**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по курсовой работе**

**по дисциплине «Компьютерная графика»**

Тема: «**Визуализация 3D объекта. 3D сцена**»

|  |  |  |
| --- | --- | --- |
| Студент гр. 6383 |  | Базаров И |
| Студент гр. 6383 |  | Василенко Т |
| Преподаватель |  | Герасимова Т.В. |

Санкт-Петербург

2019

**Содержание**

[Задание. 3](#_Toc9349041)

[Ход работы. 4](#_Toc9349042)

[1. Описание моделей геометрических фигур. 4](#_Toc9349043)

[1.1. Куб. 4](#_Toc9349044)

[1.2. Цилиндр. 5](#_Toc9349045)

[1.3. Окружность 6](#_Toc9349046)

[1.4. Изогнутый цилиндр 7](#_Toc9349047)

[2. Текстурирование фигур. 8](#_Toc9349048)

[3. Реализация возможностей управления сценой. 8](#_Toc9349049)

[4. Демонстрация работы программы. 9](#_Toc9349050)

[Вывод 12](#_Toc9349051)

[ПРИЛОЖЕНИЕ А 13](#_Toc9349052)

[ПРИЛОЖЕНИЕ Б 14](#_Toc9349053)

[ПРИЛОЖЕНИЕ В 17](#_Toc9349054)

[ПРИЛОЖЕНИЕ Г 19](#_Toc9349055)

[ПРИЛОЖЕНИЕ Д 22](#_Toc9349056)

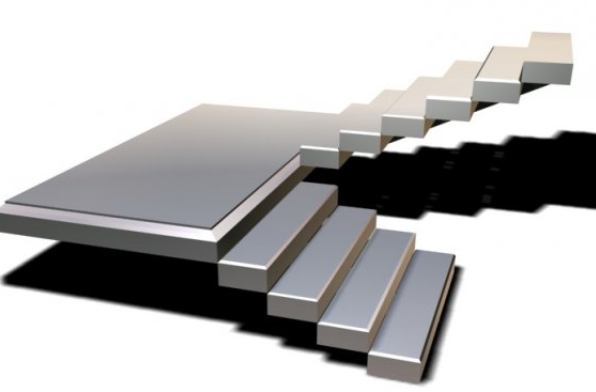
[ПРИЛОЖЕНИЕ Е 27](#_Toc9349057)

[ПРИЛОЖЕНИЕ Ж 29](#_Toc9349058)

# Задание.

Разработать программу, реализующую и визуализирующую предложенную на рис. 1 сцену, используя функции библиотеки OpenGL и языка шейдеров GLSL.

Разработанная программа должна быть пополнена возможностями управления сценой – сцену можно облететь вокруг, изменить положение источников света. Все объекты должны быть текстурированы.



**Рисунок 1 – Лестница**

# Ход работы.

Работа выполнена с помощью инструмента для разработки GUI приложений PyQt5, в основе которого лежит Python 3.7, библиотеки OpenGL для Python и языка GLSL для программирования шейдеров.

# Описание моделей геометрических фигур.

## Базисные фигуры.

Чтобы нарисовать лестницу, сначала нужно уметь рисовать базисные фигуры, в этой работе это обычный прямоугольный параллелепипед Cube, реализация в файле [figures.py](https://vk.com/away.php?to=http%3A%2F%2Ffigures.py&cc_key=). Аргументы - размеры, текстура и цвет. Он возвращает массив координат, вершин, текстурных координат, нормалей (нужно для света) и цвета вершины

class Cube:

def \_\_init\_\_(self, width, height, length, texture, color=(0.2, 0.2, 0.2, 1)):

self.x = width

self.y = height

self.z = length

self.count = 36

self.type = GL\_TRIANGLES

self.texture = texture

self.color = color

def generate(self):

return [

-self.x, -self.y, -self.z, 0.0, 0.0, -1.0, 0.0, 0.0, \*self.color,

self.x, -self.y, -self.z, 0.0, 0.0, -1.0, 1.0, 0.0, \*self.color,

self.x, self.y, -self.z, 0.0, 0.0, -1.0, 1.0, 1.0, \*self.color,

self.x, self.y, -self.z, 0.0, 0.0, -1.0, 1.0, 1.0, \*self.color,

-self.x, self.y, -self.z, 0.0, 0.0, -1.0, 0.0, 1.0, \*self.color,

-self.x, -self.y, -self.z, 0.0, 0.0, -1.0, 0.0, 0.0, \*self.color,

-self.x, -self.y, self.z, 0.0, 0.0, 1.0, 0.0, 0.0, \*self.color,

self.x, -self.y, self.z, 0.0, 0.0, 1.0, 1.0, 0.0, \*self.color,

self.x, self.y, self.z, 0.0, 0.0, 1.0, 1.0, 1.0, \*self.color,

self.x, self.y, self.z, 0.0, 0.0, 1.0, 1.0, 1.0, \*self.color,

-self.x, self.y, self.z, 0.0, 0.0, 1.0, 0.0, 1.0, \*self.color,

-self.x, -self.y, self.z, 0.0, 0.0, 1.0, 0.0, 0.0, \*self.color,

-self.x, self.y, self.z, -1.0, 0.0, 0.0, 1.0, 0.0, \*self.color,

-self.x, self.y, -self.z, -1.0, 0.0, 0.0, 1.0, 1.0, \*self.color,

-self.x, -self.y, -self.z, -1.0, 0.0, 0.0, 0.0, 1.0, \*self.color,

-self.x, -self.y, -self.z, -1.0, 0.0, 0.0, 0.0, 1.0, \*self.color,

-self.x, -self.y, self.z, -1.0, 0.0, 0.0, 0.0, 0.0, \*self.color,

-self.x, self.y, self.z, -1.0, 0.0, 0.0, 1.0, 0.0, \*self.color,

self.x, self.y, self.z, 1.0, 0.0, 0.0, 1.0, 0.0, \*self.color,

self.x, self.y, -self.z, 1.0, 0.0, 0.0, 1.0, 1.0, \*self.color,

self.x, -self.y, -self.z, 1.0, 0.0, 0.0, 0.0, 1.0, \*self.color,

self.x, -self.y, -self.z, 1.0, 0.0, 0.0, 0.0, 1.0, \*self.color,

self.x, -self.y, self.z, 1.0, 0.0, 0.0, 0.0, 0.0, \*self.color,

self.x, self.y, self.z, 1.0, 0.0, 0.0, 1.0, 0.0, \*self.color,

-self.x, -self.y, -self.z, 0.0, -1.0, 0.0, 0.0, 1.0, \*self.color,

self.x, -self.y, -self.z, 0.0, -1.0, 0.0, 1.0, 1.0, \*self.color,

self.x, -self.y, self.z, 0.0, -1.0, 0.0, 1.0, 0.0, \*self.color,

self.x, -self.y, self.z, 0.0, -1.0, 0.0, 1.0, 0.0, \*self.color,

-self.x, -self.y, self.z, 0.0, -1.0, 0.0, 0.0, 0.0, \*self.color,

-self.x, -self.y, -self.z, 0.0, -1.0, 0.0, 0.0, 1.0, \*self.color,

-self.x, self.y, -self.z, 0.0, 1.0, 0.0, 0.0, 1.0, \*self.color,

self.x, self.y, -self.z, 0.0, 1.0, 0.0, 1.0, 1.0, \*self.color,

self.x, self.y, self.z, 0.0, 1.0, 0.0, 1.0, 0.0, \*self.color,

self.x, self.y, self.z, 0.0, 1.0, 0.0, 1.0, 0.0, \*self.color,

-self.x, self.y, self.z, 0.0, 1.0, 0.0, 0.0, 0.0, \*self.color,

-self.x, self.y, -self.z, 0.0, 1.0, 0.0, 0.0, 1.0, \*self.color]

