Санкт-Петербургский Государственный

Электротехнический Университет

Кафедра МОЭВМ

Задание для лабораторной работы № 6

"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.

- проанализировать полученное задание, выделить информационные объекты и действия;

- разработать программу, реализующую представление трехмерного рисунка.

## Задание.

Разработать программу, реализующую представление разработанного вами трехмерного рисунка, используя предложенные функции библиотеки OpenGL (матрицы видового преобразования, проецирование) и язык GLSL.

## Выполнение работы.

В классе mainWindow в self.stack (QStackedWidget) добавлен виджет класса glWidget3d, который наследуется от glWidget0. Класс содержит методы отрисовки таких фигур, как куб, сфера с регулировкой полноты отрисовки, цилиндр, конус, тор, четырехугольный тор с регулировкой полноты отрисовки. Атрибут self.boxfill пренадлежит классу QCheckBox. Он необходим для управления режимом заливки: включен и выключен. Когда в нем меняется состояние, через метод stateChanged у self.boxfill передается с помощью метода connect состояние в метод self.update\_fill. 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 fill\_mode на GL\_FILL. Атрибут self.fill\_mode был добавлен в класс glWidget0, от которого наследуются класс glWidget3d. Иначе атрибуту присваивается значение GL\_LINE. В обоих случаях происходит обновление виджетов с помощью метода updateGL. Атрибут self.boxaxes пренадлежит классу QCheckBox. Он необходим для управления отображения осей системы координат: включен и выключен. Когда в нем меняется состояние, через метод stateChanged у self.boxaxes передается с помощью метода connect состояние в метод self.update\_axes. 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 axes\_flag на True. Атрибут self.axes\_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. В классе mainWindow добавлен атрибут self.lblfineness, который принадлежит классу QLabel. Он содержит текст для ползунка с регулировкой мелкости разбиения. Ползунок self.sliderfineness 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.sliderfineness. С помощью метода setValue у self.sliderfineness установлено начальное значение. Двигая ползунок, с помощью метода valueChanged у self.sliderfineness, который передает значение ползунка, и метода connect, который связывает ползунок с методом update\_fineness, метод update\_fineness 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.fineness, который добавлен в класс glWidget0, от которого наследуются класс glWidget3d. Происходит обновление виджетов с помощью метода updateGL. Дальнейшие виджеты-ползунки имеют такую же структуру и взаимодействие с методами. Отличительные моменты будут описываться. В классе mainWindow добавлен атрибут self.lbltranslate, который принадлежит классу QLabel. Он содержит текст для трех ползунков с регулировкой перемещения объектов по каждой из осей независимо от остальных. Это ползунки self.sliderxt, self.slideryt и self.sliderzt, которые взаимодействуют с методами self.update\_xt, self.update\_yt и self.update\_zt соответственно, они взаимодействуют с атрибутами self.xt, self.yt и self.zt у glWidget0, от которого наследуется glWidget3d. В классе mainWindow добавлен атрибут self.lblrotate, который принадлежит классу QLabel. Он содержит текст для трех ползунков с регулировкой поворота объектов по каждой из осей независимо от остальных. Это ползунки self.sliderxr, self.slideryr и self.sliderzr, которые взаимодействуют с методами self.update\_xr, self.update\_yr и self.update\_zr соответственно, они взаимодействуют с атрибутами self.xr, self.yr и self.zr у glWidget0, от которого наследуется glWidget3d. Использован метод у ползунков setSingleStep, чтобы задать шаг ползунка. В классе mainWindow добавлен атрибут self.lblscale, который принадлежит классу QLabel. Он содержит текст для трех ползунков с регулировкой масштабирования объектов по каждой из осей независимо от остальных. Это ползунки self.sliderxs, self.sliderys и self.sliderzs, которые взаимодействуют с методами self.update\_xs, self.update\_ys и self.update\_zs соответственно, они взаимодействуют с атрибутами self.xs, self.ys и self.zs у glWidget0, от которого наследуется glWidget3d. Использован метод у ползунков setSingleStep, чтобы задать шаг ползунка. В классе mainWindow добавлен атрибут self.lblrotatecube, который принадлежит классу QLabel. Он содержит текст для двух ползунков с регулировкой поворота куба. Это ползунки self.sliderxrcube и self.slideryrcube, которые взаимодействуют с методами self.update\_xrcube и self.update\_yrcube соответственно, они взаимодействуют с атрибутами self.xrcube и self.yrcube у