Санкт-Петербургский Государственный
Электротехнический Университет

Кафедра МОЭВМ

Задание для лабораторной работы № 7
"Реализация трехмерного объекта с использованием библиотеки OpenGL"

Выполнил: Калмак Д.А.
Факультет: ФКТИ
Группа: 0303
Преподаватель: Герасимова Т.В.

Санкт-Петербург
2023 г.

Цель работы.

- ознакомление с трехмерными объектами, освещением, тенями.

- проанализировать полученное задание, выделить информационные объекты и действия;

- разработать программу, реализующую представление трехмерной сцены с формированием различного типа проекции теней.

Задание.

Разработать программу, реализующую представление разработанной вами трехмерной сцены с добавлением возможности формирования различного типа проекций теней, используя предложенные функции OpenGL.

Выполнение работы.

В классе mainWindow в self.stack (QStackedWidget) добавлен виджет класса glWidget3d, который наследуется от glWidget0. Класс пополнен функциями освещения и методами. Для управления освещением был создан отдельный слой buttonsLayout3, принадлежащий классу QtWidgets.QVBoxLayout. Атрибут self.boxlight_flag принадлежит классу QCheckBox. Он необходим для управления режимом освещения: включен и выключен. Когда в нем меняется состояние, через метод stateChanged у self.boxlight_flag передается с помощью метода connect состояние в метод self.update_light_flag. Если активно, то есть равно Qt.Checked, то для всех виджетов в self.stack, обращение к которым осуществляется с помощью метода widget, меняется значение атрибута light_flag на True. Атрибут self.light_flag был добавлен в класс glWidget0, от которого наследуются класс glWidget3d. Иначе атрибуту присваивается значение False. В обоих случаях происходит обновление

виджетов с помощью метода updateGL. Пример метода для QCheckBox self.update_light_flag представлен ниже:

```
def update_light_flag(self, state):
    if state == Qt.Checked:
        for i in range(self.stack.__len__()):
            self.stack.widget(i).light_flag = True
            self.stack.widget(i).updateGL()
    else:
        for i in range(self.stack.__len__()):
            self.stack.widget(i).light_flag = False
            self.stack.widget(i).updateGL()
```

Работа остальных виджетов класса QCheckBox аналогична, поэтому будут описаны ключевые моменты. В классе mainWindow добавлен атрибут self.lblprojection, который принадлежит классу QLabel. Он содержит текст для формы с выбором типа проекции. Атрибут self.boxglupprojection принадлежит классу QCheckBox и связан с методом self.update_glupprojection и атрибутом glupprojection в классе glWidget0, от которого наследуется glWidget3d. Он отвечает за выставление перспективной проекции. С помощью метода setChecked у self.boxglupprojection эта форма активирована сразу. Атрибут self.boxgloprojection принадлежит классу QCheckBox и связан с методом self.update_gloprojection и атрибутом gloprojection в классе glWidget0, от которого наследуется glWidget3d. Он отвечает за выставление ортогональной проекции. В методах настроено так, что если одна форма активирована, то другая выключена. Если выключить одну, то включится другая. Если включить другую, когда включена первая, первая отключится. Описанные методы представлен ниже:

```
def update_glupprojection(self, state):
```

```
    if state == Qt.Checked:
        for i in range(self.stack.__len__()):
            self.stack.widget(i).glupprojection_flag = True
            self.stack.widget(i).gloprojection_flag = False
            self.boxgloprojection.setChecked(False)
            self.stack.widget(i).updateGL()
    else:
        for i in range(self.stack.__len__()):
            self.stack.widget(i).glupprojection_flag = False
            self.stack.widget(i).gloprojection_flag = True
            self.boxgloprojection.setChecked(True)
            self.stack.widget(i).updateGL()

def update_gloprojection(self, state):
    if state == Qt.Checked:
        for i in range(self.stack.__len__()):
            self.stack.widget(i).gloprojection_flag = True
            self.stack.widget(i).glupprojection_flag = False
            self.boxglupprojection.setChecked(False)
            self.stack.widget(i).updateGL()
    else:
        for i in range(self.stack.__len__()):
            self.stack.widget(i).gloprojection_flag = False
            self.stack.widget(i).glupprojection_flag = True
            self.boxglupprojection.setChecked(True)
            self.stack.widget(i).updateGL()
```

В классе mainWindow добавлен атрибут self.lblcolor, который принадлежит классу QLabel. Он содержит текст для ползунка с выбором источника света. Ползунок self.slidercolor принадлежит классу QSlider в горизонтальном виде с помощью параметра Qt.Orientation.Horizontal. Заданы минимальное и максимальное значения для ползунка с помощью методов setMinimum и setMaximum у self.slidercolor. С помощью метода setValue у self.slidercolor установлено начальное значение. Двигая ползунок, с помощью метода valueChanged у self.slidercolor, который передает значение ползунка, и метода connect, который связывает ползунок с методом update_color, метод update_color в классе mainWindow получает значения с ползунка. В методе для всех виджетов в self.stack, обращение к которым осуществляется с помощью метода widget, обновляется значение атрибута self.color, который добавлен в класс glWidget0, от которого наследуются класс glWidget3d. Происходит обновление виджетов с помощью метода updateGL. Пример метода для QSlider update_color представлен ниже:

```
def update_color(self, value):
    for i in range(self.stack.__len__()):
        self.stack.widget(i).color = value
        self.stack.widget(i).updateGL()
```

Дальнейшие виджеты-ползунки имеют такую же структуру и взаимодействие с методами. Отличительные моменты будут описываться. Для регулировки источника света и превращения его в прожектор добавлены следующие виджеты: self.lblcutoff и self.lblexponent, принадлежащие классу QLabel и описывающие ползунки угла пропускания света и концентрации света соответственно. Это ползунок self.slidercutoff, принадлежащий классу QSlider. Его максимальное значение 91, потому что прожектор может быть от 0 до 90 и 180 градусов по умолчанию, как

обычный источник света, и 91 соотвествует 180 в программе. С ползунком связан метод self.update_cutoff и атрибут self.cutoff класса glWidget0, от которого наследуется класс glWidget3d. Это ползунок self.sliderexponent, принадлежащий классу QSlider. Его максимальное значение 5, потому что максимальное значение концентрации 5. С ползунком связан метод self.update_exponent и атрибут self.exponent класса glWidget0, от которого наследуется класс glWidget3d. Для регулировки положения источника света были добавлены следующие виджеты: self.lbllight, принадлежащий классу QLabel и описывающий ползунки координат x, y и z. Это ползунки self.sliderxlight, self.sliderylight и self.sliderzlight, принадлежащие классу QSlider. С ползунками связаны методы self.update_xlight, self.update_ylight и self.update_zlight и атрибуты self.xlight, self.ylight и self.zlight класса glWidget0, от которого наследуется класс glWidget3d. Атрибут self.boxcolor принадлежит классу QCheckBox и связан с методом self.update_color_flag и атрибутом color_flag в классе glWidget0, от которого наследуется glWidget3d. Он отвечает за включение цветного режима. Для регулировки ослабления света с расстоянием были добавлены следующие виджеты: self.lblattenuation, принадлежащий классу QLabel и описывающий ползунки для постоянного, линейного и квадратичного коэффициента затухания. Это ползунки self.slidercattenuation, self.sliderlattenuation и self.sliderqattenuation, принадлежащие классу QSlider. С ползунками связаны методы self.update_cattenuation, self.update_lattenuation и self.update_qattenuation и атрибуты self.cattenuation, self.lattenuation и self.qattenuation класса glWidget0, от которого наследуется класс glWidget3d. Значения варьируются от 0 до 10 (в методах значение делится на 10). Для self.slidercattenuation значение установлено с помощью метода setValue на 10, в переводе 1. Такое значение является по умолчанию. Атрибут self.boxnormalize принадлежит классу QCheckBox и связан с методом

self.update_normalize и атрибутом normalize в классе glWidget0, от которого наследуется glWidget3d. Он отвечает за включение нормализации нормалей. Атрибут self.boxlocal принадлежит классу QCheckBox и связан с методом self.update_local и атрибутом local в классе glWidget0, от которого наследуется glWidget3d. Он отвечает за точку наблюдения локальную или удаленную. Атрибут self.boxtwo принадлежит классу QCheckBox и связан с методом self.update_two и атрибутом two в классе glWidget0, от которого наследуется glWidget3d. Он отвечает за правильное закрашивание обеих сторон полигона. Для регулировки отделения зеркальной составляющей цвета добавлены следующие виджеты: self.lblcontrol, принадлежащий классу QLabel и описывающий ползунок для отделения зеркальной составляющей цвета. Это ползунок self.slidercontrol, принадлежащий классу QSlider. Значения ползунка 0 и 1, что в программе соответствует режиму по умолчанию, когда величины складываются, и режиму, когда зеркальная составляющая отделяется. С ползунком связан метод self.update_control и атрибут self.control класса glWidget0, от которого наследуется класс glWidget3d. Все вышеперечисленные виджеты были добавлены в слой buttonsLayout3 с помощью метода addWidget у buttonsLayout3. Слой buttonsLayout3 добавлен в слой mainLayout с помощью метода addLayout у mainLayout.

В классе glWidget0 в инициализации сразу определаются размеры окна в координатах. Размеры хранятся в атрибутах self.xcoef и self.ycoef. В методе initilizeGL кроме перспективной проекции добавлена ортогональная, которая задается с помощью функции glOrtho(-self.xcoef, self.xcoef, -self.ycoef, self.ycoef, 0.1, 100.0). Вид проекции зависит от self.glupprojection_flag и self.gloprojection_flag, которые регулируются формой.

