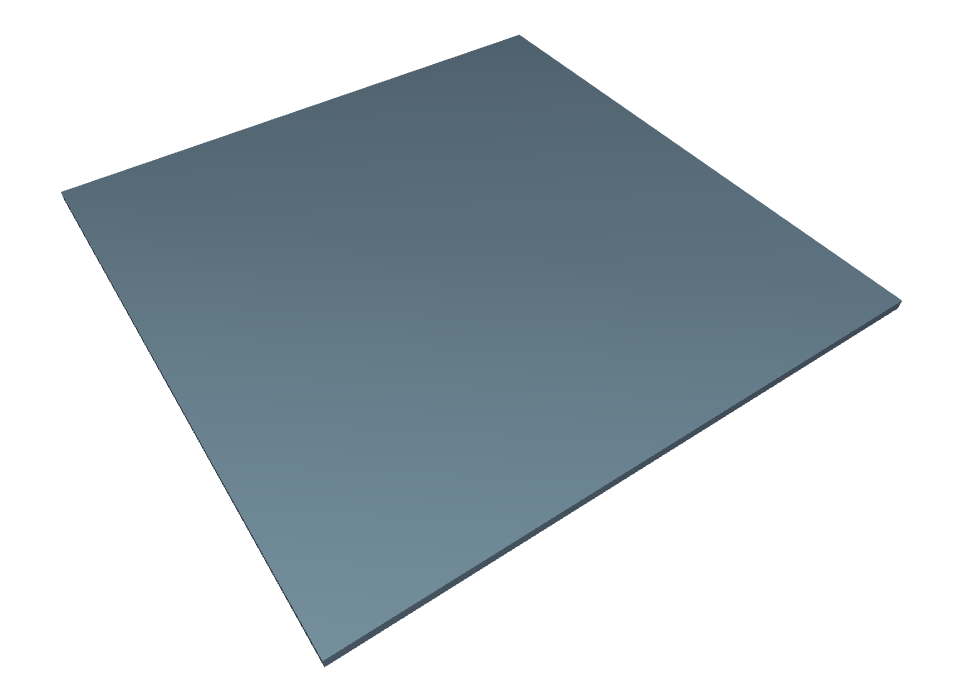
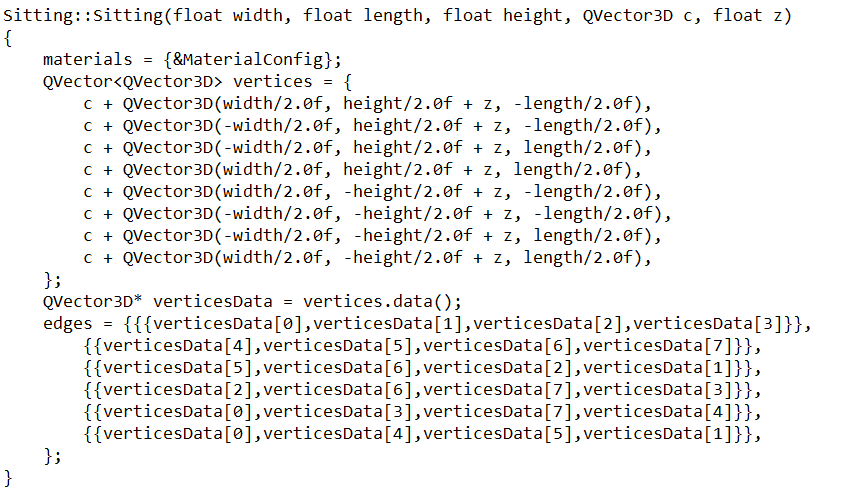


Рисунок 1 - Отрисованный параллелепипед, симулирующий пол

Листинг метода отрисовки параллелепипеда представлен ниже.



Далее, за генерацию параллелепипедов для симуляции спинки и сиденья отвечает класс *BackPlus*, который также генерирует точки параллелепипедов по формуле и сохраняет их в массиве verticesData и далее составляются шесть граней для параллелепипеда и сохраняются в массиве edges.

На рис. 2 представлен пример отрисовки сиденья и спинки.