# Текстурирование фигур.

Для создания текстур в данной работе используются стандартные функции загрузки (QGLWidget bindTexture), принимающие на вход объект типа QPixmap. Для создания объектов QPixmap необходимо указать путь до файла. После вызова функции создания текстуры возвращается идентификатор текстуры. Контекст, при котором текстура создается (например, режим наложения текстуры), можно поменять. Пример создания текстуры представлен ниже.

glEnable(GL\_TEXTURE\_2D)  
self.woodTexture = self.bindTexture(QPixmap(**"textures/дерево.jpg"**))  
self.brickTexture = self.bindTexture(QPixmap(**"textures/кирпич.jpg"**))  
self.stoneTexture = self.bindTexture(QPixmap(**"textures/камень.jpg"**))  
self.roadTexture = self.bindTexture(QPixmap(**"textures/асфальт.jpg"**))

# Реализация возможностей управления сценой.

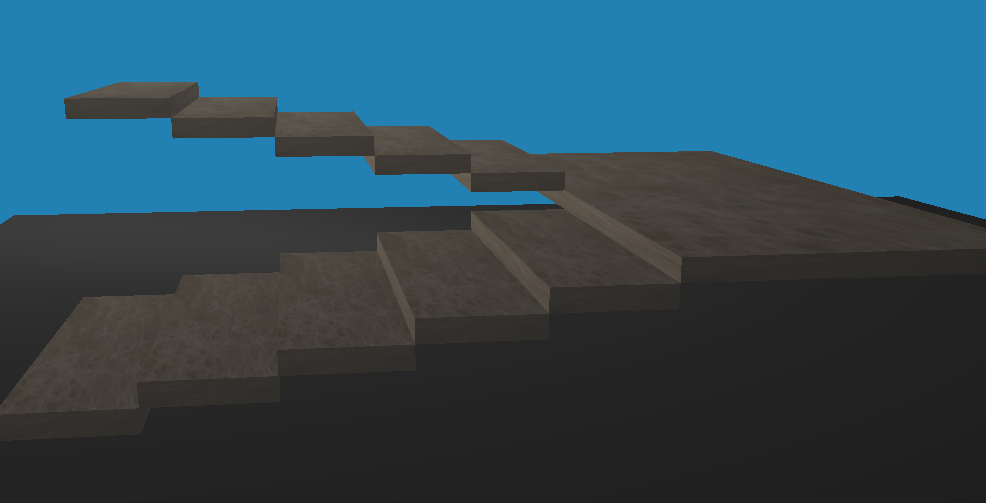
Возможность управления сценой реализована созданием камеры, с которой пользователь может взаимодействовать посредством перемещения мыши и нажатий на клавиши клавиатуры. Также пользователь может изменять положение источника света и его режим – либо статический источник света, либо свет, исходящий из камеры, который представляет собой фонарик.

Камера представляет собой три вектора cameraPos, cameraFront и cameraUp, которые изменяются при взаимодействии с пользователями и при отрисовке каждый раз изменяют матрицу вида, посредством вызова функции lookAt. Затем эта матрица передается в вершинный шейдер, где используется для передачи позиции фрагмента во фрагментный шейдер. Также вместе с матрицами проекции и модели используется для отображения положений вершин, возвращаемых вершинным шейдером.

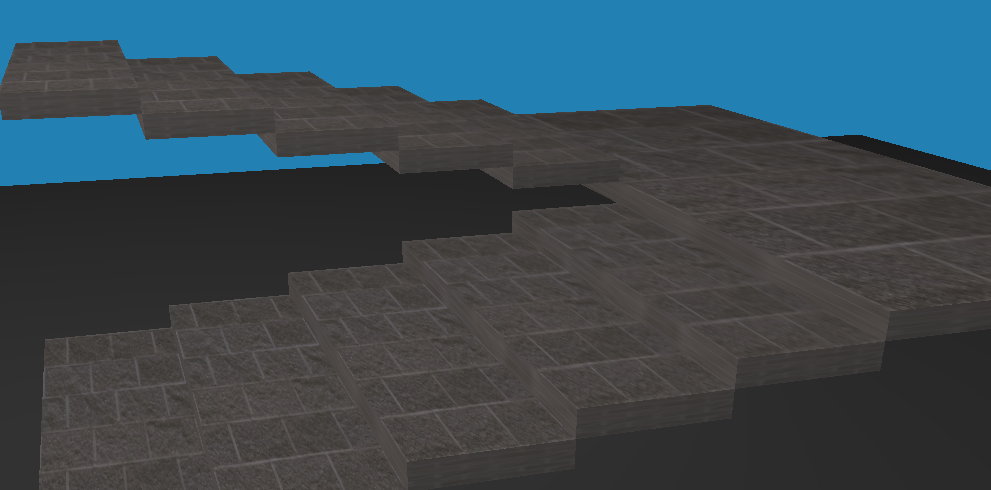
Позиция источника света задается в основной программе, где затем передается через шейдерную программу во фрагментный шейдер с помощью uniform переменной. Позиция источника света также регулируется пользователем с помощью клавиш клавиатуры. Программные коды шейдеров представлены в приложениях.

# Демонстрация работы программы.

Тестирование программы продемонстрировано ниже



**Рисунок 1: Лестница дерево**

****

**Рисунок 2: Лестница кирпич**

# Вывод

В ходе выполнения данной работы разработана программа, реализующую и визуализирующую предложенную на рис. 1 сцену. В программе используются функции библиотеки OpenGL и язык шейдеров GLSL.

Разработанная программа пополнена возможностями управления сценой – сцену можно облететь вокруг, изменить положение источников света. Все объекты сцены текстурированы.

# ПРИЛОЖЕНИЕ А

**Код файла main.py**

import sys  
  
from PyQt5 import QtWidgets  
from mainwindow import MainWindow  
  
  
def main():  
 app = QtWidgets.QApplication(sys.argv)  
 window = MainWindow()  
 window.show()  
 app.exec\_()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