glWidget0, от которого наследуется glWidget3d. Использован метод у ползунков setSingleStep, чтобы задать шаг ползунка. В классе mainWindow добавлен атрибут self.lblscalecube, который принадлежит классу QLabel. Он содержит текст для ползунка с регулировкой размера куба. Это ползунок self.sliderxscube, который взаимодействуют с методом self.update\_xscube, он взаимодействует с атрибутом self.xscube у glWidget0, от которого наследуется glWidget3d. Использован метод у ползунков setSingleStep, чтобы задать шаг ползунка. Перечисленные виджеты выше добавлены в слой buttonsLayout с помощью метода addWidget у buttonsLayout. Создан еще один слой buttonsLayout2, принадлежащий классу QtWidgets.QVBoxLayout. В классе mainWindow добавлен атрибут self.lblrotatecylinder, который принадлежит классу QLabel. Он содержит текст для двух ползунков с регулировкой поворота цилиндра. Это ползунки self.sliderxrcylinder и self.slideryrcylinder, которые взаимодействуют с методами self.update\_xrcylinder и self.update\_yrcylinder соответственно, они взаимодействуют с атрибутами self.xrcylinder и self.yrcylinder у glWidget0, от которого наследуется glWidget3d. Использован метод у ползунков setSingleStep, чтобы задать шаг ползунка. В классе mainWindow добавлен атрибут self.lblhcylinder, который принадлежит классу QLabel. Он содержит текст для ползунка с регулировкой высоты цилиндра. Это ползунок self.sliderhcylinder, который взаимодействуют с методом self.update\_hcylinder, он взаимодействует с атрибутом self.hcylinder у glWidget0, от которого наследуется glWidget3d. Использован метод у ползунков setSingleStep, чтобы задать шаг ползунка. В классе mainWindow добавлен атрибут self.lblrcylinder, который принадлежит классу QLabel. Он содержит текст для ползунка с регулировкой радиуса цилиндра. Это ползунок self.sliderrcylinder, который взаимодействуют с методом self.update\_rcylinder, он взаимодействует с атрибутом self.rcylinder у glWidget0, от которого наследуется glWidget3d. В классе mainWindow добавлен атрибут self.lblfinenesscylinder, который принадлежит классу QLabel. Он содержит текст для ползунка с регулировкой мелкости разбиения цилиндра. Это ползунок self.sliderfinenesscylinder, который взаимодействуют с методом self.update\_finenesscylinder, он взаимодействует с атрибутом self. finenesscylinder у glWidget0, от которого наследуется glWidget3d. В классе mainWindow добавлен атрибут self.lblrotatetor, который принадлежит классу QLabel. Он содержит текст для двух ползунков с регулировкой поворота тора. Это ползунки self.sliderxrtor и self.slideryrtor, которые взаимодействуют с методами self.update\_xrtor и self.update\_yrtor соответственно, они взаимодействуют с атрибутами self.xrtor и self.yrtor у glWidget0, от которого наследуется glWidget3d. Использован метод у ползунков setSingleStep, чтобы задать шаг ползунка. В классе mainWindow добавлен атрибут self.lblrotor, который принадлежит классу QLabel. Он содержит текст для ползунка с регулировкой внешнего радиуса тора. Это ползунок self.sliderrotor, который взаимодействуют с методом self.update\_rotor, он взаимодействует с атрибутом self.rotor у glWidget0, от которого наследуется glWidget3d. В классе mainWindow добавлен атрибут self.lblritor, который принадлежит классу QLabel. Он содержит текст для ползунка с регулировкой внутреннего радиуса тора. Это ползунок self.sliderritor, который взаимодействуют с методом self.update\_ritor, он взаимодействует с атрибутом self.ritor у glWidget0, от которого наследуется glWidget3d. В классе mainWindow добавлен атрибут self.lblfinenesstor, который принадлежит классу QLabel. Он содержит текст для ползунков с регулировкой мелкости разбиения тора. Это ползунки self.sliderfinenessvtor и self.sliderfinenesshtor, который взаимодействуют с методами self.update\_sliderfinenessvtor и self.update\_sliderfinenesshtor, он взаимодействует с атрибутами self.finenessvtor и self.finenesshtor у glWidget0, от которого наследуется glWidget3d. В классе mainWindow добавлен атрибут self.lblobserver, который принадлежит классу QLabel. Он содержит текст для трех ползунков с регулировкой положения наблюдателя. Это ползунки self.sliderxobserver, self.slideryobserver и self.sliderzobserver, которые взаимодействуют с методами self.update\_xobserver, self.update\_yobserver и self.update\_zobserver соответственно, они взаимодействуют с атрибутами self.xobserver, self.yobserver и self.zobserver у glWidget0, от которого наследуется glWidget3d. Перечисленные виджеты выше добавлены в слой buttonsLayout2 с помощью метода addWidget у buttonsLayout2. Слой buttonsLayout2 добавлен в слой mainLayout с помощью метода addLayout у mainLayout.