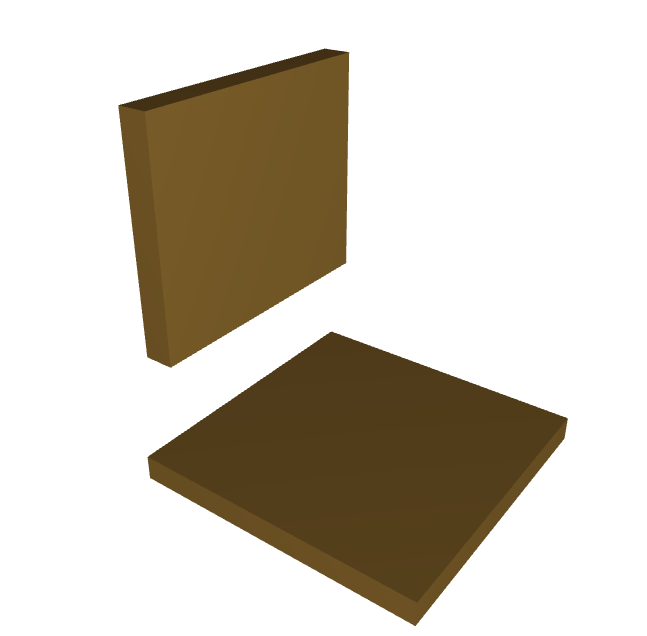
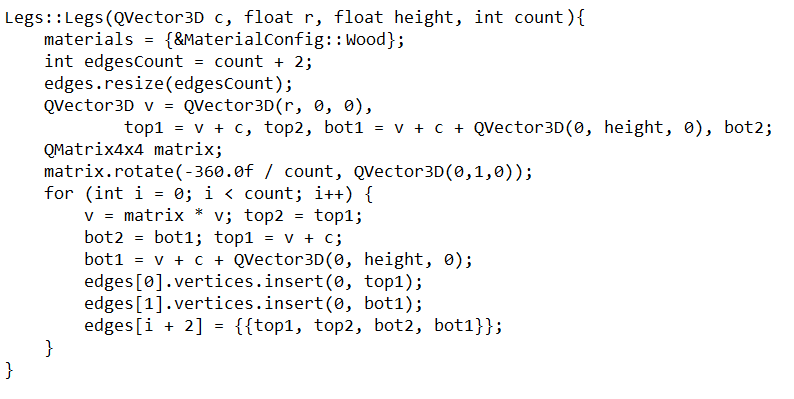


Рисунок 2 - Отрисованные параллелепипеды для спинки и сиденья

Теперь ножки. Как было сказано выше, ножки будут представлять из себя длинные цилиндры. За генерацию точек цилиндра с заданным разбиением для верхней, нижней и боковой частей и генерацию нормалей отвечает класс Legs, который принимает координату центра, радиус, высоту, количество вершин правильного многоугольника. Цилиндр это вытянутый по оси z правильный многоугольник, количество вершин которых можно задать, при достаточно большом количестве, он выглядит как цилиндр. На рис. 3 представлены отрисованные длинные цилиндры для симуляции ножек стула. Листинг метода генерации представлен ниже.



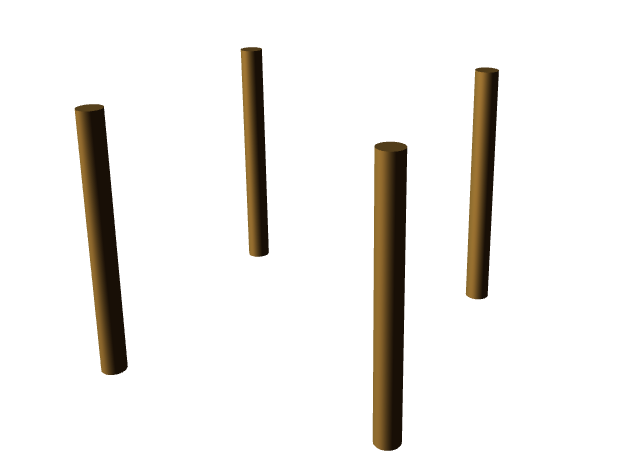


Рисунок 3 – Ножки стула

Тот же метод используется и для генерации точек для цилиндров, симулирующих палочки, находящиеся в спинке стула.

Получившиеся цилиндры показаны на рис. 4.