# ПРИЛОЖЕНИЕ Б

*Form.py***from** PyQt5 **import** QtCore, QtGui, QtWidgets  
  
  
**class** Ui\_MainWindow(object):  
 **def** setupUi(self, MainWindow):  
 MainWindow.setObjectName(**"MainWindow"**)  
 MainWindow.resize(630, 781)  
 self.centralwidget = QtWidgets.QWidget(MainWindow)  
 self.centralwidget.setObjectName(**"centralwidget"**)  
 self.verticalLayout\_13 = QtWidgets.QVBoxLayout(self.centralwidget)  
 self.verticalLayout\_13.setObjectName(**"verticalLayout\_13"**)  
 self.horizontalLayout = QtWidgets.QHBoxLayout()  
 self.horizontalLayout.setObjectName(**"horizontalLayout"**)  
 self.verticalLayout = QtWidgets.QVBoxLayout()  
 self.verticalLayout.setObjectName(**"verticalLayout"**)  
 self.verticalLayout\_3 = QtWidgets.QVBoxLayout()  
 self.verticalLayout\_3.setObjectName(**"verticalLayout\_3"**)  
 self.groupBox = QtWidgets.QGroupBox(self.centralwidget)  
 sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Preferred)  
 sizePolicy.setHorizontalStretch(0)  
 sizePolicy.setVerticalStretch(0)  
 sizePolicy.setHeightForWidth(self.groupBox.sizePolicy().hasHeightForWidth())  
 self.groupBox.setSizePolicy(sizePolicy)  
 self.groupBox.setMinimumSize(QtCore.QSize(300, 0))  
 self.groupBox.setMaximumSize(QtCore.QSize(300, 16777215))  
 self.groupBox.setObjectName(**"groupBox"**)  
 self.verticalLayout\_7 = QtWidgets.QVBoxLayout(self.groupBox)  
 self.verticalLayout\_7.setObjectName(**"verticalLayout\_7"**)  
 self.gridLayout = QtWidgets.QGridLayout()  
 self.gridLayout.setObjectName(**"gridLayout"**)  
 self.sliderViewPosZ = QtWidgets.QSlider(self.groupBox)  
 self.sliderViewPosZ.setProperty(**"value"**, 90)  
 self.sliderViewPosZ.setOrientation(QtCore.Qt.Horizontal)  
 self.sliderViewPosZ.setObjectName(**"sliderViewPosZ"**)  
 self.gridLayout.addWidget(self.sliderViewPosZ, 2, 1, 1, 1)  
 self.sliderViewPosX = QtWidgets.QSlider(self.groupBox)  
 self.sliderViewPosX.setProperty(**"value"**, 63)  
 self.sliderViewPosX.setOrientation(QtCore.Qt.Horizontal)  
 self.sliderViewPosX.setObjectName(**"sliderViewPosX"**)  
 self.gridLayout.addWidget(self.sliderViewPosX, 0, 1, 1, 1)  
 self.sliderViewPosY = QtWidgets.QSlider(self.groupBox)  
 self.sliderViewPosY.setProperty(**"value"**, 60)  
 self.sliderViewPosY.setOrientation(QtCore.Qt.Horizontal)  
 self.sliderViewPosY.setObjectName(**"sliderViewPosY"**)  
 self.gridLayout.addWidget(self.sliderViewPosY, 1, 1, 1, 1)  
 self.label = QtWidgets.QLabel(self.groupBox)  
 self.label.setObjectName(**"label"**)  
 self.gridLayout.addWidget(self.label, 0, 0, 1, 1)  
 self.label\_2 = QtWidgets.QLabel(self.groupBox)  
 self.label\_2.setObjectName(**"label\_2"**)  
 self.gridLayout.addWidget(self.label\_2, 1, 0, 1, 1)  
 self.label\_3 = QtWidgets.QLabel(self.groupBox)  
 self.label\_3.setObjectName(**"label\_3"**)  
 self.gridLayout.addWidget(self.label\_3, 2, 0, 1, 1)  
 self.verticalLayout\_7.addLayout(self.gridLayout)  
 self.verticalLayout\_3.addWidget(self.groupBox)  
 self.verticalLayout.addLayout(self.verticalLayout\_3)  
 self.verticalLayout\_6 = QtWidgets.QVBoxLayout()  
 self.verticalLayout\_6.setObjectName(**"verticalLayout\_6"**)  
 self.groupBox\_2 = QtWidgets.QGroupBox(self.centralwidget)  
 sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed, QtWidgets.QSizePolicy.Minimum)  
 sizePolicy.setHorizontalStretch(0)  
 sizePolicy.setVerticalStretch(0)  
 sizePolicy.setHeightForWidth(self.groupBox\_2.sizePolicy().hasHeightForWidth())  
 self.groupBox\_2.setSizePolicy(sizePolicy)  
 self.groupBox\_2.setMinimumSize(QtCore.QSize(300, 0))  
 self.groupBox\_2.setMaximumSize(QtCore.QSize(300, 16777215))  
 self.groupBox\_2.setObjectName(**"groupBox\_2"**)  
 self.horizontalLayout\_3 = QtWidgets.QHBoxLayout(self.groupBox\_2)  
 self.horizontalLayout\_3.setObjectName(**"horizontalLayout\_3"**)  
 self.gridLayout\_2 = QtWidgets.QGridLayout()  
 self.gridLayout\_2.setObjectName(**"gridLayout\_2"**)  
 self.sliderLightPosY = QtWidgets.QSlider(self.groupBox\_2)  
 self.sliderLightPosY.setOrientation(QtCore.Qt.Horizontal)  
 self.sliderLightPosY.setObjectName(**"sliderLightPosY"**)  
 self.gridLayout\_2.addWidget(self.sliderLightPosY, 1, 1, 1, 1)  
 self.label\_4 = QtWidgets.QLabel(self.groupBox\_2)  
 self.label\_4.setObjectName(**"label\_4"**)  
 self.gridLayout\_2.addWidget(self.label\_4, 0, 0, 1, 1)  
 self.label\_6 = QtWidgets.QLabel(self.groupBox\_2)  
 self.label\_6.setObjectName(**"label\_6"**)  
 self.gridLayout\_2.addWidget(self.label\_6, 2, 0, 1, 1)  
 self.sliderLightPosZ = QtWidgets.QSlider(self.groupBox\_2)  
 self.sliderLightPosZ.setOrientation(QtCore.Qt.Horizontal)  
 self.sliderLightPosZ.setObjectName(**"sliderLightPosZ"**)  
 self.gridLayout\_2.addWidget(self.sliderLightPosZ, 2, 1, 1, 1)  
 self.label\_5 = QtWidgets.QLabel(self.groupBox\_2)  
 self.label\_5.setObjectName(**"label\_5"**)  
 self.gridLayout\_2.addWidget(self.label\_5, 1, 0, 1, 1)  
 self.sliderLightPosX = QtWidgets.QSlider(self.groupBox\_2)  
 self.sliderLightPosX.setOrientation(QtCore.Qt.Horizontal)  
 self.sliderLightPosX.setObjectName(**"sliderLightPosX"**)  
 self.gridLayout\_2.addWidget(self.sliderLightPosX, 0, 1, 1, 1)  
 self.horizontalLayout\_3.addLayout(self.gridLayout\_2)  
 self.verticalLayout\_6.addWidget(self.groupBox\_2)  
 self.verticalLayout.addLayout(self.verticalLayout\_6)  
 self.verticalLayout\_5 = QtWidgets.QVBoxLayout()  
 self.verticalLayout\_5.setObjectName(**"verticalLayout\_5"**)  
 self.groupBox\_3 = QtWidgets.QGroupBox(self.centralwidget)  
 self.groupBox\_3.setObjectName(**"groupBox\_3"**)  
 self.horizontalLayout\_4 = QtWidgets.QHBoxLayout(self.groupBox\_3)  
 self.horizontalLayout\_4.setObjectName(**"horizontalLayout\_4"**)  
 self.verticalLayout\_12 = QtWidgets.QVBoxLayout()  
 self.verticalLayout\_12.setObjectName(**"verticalLayout\_12"**)  
 self.sliderLightIntensive = QtWidgets.QSlider(self.groupBox\_3)  
 sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed, QtWidgets.QSizePolicy.Fixed)  
 sizePolicy.setHorizontalStretch(0)  
 sizePolicy.setVerticalStretch(0)  
 sizePolicy.setHeightForWidth(self.sliderLightIntensive.sizePolicy().hasHeightForWidth())  
 self.sliderLightIntensive.setSizePolicy(sizePolicy)  
 self.sliderLightIntensive.setMinimumSize(QtCore.QSize(200, 0))  
 self.sliderLightIntensive.setProperty(**"value"**, 99)  
 self.sliderLightIntensive.setOrientation(QtCore.Qt.Horizontal)  
 self.sliderLightIntensive.setObjectName(**"sliderLightIntensive"**)  
 self.verticalLayout\_12.addWidget(self.sliderLightIntensive)  
 self.horizontalLayout\_4.addLayout(self.verticalLayout\_12)  
 self.verticalLayout\_5.addWidget(self.groupBox\_3)  
 self.verticalLayout.addLayout(self.verticalLayout\_5)  
 self.verticalLayout\_9 = QtWidgets.QVBoxLayout()  
 self.verticalLayout\_9.setObjectName(**"verticalLayout\_9"**)  
 self.groupBox\_4 = QtWidgets.QGroupBox(self.centralwidget)  
 self.groupBox\_4.setObjectName(**"groupBox\_4"**)  
 self.verticalLayout\_11 = QtWidgets.QVBoxLayout(self.groupBox\_4)  
 self.verticalLayout\_11.setObjectName(**"verticalLayout\_11"**)  
 self.verticalLayout\_10 = QtWidgets.QVBoxLayout()  
 self.verticalLayout\_10.setObjectName(**"verticalLayout\_10"**)  
 self.comboBoxStyle = QtWidgets.QComboBox(self.groupBox\_4)  
 sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed, QtWidgets.QSizePolicy.Fixed)  
 sizePolicy.setHorizontalStretch(0)  
 sizePolicy.setVerticalStretch(0)  
 sizePolicy.setHeightForWidth(self.comboBoxStyle.sizePolicy().hasHeightForWidth())  
 self.comboBoxStyle.setSizePolicy(sizePolicy)  
 self.comboBoxStyle.setMinimumSize(QtCore.QSize(300, 0))  
 self.comboBoxStyle.setObjectName(**"comboBoxStyle"**)  
 self.comboBoxStyle.addItem(**""**)  
 self.comboBoxStyle.addItem(**""**)  
 self.comboBoxStyle.addItem(**""**)  
 self.verticalLayout\_10.addWidget(self.comboBoxStyle)  
 self.verticalLayout\_11.addLayout(self.verticalLayout\_10)  
 self.verticalLayout\_9.addWidget(self.groupBox\_4)  
 self.verticalLayout\_14 = QtWidgets.QVBoxLayout()  
 self.verticalLayout\_14.setObjectName(**"verticalLayout\_14"**)  
 self.groupBox\_5 = QtWidgets.QGroupBox(self.centralwidget)  
 self.groupBox\_5.setObjectName(**"groupBox\_5"**)  
 self.horizontalLayout\_2 = QtWidgets.QHBoxLayout(self.groupBox\_5)  
 self.horizontalLayout\_2.setObjectName(**"horizontalLayout\_2"**)  
 self.verticalLayout\_15 = QtWidgets.QVBoxLayout()  
 self.verticalLayout\_15.setObjectName(**"verticalLayout\_15"**)  
 self.comboBoxTexture = QtWidgets.QComboBox(self.groupBox\_5)  
 sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed, QtWidgets.QSizePolicy.Fixed)  
 sizePolicy.setHorizontalStretch(0)  
 sizePolicy.setVerticalStretch(0)  
 sizePolicy.setHeightForWidth(self.comboBoxTexture.sizePolicy().hasHeightForWidth())  
 self.comboBoxTexture.setSizePolicy(sizePolicy)  
 self.comboBoxTexture.setMinimumSize(QtCore.QSize(300, 0))  
 self.comboBoxTexture.setObjectName(**"comboBoxTexture"**)  
 self.comboBoxTexture.addItem(**""**)  
 self.comboBoxTexture.addItem(**""**)  
 self.comboBoxTexture.addItem(**""**)  
 self.comboBoxTexture.addItem(**""**)  
 self.comboBoxTexture.addItem(**""**)  
 self.verticalLayout\_15.addWidget(self.comboBoxTexture)  
 self.horizontalLayout\_2.addLayout(self.verticalLayout\_15)  
 self.verticalLayout\_14.addWidget(self.groupBox\_5)  
 self.groupBox\_7 = QtWidgets.QGroupBox(self.centralwidget)  
 self.groupBox\_7.setObjectName(**"groupBox\_7"**)  
 self.horizontalLayout\_5 = QtWidgets.QHBoxLayout(self.groupBox\_7)  
 self.horizontalLayout\_5.setObjectName(**"horizontalLayout\_5"**)  
 self.verticalLayout\_16 = QtWidgets.QVBoxLayout()  
 self.verticalLayout\_16.setObjectName(**"verticalLayout\_16"**)  
 self.pushButton = QtWidgets.QPushButton(self.groupBox\_7)  
 self.pushButton.setMinimumSize(QtCore.QSize(250, 0))  
 self.pushButton.setObjectName(**"pushButton"**)  
 self.verticalLayout\_16.addWidget(self.pushButton)  
 self.pushButton\_2 = QtWidgets.QPushButton(self.groupBox\_7)  
 self.pushButton\_2.setMinimumSize(QtCore.QSize(250, 0))  
 self.pushButton\_2.setObjectName(**"pushButton\_2"**)  
 self.verticalLayout\_16.addWidget(self.pushButton\_2)  
 self.horizontalLayout\_5.addLayout(self.verticalLayout\_16)  
 self.verticalLayout\_14.addWidget(self.groupBox\_7)  
 self.verticalLayout\_9.addLayout(self.verticalLayout\_14)  
 self.verticalLayout.addLayout(self.verticalLayout\_9)  
 self.groupBox\_8 = QtWidgets.QGroupBox(self.centralwidget)  
 self.groupBox\_8.setObjectName(**"groupBox\_8"**)  
 self.horizontalLayout\_6 = QtWidgets.QHBoxLayout(self.groupBox\_8)  
 self.horizontalLayout\_6.setObjectName(**"horizontalLayout\_6"**)  
 self.verticalLayout\_17 = QtWidgets.QVBoxLayout()  
 self.verticalLayout\_17.setObjectName(**"verticalLayout\_17"**)  
 self.spinBox = QtWidgets.QSpinBox(self.groupBox\_8)  
 self.spinBox.setMinimum(1)  
 self.spinBox.setMaximum(50)  
 self.spinBox.setProperty(**"value"**, 5)  
 self.spinBox.setObjectName(**"spinBox"**)  
 self.verticalLayout\_17.addWidget(self.spinBox)  
 self.horizontalLayout\_6.addLayout(self.verticalLayout\_17)  
 self.verticalLayout.addWidget(self.groupBox\_8)  
 spacerItem = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Expanding)  
 self.verticalLayout.addItem(spacerItem)  
 self.check1 = QtWidgets.QCheckBox(self.centralwidget)  
 self.check1.setObjectName(**"check1"**)  
 self.verticalLayout.addWidget(self.check1)  
 self.horizontalLayout.addLayout(self.verticalLayout)  
 self.widget\_layout = QtWidgets.QVBoxLayout()  
 self.widget\_layout.setObjectName(**"widget\_layout"**)  
 self.horizontalLayout.addLayout(self.widget\_layout)  
 self.verticalLayout\_13.addLayout(self.horizontalLayout)  
 MainWindow.setCentralWidget(self.centralwidget)  
  