В классе glWidget3d в методе paintGL задается положение наблюдателя с помощью функции gluLookAt, в котором меняются параметры self.xobserver, self.yobserver и self.zobserver ползунками. С помощью функции glTranslatef осуществляется перемещение объектов по каждой из осей, в ней используются параметры self.xt, self.yt и self.zt, которые меняются ползунками. С помощью функций glRotatef осуществляется поворот объектов по каждой из осей, в ней используются параметры self.xr, self.yr и self.zr, которые меняются ползунками. С помощью функций glScalef осуществляется масштабирование объектов по каждой из осей, в ней используются параметры self.xs, self.ys и self.zs, которые меняются ползунками. Для сохранения и восстановления матриц используются функции glPushMatrix и glPopMatrix. Если self.axes\_flag равен True, то с помощью функции glBegin(GL\_LINES) отображаются оси системы координат. Для кубов настроены цвета, начальное положение с помощью функций glTranslatef и glRotatef, glPolygoneMode, который регулируется атрибутом self.fill\_mode, у которого значение меняется формой. Повороты и изменение размера от ползунков реализуются с помощью функций glRotatef и glScalef, которые управляются атрибутами self.xrcube, self.yrcube, self.xscube, self.xscube, self.xscube. Отрисовка куба происходит с помощью метода self.draw\_cube у класса glWidget3d. В методе используются функции glBegin(GL\_QUADS) для шести граней куба. Для сфер настроены цвета, начальное положение с помощью функций glTranslatef и glRotatef, glPolygoneMode, который регулируется атрибутом self.fill\_mode, у которого значение меняется формой. Отрисовка сферы происходит с помощью метода self.draw\_sphere у класса glWidget3d. Метод принимает радиус сферы, мелкость разбиения stacks и slices от ползунка, который управляет атрибутом self.fineness, полнота отрисовки. В методе для сферы запускаются два цикла: первый по stacks, причем домноженный на полноту отрисовки, и второй вложенный по slices. Для соединения кусочков сферы используется функция glBegin(GL\_QUAD\_STRIP). Для цилиндра настроен цвет, начальное положение с помощью функций glTranslatef и glRotatef, glPolygoneMode, который регулируется атрибутом self.fill\_mode, у которого значение меняется формой. Повороты от ползунков реализуются с помощью функций glRotatef, которые управляются атрибутами self.xrcylinder и self.yrcylinder. Изменение радиуса, высоты и мелкости разбиения цилиндра регулируется ползунками, которые меняют параметры self.rcylinder, self.hcylinder, self.finenesscylinder, которые передаются в функцию отрисовки цилиндра, либо self.fineness, регулирующийся ползунком. Отрисовка цилиндра происходит с помощью метода self.draw\_cylinder у класса glWidget3d. Метод принимает радиус, высоту, мелкость разбиения slices. В методе для цилиндра запускается цикл для вычисления координат. Для двух концов используется функция glBegin(GL\_TRIANGLE\_FAN), а для тубуса glBegin(GL\_TRIANGLE\_STRIP). Для конуса настроен цвет, начальное положение с помощью функций glTranslatef и glRotatef, glPolygoneMode, который регулируется атрибутом self.fill\_mode, у которого значение меняется формой. Отрисовка цилиндра происходит с помощью метода self.draw\_cylinder у класса glWidget3d. Метод принимает радиус, высоту, мелкость разбиения slices, которая регулируется ползунком, который меняет атрибут self.fineness. В методе для конуса запускается цикл для вычисления координат. Для отрисовки используется функция glBegin(GL\_TRIANGLE\_FAN). Для торов настроен цвет, начальное положение с помощью функций glTranslatef и glRotatef, glPolygoneMode, который регулируется атрибутом self.fill\_mode, у которого значение меняется формой. Повороты от ползунков реализуются с помощью функций glRotatef, которые управляются атрибутами self.xrtor и self.yrtor. Изменение внешнего радиуса, внутреннего радииуса и мелкости разбиения тора stacks и slices регулируется ползунками, которые меняют параметры self.rotor, self.ritor, self.finenessvtor и self.finenesshtor, которые передаются в функцию отрисовки тора, либо self.fineness, регулирующийся ползунком. Отрисовка тора происходит с помощью метода self.draw\_tor у класса glWidget3d. Метод принимает внешний радиус, внутренний радиус, мелкость разбиения stacks и slices. В методе для тора запускается два цикла: первый по stacks, второй вложенный по slices. Для соединения кусочков тора используется функция glBegin(GL\_QUAD\_STRIP). Для четырехугольных торов настроен цвет, начальное положение с помощью функций glTranslatef и glRotatef, glPolygoneMode, который регулируется атрибутом self.fill\_mode, у которого значение меняется формой. Изменение мелкости разбиения четырехугольного тора slices регулируется ползунком, который меняет параметр self.fineness, который передается в функцию отрисовки четырехугольного тора. Отрисовка четырехугольного тора происходит с помощью метода self.draw\_quad\_tor у класса glWidget3d. Метод принимает высоту, внешний радиус, мелкость разбиения slices, часть для внутреннего радиуса, полнота отрисовки. В методе для тора для отрисовки используется функция glBegin(GL\_QUADS), чтобы отрисовывать по шесть граней.