В классе glWidget3d, который наследуется от glWidget0, в методе paintGL происходит настройка освещения. Аналогично методу initilizeGL в методе paintGL происходит выбор вида проекции. Выбор проекции представлен ниже:

```
glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    aspect = self.w / self.h
    if self.glupprojection_flag:
       gluPerspective(45.0, aspect, 0.1, 100.0)
    elif self.gloprojection_flag:
       glOrtho(-self.xcoef, self.xcoef, -self.ycoef, self.ycoef, 0.1, 100.0)
    glMatrixMode(GL_MODELVIEW)
```

Если форма, которая отвечает за атрибут self.light_flag, активирована, то открывается блок с настройкой света. В переменную myLightPosition записываются значения положения источника света. Положение управляют атрибуты self.xlight, self.ylight и self.zlight, которые регулируются ползунками. С помощью функции glLightfv применяется положение источника света:

```
myLightPosition = [self.xlight, self.ylight, self.zlight, 1]
glLightfv(GL_LIGHT0, GL_POSITION, myLightPosition)
```

С помощью функции *glEnable(GL_LIGHTING)* включается освещение. С помощью функции *glEnable(GL_LIGHT0)* включается источник света. В зависимости от значения self.color, который управляется ползунком, выстанавляется цвет для фонового, диффузного и зеркального освещения. Эти цвета устанавливаются с помощью функции glLightfv для GL_AMBIENT, GL_DIFFUSE и GL_SPECULAR:

```
glLightfv(GL_LIGHT0, GL_AMBIENT, amb0)
glLightfv(GL_LIGHT0, GL_DIFFUSE, diff0)
glLightfv(GL_LIGHT0, GL_SPECULAR, spec0)
```

В зависимости от значения self.cutoff определяется угол пропускания света и источник света может стать прожектором. Значение self.cutoff управляется ползунком, и по умолчанию угол равен 180, иначе от 0 до 90. Значение угла устанавливается с помощью функции glLightf для GL_SPOT_CUTOFF:

```
if self.cutoff == 91:
    glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 180)
else:
    glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, self.cutoff)
```

В зависимости от значения self.exponent, который управляется ползунком, определяется концентрация света, по умолчанию она равна 0. Значение концентрации устанавливается с помощью функции glLightf для GL_SPOT_EXPONENT:

```
glLightf(GL_LIGHT0, GL_SPOT_EXPONENT, self.exponent)
```

Направление света задается с помощью функции glLightfv для GL_SPOT_DIRECTION.

```
glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, dir)
```

В зависимости от значения self.color_flag, который управляется формой, включается цвет с помощью функции *glEnable(GL_COLOR_MATERIAL)* или выключается *glDisable(GL_COLOR_MATERIAL)*. В зависимости от значений self.cattenuation, self.lattenuation и self.qattenuation, которые регулируются ползунками и отвечают за ослабление света с расстоянием по постоянному, линейному и квадратичному коэффициентам затухания. Коэффициенты затухания устанавливаются с помощью функции glLightf для GL_CONSTANT_ATTENUATION, GL_LINEAR_ATTENUATION и GL_QUADRATIC_ATTENUATION:

```
glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, self.cattenuation)
glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, self.lattenuation)
```

glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, self.qattenuation)

В зависимости от значения self.normalize, который регулируется формой, включается режим нормализации нормалей с помощью функции *glEnable(GL_NORMALIZE)* и выключается с помощью функции *glDisable(GL_NORMALIZE)*. В зависимости от значения self.local, который управляется формой, определяется точка наблюдения локальная или удаленная. Значение устанавливается с помощью функции glLightModeli для GL_LIGHT_MODEL_LOCAL_VIEWER:

glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, self.local)

В зависимости от значения self.two, который управляется формой, определяется правильное закрашивание обеих сторон полигона. Значение устанавливается с помощью функции glLightModeli для GL_LIGHT_MODEL_TWO_SIDE:

glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, self.two)

В зависимости от значения self.control, который управляется ползунком, устанавливается отделение зеркальной составляющей цвета, либо по умолчанию все складывается, либо зеркальная составляющая отделяется. Значение устанавливается с помощью функции glLightModeli для GL_LIGHT_MODEL_COLOR_CONTROL и значения равны GL_SINGLE_COLOR или GL_SEPARATE_SPECULAR_COLOR:

```
if self.control == 0:
    glLightModeli(GL_LIGHT_MODEL_COLOR_CONTROL,
GL_SINGLE_COLOR)
elif self.control == 1:
    glLightModelf(GL_LIGHT_MODEL_COLOR_CONTROL,
GL_SEPARATE_SPECULAR_COLOR)
```

ТЕСТИРОВАНИЕ.