 self.retranslateUi(MainWindow)  
 QtCore.QMetaObject.connectSlotsByName(MainWindow)  
  
 **def** retranslateUi(self, MainWindow):  
 \_translate = QtCore.QCoreApplication.translate  
 MainWindow.setWindowTitle(\_translate(**"MainWindow"**, **"Курсовая работа"**))  
 self.groupBox.setTitle(\_translate(**"MainWindow"**, **"Положение наблюдателя"**))  
 self.label.setText(\_translate(**"MainWindow"**, **"<html><head/><body><p align=\"center\">X</p></body></html>"**))  
 self.label\_2.setText(\_translate(**"MainWindow"**, **"<html><head/><body><p align=\"center\">Y</p></body></html>"**))  
 self.label\_3.setText(\_translate(**"MainWindow"**, **"<html><head/><body><p align=\"center\">Z</p></body></html>"**))  
 self.groupBox\_2.setTitle(\_translate(**"MainWindow"**, **"Положение источника света"**))  
 self.label\_4.setText(\_translate(**"MainWindow"**, **"<html><head/><body><p align=\"center\">X</p></body></html>"**))  
 self.label\_6.setText(\_translate(**"MainWindow"**, **"<html><head/><body><p align=\"center\">Z</p></body></html>"**))  
 self.label\_5.setText(\_translate(**"MainWindow"**, **"<html><head/><body><p align=\"center\">Y</p></body></html>"**))  
 self.groupBox\_3.setTitle(\_translate(**"MainWindow"**, **"Интенсивность"**))  
 self.groupBox\_4.setTitle(\_translate(**"MainWindow"**, **"Отрисовка"**))  
 self.comboBoxStyle.setItemText(0, \_translate(**"MainWindow"**, **"GL\_FILL"**))  
 self.comboBoxStyle.setItemText(1, \_translate(**"MainWindow"**, **"GL\_LINE"**))  
 self.comboBoxStyle.setItemText(2, \_translate(**"MainWindow"**, **"GL\_POINT"**))  
 self.groupBox\_5.setTitle(\_translate(**"MainWindow"**, **"Текстура"**))  
 self.comboBoxTexture.setItemText(0, \_translate(**"MainWindow"**, **"дерево"**))  
 self.comboBoxTexture.setItemText(1, \_translate(**"MainWindow"**, **"камень"**))  
 self.comboBoxTexture.setItemText(2, \_translate(**"MainWindow"**, **"асфальт"**))  
 self.comboBoxTexture.setItemText(3, \_translate(**"MainWindow"**, **"кирпич"**))  
 self.comboBoxTexture.setItemText(4, \_translate(**"MainWindow"**, **"без текстуры"**))  
 self.groupBox\_7.setTitle(\_translate(**"MainWindow"**, **"Цвет"**))  
 self.pushButton.setText(\_translate(**"MainWindow"**, **"Произвольный цвет"**))  
 self.pushButton\_2.setText(\_translate(**"MainWindow"**, **"Без цвета"**))  
 self.groupBox\_8.setTitle(\_translate(**"MainWindow"**, **"Количество ступенек"**))  
 self.check1.setText(\_translate(**"MainWindow"**, **"Режим камеры"**))

# ПРИЛОЖЕНИЕ В

Glwidget.py

**import** copy  
**import** numpy **as** np  
**import** ctypes  
**import** random  
**from** math **import** cos, sin, radians  
  
**from** OpenGL.GL **import** \*  
**from** PyQt5.QtCore **import** Qt, QPointF  
**from** PyQt5.QtGui **import** \*  
**from** PyQt5.QtOpenGL **import** \*  
  
**from** figures **import** \*  
  
  
**class** glWidget(QGLWidget):  
  
 **def** \_\_init\_\_(self, parent):  
  
 QGLWidget.\_\_init\_\_(self, parent)  
 self.rotationX = 0  
 self.rotationY = 0  
 self.rotationZ = 0  
  
 self.is\_pressed = **False** self.firstMouse = **True** self.isViewingCamera = **True** self.yaw = -100.0  
 self.pitch = -10.0  
 self.scale = 5.0  
  
 fx = cos(radians(self.yaw)) \* cos(radians(self.pitch))  
 fy = sin(radians(self.pitch))  
 fz = sin(radians(self.yaw)) \* cos(radians(self.pitch))  
 self.cameraFront = QVector3D(fx, fy, fz)  
 self.cameraPos = QVector3D(5.0, 4.0, 16.0)  
 self.cameraUp = QVector3D(0.0, 1.0, 0.0)  
  
 self.LightIntensity = QVector3D(1.0, 1.0, 1.0)  
 self.LightPosition = QVector4D(9.0, 5.0, 13.0, 0.0)  
  
 self.polygon\_mode = GL\_FILL  
  
 self.projection\_matrix = QMatrix4x4()  
 self.projection\_matrix.perspective(45.0, self.width() / self.height(), 0.1, 100.0)  
 self.current\_view = QMatrix4x4()  
 self.current\_model = QMatrix4x4()  
  
 self.changeColor((0.2, 0.2, 0.2, 1))  
 self.changeStepCount(5)  
  
 **def** resizeGL(self, w, h):  
  
 glMatrixMode(GL\_PROJECTION)  
 glLoadIdentity()  
 glViewport(0, 0, w, h)  
 glMatrixMode(GL\_MODELVIEW)  
  
 **def** paintGL(self):  
 glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT) *# очистка буферов* glEnable(GL\_DEPTH\_TEST)  
  
 self.shaderProgram.setUniformValue(**'light.position'**, self.LightPosition)  
 self.shaderProgram.setUniformValue(**'light.direction'**, self.cameraFront)  
 self.shaderProgram.setUniformValue(**'ViewPosition'**, self.cameraPos)  
 self.shaderProgram.setUniformValue(**'LightIntensity'**, self.LightIntensity)  
  
 self.updateMatrices()  
  
 glPointSize(4)  
 glLineWidth(2)  
 glPolygonMode(GL\_FRONT\_AND\_BACK, self.polygon\_mode)  
 self.drawLight()  
  
 self.drawScene()  
  
 glDisable(GL\_DEPTH\_TEST)  
  
 **def** drawScene(self):  
 step\_width = 4  
 step\_height = 0.3  
 step\_length = 1  
  
 self.drawFigure(Cube(3, 0.3, 8, self.currentTexture, self.current\_color))  
 **for** i **in** range(self.step\_count):  
 self.drawFigure(Cube(step\_length, step\_height, step\_width, self.currentTexture, self.current\_color),  
 offset=(2 \* step\_length \* (i + 2),  
 2 \* step\_height \* (i + 1),  
 step\_width))  
 self.drawFigure(Cube(step\_length, step\_height, step\_width, self.currentTexture, self.current\_color),  
 offset=(2 \* step\_length \* (i + 2),  
 -2 \* step\_height \* (i + 1),  
 -step\_width))  
 self.drawFigure(Cube(3 \* step\_length \* self.step\_count, 0.3, 3 \* step\_length \* self.step\_count,  
 -1, (0.25, 0.25, 0.25, 1)),  
 offset=(step\_length \* self.step\_count / 2, -2 \* step\_height \* (self.step\_count + 1), 0))  
  
 **def** initializeGL(self):  
 glClearColor(0.133, 0.5, 0.7, 1.0)  
  
 *# VBO* self.\_vertexBuffer = glGenBuffers(1)  
 glBindBuffer(GL\_ARRAY\_BUFFER, self.\_vertexBuffer)  
 *# IBO* self.\_indexBuffer = glGenBuffers(1)  
 glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, self.\_indexBuffer)  
  
 self.shaderProgram = self.initShaderProgram()  
  
 self.attribVertexColor = self.shaderProgram.attributeLocation(**"color\_vertex"**)  
 self.attribVertexPosition = self.shaderProgram.attributeLocation(**"position\_vertex"**)  
 self.attribVertexNormal = self.shaderProgram.attributeLocation(**"normal\_vertex"**)  
 self.attribVertexTexCoord = self.shaderProgram.attributeLocation(**"texcoord\_vertex"**)  
  
 glEnable(GL\_TEXTURE\_2D)  
 self.woodTexture = self.bindTexture(QPixmap(**"textures/дерево.jpg"**))  
 self.brickTexture = self.bindTexture(QPixmap(**"textures/кирпич.jpg"**))  
 self.stoneTexture = self.bindTexture(QPixmap(**"textures/камень.jpg"**))  
 self.roadTexture = self.bindTexture(QPixmap(**"textures/асфальт.jpg"**))  
  
 self.currentTexture = self.woodTexture  
  
 **def** changeStepCount(self, value):  
 self.step\_count = value  
  
 **def** changeColor(self, color=**None**):  
 **if** color:  
 self.current\_color = color  
 **else**:  
 self.current\_color = (random.random(), random.random(), random.random(), 1)  
  
