Санкт-Петербургский Государственный

Электротехнический Университет

Кафедра МОЭВМ

Задание для лабораторной работы № 1

"Примитивы OpenGL"

| Студенты гр. 1384 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Усачева Д.В.  Пчелинцева К.Р. |
| --- | --- | --- |
| Преподаватель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Герасимова Т.В. |

Санкт-Петербург

2024 г.

**Цель, требования и рекомендации к выполнению задания**

* ознакомление с основными примитивами OpenGL.
* освоение возможности подключения графической библиотеки в среду разработки.

Требования и рекомендации к выполнению задания:

* проанализировать полученное задание, выделить информационные объекты и действия;
* разработать программу с использованием требуемых примитивов и атрибутов.

**Задание**

Разработать программу, реализующую представление определенного набора примитивов из имеющихся в библиотеке OpenGL (GL\_POINT, GL\_LINES, GL\_LINE\_STRIP, GL\_LINE\_LOOP, GL\_TRIANGLES, GL\_TRIANGLE\_STRIP, GL\_TRIANGLE\_FAN, GL\_QUADS, GL\_QUAD\_STRIP, GL\_POLYGON).

Разработанная на базе шаблона программа должна быть пополнена возможностями остановки интерактивно различных атрибутов примитивов рисования через вызов соответствующих элементов интерфейса пользователя

**Общие сведения**

В данной лабораторной работе должны быть рассмотрены следующие примитивы:

*GL\_POINTS* – каждая вершина рассматривается как отдельная точка, параметры которой не зависят от параметров остальных заданных точек. При этом вершина n определяет точку n. Рисуется N точек (n – номер текущей вершины, N – общее число вершин).

Основой графики OpenGL являются вершины. Для их определения используется команда glVertex.

void glVertex[2 3 4][s i f d](type coord)

Вызов команды определяется четырьмя координатами x, y, z и w. При этом вызов glVertex2\* устанавливает координаты x и y, координата z полагается равной 0, а w – 1. Вызов glVertex3\* устанавливает координаты x, y, z, а w равно 1.

*GL\_LINES* – каждая пара вершин рассматривается как независимый отрезок. Первые две вершины определяют первый отрезок, следующие две – второй отрезок и т.д., вершины (2n-1) и 2n определяют отрезок n. Всего рисуется N/2 линий. Если число вершин нечетно, то последняя просто игнорируется.

*GL\_LINE\_STRIP* – в этом режиме рисуется последовательность из одного или нескольких связанных отрезков. Первая вершина задает начало первого отрезка, а вторая – конец первого, который является также началом второго. В общем случае, вершина n (n > 1) определяет начало отрезка n и конец отрезка (n - 1). Всего рисуется (N - 1) отрезок.

*GL\_LINE\_LOOP* – осуществляется рисование замкнутой кривой линии. Первая вершина задает начало первого отрезка, а вторая – конец первого, который является также началом второго. В общем случае, вершина n (n > 1) определяет начало отрезка n и конец отрезка (n - 1). Первая вершина является концом последнего отрезка. Всего рисуется N отрезков.

*GL\_TRIANGLES* – каждая тройка вершин рассматривается как независимый треугольник. Вершины (3n-2), (3n-1), 3n (в таком порядке) определяют треугольник n. Если число вершин не кратно 3, то оставшиеся ( одна или две) вершины игнорируются. Всего рисуется N/3 треугольника.

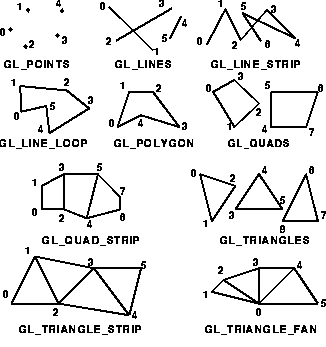
*GL\_TRIANGLE\_STRIP* - в этом режиме рисуется группа связанных треугольников, имеющих общую грань. Первые три вершины определяют первый треугольник, вторая, третья и четвертая – второй и т.д. для нечетного n вершины n, (n+1) и (n+2) определяют треугольник n. Для четного n треугольник определяют вершины (n+1), n и (n+2). Всего рисуется (N-2) треугольника.