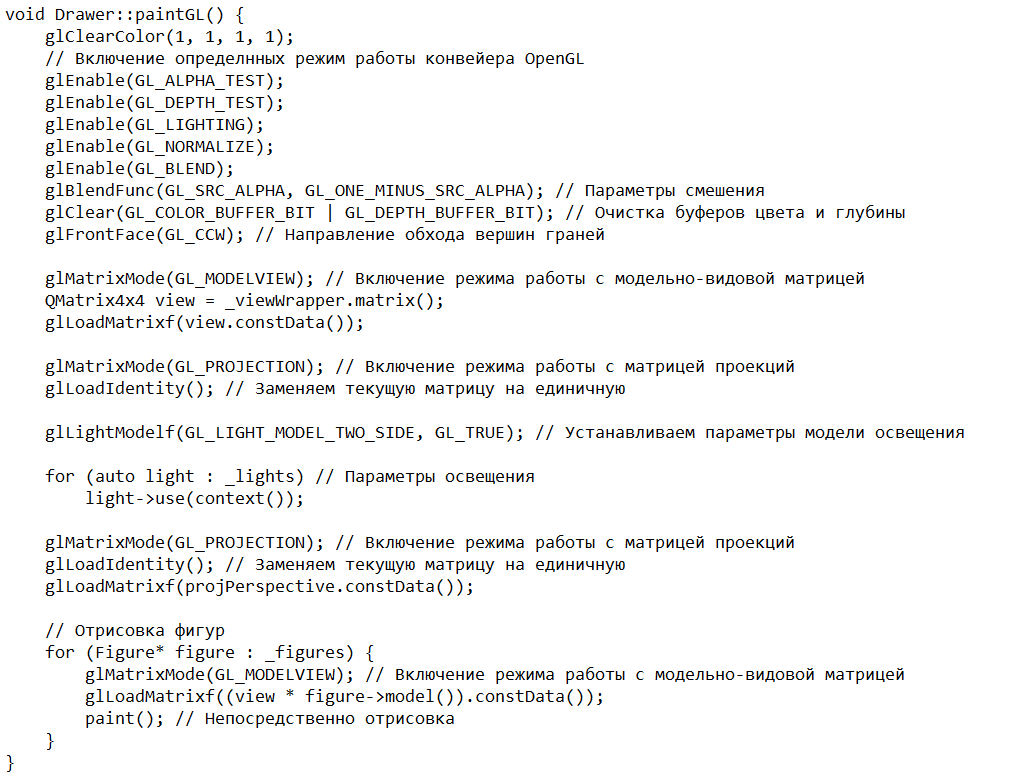


Рисунок 4 – Часть спинки стула

Каждый класс, генерирующий примитивы для деталей, является наследником абстрактного класса *Figure*, в котором реализована логика отрисовки *OpenGL* уровня.

Отрисовка реализована следующим образом: генерируются и связываются буферы *VBO*, в котором хранятся координаты вершин, также индексированный буфер *VBO* и *VAO* буфер, в котором хранится материал вершин. При отрисовке фигуры разбирается массив *edges* в массивы *verticesBuffer*, в котором хранится информация о вершине, её координаты и нормаль, а также создается массив *indicesBuffer*, который хранит в себе индексы вершин предыдущего буфера для оптимизации отрисовки. Все части собираются вместе и отрисовываются с помощью функций *PaintGL*, *paint* и *malloc*. В *paint* настраивается матрица проекции.

В листинге показана функция *PaintGL.*



Стул с разных сторон показан на рис. 5-7.