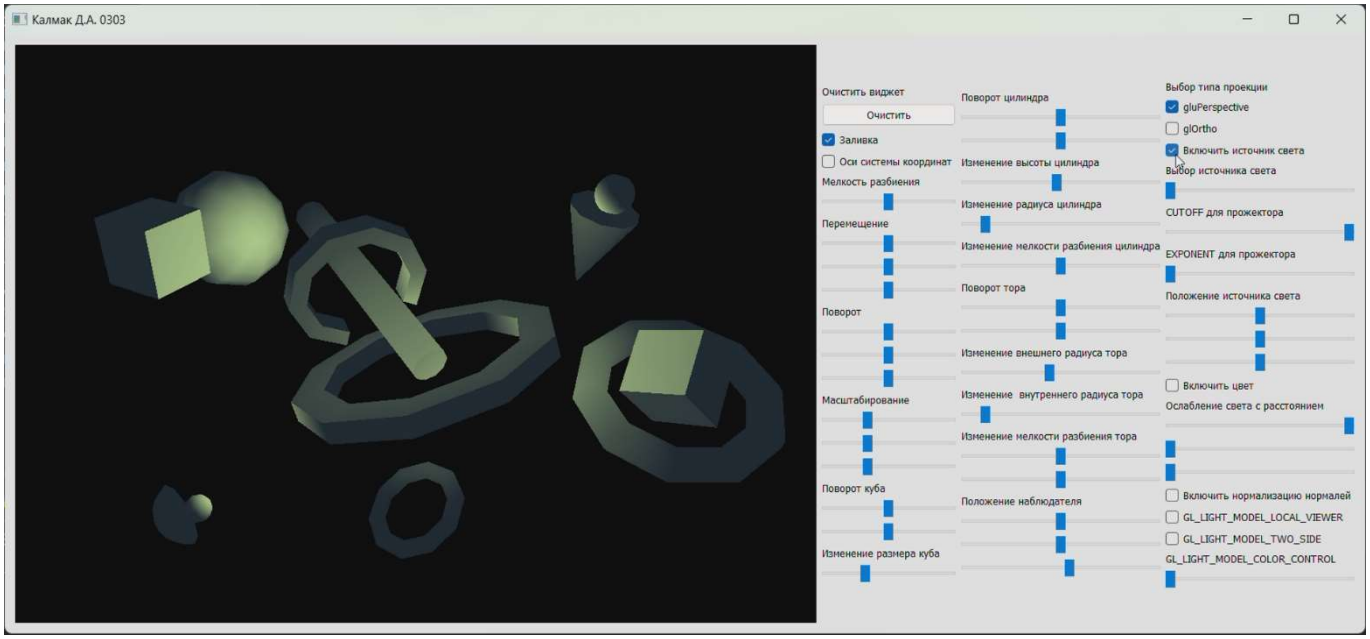

Рисунок 1 – Трехмерная сцена с включенным источником света

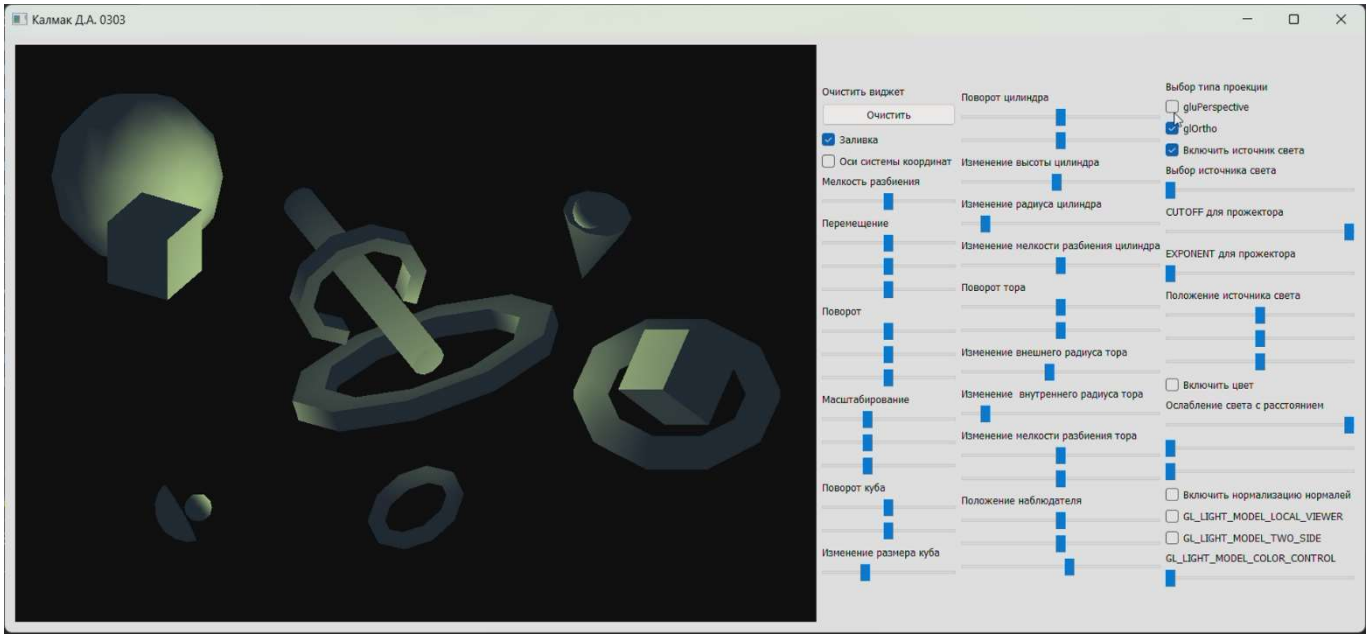

Рисунок 2 – Сменен тип проекции

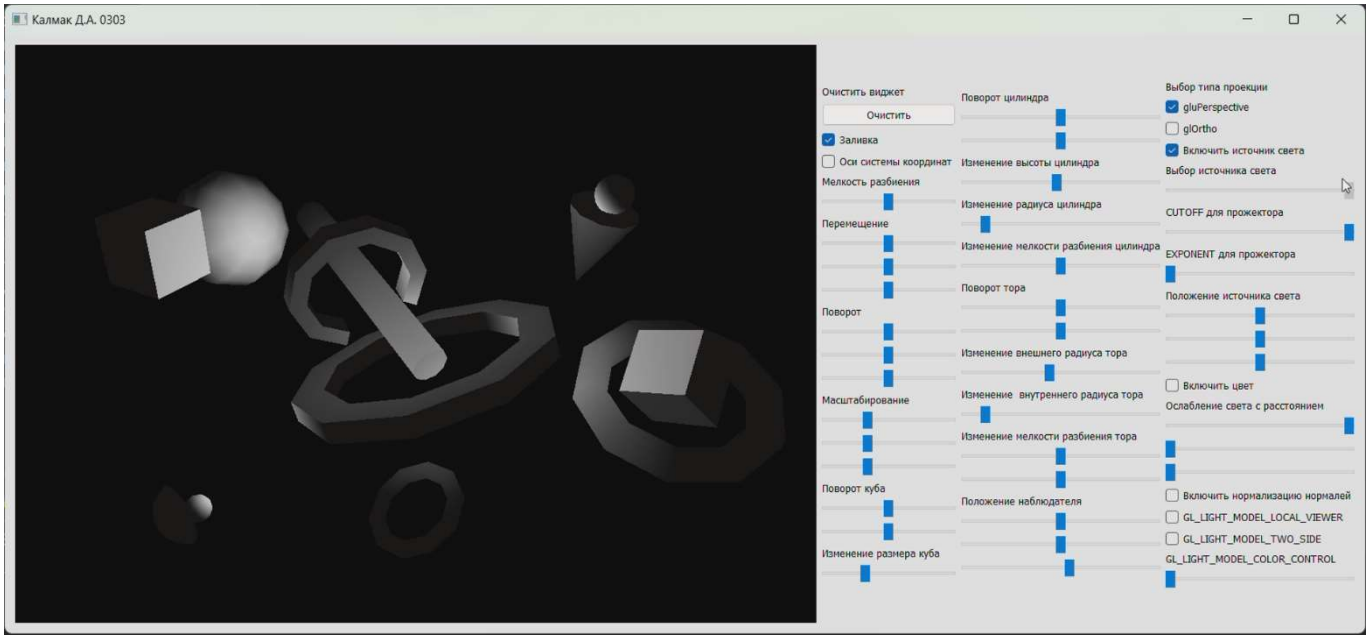

Рисунок 3 – Изменены параметры освещения

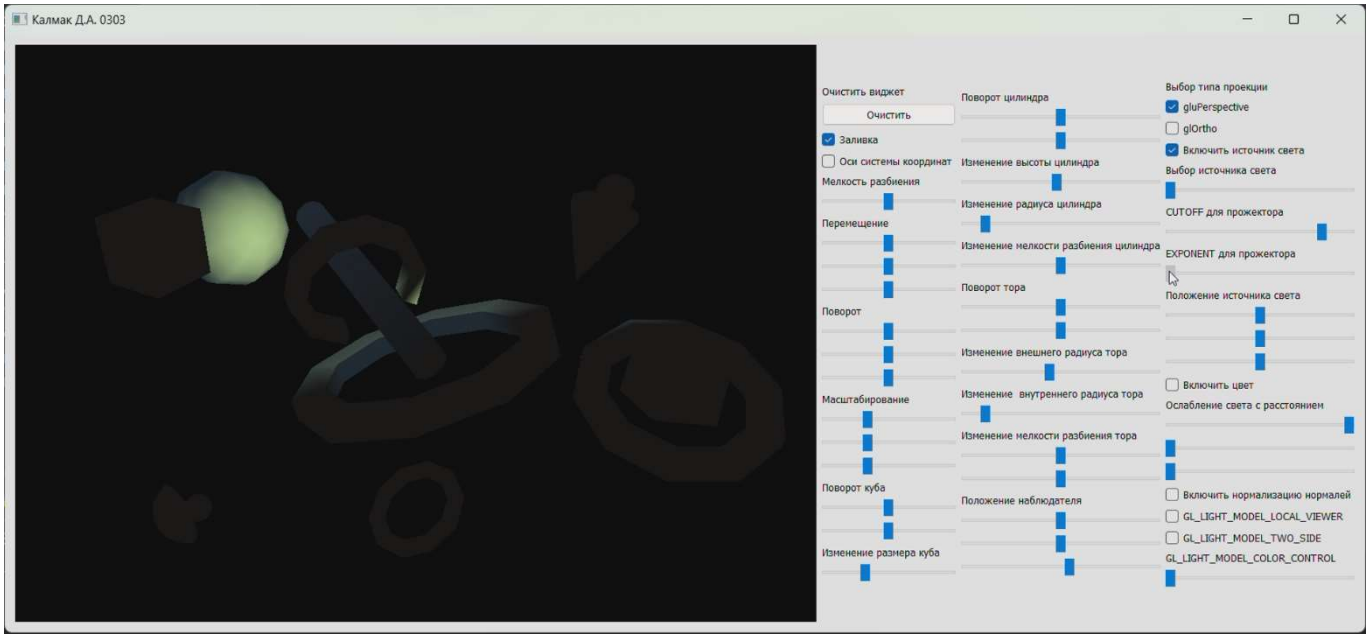

Рисунок 4 – Источник света стал прожектором

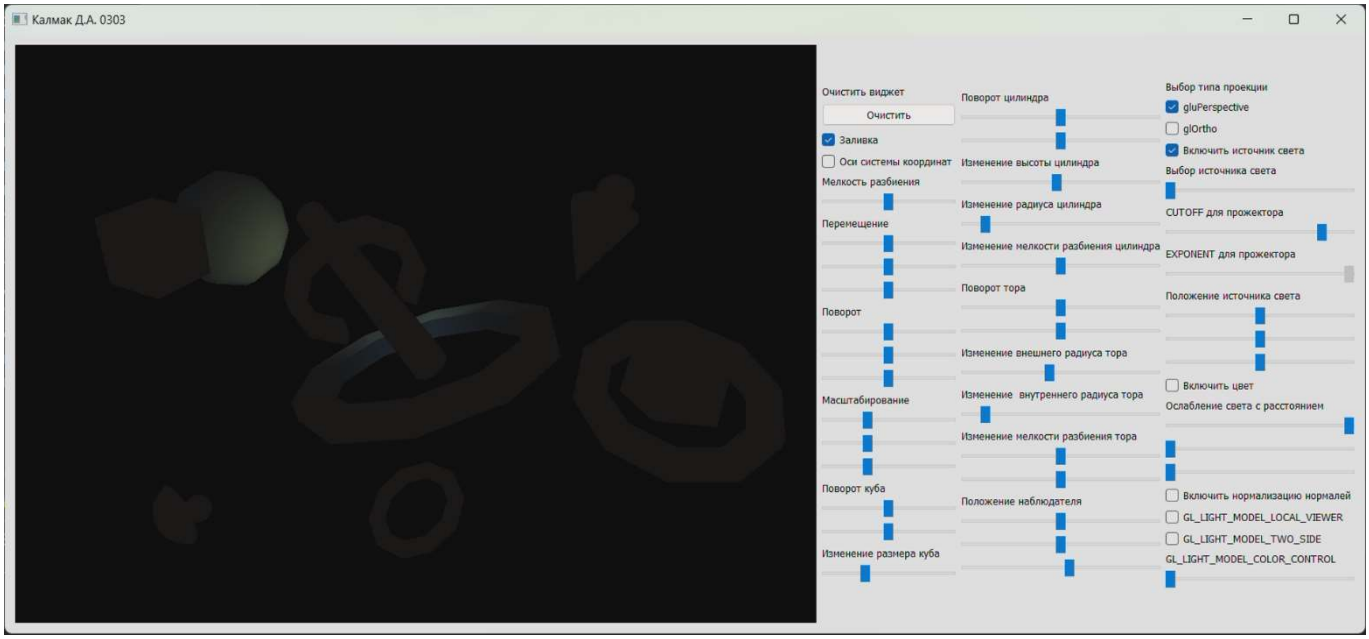

Рисунок 5 – Изменена концентрация света

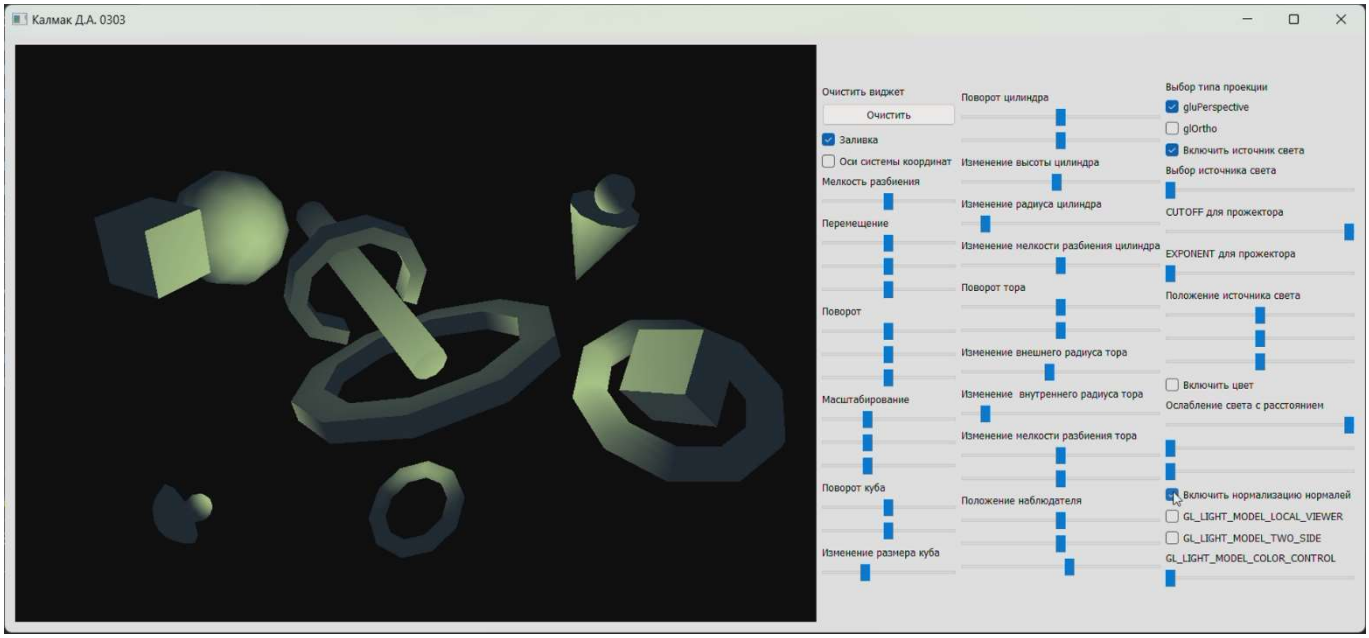

Рисунок 6 – Включена нормализация нормалей

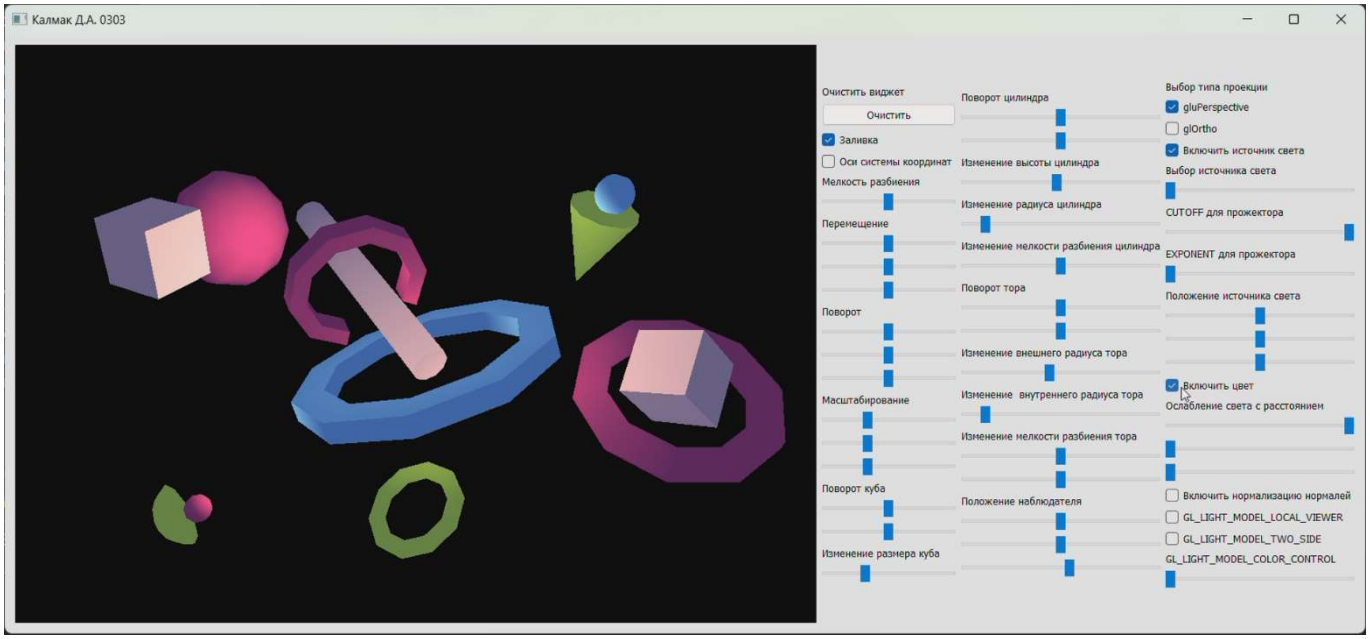

Рисунок 7 – Включен цвет для трехмерной сцены

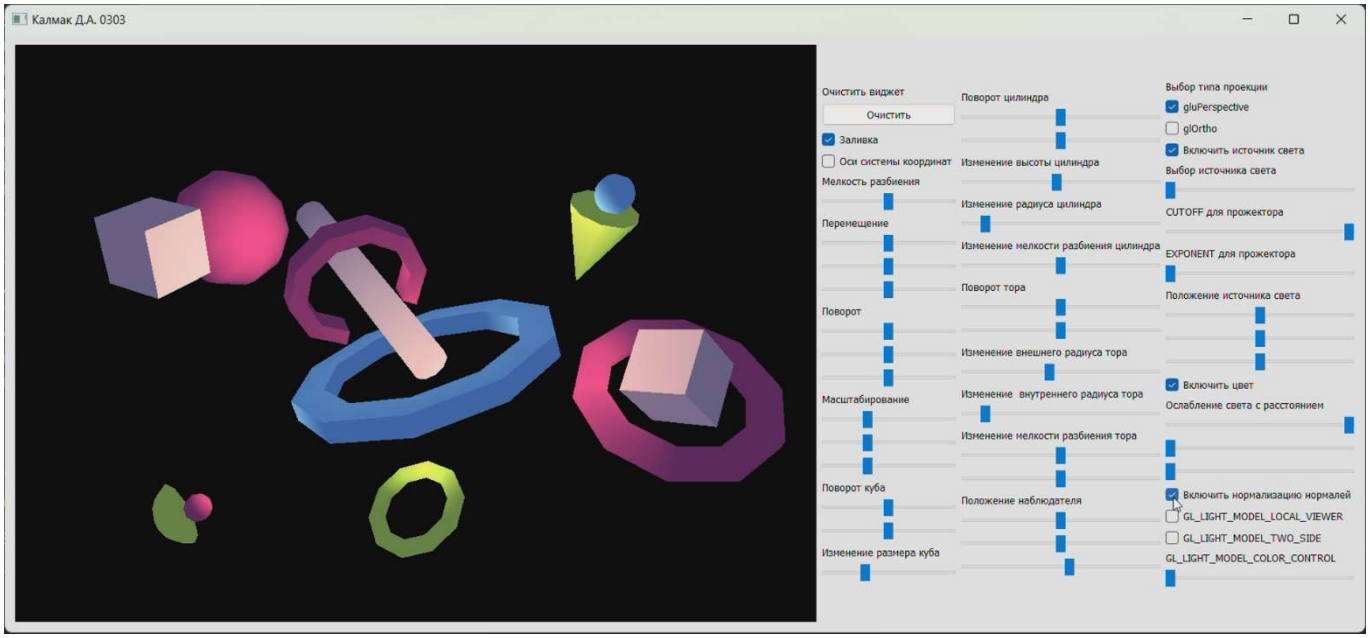

Рисунок 8 – Включена нормализация нормалей

Вывод.

В результате выполнения лабораторной работы была разработана программа, реализующая представление разработанной трехмерной сцены с добавлением возможности формирования различного типа проекций теней.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
import math
import sys
from OpenGL.GL import *
from OpenGL.GLU import *
from PyQt5.QtCore import Qt, QTimer
from PyQt5.QtGui import QOpenGLShaderProgram, QOpenGLShader
from PyQt5.QtOpenGL import *
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import (QWidget, QLabel,
                             QComboBox, QStackedWidget, QSlider, QCheckBox,
QPushButton)


class mainWindow(QWidget):
    def __init__(self, parent=None):
        super(mainWindow, self).__init__()
        self.stack = QStackedWidget()
        self.stack.addWidget(glWidget3d())

        buttonsLayout = QtWidgets.QVBoxLayout()
        self.lblclear = QLabel("Очистить виджет", self)
        self.btnclear = QPushButton("Очистить", self)
        self.btnclear.clicked.connect(self.update_clear)
        self.boxfill = QCheckBox("Заливка", self)
        self.boxfill.stateChanged.connect(self.update_fill)
        self.boxfill.setChecked(True)
        self.boxaxes = QCheckBox("Оси системы координат", self)
        self.boxaxes.stateChanged.connect(self.update_axes)
        self.lblfineness = QLabel("Мелкость разбиения", self)
        self.sliderfineness = QSlider(Qt.Orientation.Horizontal, self)
        self.sliderfineness.setMinimum(5)
        self.sliderfineness.setMaximum(15)
        self.sliderfineness.setValue(10)
        self.sliderfineness.valueChanged.connect(self.update_fineness)

        self.lbltranslate = QLabel("Перемещение", self)
        self.sliderxt = QSlider(Qt.Orientation.Horizontal, self)
        self.sliderxt.setMinimum(-10)
        self.sliderxt.setMaximum(10)
        self.sliderxt.setValue(0)
        self.sliderxt.valueChanged.connect(self.update_xt)
        self.slideryt = QSlider(Qt.Orientation.Horizontal, self)