## Тестирование.

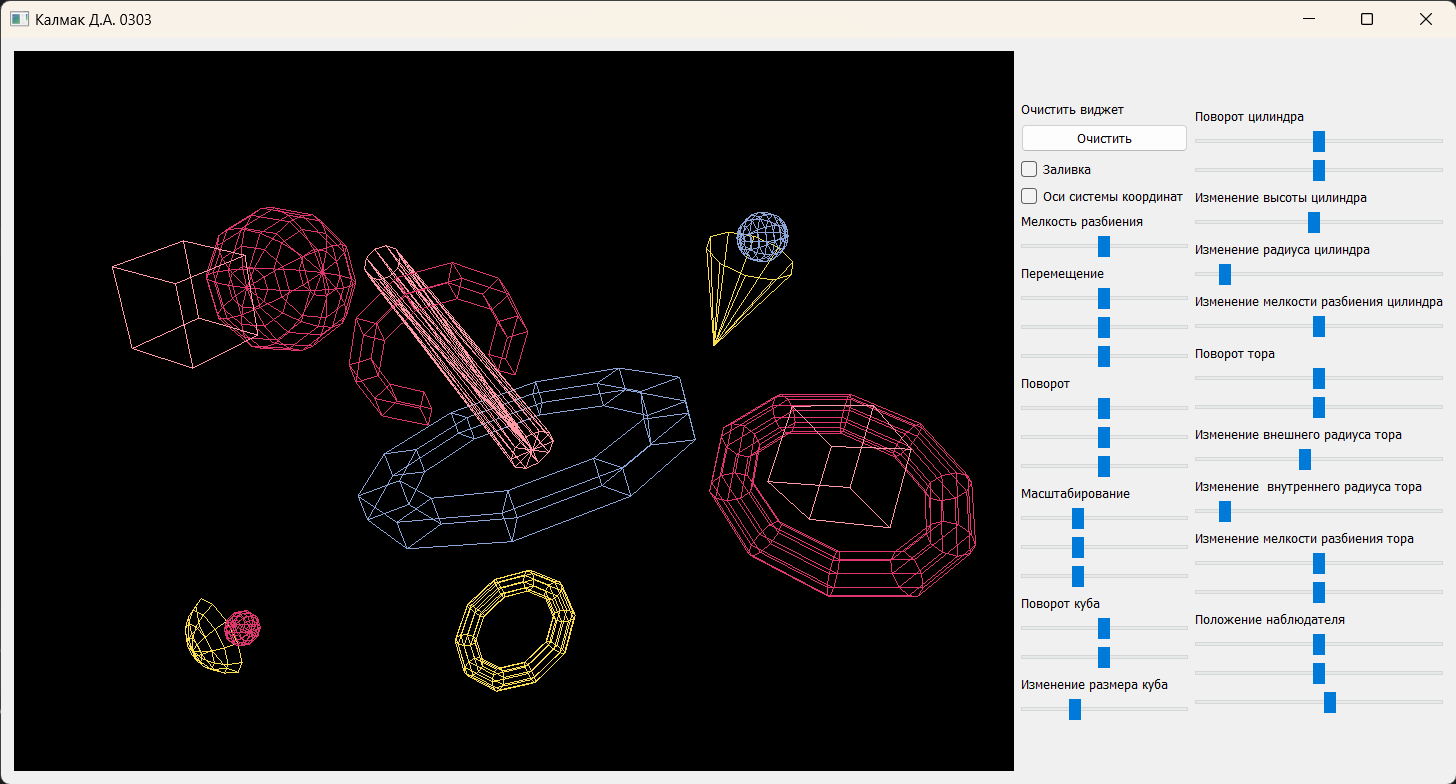


Рисунок 1 – Начальное состояние трехмерного рисунка

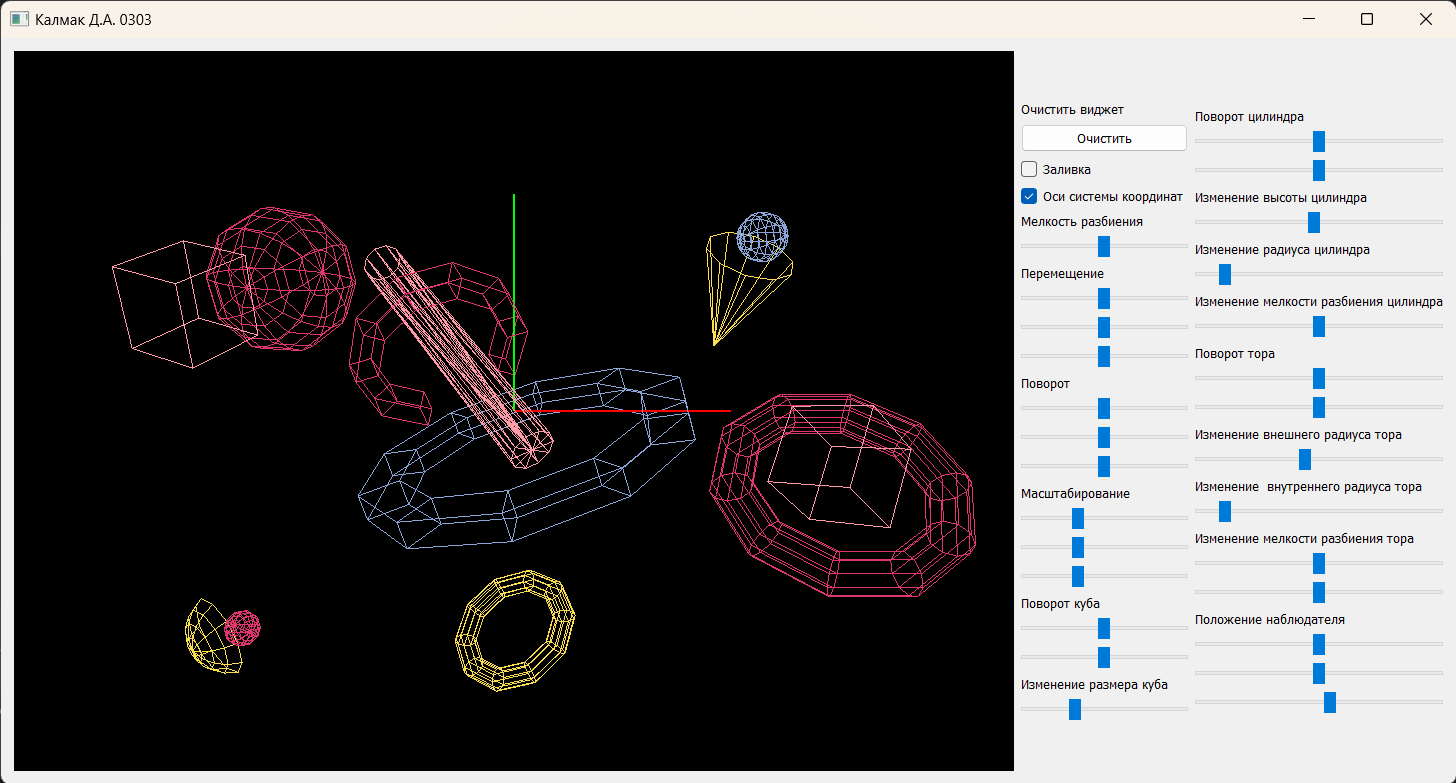