*GL\_TRIANGLE\_FAN* - в этом режиме рисуется группа связанных треугольников, имеющих общие грани и одну общую вершину. Первые три вершины определяют первый треугольник, первая, третья и четвертая – второй и т.д. Всего рисуется (N-2) треугольника.

*GL\_QUADS* – каждая группа из четырех вершин рассматривается как независимый четырехугольник. Вершины (4n-3), (4n-2), (4n-1) и 4n определяют четырехугольник n. Если число вершин не кратно 4, то оставшиеся (одна, две или три) вершины игнорируются. Всего рисуется N/4 четырехугольника.

*GL\_QUAD\_STRIP* – рисуется группа четырехугольников, имеющих общую грань. Первая группа из четырех вершин задает первый четырехугольник. Третья, четвертая, пятая и шестая задают второй четырехугольник.

*GL\_POLYGON –* задет многоугольник. При этом число вершин равно числу вершин рисуемого многоугольника.



GL\_FRONT - для лицевых граней, GL\_BACK - для обратных граней, GL\_FRONT\_AND\_BACK - для всех граней. Параметр mode может быть равен:

GL\_POINT при таком режиме будут отображаться только вершины многоугольников.

GL\_LINE при таком режиме многоугольник будет представляться набором отрезков.

GL\_FILL при таком режиме многоугольники будут закрашиваться текущим цветом с учетом освещения, и этот режим установлен по умолчанию.

**Выполнение работы**

Данная работа выполнена на операционной системе Windows 11. Приложение было создано на Python 3.11 с применением библиотеки OpenGL, которая была установлена через клонирование репозитория с кодом PyOpenGL, а также с помощью команды pip install в среде разработки PyCharm. Для создания интерфейса использовалась библиотека PyQt6. Интерфейс был разработан интерактивно в программе QtDesigner, после чего файл формата .ui был конвертирован в .py с помощью команды pyuic6 input.ui -o design.py.

В программе была реализована демонстрация основных примитивов (GL\_POINT, GL\_LINES, GL\_LINE\_STRIP, GL\_LINE\_LOOP, GL\_TRIANGLES, GL\_TRIANGLE\_STRIP, GL\_TRIANGLE\_FAN, GL\_QUADS, GL\_QUAD\_STRIP, GL\_POLYGON). В объекте combobox в приложении предоставлен выбор примитивов, которые могут быть отображены в OpenGL окне.

**Тестирование**

Представления результатов тестирования представлены на снимках экрана.