        self.slideryt.setMinimum(-10)
        self.slideryt.setMaximum(10)
        self.slideryt.setValue(0)
        self.slideryt.valueChanged.connect(self.update_yt)
        self.sliderzt = QSlider(Qt.Orientation.Horizontal, self)
        self.sliderzt.setMinimum(-10)
        self.sliderzt.setMaximum(10)
        self.sliderzt.setValue(0)
        self.sliderzt.valueChanged.connect(self.update_zt)
```

```
self.lblrotate = QLabel("Поворот", self)
self.sliderxr = QSlider(Qt.Orientation.Horizontal, self)
self.sliderxr.setMinimum(-30)
self.sliderxr.setMaximum(30)
self.sliderxr.setValue(0)
self.sliderxr.setSingleStep(5)
self.sliderxr.valueChanged.connect(self.update_xr)
self.slideryr = QSlider(Qt.Orientation.Horizontal, self)
self.slideryr.setMinimum(-30)
self.slideryr.setMaximum(30)
self.slideryr.setValue(0)
self.slideryr.setSingleStep(5)
self.slideryr.valueChanged.connect(self.update_yr)
self.sliderzr = QSlider(Qt.Orientation.Horizontal, self)
self.sliderzr.setMinimum(-30)
self.sliderzr.setMaximum(30)
self.sliderzr.setValue(0)
self.sliderzr.setSingleStep(5)
self.sliderzr.valueChanged.connect(self.update_zr)
self.lblscale = QLabel("Масштабирование", self)
self.sliderxs = QSlider(Qt.Orientation.Horizontal, self)
self.sliderxs.setMinimum(0)
self.sliderxs.setMaximum(30)
self.sliderxs.setValue(10)
self.sliderxs.setSingleStep(5)
self.sliderxs.valueChanged.connect(self.update_xs)
self.sliderys = QSlider(Qt.Orientation.Horizontal, self)
self.sliderys.setMinimum(0)
self.sliderys.setMaximum(30)
self.sliderys.setValue(10)
self.sliderys.setSingleStep(5)
self.sliderys.valueChanged.connect(self.update_ys)
self.sliderzs = QSlider(Qt.Orientation.Horizontal, self)
self.sliderzs.setMinimum(0)
self.sliderzs.setMaximum(30)
self.sliderzs.setValue(10)
self.sliderzs.setSingleStep(5)
self.sliderzs.valueChanged.connect(self.update_zs)

self.lblrotatecube = QLabel("Поворот куба", self)
self.sliderxrcube = QSlider(Qt.Orientation.Horizontal, self)
self.sliderxrcube.setMinimum(-30)
self.sliderxrcube.setMaximum(30)
self.sliderxrcube.setValue(0)
self.sliderxrcube.setSingleStep(5)
self.sliderxrcube.valueChanged.connect(self.update_xrcube)
self.slideryrcube = QSlider(Qt.Orientation.Horizontal, self)
self.slideryrcube.setMinimum(-30)
self.slideryrcube.setMaximum(30)
self.slideryrcube.setValue(0)
self.slideryrcube.setSingleStep(5)
self.slideryrcube.valueChanged.connect(self.update_yrcube)
self.lblscalecube = QLabel("Изменение размера куба", self)
self.sliderxscube = QSlider(Qt.Orientation.Horizontal, self)
self.sliderxscube.setMinimum(1)
self.sliderxscube.setMaximum(30)
self.sliderxscube.setValue(10)
```