Рисунок 2 – Показаны оси системы координат

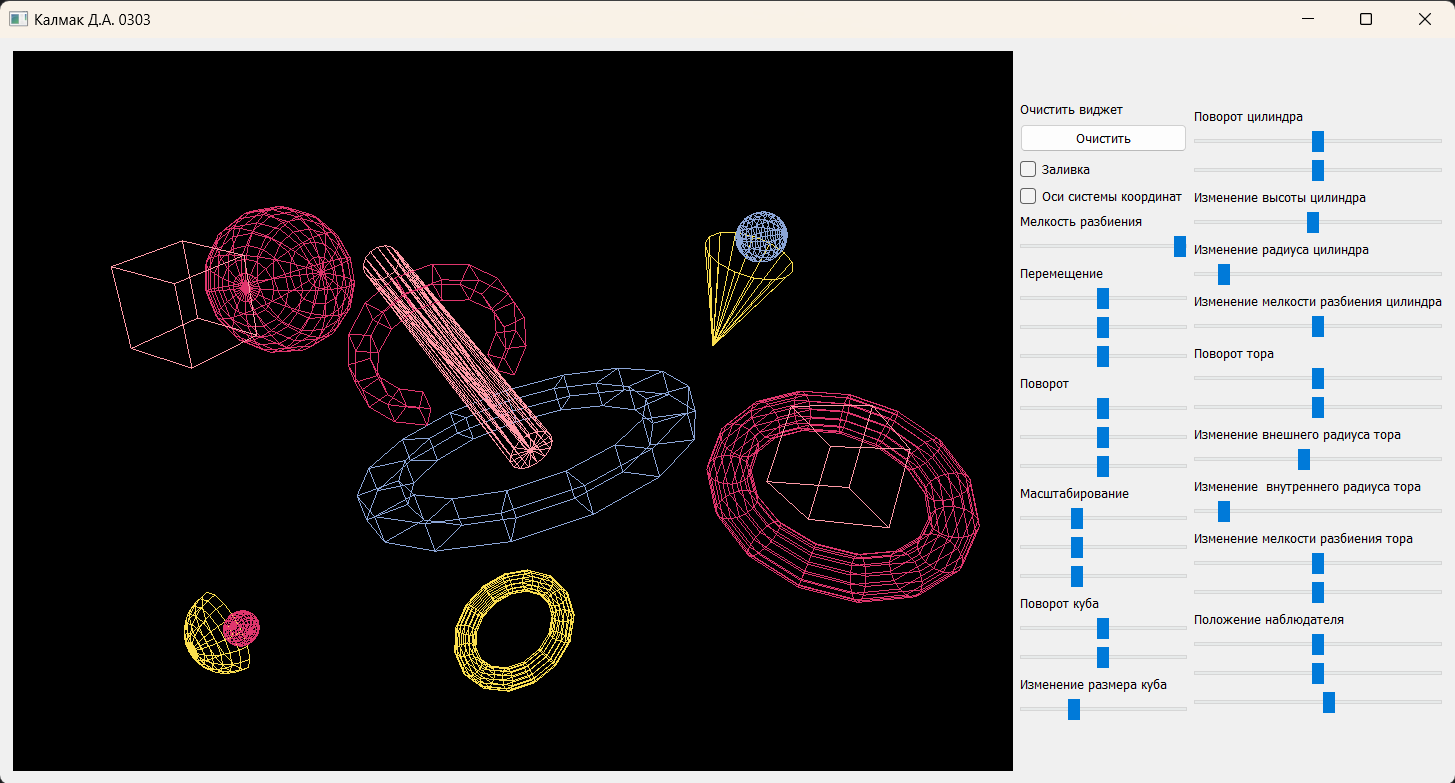


Рисунок 3 – Увеличена мелкость разбиения

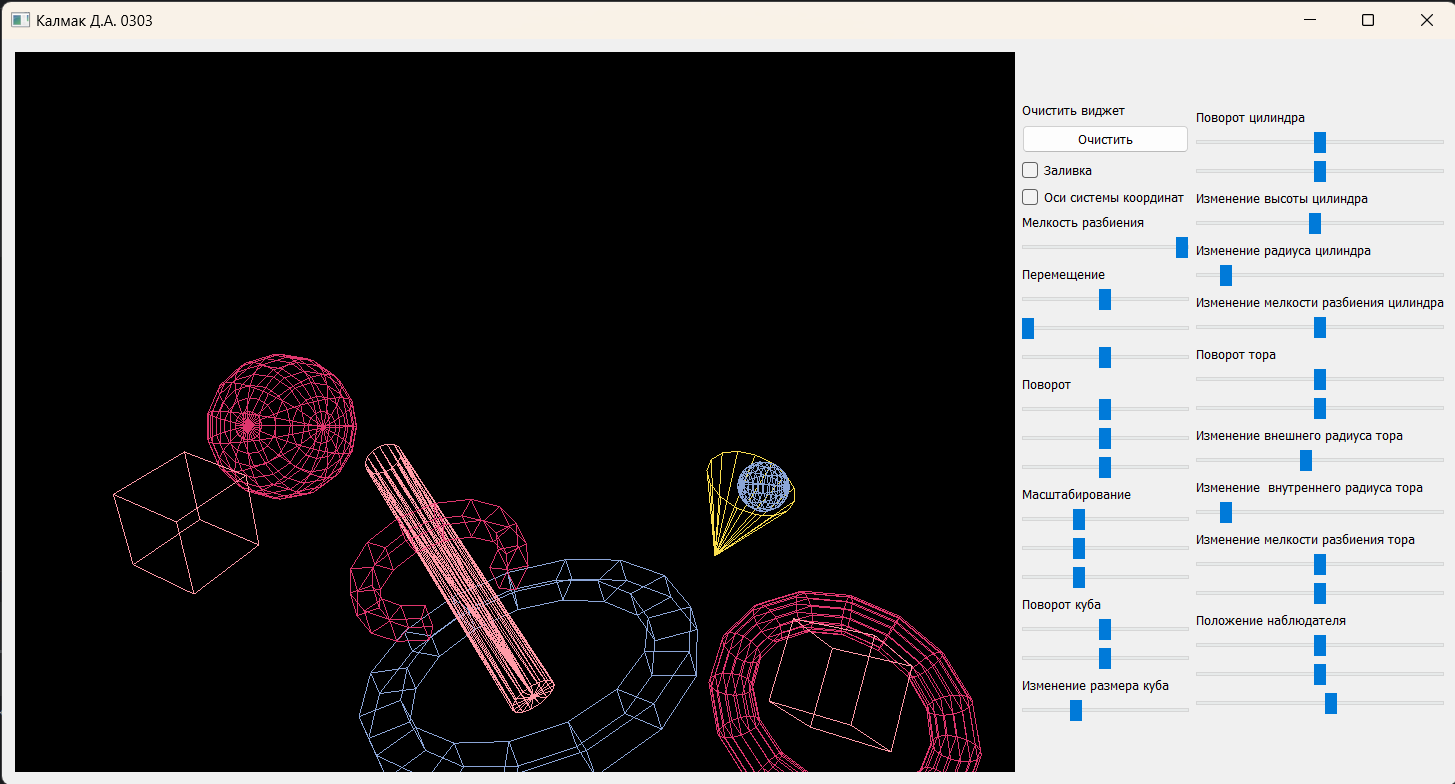


Рисунок 4 – Перемещение объектов по оси y