Рисунок 1 — GL\_POINTS

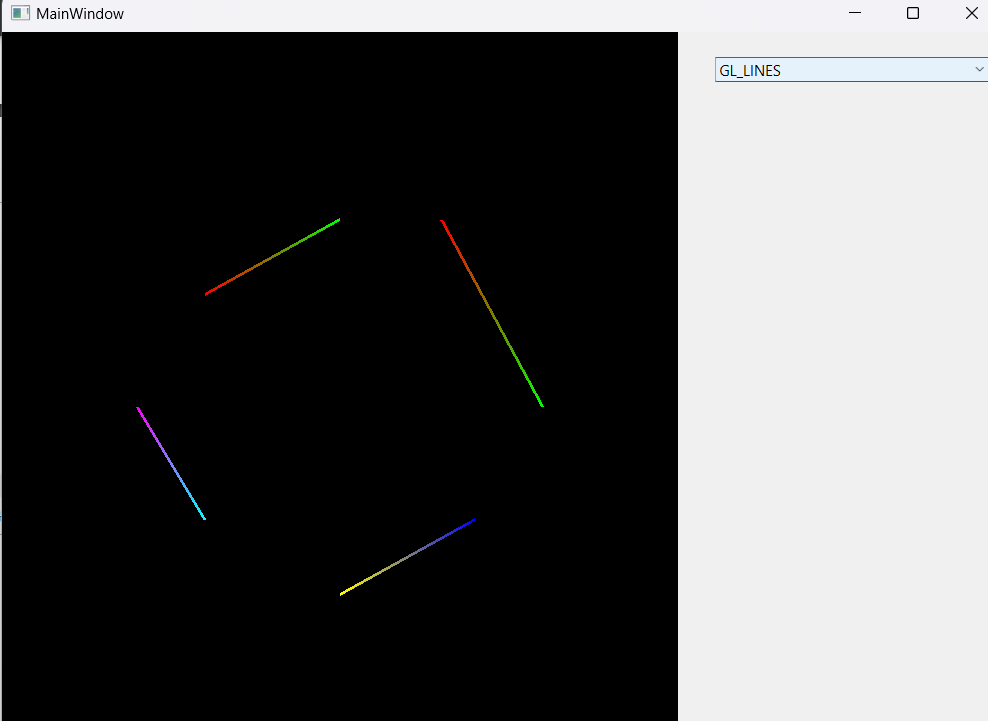


Рисунок 2 — GL\_LINES



Рисунок 3 — GL\_LINE\_STRIP

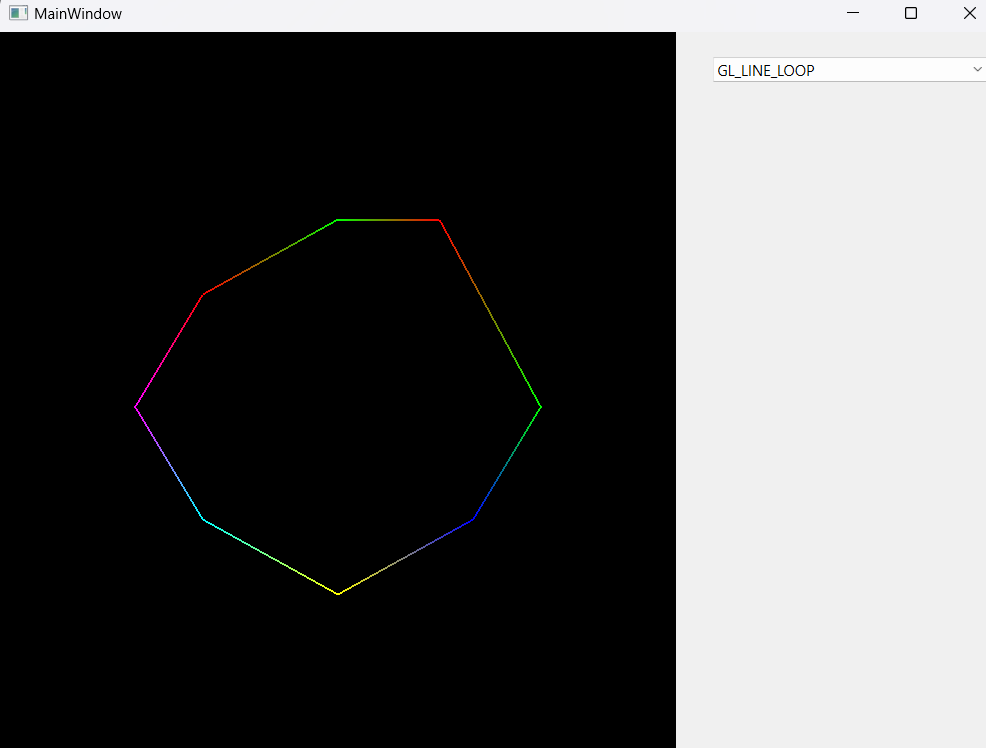


Рисунок 4 — GL\_LINE\_LOOP