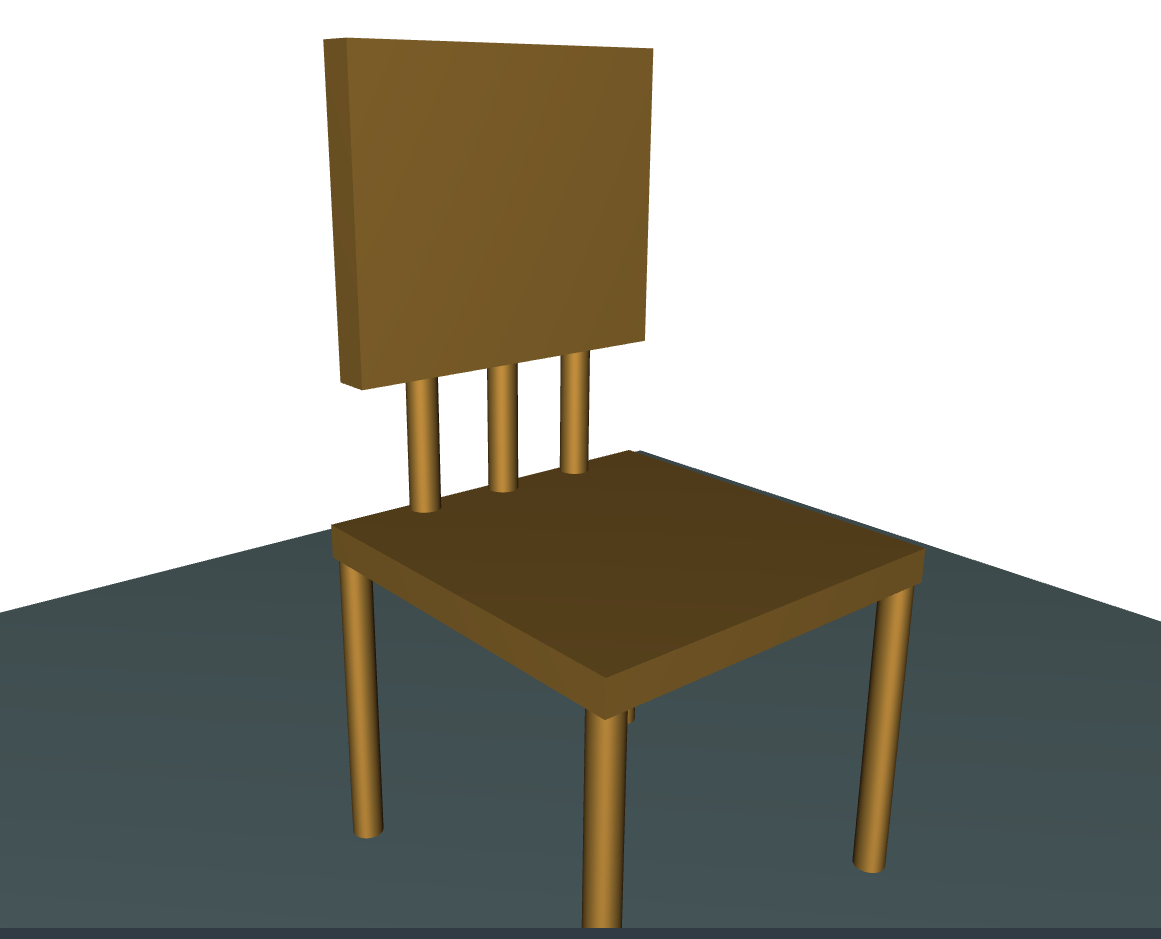


Рисунок 5 - Стул спереди

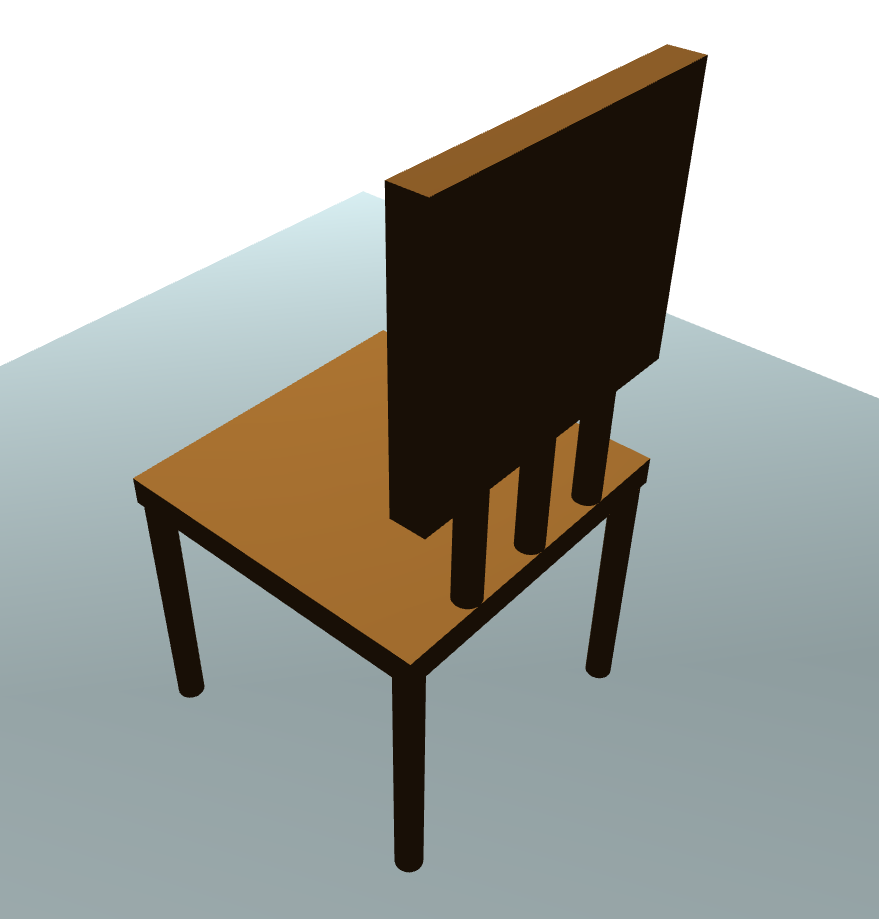


Рисунок 6 - Стул сзади



Рисунок 7 - Стул снизу

Материал описан в классе *Material*, который хранит состояние материала фигуры, а также в данном классе есть метод *use*, который принимает текущий контекст *OpenGL* и применяет текущий материал для точки. Освещение реализовано похожим образом.