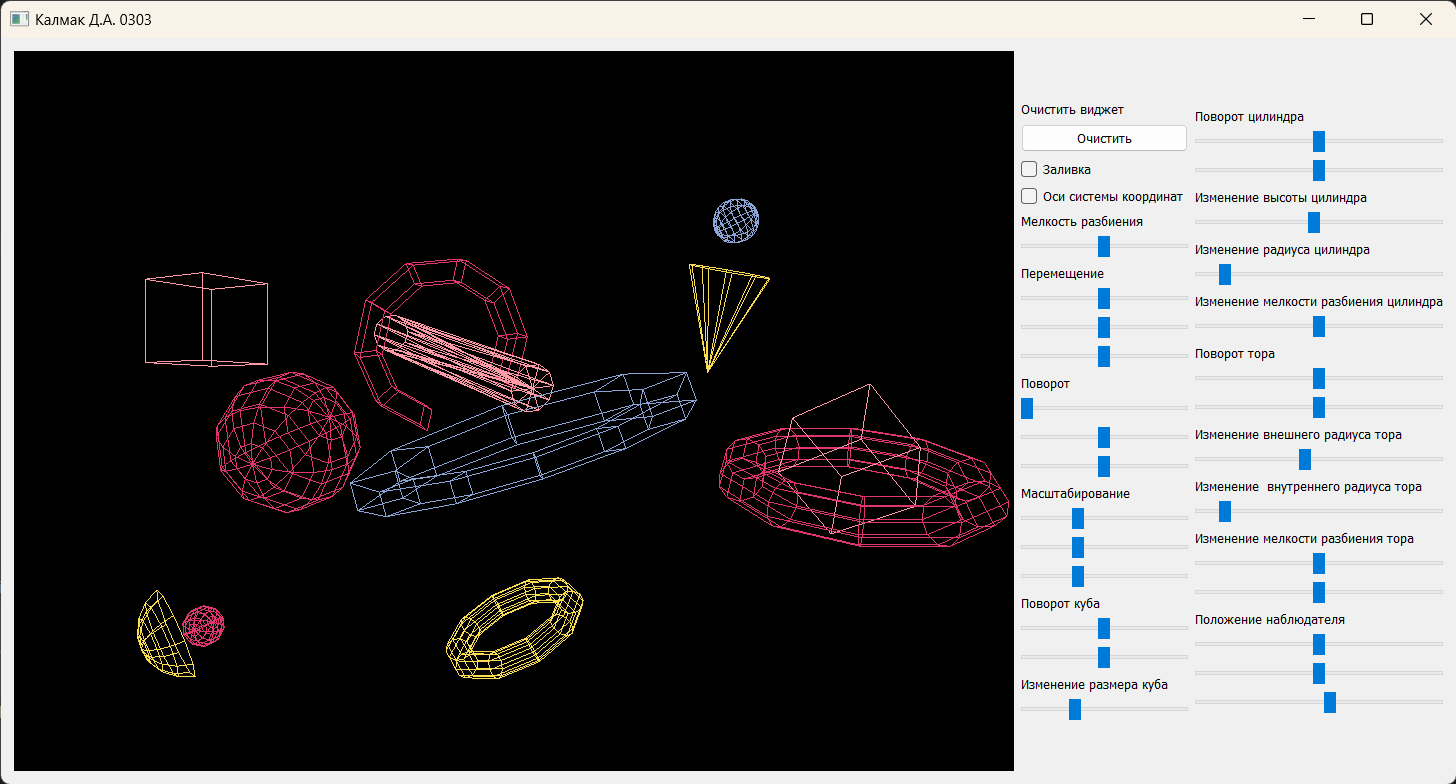


Рисунок 5 – Поворот объектов по оси x

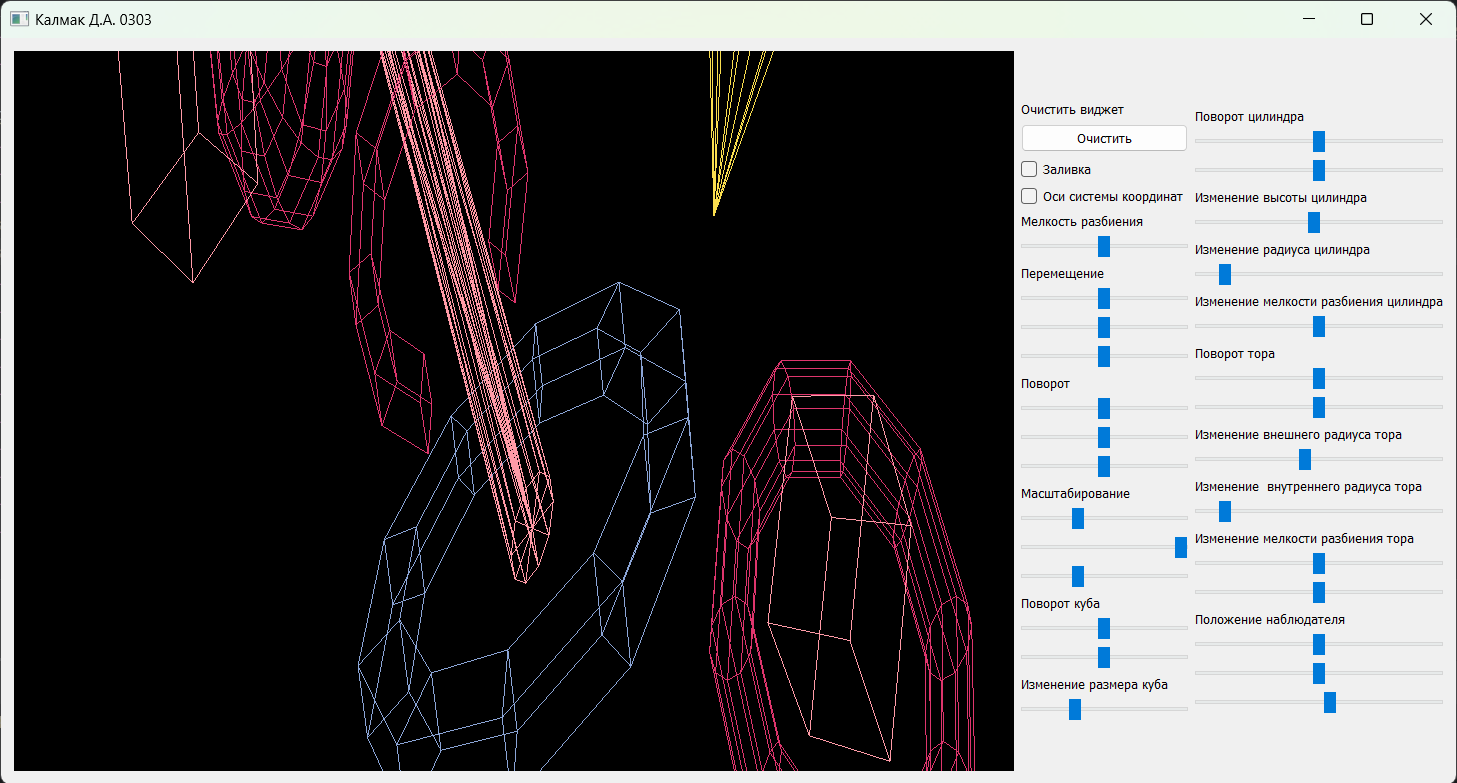


Рисунок 6 – Масштабирование по оси y