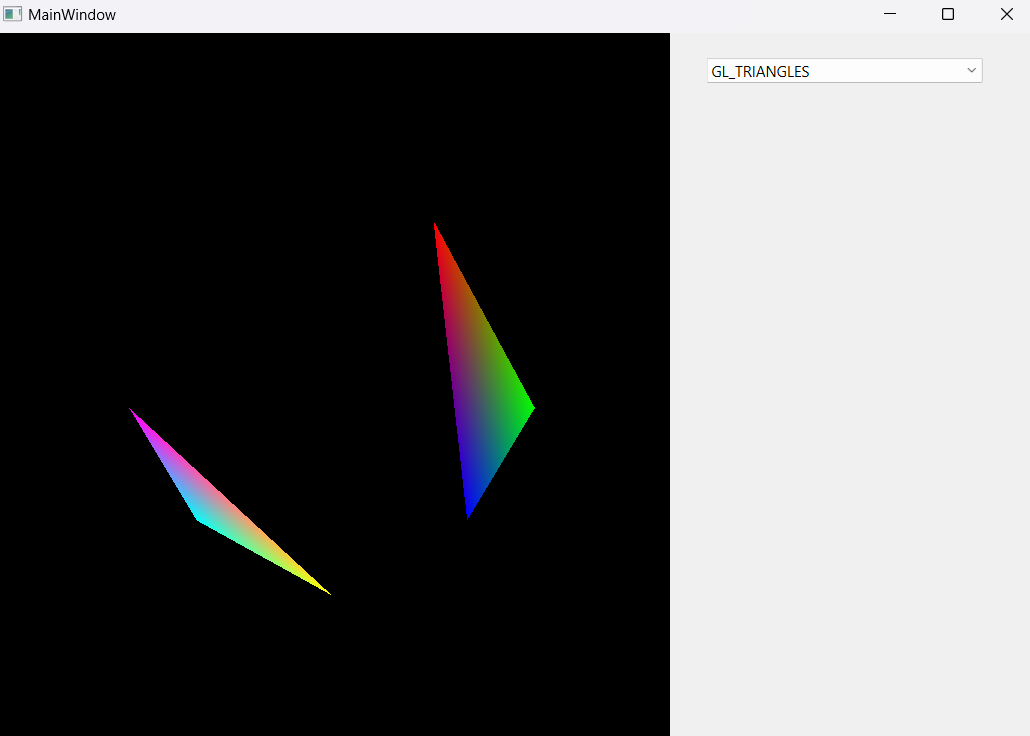


Рисунок 5 — GL\_TRIANGLES

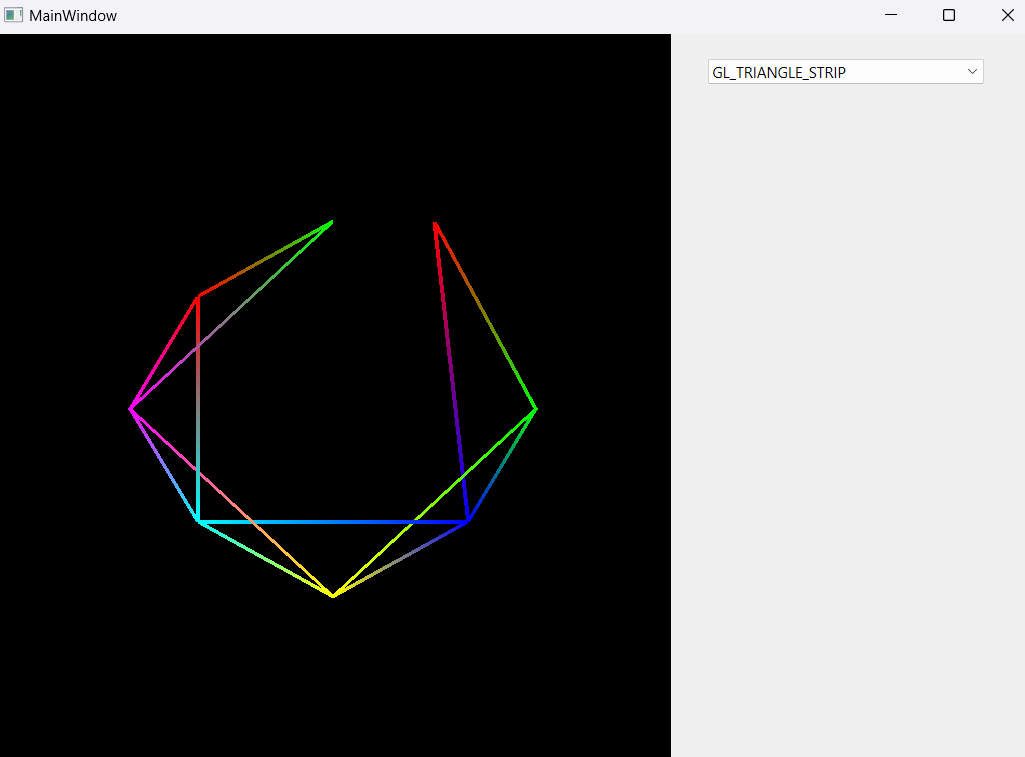


Рисунок 6 — GL\_TRIANGLE\_STRIP

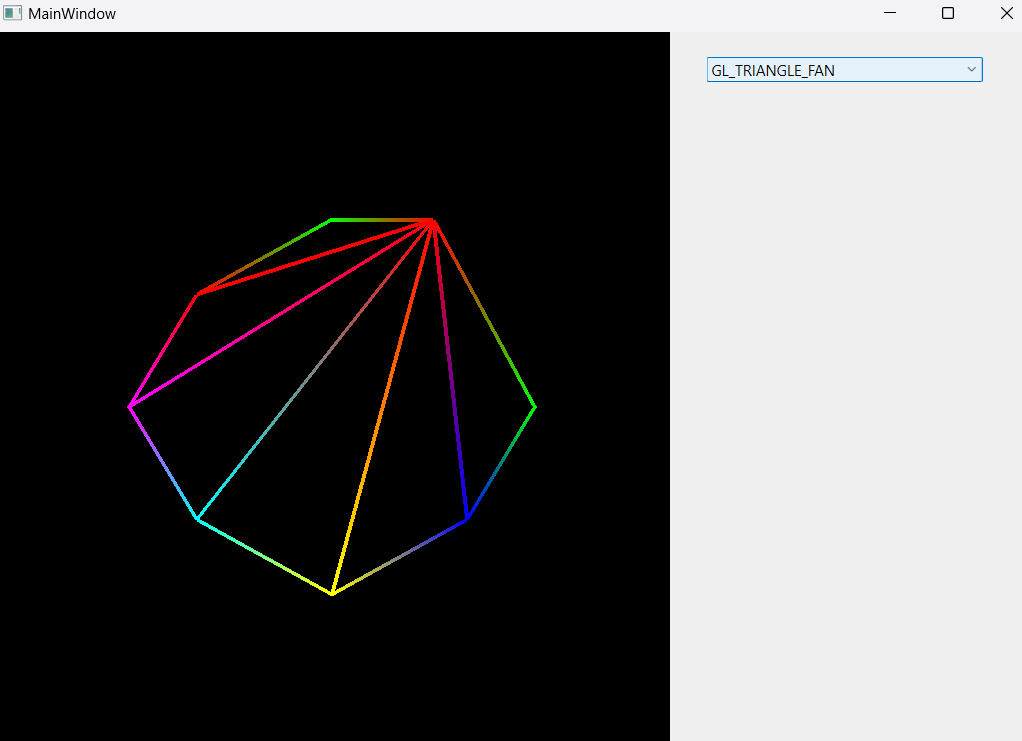


Рисунок 7 — GL\_TRIANGLE\_FAN