```
self.sliderxscube.setSingleStep(5)
self.sliderxscube.valueChanged.connect(self.update_xscube)

buttonsLayout.addStretch()
buttonsLayout.addWidget(self.lblclear)
buttonsLayout.addWidget(self.btnclear)
buttonsLayout.addWidget(self.boxfill)
buttonsLayout.addWidget(self.boxaxes)
buttonsLayout.addWidget(self.lblfineness)
buttonsLayout.addWidget(self.sliderfineness)
buttonsLayout.addWidget(self.lbltranslate)
buttonsLayout.addWidget(self.sliderxt)
buttonsLayout.addWidget(self.slideryt)
buttonsLayout.addWidget(self.sliderzt)
buttonsLayout.addWidget(self.lblrotate)
buttonsLayout.addWidget(self.sliderxr)
buttonsLayout.addWidget(self.slideryr)
buttonsLayout.addWidget(self.sliderzr)
buttonsLayout.addWidget(self.lblscale)
buttonsLayout.addWidget(self.sliderxs)
buttonsLayout.addWidget(self.sliderys)
buttonsLayout.addWidget(self.sliderzs)
buttonsLayout.addWidget(self.lblrotatecube)
buttonsLayout.addWidget(self.sliderxrcube)
buttonsLayout.addWidget(self.slideryrcube)
buttonsLayout.addWidget(self.lblscalecube)
buttonsLayout.addWidget(self.sliderxscube)
buttonsLayout.addStretch()

buttonsLayout2 = QtWidgets.QVBoxLayout()
self.lblrotatecylinder = QLabel("Поворот цилиндра", self)
self.sliderxrcylinder = QSlider(Qt.Orientation.Horizontal, self)
self.sliderxrcylinder.setMinimum(-30)
self.sliderxrcylinder.setMaximum(30)
self.sliderxrcylinder.setValue(0)
self.sliderxrcylinder.setSingleStep(5)
self.sliderxrcylinder.valueChanged.connect(self.update_xrcylinder)
self.slideryrcylinder = QSlider(Qt.Orientation.Horizontal, self)
self.slideryrcylinder.setMinimum(-30)
self.slideryrcylinder.setMaximum(30)
self.slideryrcylinder.setValue(0)
self.slideryrcylinder.setSingleStep(5)
self.slideryrcylinder.valueChanged.connect(self.update_yrcylinder)
self.lblhcylinder = QLabel("Изменение высоты цилиндра", self)
self.sliderhcylinder = QSlider(Qt.Orientation.Horizontal, self)
self.sliderhcylinder.setMinimum(1)
self.sliderhcylinder.setMaximum(30)
self.sliderhcylinder.setValue(15)
self.sliderhcylinder.setSingleStep(5)
self.sliderhcylinder.valueChanged.connect(self.update_hcylinder)
self.lblrcylinder = QLabel("Изменение радиуса цилиндра", self)
self.sliderrcylinder = QSlider(Qt.Orientation.Horizontal, self)
self.sliderrcylinder.setMinimum(0)
self.sliderrcylinder.setMaximum(10)
self.sliderrcylinder.setValue(1)
self.sliderrcylinder.valueChanged.connect(self.update_rcylinder)
self.lblfinenesscylinder = QLabel("Изменение мелкости разбиения цилиндра",
```

```
self)
        self.sliderfinenesscylinder = QSlider(Qt.Orientation.Horizontal, self)
        self.sliderfinenesscylinder.setMinimum(5)
        self.sliderfinenesscylinder.setMaximum(15)
        self.sliderfinenesscylinder.setValue(10)

self.sliderfinenesscylinder.valueChanged.connect(self.update_finenesscylinder)

        self.lblrotatetor = QLabel("Поворот тора", self)
        self.sliderxrtor = QSlider(Qt.Orientation.Horizontal, self)
        self.sliderxrtor.setMinimum(-30)
        self.sliderxrtor.setMaximum(30)
        self.sliderxrtor.setValue(0)
        self.sliderxrtor.setSingleStep(5)
        self.sliderxrtor.valueChanged.connect(self.update_xrtor)
        self.slideryrtor = QSlider(Qt.Orientation.Horizontal, self)
        self.slideryrtor.setMinimum(-30)
        self.slideryrtor.setMaximum(30)
        self.slideryrtor.setValue(0)
        self.slideryrtor.setSingleStep(5)
        self.slideryrtor.valueChanged.connect(self.update_yrtor)
        self.lblrotor = QLabel("Изменение внешнего радиуса тора", self)
        self.sliderrotor = QSlider(Qt.Orientation.Horizontal, self)
        self.sliderrotor.setMinimum(1)
        self.sliderrotor.setMaximum(10)
        self.sliderrotor.setValue(5)
        self.sliderrotor.valueChanged.connect(self.update_rotor)
        self.lblritor = QLabel("Изменение  внутреннего радиуса тора", self)
        self.sliderritor = QSlider(Qt.Orientation.Horizontal, self)
        self.sliderritor.setMinimum(0)
        self.sliderritor.setMaximum(10)
        self.sliderritor.setValue(1)
        self.sliderritor.valueChanged.connect(self.update_ritor)
        self.lblfinenesstor = QLabel("Изменение мелкости разбиения тора", self)
        self.sliderfinenessvtor = QSlider(Qt.Orientation.Horizontal, self)
        self.sliderfinenessvtor.setMinimum(5)
        self.sliderfinenessvtor.setMaximum(15)
        self.sliderfinenessvtor.setValue(10)
        self.sliderfinenessvtor.valueChanged.connect(self.update_finenessvtor)
        self.sliderfinenesshtor = QSlider(Qt.Orientation.Horizontal, self)
        self.sliderfinenesshtor.setMinimum(5)
        self.sliderfinenesshtor.setMaximum(15)
        self.sliderfinenesshtor.setValue(10)
        self.sliderfinenesshtor.valueChanged.connect(self.update_finenesshtor)

        self.lblobserver = QLabel("Положение наблюдателя", self)
        self.sliderxobserver = QSlider(Qt.Orientation.Horizontal, self)
        self.sliderxobserver.setMinimum(-10)
        self.sliderxobserver.setMaximum(10)
        self.sliderxobserver.setValue(0)
        self.sliderxobserver.valueChanged.connect(self.update_xobserver)
        self.slideryobserver = QSlider(Qt.Orientation.Horizontal, self)
        self.slideryobserver.setMinimum(-10)
        self.slideryobserver.setMaximum(10)
        self.slideryobserver.setValue(0)
        self.slideryobserver.valueChanged.connect(self.update_yobserver)
        self.sliderzobserver = QSlider(Qt.Orientation.Horizontal, self)
```

```
self.sliderzobserver.setMinimum(-10)
self.sliderzobserver.setMaximum(10)
self.sliderzobserver.setValue(1)
self.sliderzobserver.valueChanged.connect(self.update_zobserver)

buttonsLayout2.addStretch()
buttonsLayout2.addWidget(self.lblrotatecylinder)
buttonsLayout2.addWidget(self.sliderxrcylinder)
buttonsLayout2.addWidget(self.slideryrcylinder)
buttonsLayout2.addWidget(self.lblhcylinder)
buttonsLayout2.addWidget(self.sliderhcylinder)
buttonsLayout2.addWidget(self.lblrcylinder)
buttonsLayout2.addWidget(self.sliderrcylinder)
buttonsLayout2.addWidget(self.lblfinenesscylinder)
buttonsLayout2.addWidget(self.sliderfinenesscylinder)
buttonsLayout2.addWidget(self.lblrotatetor)
buttonsLayout2.addWidget(self.sliderxrtor)
buttonsLayout2.addWidget(self.slideryrtor)
buttonsLayout2.addWidget(self.lblrotor)
buttonsLayout2.addWidget(self.sliderrotor)
buttonsLayout2.addWidget(self.lblritor)
buttonsLayout2.addWidget(self.sliderritor)
buttonsLayout2.addWidget(self.lblfinenesstor)
buttonsLayout2.addWidget(self.sliderfinenessvtor)
buttonsLayout2.addWidget(self.sliderfinenesshtor)
buttonsLayout2.addWidget(self.lblobserver)
buttonsLayout2.addWidget(self.sliderxobserver)
buttonsLayout2.addWidget(self.slideryobserver)
buttonsLayout2.addWidget(self.sliderzobserver)
buttonsLayout2.addStretch()

buttonsLayout3 = QtWidgets.QVBoxLayout()
self.boxlight_flag = QCheckBox("Включить источник света", self)
self.boxlight_flag.stateChanged.connect(self.update_light_flag)
self.lblprojection = QLabel("Выбор типа проекции", self)
self.boxglupprojection = QCheckBox("gluPerspective", self)
self.boxglupprojection.setChecked(True)
self.boxglupprojection.stateChanged.connect(self.update_glupprojection)
self.boxgloprojection = QCheckBox("glOrtho", self)
self.boxgloprojection.stateChanged.connect(self.update_gloprojection)

self.lblcolor = QLabel("Выбор источника света", self)
self.slidercolor = QSlider(Qt.Orientation.Horizontal, self)
self.slidercolor.setMinimum(0)
self.slidercolor.setMaximum(1)
self.slidercolor.setValue(0)
self.slidercolor.valueChanged.connect(self.update_color)

self.lblcutoff = QLabel("CUTOFF для прожектора", self)
self.slidercutoff = QSlider(Qt.Orientation.Horizontal, self)
self.slidercutoff.setMinimum(0)
self.slidercutoff.setMaximum(91)
self.slidercutoff.setValue(91)
self.slidercutoff.valueChanged.connect(self.update_cutoff)

self.lblexponent = QLabel("EXPONENT для прожектора", self)
self.sliderexponent = QSlider(Qt.Orientation.Horizontal, self)
```

```
self.sliderexponent.setMinimum(0)
self.sliderexponent.setMaximum(5)
self.sliderexponent.setValue(0)
self.sliderexponent.valueChanged.connect(self.update_exponent)

self.lbllight = QLabel("Положение источника света", self)
self.sliderxlight = QSlider(Qt.Orientation.Horizontal, self)
self.sliderxlight.setMinimum(-10)
self.sliderxlight.setMaximum(10)
self.sliderxlight.setValue(0)
self.sliderxlight.valueChanged.connect(self.update_xlight)
self.sliderylight = QSlider(Qt.Orientation.Horizontal, self)
self.sliderylight.setMinimum(-10)
self.sliderylight.setMaximum(10)
self.sliderylight.setValue(0)
self.sliderylight.valueChanged.connect(self.update_ylight)
self.sliderzlight = QSlider(Qt.Orientation.Horizontal, self)
self.sliderzlight.setMinimum(-10)
self.sliderzlight.setMaximum(10)
self.sliderzlight.setValue(0)
self.sliderzlight.valueChanged.connect(self.update_zlight)

self.boxcolor = QCheckBox("Включить цвет", self)
self.boxcolor.stateChanged.connect(self.update_color_flag)

self.lblattenuation = QLabel("Ослабление света с расстоянием", self)
self.slidercattenuation = QSlider(Qt.Orientation.Horizontal, self)
self.slidercattenuation.setMinimum(0)
self.slidercattenuation.setMaximum(10)
self.slidercattenuation.setValue(10)
self.slidercattenuation.valueChanged.connect(self.update_cattenuation)
self.sliderlattenuation = QSlider(Qt.Orientation.Horizontal, self)
self.sliderlattenuation.setMinimum(0)
self.sliderlattenuation.setMaximum(10)
self.sliderlattenuation.setValue(0)
self.sliderlattenuation.valueChanged.connect(self.update_lattenuation)
self.sliderqattenuation = QSlider(Qt.Orientation.Horizontal, self)
self.sliderqattenuation.setMinimum(0)
self.sliderqattenuation.setMaximum(10)
self.sliderqattenuation.setValue(0)
self.sliderqattenuation.valueChanged.connect(self.update_qattenuation)

self.boxnormalize = QCheckBox("Включить нормализацию нормалей", self)
self.boxnormalize.stateChanged.connect(self.update_normalize)

self.boxlocal = QCheckBox("GL_LIGHT_MODEL_LOCAL_VIEWER", self)
self.boxlocal.stateChanged.connect(self.update_local)

self.boxtwo = QCheckBox("GL_LIGHT_MODEL_TWO_SIDE", self)
self.boxtwo.stateChanged.connect(self.update_two)

self.lblcontrol = QLabel("GL_LIGHT_MODEL_COLOR_CONTROL", self)
self.slidercontrol = QSlider(Qt.Orientation.Horizontal, self)
self.slidercontrol.setMinimum(0)
self.slidercontrol.setMaximum(1)
self.slidercontrol.setValue(0)
self.slidercontrol.valueChanged.connect(self.update_control)
```