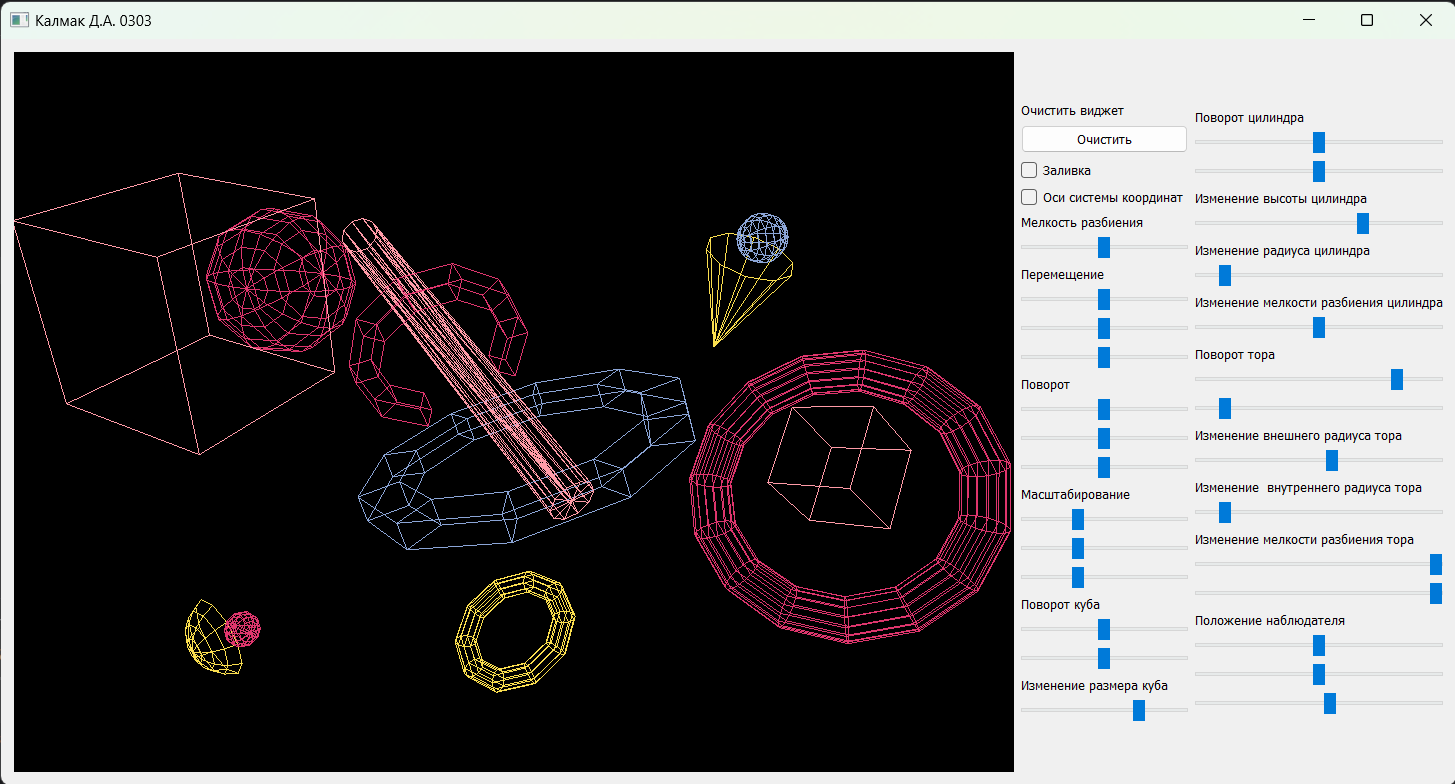


Рисунок 7 – Увеличен размер куба, изменена высота цилиндра, повернут тор, увеличен внешний радиус тора, изменена мелкость разбиения тора