Рисунок 8 — GL\_QUADS

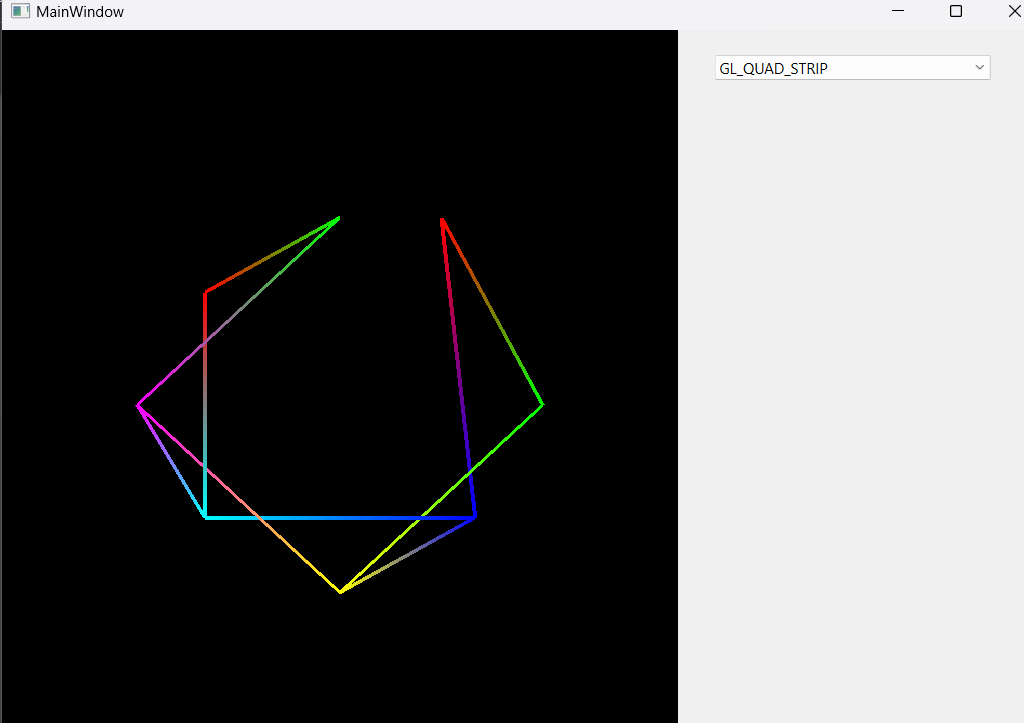


Рисунок 9 — GL\_QUAD\_STRIP

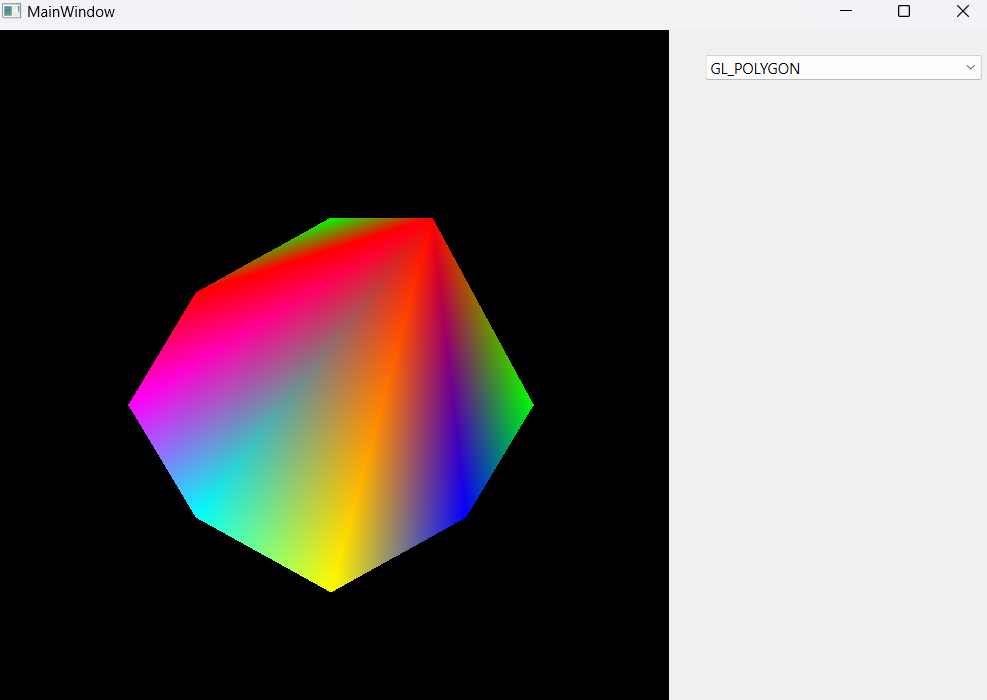


Рисунок 10 — GL\_POLYGON

**Вывод**

В результате выполнения лабораторной работы была разработана программа, создающая графические примитивы OpenGL. Программа работает корректно. При выполнении работы были приобретены навыки работы с графической библиотекой OpenGL.