```
    buttonsLayout3.addStretch()
    buttonsLayout3.addWidget(self.lblprojection)
    buttonsLayout3.addWidget(self.boxglupprojection)
    buttonsLayout3.addWidget(self.boxgloprojection)
    buttonsLayout3.addWidget(self.boxlight_flag)
    buttonsLayout3.addWidget(self.lblcolor)
    buttonsLayout3.addWidget(self.slidercolor)
    buttonsLayout3.addWidget(self.lblcutoff)
    buttonsLayout3.addWidget(self.slidercutoff)
    buttonsLayout3.addWidget(self.lblexponent)
    buttonsLayout3.addWidget(self.sliderexponent)
    buttonsLayout3.addWidget(self.lbllight)
    buttonsLayout3.addWidget(self.sliderxlight)
    buttonsLayout3.addWidget(self.sliderylight)
    buttonsLayout3.addWidget(self.sliderzlight)
    buttonsLayout3.addWidget(self.boxcolor)
    buttonsLayout3.addWidget(self.lblattenuation)
    buttonsLayout3.addWidget(self.slidercattenuation)
    buttonsLayout3.addWidget(self.sliderlattenuation)
    buttonsLayout3.addWidget(self.sliderqattenuation)
    buttonsLayout3.addWidget(self.boxnormalize)
    buttonsLayout3.addWidget(self.boxlocal)
    buttonsLayout3.addWidget(self.boxtwo)
    buttonsLayout3.addWidget(self.lblcontrol)
    buttonsLayout3.addWidget(self.slidercontrol)
    buttonsLayout3.addStretch()

    mainLayout = QtWidgets.QHBoxLayout()
    widgetLayout = QtWidgets.QHBoxLayout()
    widgetLayout.addWidget(self.stack)
    mainLayout.addLayout(widgetLayout)
    mainLayout.addLayout(buttonsLayout)
    mainLayout.addLayout(buttonsLayout2)
    mainLayout.addLayout(buttonsLayout3)
    self.setLayout(mainLayout)
    self.setWindowTitle("Калмак Д.А. 0303")

def update_clear(self):
    for i in range(self.stack.__len__()):
        self.stack.widget(i).clearstatus = True
        self.stack.widget(i).updateGL()

def update_shader(self, state):
    if state == Qt.Checked:
        for i in range(self.stack.__len__()):
            self.stack.widget(i).shader_flag = True
            self.stack.widget(i).updateGL()
    else:
        for i in range(self.stack.__len__()):
            self.stack.widget(i).shader_flag = False
            self.stack.widget(i).updateGL()

def update_fill(self, state):
    if state == Qt.Checked:
        for i in range(self.stack.__len__()):
            self.stack.widget(i).fill_mode = GL_FILL
```

```
            self.stack.widget(i).updateGL()
    else:
        for i in range(self.stack.__len__()):
            self.stack.widget(i).fill_mode = GL_LINE
            self.stack.widget(i).updateGL()

def update_axes(self, state):
    if state == Qt.Checked:
        for i in range(self.stack.__len__()):
            self.stack.widget(i).axes_flag = True
            self.stack.widget(i).updateGL()
    else:
        for i in range(self.stack.__len__()):
            self.stack.widget(i).axes_flag = False
            self.stack.widget(i).updateGL()

def update_fineness(self, value):
    for i in range(self.stack.__len__()):
        self.stack.widget(i).fineness = value
        self.stack.widget(i).updateGL()

def update_xt(self, value):
    for i in range(self.stack.__len__()):
        self.stack.widget(i).xt = value / 10
        self.stack.widget(i).updateGL()

def update_yt(self, value):
    for i in range(self.stack.__len__()):
        self.stack.widget(i).yt = value / 10
        self.stack.widget(i).updateGL()

def update_zt(self, value):
    for i in range(self.stack.__len__()):
        self.stack.widget(i).zt = value / 10
        self.stack.widget(i).updateGL()

def update_xr(self, value):
    for i in range(self.stack.__len__()):
        self.stack.widget(i).xr = value
        self.stack.widget(i).updateGL()

def update_yr(self, value):
    for i in range(self.stack.__len__()):
        self.stack.widget(i).yr = value
        self.stack.widget(i).updateGL()

def update_zr(self, value):
    for i in range(self.stack.__len__()):
        self.stack.widget(i).zr = value
        self.stack.widget(i).updateGL()

def update_xs(self, value):
    for i in range(self.stack.__len__()):
        self.stack.widget(i).xs = value / 10
        self.stack.widget(i).updateGL()

def update_ys(self, value):
```

```
    for i in range(self.stack.__len__()):
        self.stack.widget(i).ys = value / 10
        self.stack.widget(i).updateGL()

def update_zs(self, value):
    for i in range(self.stack.__len__()):
        self.stack.widget(i).zs = value / 10
        self.stack.widget(i).updateGL()

def update_xrcube(self, value):
    for i in range(self.stack.__len__()):
        self.stack.widget(i).xrcube = value
        self.stack.widget(i).updateGL()

def update_yrcube(self, value):
    for i in range(self.stack.__len__()):
        self.stack.widget(i).yrcube = value
        self.stack.widget(i).updateGL()

def update_xscube(self, value):
    for i in range(self.stack.__len__()):
        self.stack.widget(i).xscube = value / 10
        self.stack.widget(i).updateGL()

def update_xrcylinder(self, value):
    for i in range(self.stack.__len__()):
        self.stack.widget(i).xrcylinder = value
        self.stack.widget(i).updateGL()

def update_yrcylinder(self, value):
    for i in range(self.stack.__len__()):
        self.stack.widget(i).yrcylinder = value
        self.stack.widget(i).updateGL()

def update_hcylinder(self, value):
    for i in range(self.stack.__len__()):
        self.stack.widget(i).hcylinder = value / 10
        self.stack.widget(i).updateGL()

def update_rcylinder(self, value):
    for i in range(self.stack.__len__()):
        self.stack.widget(i).rcylinder = value / 10
        self.stack.widget(i).updateGL()

def update_finenesscylinder(self, value):
    for i in range(self.stack.__len__()):
        self.stack.widget(i).finenesscylinder = value
        self.stack.widget(i).updateGL()

def update_xrtor(self, value):
    for i in range(self.stack.__len__()):
        self.stack.widget(i).xrtor = value
        self.stack.widget(i).updateGL()

def update_yrtor(self, value):
    for i in range(self.stack.__len__()):
        self.stack.widget(i).yrtor = value
```

```
            self.stack.widget(i).updateGL()

    def update_rotor(self, value):
        for i in range(self.stack.__len__()):
            self.stack.widget(i).rotor = value / 10
            self.stack.widget(i).updateGL()

    def update_ritor(self, value):
        for i in range(self.stack.__len__()):
            self.stack.widget(i).ritor = value / 10
            self.stack.widget(i).updateGL()

    def update_finenessvtor(self, value):
        for i in range(self.stack.__len__()):
            self.stack.widget(i).finenessvtor = value
            self.stack.widget(i).updateGL()

    def update_finenesshtor(self, value):
        for i in range(self.stack.__len__()):
            self.stack.widget(i).finenesshtor = value
            self.stack.widget(i).updateGL()

    def update_xobserver(self, value):
        for i in range(self.stack.__len__()):
            self.stack.widget(i).xobserver = value / 10
            self.stack.widget(i).updateGL()

    def update_yobserver(self, value):
        for i in range(self.stack.__len__()):
            self.stack.widget(i).yobserver = value / 10
            self.stack.widget(i).updateGL()

    def update_zobserver(self, value):
        for i in range(self.stack.__len__()):
            self.stack.widget(i).zobserver = value / 10
            self.stack.widget(i).updateGL()

    def update_glupprojection(self, state):
        if state == Qt.Checked:
            for i in range(self.stack.__len__()):
                self.stack.widget(i).glupprojection_flag = True
                self.stack.widget(i).gloprojection_flag = False
                self.boxgloprojection.setChecked(False)
                self.stack.widget(i).updateGL()
        else:
            for i in range(self.stack.__len__()):
                self.stack.widget(i).glupprojection_flag = False
                self.stack.widget(i).gloprojection_flag = True
                self.boxgloprojection.setChecked(True)
                self.stack.widget(i).updateGL()

    def update_gloprojection(self, state):
        if state == Qt.Checked:
            for i in range(self.stack.__len__()):
                self.stack.widget(i).gloprojection_flag = True
                self.stack.widget(i).glupprojection_flag = False
                self.boxglupprojection.setChecked(False)
```

```
            self.stack.widget(i).updateGL()
    else:
        for i in range(self.stack.__len__()):
            self.stack.widget(i).gloprojection_flag = False
            self.stack.widget(i).glupprojection_flag = True
            self.boxglupprojection.setChecked(True)
            self.stack.widget(i).updateGL()

def update_color(self, value):
    for i in range(self.stack.__len__()):
        self.stack.widget(i).color = value
        self.stack.widget(i).updateGL()

def update_cutoff(self, value):
    for i in range(self.stack.__len__()):
        self.stack.widget(i).cutoff = value
        self.stack.widget(i).updateGL()

def update_exponent(self, value):
    for i in range(self.stack.__len__()):
        self.stack.widget(i).exponent = value
        self.stack.widget(i).updateGL()

def update_xlight(self, value):
    for i in range(self.stack.__len__()):
        self.stack.widget(i).xlight = value / 10
        self.stack.widget(i).updateGL()

def update_ylight(self, value):
    for i in range(self.stack.__len__()):
        self.stack.widget(i).ylight = value / 10
        self.stack.widget(i).updateGL()

def update_zlight(self, value):
    for i in range(self.stack.__len__()):
        self.stack.widget(i).zlight = value / 10
        self.stack.widget(i).updateGL()

def update_color_flag(self, state):
    if state == Qt.Checked:
        for i in range(self.stack.__len__()):
            self.stack.widget(i).color_flag = True
            self.stack.widget(i).updateGL()
    else:
        for i in range(self.stack.__len__()):
            self.stack.widget(i).color_flag = False
            self.stack.widget(i).updateGL()

def update_cattenuation(self, value):
    for i in range(self.stack.__len__()):
        self.stack.widget(i).cattenuation = value / 10
        self.stack.widget(i).updateGL()

def update_lattenuation(self, value):
    for i in range(self.stack.__len__()):
        self.stack.widget(i).lattenuation = value / 10
        self.stack.widget(i).updateGL()
```