 **def** mousePressEvent(self, event):  
 self.lastPos = event.pos()  
 self.is\_pressed = **True  
  
 def** mouseReleaseEvent(self, event):  
 self.is\_pressed = **False  
  
 def** mouseMoveEvent(self, event):  
 **if** self.isViewingCamera:  
 self.viewingCameraMode(event.x(), event.y())  
 **else**:  
 self.spinMode(event)  
 self.updateGL()  
  
 **def** wheelEvent(self, event):  
 self.LightPosition.setY(self.LightPosition.y() + event.angleDelta().y() / 120)  
 self.updateGL()  
  
 **def** viewingCameraMode(self, x, y):  
 sensitivity = 0.15  
  
 dx = (x - self.lastPos.x()) \* sensitivity  
 dy = (self.lastPos.y() - y) \* sensitivity  
  
 self.lastPos = QPointF(x, y)  
  
 self.yaw += dx  
 self.pitch += dy  
  
 **if** (self.pitch > 89.0):  
 self.pitch = 89.0  
 **if** (self.pitch < -89.0):  
 self.pitch = -89.0  
  
 self.cameraFront.setX(cos(radians(self.yaw)) \* cos(radians(self.pitch)))  
 self.cameraFront.setY(sin(radians(self.pitch)))  
 self.cameraFront.setZ(sin(radians(self.yaw)) \* cos(radians(self.pitch)))  
 self.cameraFront.normalize()  
  
 **def** spinMode(self, event):  
 speed = 180  
 **if** self.is\_pressed:  
 dx = (event.x() - self.lastPos.x()) / self.width()  
 dy = (event.y() - self.lastPos.y()) / self.height()  
 **if** (event.buttons() == Qt.LeftButton):  
 self.rotationX += speed \* dy  
 self.rotationY += speed \* dx  
 **elif** (event.buttons() == Qt.RightButton):  
 self.rotationX += speed \* dy  
 self.rotationZ += speed \* dx  
 self.lastPos = event.pos()  
  
 **def** keyboardCallBack(self, event):  
 cameraSpeed = 0.5  
 **if** event.key() == ord(**'W'**) **or** event.key() == 1062:  
 self.cameraPos += cameraSpeed \* self.cameraFront  
 **if** event.key() == ord(**'A'**) **or** event.key() == 1060:  
 cross = QVector3D.crossProduct(self.cameraFront, self.cameraUp)  
 cross.normalize()  
 self.cameraPos -= cross \* cameraSpeed  
 **if** event.key() == ord(**'S'**) **or** event.key() == 1067:  
 self.cameraPos -= cameraSpeed \* self.cameraFront  
 **if** event.key() == ord(**'D'**) **or** event.key() == 1042:  
 cross = QVector3D.crossProduct(self.cameraFront, self.cameraUp)  
 cross.normalize()  
 self.cameraPos += cross \* cameraSpeed  
 **if** event.key() == 16777235: *# ^* self.LightPosition.setZ(self.LightPosition.z() - 1)  
 **if** event.key() == 16777234: *# <-* self.LightPosition.setX(self.LightPosition.x() - 1)  
 **if** event.key() == 16777237: *# v* self.LightPosition.setZ(self.LightPosition.z() + 1)  
 **if** event.key() == 16777236: *# ->* self.LightPosition.setX(self.LightPosition.x() + 1)  
  
 **def** initShaderProgram(self):  
 shaderProgram = QOpenGLShaderProgram()  
 shaderProgram.addShaderFromSourceFile(QOpenGLShader.Vertex, **"shaders/shader.vert"**)  
 shaderProgram.addShaderFromSourceFile(QOpenGLShader.Fragment, **"shaders/shader.frag"**)  
 shaderProgram.link()  
 shaderProgram.bind()  
  
 *# light* shaderProgram.setUniformValue(**'light.ambient'**, QVector3D(0.2, 0.2, 0.2))  
 shaderProgram.setUniformValue(**'light.diffuse'**, QVector3D(0.5, 0.5, 0.5))  
 shaderProgram.setUniformValue(**'light.specular'**, QVector3D(0.1, 0.1, 0.1))  
 shaderProgram.setUniformValue(**'light.constant'**, 1.0)  
 shaderProgram.setUniformValue(**'light.linear'**, 0.014)  
 shaderProgram.setUniformValue(**'light.quadratic'**, 0.0007)  
 shaderProgram.setUniformValue(**'light.cutOff'**, cos(radians(12.5)))  
 shaderProgram.setUniformValue(**'light.outerCutOff'**, cos(radians(17.5)))  
 *# material* shaderProgram.setUniformValue(**'material.specular'**, QVector3D(0.5, 0.5, 0.5))  
 shaderProgram.setUniformValue(**'material.shininess'**, 10.0)  
  
 **return** shaderProgram  
  
 **def** updateMatrices(self):  
  
 view, model = QMatrix4x4(), QMatrix4x4()  
  
 view.lookAt(self.cameraPos, self.cameraPos + self.cameraFront, self.cameraUp)  
  
 model.rotate(self.rotationX, QVector3D(1.0, 0.0, 0.0))  
 model.rotate(self.rotationY, QVector3D(0.0, 1.0, 0.0))  
 model.rotate(self.rotationZ, QVector3D(0.0, 0.0, 1.0))  
  
 self.current\_view = view  
 self.current\_model = model  
  
 self.setUniformMatrix(model, view)  
  
 **def** setUniformMatrix(self, model, view):  
 modelview = view \* model  
 self.shaderProgram.setUniformValue(**"MVP"**, self.projection\_matrix \* modelview)  
 self.shaderProgram.setUniformValue(**"NormalMatrix"**, model.normalMatrix())  
 self.shaderProgram.setUniformValue(**"ModelMatrix"**, model)  
  
 **def** drawFigure(self, figure, offset=(0, 0, 0), rotate=(0, 0, 0), scale=(1, 1, 1)):  
 self.setDefaultAttribPointers()  
 self.enableAttributeArrays()  
 model = copy.deepcopy(self.current\_model)  
 model.translate(\*offset)  
 model.rotate(rotate[0], QVector3D(1.0, 0.0, 0.0))  
 model.rotate(rotate[1], QVector3D(0.0, 1.0, 0.0))  
 model.rotate(rotate[2], QVector3D(0.0, 0.0, 1.0))  
 model.scale(\*scale)  
 self.setUniformMatrix(model, self.current\_view)  
  
 glBindTexture(GL\_TEXTURE\_2D, figure.texture)  
 glBufferData(GL\_ARRAY\_BUFFER, np.array(figure.generate(), dtype=**'float32'**), GL\_STATIC\_DRAW)  
 glDrawArrays(figure.type, 0, figure.count)  
 self.disableAttributeArrays()  
  
 self.setUniformMatrix(self.current\_model, self.current\_view)  
  