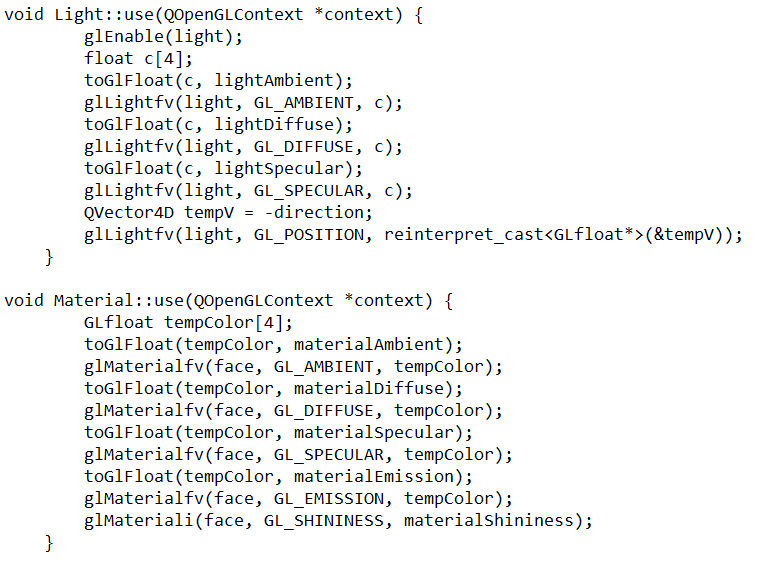
*Параметры света:*

Параметр *GL\_AMBIENT* определяет цвет фонового освещения, параметр *GL\_DIFFUSE* определяет цвет диффузного освещения, параметр *GL\_SPECULAR* определяет цвет зеркального отражения, параметр *GL\_POSITION* определяет положение источника света.

*Параметры материала:*

Параметр *GL\_AMBIENT* определяет цвет материала в тени, параметр *GL\_DIFFUSE* определяет цвет диффузного отражения материала, параметр *GL\_SPECULAR* определяет цвет отраженного света, параметр *GL\_SHININESS* определяет степень зеркального отражения материала.

Реализация материала и освещения в методе *use* представлена в листинге ниже.



Класс камеры вычисляет нормализованные вектора пространства камеры. При нажатии клавиш *W, A, S, D, Space, Shift* позиция камеры смещается вдоль вычисленных векторов.

При зажатии левой кнопки мыши и перемещении, вычисляется смещение в координатах. Таким образом реализовано свободное перемещение по сцене.



Рисунок 8 - Свободное перемещение по сцене.

**Выводы.**

В ходе выполнения курсовой работы были получены навыки построения модели, настройки материалов, наложения текстур, использования алгоритма освещения средствами последней спецификации *OpenGL*.

**Список использованных источников**

1. https://learnopengl.com

2. http://www.opengl-tutorial.org/ru/

3. https://habr.com/ru/post/336166/

4. https://ravesli.com/urok-6-tekstury-v-opengl/

**ПРИЛОЖЕНИЕ А**

**MAINWINDOW.CPP**

#include "drawer.h"

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <table.h>

bool MainWindow::count = false;

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

drawer = new Drawer();

ui->setupUi(this);

ui->verticalLayout->addWidget(drawer, 1);

ui->unbindLight->setDisabled(true);

float splitDensity = 3.0f;

// table.bigPanel->split(splitDensity);

// table.bigPanelSides->split(splitDensity);

// table.smallPanel->split(splitDensity);

// table.smallPanelSides->split(splitDensity);

for (Figure\* f : table.legs)

f->split(splitDensity);

for (Figure\* f : table.smallLegs)

f->split(splitDensity);

for (Figure\* f : table.ironLegs)

f->split(splitDensity);

for (Figure\* f : table.legs)

drawer->addFigure(f);

for (Figure\* f : table.smallLegs)

drawer->addFigure(f);

for (Figure\* f : table.ironLegs)

drawer->addFigure(f);