```
    def update_qattenuation(self, value):
        for i in range(self.stack.__len__()):
            self.stack.widget(i).qattenuation = value / 10
            self.stack.widget(i).updateGL()

    def update_light_flag(self, state):
        if state == Qt.Checked:
            for i in range(self.stack.__len__()):
                self.stack.widget(i).light_flag = True
                self.stack.widget(i).updateGL()
        else:
            for i in range(self.stack.__len__()):
                self.stack.widget(i).light_flag = False
                self.stack.widget(i).updateGL()

    def update_normalize(self, state):
        if state == Qt.Checked:
            for i in range(self.stack.__len__()):
                self.stack.widget(i).normalize = True
                self.stack.widget(i).updateGL()
        else:
            for i in range(self.stack.__len__()):
                self.stack.widget(i).normalize = False
                self.stack.widget(i).updateGL()

    def update_local(self, state):
        if state == Qt.Checked:
            for i in range(self.stack.__len__()):
                self.stack.widget(i).local = True
                self.stack.widget(i).updateGL()
        else:
            for i in range(self.stack.__len__()):
                self.stack.widget(i).local = False
                self.stack.widget(i).updateGL()

    def update_two(self, state):
        if state == Qt.Checked:
            for i in range(self.stack.__len__()):
                self.stack.widget(i).two = True
                self.stack.widget(i).updateGL()
        else:
            for i in range(self.stack.__len__()):
                self.stack.widget(i).two = False
                self.stack.widget(i).updateGL()

    def update_control(self, value):
        for i in range(self.stack.__len__()):
            self.stack.widget(i).control = value
            self.stack.widget(i).updateGL()


class glWidget0(QGLWidget):
    def __init__(self, parent=None):
        QGLWidget.__init__(self, parent)
        self.setMinimumSize(1000, 720)
        self.w = 1000
```

```
self.h = 720
self.xy = []
self.clearstatus = False
self.time = 0
self.shader_program = QOpenGLShaderProgram()
self.shader_flag = False
self.fill_mode = GL_LINE
self.axes_flag = False
self.fineness = 10
self.xt = 0
self.yt = 0
self.zt = 0
self.xr = 0
self.yr = 0
self.zr = 0
self.xs = 1
self.ys = 1
self.zs = 1
self.xrcube = 0
self.yrcube = 0
self.xscube = 1
self.xrcylinder = 0
self.yrcylinder = 0
self.hcylinder = 1.5
self.rcylinder = 0.1
self.finenesscylinder = 10
self.xrtor = 0
self.yrtor = 0
self.rotor = 0.5
self.ritor = 0.1
self.finenessvtor = 10
self.finenesshtor = 10
self.xobserver = 0
self.yobserver = 0
self.zobserver = 0.1
a = self.w / self.h
t = math.tan(45 / 2 * math.pi / 180) * 2
self.xcoef = 4 * a * (t / 2)
self.ycoef = 4 * (t / 2)
self.glupprojection_flag = True
self.gloprojection_flag = False
self.color = 0
self.cutoff = 91
self.exponent = 0
self.xlight = 0
self.ylight = 0
self.zlight = 0
self.color_flag = False
self.cattenuation = 1
self.lattenuation = 0
self.qattenuation = 0
self.light_flag = False
self.normalize = False
self.local = False
self.two = False
self.control = 0
```

```
    def initializeGL(self):
        glClearColor(0.0, 0.0, 0.0, 0.1)
        glClearDepth(1.0)
        glDepthFunc(GL_LESS)
        glEnable(GL_DEPTH_TEST)
        glShadeModel(GL_SMOOTH)
        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()
        if self.glupprojection_flag:
            gluPerspective(45.0, 750/720, 0.1, 100.0)
        elif self.gloprojection_flag:
            glOrtho(-self.xcoef, self.xcoef, -self.ycoef, self.ycoef, 0.1, 100.0)
        glMatrixMode(GL_MODELVIEW)
        self.shader_program.addShaderFromSourceFile(QOpenGLShader.Vertex, "v5.vert")
        self.shader_program.addShaderFromSourceFile(QOpenGLShader.Fragment,
"f5.frag")
        self.shader_program.link()

    def paintGL(self):
        pass

    def resizeGL(self, w, h):
        self.w = w
        self.h = h
        glViewport(0, 0, w, h)
        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()
        aspect = w / h
        if self.glupprojection_flag:
            gluPerspective(45.0, aspect, 0.1, 100.0)
        elif self.gloprojection_flag:
            glOrtho(-self.xcoef, self.xcoef, -self.ycoef, self.ycoef, 0.1, 100.0)
        glMatrixMode(GL_MODELVIEW)


class glWidget3d(glWidget0):
    def paintGL(self):
        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()
        aspect = self.w / self.h
        if self.glupprojection_flag:
            gluPerspective(45.0, aspect, 0.1, 100.0)
        elif self.gloprojection_flag:
            glOrtho(-self.xcoef, self.xcoef, -self.ycoef, self.ycoef, 0.1, 100.0)
        glMatrixMode(GL_MODELVIEW)

        if self.light_flag:
            myLightPosition = [self.xlight, self.ylight, self.zlight, 1]
            glLightfv(GL_LIGHT0, GL_POSITION, myLightPosition)
            glEnable(GL_LIGHTING)
            glEnable(GL_LIGHT0)

            if self.color == 0:
                amb0 = [0.2, 0.4, 0.6, 1.0]
                diff0 = [0.8, 0.9, 0.5, 1.0]
                spec0 = [1.0, 0.8, 1.0, 1.0]
            elif self.color == 1:
```

```
                amb0 = [0.0, 0.0, 0.0, 1.0]
                diff0 = [1.0, 1.0, 1.0, 1.0]
                spec0 = [1.0, 1.0, 1.0, 1.0]
            glLightfv(GL_LIGHT0, GL_AMBIENT, amb0)
            glLightfv(GL_LIGHT0, GL_DIFFUSE, diff0)
            glLightfv(GL_LIGHT0, GL_SPECULAR, spec0)

            if self.cutoff == 91:
                glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 180)
            else:
                glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, self.cutoff)
            glLightf(GL_LIGHT0, GL_SPOT_EXPONENT, self.exponent)
            dir = [0, 0, -1]
            glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, dir)

            if self.color_flag == True:
                glEnable(GL_COLOR_MATERIAL)
            else:
                glDisable(GL_COLOR_MATERIAL)

            glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, self.cattenuation)
            glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, self.lattenuation)
            glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, self.qattenuation)

            if self.normalize:
                glEnable(GL_NORMALIZE)
            else:
                glDisable(GL_NORMALIZE)

            glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, self.local)
            glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, self.two)
            if self.control == 0:
                glLightModeli(GL_LIGHT_MODEL_COLOR_CONTROL, GL_SINGLE_COLOR)
            elif self.control == 1:
                glLightModelf(GL_LIGHT_MODEL_COLOR_CONTROL,
GL_SEPARATE_SPECULAR_COLOR)

        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glLoadIdentity()
        glTranslatef(0, 0, -4.0)
        gluLookAt(
            self.xobserver, self.yobserver, self.zobserver,
            0, 0, 0,
            0, 1, 0,
        )
        glTranslatef(0, 0, 0.1)
        glTranslatef(self.xt, self.yt, self.zt)
        glRotatef(self.xr, 1, 0, 0)
        glRotatef(self.yr, 0, 1, 0)
        glRotatef(self.zr, 0, 0, 1)
        glScalef(self.xs, 1, 1)
        glScalef(1, self.ys, 1)
        glScalef(1, 1, self.zs)
        glPushMatrix()
        # glDepthMask(GL_FALSE)
        # glEnable(GL_BLEND)
        # glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)
```

```
if self.axes_flag:
    glLineWidth(2.0)
    glColor4f(1, 0, 0, 1)
    glBegin(GL_LINES)
    glVertex3f(0, 0, 0)
    glVertex3f(1, 0, 0)
    glEnd()
    glColor4f(0, 1, 0, 1)
    glBegin(GL_LINES)
    glVertex3f(0, 0, 0)
    glVertex3f(0, 1, 0)
    glEnd()
    glColor4f(0, 0, 1, 1)
    glBegin(GL_LINES)
    glVertex3f(0, 0, 0)
    glVertex3f(0, 0, 1)
    glEnd()
    glLineWidth(1.0)

# Кубы
# 1
glColor4f(1, 0.6078, 0.6549, 1)
glTranslatef(-1.5, 0.5, 0.0)
glPolygonMode(GL_FRONT_AND_BACK, self.fill_mode)
glRotatef(30, 1, 0, 0)
glRotatef(60, 0, 1, 0)
glRotatef(45, 0, 0, 0)
glRotatef(self.xrcube, 1, 0, 0)
glRotatef(self.yrcube, 0, 1, 0)
glScalef(self.xscube, self.xscube, self.xscube)
self.draw_cube()
glPopMatrix()
# 2
glPushMatrix()
glColor4f(1, 0.6078, 0.6549, 1)
glTranslatef(1.5, -0.25, 0.0)
glPolygonMode(GL_FRONT_AND_BACK, self.fill_mode)
glRotatef(50, 0, -1, 0)
glRotatef(30, -1, 0, 0)
self.draw_cube()
glPopMatrix()

# Сферы
# 1
glPushMatrix()
glColor4f(0.8745, 0.2118, 0.4274, 1)
glTranslatef(-1.6, 0.9, -2)
glPolygonMode(GL_FRONT_AND_BACK, self.fill_mode)
glRotatef(45, 0, 1, 0)
self.draw_sphere(0.5, self.fineness, self.fineness, 1)
glPopMatrix()
# 2
glPushMatrix()
glColor4f(0.5451, 0.6471, 0.8392, 1)
glTranslatef(1.0, 0.7, 0.5)
glPolygonMode(GL_FRONT_AND_BACK, self.fill_mode)
glRotatef(45, 0, 1, 0)
```

```
self.draw_sphere(0.1, self.fineness, self.fineness, 1)
glPopMatrix()
# 3
glPushMatrix()
glColor4f(0.9804, 0.8706, 0.3098, 1)
glTranslatef(-1.3, -1, 0)
glPolygonMode(GL_FRONT_AND_BACK, self.fill_mode)
glRotatef(80, 1, 0, 0)
glRotatef(120, 0, 1, 0)
self.draw_sphere(0.2, self.fineness, self.fineness, 0.5)
glPopMatrix()
# 4
glPushMatrix()
glColor4f(0.8745, 0.2118, 0.4274, 1)
glTranslatef(-1.25, -1, 0)
glPolygonMode(GL_FRONT_AND_BACK, self.fill_mode)
glRotatef(45, 0, 1, 0)
self.draw_sphere(0.08, self.fineness, self.fineness, 1)
glPopMatrix()

# Цилиндр
glPushMatrix()
glColor4f(1, 0.6078, 0.6549, 1)
glTranslatef(-0.3, 0.3, 0)
glPolygonMode(GL_FRONT_AND_BACK, self.fill_mode)
glRotatef(45, 1, 0, 0)
glRotatef(30, 0, 1, 0)
glRotatef(self.xrcylinder, 1, 0, 0)
glRotatef(self.yrcylinder, 0, 1, 0)
if self.fineness != 10:
    self.draw_cylinder(0.1, 1.5, self.fineness)
else:
    self.draw_cylinder(self.rcylinder, self.hcylinder, self.finenesscylinder)
glPopMatrix()