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

main.py

from PyQt6 import QtCore, QtWidgets   
import sys   
from OpenGL.GL import \*   
from OpenGL.GLU import \*   
from OpenGL.GLUT import \*   
from PyQt6.QtWidgets import QApplication, QMainWindow, QVBoxLayout, QComboBox, QWidget   
from PyQt6.QtOpenGLWidgets import QOpenGLWidget   
from commands import commands   
   
class MyGLWidget(QOpenGLWidget):   
 def \_\_init\_\_(self,parent):   
 super(MyGLWidget, self).\_\_init\_\_(parent)   
 self.current\_mode = 'GL\_POINTS'   
   
 def paintGL(self):   
 if self.current\_mode in commands:   
 commands[self.current\_mode]()   
 self.update()   
   
   
class Ui\_MainWindow(object):   
 def setupUi(self, MainWindow):   
 MainWindow.setObjectName("MainWindow")   
 MainWindow.resize(800, 600)   
 self.centralwidget = QtWidgets.QWidget(parent=MainWindow)   
 self.centralwidget.setObjectName("centralwidget")   
 self.glWidget = MyGLWidget(parent=self.centralwidget)   
 self.glWidget.setGeometry(QtCore.QRect(0, 0, 541, 600))   
 self.comboBox = QtWidgets.QComboBox(parent=self.centralwidget)   
 self.comboBox.setGeometry(QtCore.QRect(570, 20, 221, 20))   
 self.comboBox.setObjectName("comboBox")   
 self.comboBox.addItem("")   
 self.comboBox.addItem("")   
 self.comboBox.addItem("")   
 self.comboBox.addItem("")   
 self.comboBox.addItem("")   
 self.comboBox.addItem("")   
 self.comboBox.addItem("")   
 self.comboBox.addItem("")   
 self.comboBox.addItem("")   
 self.comboBox.addItem("")   
 MainWindow.setCentralWidget(self.centralwidget)   
 self.menubar = QtWidgets.QMenuBar(parent=MainWindow)   
 self.menubar.setGeometry(QtCore.QRect(0, 0, 800, 26))   
 self.menubar.setObjectName("menubar")   
 MainWindow.setMenuBar(self.menubar)   
 self.statusbar = QtWidgets.QStatusBar(parent=MainWindow)   
 self.statusbar.setObjectName("statusbar")   
 MainWindow.setStatusBar(self.statusbar)   
   
 self.retranslateUi(MainWindow)   
 QtCore.QMetaObject.connectSlotsByName(MainWindow)   
   
 def retranslateUi(self, MainWindow):   
 \_translate = QtCore.QCoreApplication.translate   
 MainWindow.setWindowTitle(\_translate("MainWindow", "MainWindow"))   
 self.comboBox.setCurrentText(\_translate("MainWindow", "GL\_POINTS"))   
 self.comboBox.setItemText(0, \_translate("MainWindow", "GL\_POINTS"))   
 self.comboBox.setItemText(1, \_translate("MainWindow", "GL\_LINES"))   
 self.comboBox.setItemText(2, \_translate("MainWindow", "GL\_LINE\_STRIP"))   
 self.comboBox.setItemText(3, \_translate("MainWindow", "GL\_LINE\_LOOP"))   
 self.comboBox.setItemText(4, \_translate("MainWindow", "GL\_TRIANGLES"))   
 self.comboBox.setItemText(5, \_translate("MainWindow", "GL\_TRIANGLE\_STRIP"))   
 self.comboBox.setItemText(6, \_translate("MainWindow", "GL\_TRIANGLE\_FAN"))   
 self.comboBox.setItemText(7, \_translate("MainWindow", "GL\_QUADS"))   
 self.comboBox.setItemText(8, \_translate("MainWindow", "GL\_QUAD\_STRIP"))   
 self.comboBox.setItemText(9, \_translate("MainWindow", "GL\_POLYGON"))   
   
 def onComboBoxChanged(self):   
 print(self.comboBox.currentText())   
 self.glWidget.current\_mode = self.comboBox.currentText()   
 self.glWidget.update()   
   