// drawer->addFigure(table.smallPanel);

// drawer->addFigure(table.smallPanelSides);

// drawer->addFigure(table.bigPanel);

drawer->addFigure(table.sitting);

// QObject::connect(ui->attachLightCheckbox, SIGNAL(clicked()), this, SLOT(lightClicked()));

}

MainWindow::~MainWindow()

{

delete ui;

}

void MainWindow::on\_bindLight\_clicked()

{

LightConfig::base.attachViewWrapper(&drawer->viewWrapper());

ui->bindLight->setDisabled(true);

ui->unbindLight->setDisabled(false);

}

void MainWindow::on\_unbindLight\_clicked()

{

LightConfig::base.detachViewWrapper();

ui->bindLight->setDisabled(false);

ui->unbindLight->setDisabled(true);

}

void MainWindow::on\_firstSettings\_clicked()

{

LightConfig::base.changeConfig(GL\_LIGHT0,

QVector4D(40,40,40, 1),

QColor(62, 62, 62),

QColor(184, 246, 255),

QColor(255, 255, 255));

ui->firstSettings->setDisabled(true);

ui->secondSettings->setDisabled(false);

ui->thirdSettings->setDisabled(false);

}

void MainWindow::on\_secondSettings\_clicked()

{

LightConfig::base.changeConfig(GL\_LIGHT1,

QVector3D(1, 1, 1),

QColor(12, 12, 62),

QColor(90, 123, 128),

QColor(75, 128, 118));

ui->firstSettings->setDisabled(false);

ui->secondSettings->setDisabled(true);

ui->thirdSettings->setDisabled(false);

}

void MainWindow::on\_thirdSettings\_clicked()

{

LightConfig::base.changeConfig(GL\_LIGHT1,

QVector3D(12, 12, 1),

QColor(62, 12, 62),

QColor(90, 23, 18),

QColor(5, 128, 118));

ui->firstSettings->setDisabled(false);

ui->secondSettings->setDisabled(false);

ui->thirdSettings->setDisabled(true);

}

**ПРИЛОЖЕНИЕ Б**

**КЛАСС КАМЕРЫ**

#include "camera.h"

Camera::Camera(QVector3D pos, QVector3D worldUp): pos(pos), worldUp(worldUp),yaw(YAW),pitch(PITCH),front({0.0f,0.0f,-1.0f}), movementSpeed(0.1f)

{

sens=0.1f;

updateCamVectors();

}

QMatrix4x4 Camera::getMatrix()

{

QMatrix4x4 a;

a.lookAt(this->pos,this->pos+this->front,this->up);

return a;

}

void Camera::changeYawAndPitch(float yaw, float pitch)

{

this->yaw += yaw\*sens;

this->pitch += pitch\*sens;

// Make sure that when pitch is out of bounds, screen doesn't get flipped

if (true)

{

if (this->pitch > 89.0f)

this->pitch = 89.0f;

if (this->pitch < -89.0f)

this->pitch = -89.0f;

}

// Update Front, Right and Up Vectors using the updated Eular angles

this->updateCamVectors();

}

void Camera::moveCam(QSet<int> \*keys)

{

if(keys->contains(Qt::Key\_W))

this->pos+=this->movementSpeed\*this->front;

if(keys->contains(Qt::Key\_S))

this->pos-=this->movementSpeed\*this->front;

if(keys->contains(Qt::Key\_A))

this->pos-=this->right\*this->movementSpeed;

if(keys->contains(Qt::Key\_D))

this->pos+=this->right\*this->movementSpeed;

if(keys->contains(Qt::Key\_Space))

this->pos+=this->up\*this->movementSpeed;

if(keys->contains(Qt::Key\_Control))

this->pos-=this->up\*this->movementSpeed;

}

void Camera::updateCamVectors()

{

QVector3D front;

front.setX( cosf(qDegreesToRadians(this->yaw)) \* cosf(qDegreesToRadians(this->pitch)) );

front.setY( sinf(qDegreesToRadians(this->pitch)) );

front.setZ( sinf(qDegreesToRadians(this->yaw)) \* cosf(qDegreesToRadians(this->pitch)) );

this->front=front;

this->front.normalize();

this->right = QVector3D::normal(this->front, this->worldUp); // Normalize the vectors, because their length gets closer to 0 the more you look up or down which results in slower movement.

this->up = QVector3D::normal(this->right, this->front);

}