# Конус
glPushMatrix()
glColor4f(0.9804, 0.8706, 0.3098, 1)
glTranslatef(1, 0.5, 0)
glPolygonMode(GL_FRONT_AND_BACK, self.fill_mode)
glRotatef(130, 1, 0, 0)
glRotatef(10, 0, -1, 0)
self.draw_cone(0.2, 0.5, self.fineness)
glPopMatrix()

# Торы
# 1
glPushMatrix()
glTranslatef(0, -1, 0)
glColor4f(0.9804, 0.8706, 0.3098, 1)
glPolygonMode(GL_FRONT_AND_BACK, self.fill_mode)
glRotatef(45, -1, 0, 0)
glRotatef(25, 0, -1, 0)
self.draw_tor(0.25, 0.05, self.fineness, self.fineness)
glPopMatrix()
# 2
glPushMatrix()
```

```
        glTranslatef(1.5, -0.35, 0)
        glColor4f(0.8745, 0.2118, 0.4274, 1)
        glPolygonMode(GL_FRONT_AND_BACK, self.fill_mode)
        glRotatef(45, -1, 0, 0)
        glRotatef(self.xrtor, 1, 0, 0)
        glRotatef(self.yrtor, 0, 1, 0)
        if self.fineness != 10:
            self.draw_tor(0.5, 0.1, self.fineness, self.fineness)
        else:
            self.draw_tor(self.rotor, self.ritor, self.finenessvtor,
self.finenesshtor)
        glPopMatrix()

        # Четырехугольные торы
        # 1
        glPushMatrix()
        glColor4f(0.8745, 0.2118, 0.4274, 1)
        glTranslatef(-0.4, 0.4, 0)
        glPolygonMode(GL_FRONT_AND_BACK, self.fill_mode)
        glRotatef(60, -1, 0, 0)
        glRotatef(30, 0, 1, 0)
        glRotatef(25, 0, 0, 1)
        self.draw_quad_tor(0.125, 0.4, self.fineness, 4, 0.8)
        glPopMatrix()
        # 2
        glPushMatrix()
        glColor4f(0.5451, 0.6471, 0.8392, 1)
        glPolygonMode(GL_FRONT_AND_BACK, self.fill_mode)
        glRotatef(45, 0, 1, 0)
        glRotatef(25, 0, 0, 1)
        self.draw_quad_tor(0.125, 0.8, self.fineness, 4, 1)
        glPopMatrix()

        if self.clearstatus:
            glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
            self.xy = []
            self.clearstatus = False
        # glDepthMask(GL_TRUE)
        # glDisable(GL_BLEND)
        glDisable(GL_LIGHTING)
        glDisable(GL_LIGHT0)

    def mousePressEvent(self, event):
        a = self.w / self.h
        t = math.tan(45 / 2 * math.pi / 180) * 2
        self.xcoef = 4 * a * (t / 2)
        self.ycoef = 4 * (t / 2)
        xpos = (-(self.w / 2) + event.pos().x()) / self.w * 2 * self.xcoef
        ypos = -(-(self.h / 2) + event.pos().y()) / self.h * 2 * self.ycoef
        if len(self.xy) < 7:
            self.xy.append([xpos, ypos, 0])
            # print(len(self.xy))
        self.updateGL()
        super().mousePressEvent(event)

    def draw_cube(self):
        glBegin(GL_QUADS)
```

```
    glNormal3f(0, 0, 1)
    glVertex3f(0.2, 0.2, 0.2)
    glVertex3f(-0.2, 0.2, 0.2)
    glVertex3f(-0.2, -0.2, 0.2)
    glVertex3f(0.2, -0.2, 0.2)
    glEnd()

    glBegin(GL_QUADS)
    glNormal3f(0, 0, -1)
    glVertex3f(0.2, 0.2, -0.2)
    glVertex3f(0.2, -0.2, -0.2)
    glVertex3f(-0.2, -0.2, -0.2),
    glVertex3f(-0.2, 0.2, -0.2)
    glEnd()

    glBegin(GL_QUADS)
    glNormal3f(-1, 0, 0)
    glVertex3f(-0.2, 0.2, -0.2)
    glVertex3f(-0.2, 0.2, 0.2)
    glVertex3f(-0.2, -0.2, 0.2)
    glVertex3f(-0.2, -0.2, -0.2)
    glEnd()

    glBegin(GL_QUADS)
    glNormal3f(1, 0, 0)
    glVertex3f(0.2, 0.2, 0.2)
    glVertex3f(0.2, -0.2, 0.2)
    glVertex3f(0.2, -0.2, -0.2)
    glVertex3f(0.2, 0.2, -0.2)
    glEnd()

    glBegin(GL_QUADS)
    glNormal3f(0, 1, 0)
    glVertex3f(-0.2, 0.2, -0.2)
    glVertex3f(-0.2, 0.2, 0.2)
    glVertex3f(0.2, 0.2, 0.2)
    glVertex3f(0.2, 0.2, -0.2)
    glEnd()

    glBegin(GL_QUADS)
    glNormal3f(0, -1, 0)
    glVertex3f(-0.2, -0.2, -0.2)
    glVertex3f(0.2, -0.2, -0.2)
    glVertex3f(0.2, -0.2, 0.2)
    glVertex3f(-0.2, -0.2, 0.2)
    glEnd()

def draw_sphere(self, r, stacks, slices, part):
    for i in range(0, int((stacks + 1) * part)):
        stack1 = math.pi * (-0.5 + (i - 1) / stacks)
        z1 = math.sin(stack1)
        zr1 = math.cos(stack1)
        stack2 = math.pi * (-0.5 + i / stacks)
        z2 = math.sin(stack2)
        zr2 = math.cos(stack2)
        glBegin(GL_QUAD_STRIP)
        for j in range(0, slices + 1):
```

```
            ang = 2 * math.pi * (j - 1) / slices
            x = math.cos(ang)
            y = math.sin(ang)
            glNormal3f(x * zr1, y * zr1, z1)
            glVertex3f(r * x * zr1, r * y * zr1, r * z1)
            glNormal3f(x * zr2, y * zr2, z2)
            glVertex3f(r * x * zr2, r * y * zr2, r * z2)
        glEnd()

def draw_cylinder(self, r, h, slices):
    coords = []
    for i in range(slices + 1):
        angle = 2 * math.pi * (i / slices)
        x = r * math.cos(angle)
        y = r * math.sin(angle)
        coords.append((x, y))

    glBegin(GL_TRIANGLE_FAN)
    glNormal3f(0, 0, -h / 2)
    glVertex(0, 0, h / 2)
    for (x, y) in coords:
        z = h / 2
        glVertex(x, y, z)
    glEnd()

    glBegin(GL_TRIANGLE_FAN)
    glVertex(0, 0, h / 2)
    for (x, y) in coords:
        z = -h / 2
        glNormal3f(x, y, z)
        glVertex(x, y, z)
    glEnd()

    glBegin(GL_TRIANGLE_STRIP)
    for (x, y) in coords:
        z = h / 2
        glVertex(x, y, z)
        glVertex(x, y, -z)
    glEnd()

def draw_cone(self, r, h, slices):
    coords = []
    for i in range(int(slices) + 1):
        angle = 2 * math.pi * (i / slices)
        x = r * math.cos(angle)
        y = r * math.sin(angle)
        coords.append((x, y))

    glBegin(GL_TRIANGLE_FAN)
    glNormal(0, 0, -h / 2)
    glVertex(0, 0, -h / 2)
    for (x, y) in coords:
        z = -h / 2
        glVertex(x, y, z)
    glEnd()

    glBegin(GL_TRIANGLE_FAN)
```

```
        glNormal(0, 0, h / 2)
        glVertex(0, 0, h / 2)
        for (x, y) in coords:
            z = -h / 2
            glNormal3f(x, y, -z)
            glVertex(x, y, z)
        glEnd()

    def draw_tor(self, ro, ri, stacks, slices):
        for i in range(0, stacks):
            glBegin(GL_QUAD_STRIP)
            for j in range(0, slices+1):
                for k in range(1, -1, -1):
                    s = (i + k) % stacks + 0.5
                    t = j % slices
                    x = (ro + ri * math.cos(s * 2 * math.pi / stacks)) * math.cos(t *
2 * math.pi / slices)
                    y = (ro + ri * math.cos(s * 2 * math.pi / stacks)) * math.sin(t *
2 * math.pi / slices)
                    z = ri * math.sin(s * 2 * math.pi / stacks)
                    glNormal3f(x, y, z)
                    glVertex3f(x, y, z)
            glEnd()

    def draw_quad_tor(self, h, r, slices, r_part, part):
        ri = r / r_part
        glBegin(GL_QUADS)
        for i in range(0, int(slices * part)):
            x = r * math.cos(i * 2 * math.pi / slices)
            y = -h * r_part / 2
            z = r * math.sin(i * 2 * math.pi / slices)
            x1 = (r - ri) * math.cos(i * 2 * math.pi / slices)
            z1 = (r - ri) * math.sin(i * 2 * math.pi / slices)
            x2 = r * math.cos((i + 1) * 2 * math.pi / slices)
            z2 = r * math.sin((i + 1) * 2 * math.pi / slices)
            x3 = (r - ri) * math.cos((i + 1) * 2 * math.pi / slices)
            z3 = (r - ri) * math.sin((i + 1) * 2 * math.pi / slices)
            # лево
            glNormal3f(-1, 0, 0)
            glVertex3f(x, y, z)
            glVertex3f(x, y + h, z)
            glVertex3f(x1, y + h, z1)
            glVertex3f(x1, y, z1)
            # перед
            glNormal3f(0, 0, 1)
            glVertex3f(x, y, z)
            glVertex3f(x2, y, z2)
            glVertex3f(x2, y + h, z2)
            glVertex3f(x, y + h, z)
            # право
            glNormal3f(1, 0, 0)
            glVertex3f(x2, y, z2)
            glVertex3f(x2, y + h, z2)
            glVertex3f(x3, y + h, z3)
            glVertex3f(x3, y, z3)
            # зад
            glNormal3f(0, 0, -1)
```

```
            glVertex3f(x1, y, z1)
            glVertex3f(x1, y + h, z1)
            glVertex3f(x3, y + h, z3)
            glVertex3f(x3, y, z3)
            # низ
            glNormal3f(0, -1, 0)
            glVertex3f(x, y, z)
            glVertex3f(x1, y, z1)
            glVertex3f(x3, y, z3)
            glVertex3f(x2, y, z2)
            # верх
            glNormal3f(0, 1, 0)
            glVertex3f(x, y + h, z)
            glVertex3f(x1, y + h, z1)
            glVertex3f(x3, y + h, z3)
            glVertex3f(x2, y + h, z2)
        glEnd()


if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    qWindow = QtWidgets.QMainWindow()
    window = mainWindow(qWindow)
    window.show()
    sys.exit(app.exec_())
```