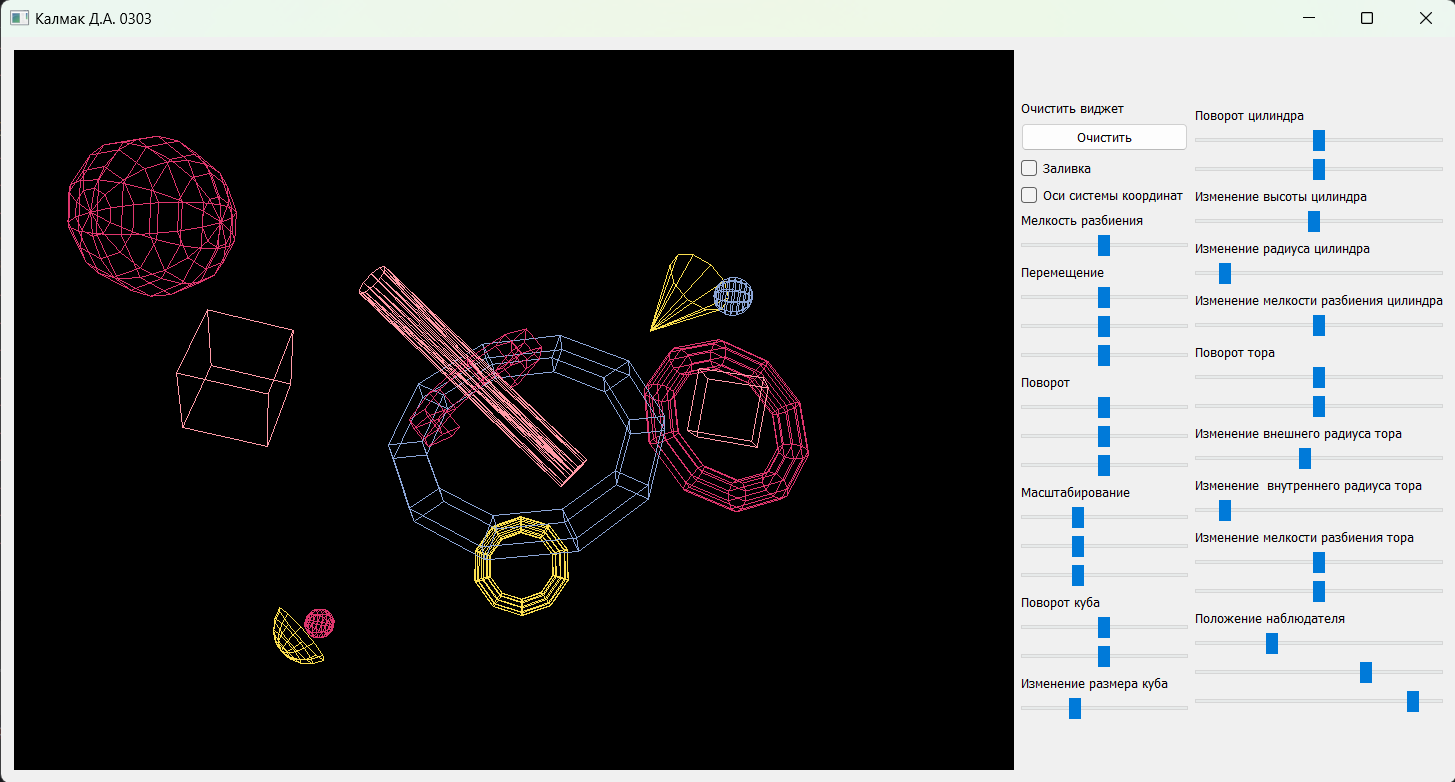


Рисунок 8 – Изменено положение наблюдателя

## Вывод.

В результате выполнения лабораторной работы была разработана программа, реализующая представление трехмерного рисунка.

## 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.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()  
  
 mainLayout = QtWidgets.QHBoxLayout()  
 widgetLayout = QtWidgets.QHBoxLayout()  
 widgetLayout.addWidget(self.stack)  
 mainLayout.addLayout(widgetLayout)  
 mainLayout.addLayout(buttonsLayout)  
 mainLayout.addLayout(buttonsLayout2)  
 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()  
  
  
class glWidget0(QGLWidget):  
 def \_\_init\_\_(self, parent=None):  
 QGLWidget.\_\_init\_\_(self, parent)  
 self.setMinimumSize(1000, 720)  
 self.w = 480  
 self.h = 480  
 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  
  
 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()  
 gluPerspective(45.0, 750/720, 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  
 gluPerspective(45.0, aspect, 0.1, 100)  
 glMatrixMode(GL\_MODELVIEW)  
  
  
class glWidget3d(glWidget0):  
 def paintGL(self):  
 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)  
  
 def mousePressEvent(self, event):  
 a = self.w / self.h  
 t = math.tan(45 / 2 \* math.pi / 180) \* 2  
 xcoef = 4 \* a \* (t / 2)  
 ycoef = 4 \* (t / 2)  
 xpos = (-(self.w / 2) + event.pos().x()) / self.w \* 2 \* xcoef  
 ypos = -(-(self.h / 2) + event.pos().y()) / self.h \* 2 \* 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)  
 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)  
 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)  
 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)  
 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)  
 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)  
 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)  
 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  
 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)  
 # 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  
 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)  
 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)  
  
 glVertex3f(x, y, z)  
 glVertex3f(x, y + h, z)  
 glVertex3f(x1, y + h, z1)  
 glVertex3f(x1, y, z1)  
  
 glVertex3f(x, y, z)  
 glVertex3f(x2, y, z2)  
 glVertex3f(x2, y + h, z2)  
 glVertex3f(x, y + h, z)  
  
 glVertex3f(x2, y, z2)  
 glVertex3f(x2, y + h, z2)  
 glVertex3f(x3, y + h, z3)  
 glVertex3f(x3, y, z3)  
  
 glVertex3f(x1, y, z1)  
 glVertex3f(x1, y + h, z1)  
 glVertex3f(x3, y + h, z3)  
 glVertex3f(x3, y, z3)  
  
 glVertex3f(x, y, z)  
 glVertex3f(x1, y, z1)  
 glVertex3f(x3, y, z3)  
 glVertex3f(x2, y, z2)  
  
 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\_())