if \_\_name\_\_ == "\_\_main\_\_":   
 app = QtWidgets.QApplication(sys.argv)   
 MainWindow = QtWidgets.QMainWindow()   
 ui = Ui\_MainWindow()   
 ui.setupUi(MainWindow)   
 ui.comboBox.currentIndexChanged.connect(ui.onComboBoxChanged)   
 MainWindow.show()   
 sys.exit(app.exec())

commands.py

from OpenGL.GL import \*

from OpenGL.GLUT import \*

colors = [[1.0, 0.0, 0.0, 1.0],

[0.0, 1.0, 0.0, 1.0],

[0.0, 0.0, 1.0, 1.0],

[1.0, 1.0, 0.0, 1.0],

[0.0, 1.0, 1.0, 1.0],

[1.0, 0.0, 1.0, 1.0]]

vertex = [[0.3, 0.5],

[0.6, 0],

[0.4, -0.3],

[0, -0.5],

[-0.4, -0.3],

[-0.6, 0],

[-0.4, 0.3],

[0, 0.5]]

def points():

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

glPointSize(3)

glBegin(GL\_POINTS)

for i in range(len(vertex)):

glColor4f(\*colors[i%len(colors)])

glVertex2f(vertex[i][0], vertex[i][1])

glEnd()

def lines():

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

glLineWidth(3)

glBegin(GL\_LINES)

for i in range(len(vertex)):

glColor4f(\*colors[i%len(colors)])

glVertex2f(vertex[i][0], vertex[i][1])

glEnd()

def lineStrip():

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

glLineWidth(3)

glBegin(GL\_LINE\_STRIP)

for i in range(len(vertex)):

glColor4f(\*colors[i%len(colors)])

glVertex2f(vertex[i][0], vertex[i][1])

glEnd()

def lineLoop():

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

glLineWidth(2)

glBegin(GL\_LINE\_LOOP)

for i in range(len(vertex)):

glColor4f(\*colors[i%len(colors)])

glVertex2f(vertex[i][0], vertex[i][1])

glEnd()

def triangles():

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

glLineWidth(4)

glBegin(GL\_TRIANGLES)

for i in range(len(vertex)):

glColor4f(\*colors[i%len(colors)])

glVertex2f(vertex[i][0], vertex[i][1])

glEnd()

def triangleStrip():

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

glLineWidth(4)

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE)

glBegin(GL\_TRIANGLE\_STRIP)

for i in range(len(vertex)):

glColor4f(\*colors[i%len(colors)])

glVertex2f(vertex[i][0], vertex[i][1])

glEnd()

def triangleFan():

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

glLineWidth(4)

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE)

glBegin(GL\_TRIANGLE\_FAN)

for i in range(len(vertex)):

glColor4f(\*colors[i%len(colors)])

glVertex2f(vertex[i][0], vertex[i][1])

glEnd()

def quads():

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

glLineWidth(4)

glBegin(GL\_QUADS)

for i in range(len(vertex)):

glColor4f(\*colors[i%len(colors)])

glVertex2f(vertex[i][0], vertex[i][1])

glEnd()

def quadStrip():

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

glLineWidth(4)

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE)

glBegin(GL\_QUAD\_STRIP)

for i in range(len(vertex)):

glColor4f(\*colors[i%len(colors)])

glVertex2f(vertex[i][0], vertex[i][1])

glEnd()

def polygon():

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

glLineWidth(4)

glBegin(GL\_POLYGON)

for i in range(len(vertex)):

glColor4f(\*colors[i%len(colors)])

glVertex2f(vertex[i][0], vertex[i][1])

glEnd()

commands = {

"GL\_POINTS" : points,

"GL\_LINES" : lines,

"GL\_LINE\_STRIP" : lineStrip,

"GL\_LINE\_LOOP" : lineLoop,

"GL\_TRIANGLES": triangles,

"GL\_TRIANGLE\_STRIP": triangleStrip,

"GL\_TRIANGLE\_FAN" : triangleFan,

"GL\_QUADS" : quads,

"GL\_QUAD\_STRIP" : quadStrip,

"GL\_POLYGON" : polygon

}