 **def** drawLight(self):  
 self.drawFigure(Cube(0.1, 0.1, 0.1, -1, (1, 1, 0, 1)),  
 offset=(self.LightPosition.x(), self.LightPosition.y(), self.LightPosition.z()))  
  
 **def** enableAttributeArrays(self):  
 self.shaderProgram.enableAttributeArray(self.attribVertexPosition)  
 self.shaderProgram.enableAttributeArray(self.attribVertexTexCoord)  
 self.shaderProgram.enableAttributeArray(self.attribVertexNormal)  
 self.shaderProgram.enableAttributeArray(self.attribVertexColor)  
  
 **def** disableAttributeArrays(self):  
 self.shaderProgram.disableAttributeArray(self.attribVertexPosition)  
 self.shaderProgram.disableAttributeArray(self.attribVertexTexCoord)  
 self.shaderProgram.disableAttributeArray(self.attribVertexNormal)  
 self.shaderProgram.disableAttributeArray(self.attribVertexColor)  
  
 **def** setDefaultAttribPointers(self):  
 stride = 12 \* ctypes.sizeof(ctypes.c\_float)  
 glVertexAttribPointer(self.attribVertexPosition, 3, GL\_FLOAT, GL\_FALSE, stride, **None**)  
 glVertexAttribPointer(self.attribVertexNormal, 3, GL\_FLOAT, GL\_FALSE, stride,  
 ctypes.c\_void\_p(3 \* ctypes.sizeof(ctypes.c\_float)))  
 glVertexAttribPointer(self.attribVertexTexCoord, 2, GL\_FLOAT, GL\_FALSE, stride,  
 ctypes.c\_void\_p(6 \* ctypes.sizeof(ctypes.c\_float)))  
 glVertexAttribPointer(self.attribVertexColor, 4, GL\_FLOAT, GL\_FALSE, stride,  
 ctypes.c\_void\_p(8 \* ctypes.sizeof(ctypes.c\_float)))

# ПРИЛОЖЕНИЕ Г

mainwindow.ui

**from** PyQt5 **import** QtWidgets, QtGui  
**from** PyQt5.QtWidgets **import** QCheckBox  
  
**from** ui.form **import** Ui\_MainWindow  
**from** glwidget **import** glWidget, GL\_FILL, GL\_LINE, GL\_POINT  
  
  
**class** MainWindow(QtWidgets.QMainWindow, Ui\_MainWindow):  
  
 **def** \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setupUi(self)  
  
 self.glwidget = glWidget(self)  
 self.glwidget.setMinimumSize(700, 100)  
 self.widget\_layout.addWidget(self.glwidget)  
  
 **def** check1F():  
 self.glwidget.isViewingCamera = self.check1.isChecked()  
 self.check1.stateChanged.connect(check1F)  
 self.sliderLightIntensive.valueChanged.connect(self.changeLightIntensive)  
 self.comboBoxStyle.currentIndexChanged.connect(self.changeDrawStyle)  
 self.comboBoxTexture.currentTextChanged.connect(self.changeTexture)  
  
 self.sliderLightPosX.valueChanged.connect(self.changeLightPosX)  
 self.sliderLightPosY.valueChanged.connect(self.changeLightPosY)  
 self.sliderLightPosZ.valueChanged.connect(self.changeLightPosZ)  
  
 self.sliderViewPosX.valueChanged.connect(self.changeViewPosX)  
 self.sliderViewPosY.valueChanged.connect(self.changeViewPosY)  
 self.sliderViewPosZ.valueChanged.connect(self.changeViewPosZ)  
  
 self.pushButton.clicked.connect(self.changeColor)  
 self.pushButton\_2.clicked.connect(self.changeColor2)  
  
 self.spinBox.valueChanged.connect(self.changeStepCount)  
  
 self.glwidget.setFocus()  
  
 **def** changeStepCount(self, value):  
 self.glwidget.changeStepCount(value)  
 self.glwidget.setFocus()  
 self.glwidget.updateGL()  
  
 **def** changeColor2(self):  
 self.glwidget.changeColor((0.2, 0.2, 0.2, 1))  
 self.glwidget.setFocus()  
 self.glwidget.updateGL()  
  
 **def** changeColor(self):  
 self.glwidget.changeColor()  
 self.glwidget.setFocus()  
 self.glwidget.updateGL()  
  
 **def** keyPressEvent(self, event):  
 self.glwidget.keyboardCallBack(event)  
 **if** event.key() % 32 == 0:  
 self.check1.setChecked(**not** self.check1.isChecked())  
 self.glwidget.updateGL()  
  
 **def** changeLightIntensive(self, val):  
 self.glwidget.LightIntensity.setX(val / 100)  
 self.glwidget.LightIntensity.setY(val / 100)  
 self.glwidget.LightIntensity.setZ(val / 100)  
 self.glwidget.setFocus()  
 self.glwidget.updateGL()  
  
 **def** changeDrawStyle(self, value):  
 **if** value **is** 1:  
 type\_draw = GL\_LINE  
 **elif** value **is** 2:  
 type\_draw = GL\_POINT  
 **else**:  
 type\_draw = GL\_FILL  
  
 self.glwidget.polygon\_mode = type\_draw  
 self.glwidget.setFocus()  
 self.glwidget.updateGL()  
  
 **def** changeTexture(self, value):  
 **if** value == **'дерево'**:  
 self.glwidget.currentTexture = self.glwidget.woodTexture  
 **if** value == **'камень'**:  
 self.glwidget.currentTexture = self.glwidget.stoneTexture  
 **if** value == **'асфальт'**:  
 self.glwidget.currentTexture = self.glwidget.roadTexture  
 **if** value == **'кирпич'**:  
 self.glwidget.currentTexture = self.glwidget.brickTexture  
 **if** value == **'без текстуры'**:  
 self.glwidget.currentTexture = -1  
  
 self.glwidget.setFocus()  
 self.glwidget.updateGL()  
  
 **def** changeLightPosX(self, value):  
 self.glwidget.LightPosition.setX(value / 5 - 5)  
 self.glwidget.setFocus()  
 self.glwidget.updateGL()  
  
 **def** changeLightPosY(self, value):  
 self.glwidget.LightPosition.setY(value / 5 - 5)  
 self.glwidget.setFocus()  
 self.glwidget.updateGL()  
  
 **def** changeLightPosZ(self, value):  
 self.glwidget.LightPosition.setZ(value / 5 - 5)  
 self.glwidget.setFocus()  
 self.glwidget.updateGL()  
  
  
 **def** changeViewPosX(self, value):  
 self.glwidget.cameraPos.setX(value / 2.5 - 20)  
 self.glwidget.setFocus()  
 self.glwidget.updateGL()  
  
 **def** changeViewPosY(self, value):  
 self.glwidget.cameraPos.setY(value / 2.5 - 20)  
 self.glwidget.setFocus()  
 self.glwidget.updateGL()  
  
 **def** changeViewPosZ(self, value):  
 self.glwidget.cameraPos.setZ(value / 2.5 - 20)  
 self.glwidget.setFocus()  
 self.glwidget.updateGL()  
  
  
  
  
  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 **pass**

# ПРИЛОЖЕНИЕ Д

Figures.py

**from** OpenGL.raw.GL.VERSION.GL\_1\_1 **import** GL\_TRIANGLES  
  
  
**class** Cube:  
 **def** \_\_init\_\_(self, width, height, length, texture, color=(0.2, 0.2, 0.2, 1)):  
 self.x = width  
 self.y = height  
 self.z = length  
 self.count = 36  
 self.type = GL\_TRIANGLES  
 self.texture = texture  
 self.color = color  
  
 **def** generate(self):  
 **return** [  
 -self.x, -self.y, -self.z, 0.0, 0.0, -1.0, 0.0, 0.0, \*self.color,  
 self.x, -self.y, -self.z, 0.0, 0.0, -1.0, 1.0, 0.0, \*self.color,  
 self.x, self.y, -self.z, 0.0, 0.0, -1.0, 1.0, 1.0, \*self.color,  
 self.x, self.y, -self.z, 0.0, 0.0, -1.0, 1.0, 1.0, \*self.color,  
 -self.x, self.y, -self.z, 0.0, 0.0, -1.0, 0.0, 1.0, \*self.color,  
 -self.x, -self.y, -self.z, 0.0, 0.0, -1.0, 0.0, 0.0, \*self.color,  
  
 -self.x, -self.y, self.z, 0.0, 0.0, 1.0, 0.0, 0.0, \*self.color,  
 self.x, -self.y, self.z, 0.0, 0.0, 1.0, 1.0, 0.0, \*self.color,  
 self.x, self.y, self.z, 0.0, 0.0, 1.0, 1.0, 1.0, \*self.color,  
 self.x, self.y, self.z, 0.0, 0.0, 1.0, 1.0, 1.0, \*self.color,  
 -self.x, self.y, self.z, 0.0, 0.0, 1.0, 0.0, 1.0, \*self.color,  
 -self.x, -self.y, self.z, 0.0, 0.0, 1.0, 0.0, 0.0, \*self.color,  
  
 -self.x, self.y, self.z, -1.0, 0.0, 0.0, 1.0, 0.0, \*self.color,  
 -self.x, self.y, -self.z, -1.0, 0.0, 0.0, 1.0, 1.0, \*self.color,  
 -self.x, -self.y, -self.z, -1.0, 0.0, 0.0, 0.0, 1.0, \*self.color,  
 -self.x, -self.y, -self.z, -1.0, 0.0, 0.0, 0.0, 1.0, \*self.color,  
 -self.x, -self.y, self.z, -1.0, 0.0, 0.0, 0.0, 0.0, \*self.color,  
 -self.x, self.y, self.z, -1.0, 0.0, 0.0, 1.0, 0.0, \*self.color,  
  
 self.x, self.y, self.z, 1.0, 0.0, 0.0, 1.0, 0.0, \*self.color,  
 self.x, self.y, -self.z, 1.0, 0.0, 0.0, 1.0, 1.0, \*self.color,  
 self.x, -self.y, -self.z, 1.0, 0.0, 0.0, 0.0, 1.0, \*self.color,  
 self.x, -self.y, -self.z, 1.0, 0.0, 0.0, 0.0, 1.0, \*self.color,  
 self.x, -self.y, self.z, 1.0, 0.0, 0.0, 0.0, 0.0, \*self.color,  
 self.x, self.y, self.z, 1.0, 0.0, 0.0, 1.0, 0.0, \*self.color,  
  
 -self.x, -self.y, -self.z, 0.0, -1.0, 0.0, 0.0, 1.0, \*self.color,  
 self.x, -self.y, -self.z, 0.0, -1.0, 0.0, 1.0, 1.0, \*self.color,  
 self.x, -self.y, self.z, 0.0, -1.0, 0.0, 1.0, 0.0, \*self.color,  
 self.x, -self.y, self.z, 0.0, -1.0, 0.0, 1.0, 0.0, \*self.color,  
 -self.x, -self.y, self.z, 0.0, -1.0, 0.0, 0.0, 0.0, \*self.color,  
 -self.x, -self.y, -self.z, 0.0, -1.0, 0.0, 0.0, 1.0, \*self.color,  
  
 -self.x, self.y, -self.z, 0.0, 1.0, 0.0, 0.0, 1.0, \*self.color,  
 self.x, self.y, -self.z, 0.0, 1.0, 0.0, 1.0, 1.0, \*self.color,  
 self.x, self.y, self.z, 0.0, 1.0, 0.0, 1.0, 0.0, \*self.color,  
 self.x, self.y, self.z, 0.0, 1.0, 0.0, 1.0, 0.0, \*self.color,  
 -self.x, self.y, self.z, 0.0, 1.0, 0.0, 0.0, 0.0, \*self.color,  
 -self.x, self.y, -self.z, 0.0, 1.0, 0.0, 0.0, 1.0, \*self.color]

# ПРИЛОЖЕНИЕ Е

Shader.frag

#version 330 core  
  
**in vec4** position\_;  
**in vec3** normal\_;  
**in vec4** color\_;  
**in vec2** texcoord\_;  
  
**struct** Material {  
 **sampler2D** diffuse;  
 **vec3** specular;  
 **float** shininess;  
};  
  
**struct** Light {  
 **vec4** position;  
 **vec3** direction;  
 **float** cutOff;  
 **float** outerCutOff;  
  
 **vec3** ambient;  
 **vec3** diffuse;  
 **vec3** specular;  
  
 **float** constant;  
 **float** linear;  
 **float** quadratic;  
};  
  
  
**uniform vec3** ViewPosition;  
**uniform vec3** LightIntensity;  
**uniform** Material material;  
**uniform** Light light;  
  
**out vec4** OutColor;  
  
  
  
**void** main(){  
 **vec3** color;  
  
 **vec3** normal = normalize(normal\_);  
 **vec3** lightDir;  
 **vec3** view;  
 **float** intensity = 1.0;  
  
 **if**(light.position.w == 0.0) {  
 lightDir = normalize(light.position.xyz);  
 view = normalize(-position\_.xyz);  
 }  
 **else** {  
 lightDir = normalize(light.position.xyz - position\_.xyz);  
 view = normalize(ViewPosition - position\_.xyz);  
 }  
  
 **vec3** reflect = reflect(-lightDir, normal);  
  
 *// затухание* **float** distance = length(light.position.xyz - position\_.xyz);  
 **float** attenuation = 1.0 / (light.constant + light.linear \* distance + light.quadratic \* (distance \* distance));  
  
 *// окружающий свет* color = light.ambient;  
  
 *// добавление размытия* **float** dotNL = max(dot(normal, lightDir), 0.0);  
 color += light.diffuse \* dotNL \* intensity;  
  
 *// добавление текстуры* color \*= texture2D(material.diffuse, texcoord\_).rgb;  
  
 *// Добавление цвета* color += color\_.rgb;  
  
 *// Добавление бликов* **float** dotNR = max(dot(view, reflect), 0.0);  
 color += pow(dotNR, material.shininess) \* light.specular \* material.specular \* intensity;  
  
 color \*= attenuation;  
  
 OutColor = **vec4**(LightIntensity \* color, 1.0f);  
}

# ПРИЛОЖЕНИЕ Ж

**Shader.vert**

#version 330 core  
  
  
**in vec3** position\_vertex;  
**in vec3** normal\_vertex;  
**in vec2** texcoord\_vertex;  
**in vec4** color\_vertex;  
  
**out vec4** position\_;  
**out vec3** normal\_;  
**out vec4** color\_;  
**out vec2** texcoord\_;  
  
**uniform mat3** NormalMatrix;  
**uniform mat4** MVP;  
**uniform mat4** ModelMatrix;  
  
  
**void** main(){  
   
 position\_ = ModelMatrix \* **vec4**(position\_vertex, 1.0);  
 normal\_ = NormalMatrix \* normal\_vertex;  
 color\_ = color\_vertex;  
 texcoord\_ = texcoord\_vertex;  
  
 gl\_Position = MVP \* **vec4**(position\_vertex, 1.0);  
}