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.

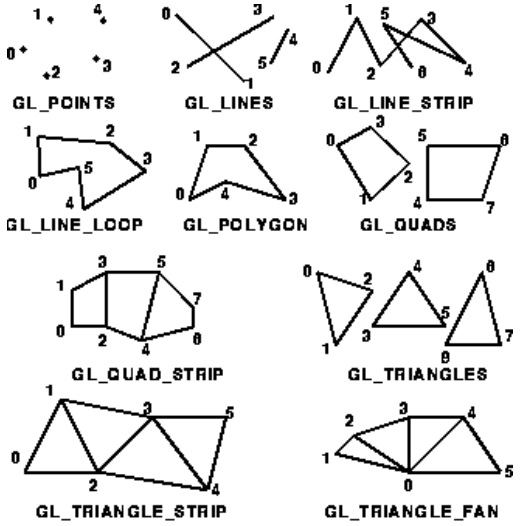
GL_TRIANGLE_STRIP - в этом режиме рисуется группа связанных треугольников, имеющих общую грань. Первые три вершины определяют первый треугольник, вторая, третья и четвертая – второй и т.д. для нечетного n вершины n, (n+1) и (n+2) определяют треугольник n. Для четного n треугольник определяют вершины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

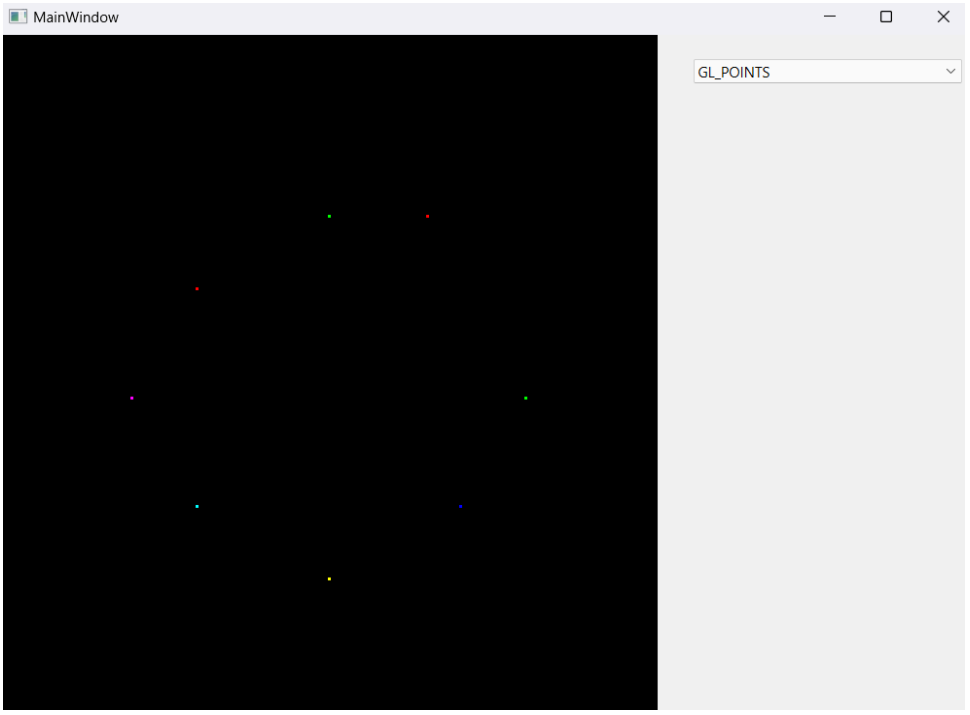


Рисунок 1 — GL_POINTS

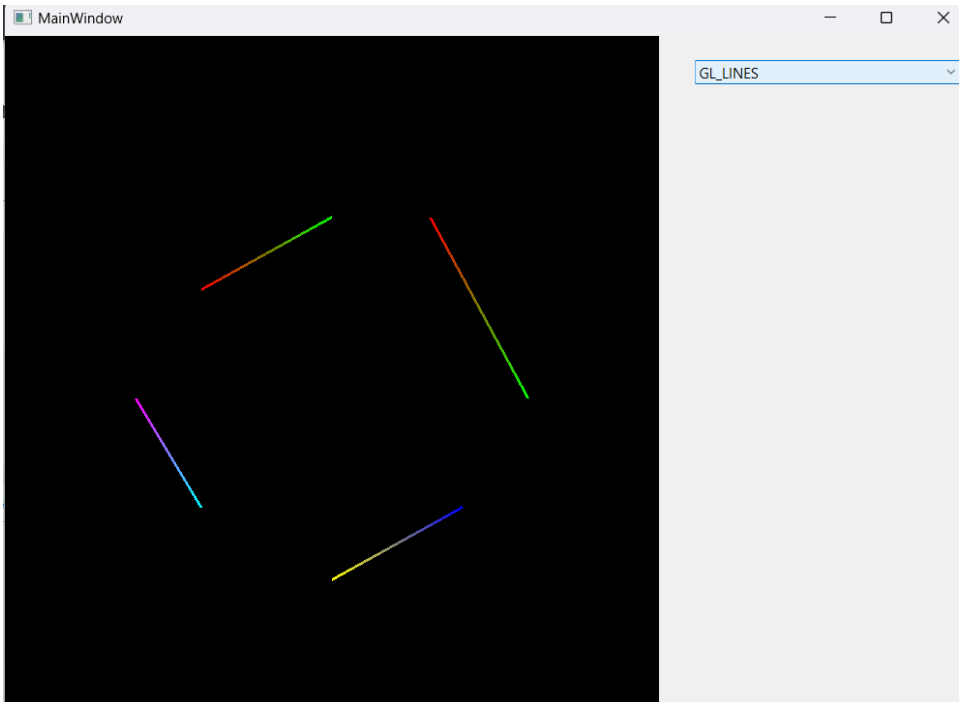


Рисунок 2 — GL_LINES

Рисунок 3 — GL_LINE_STRIP

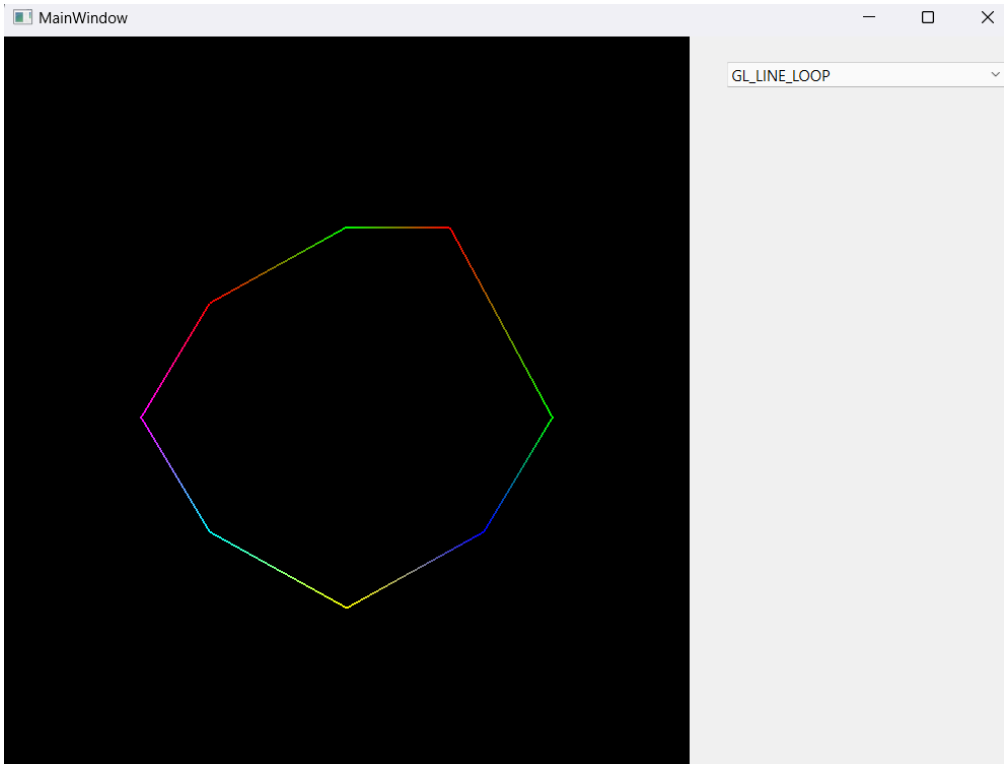


Рисунок 4 — GL_LINE_LOOP

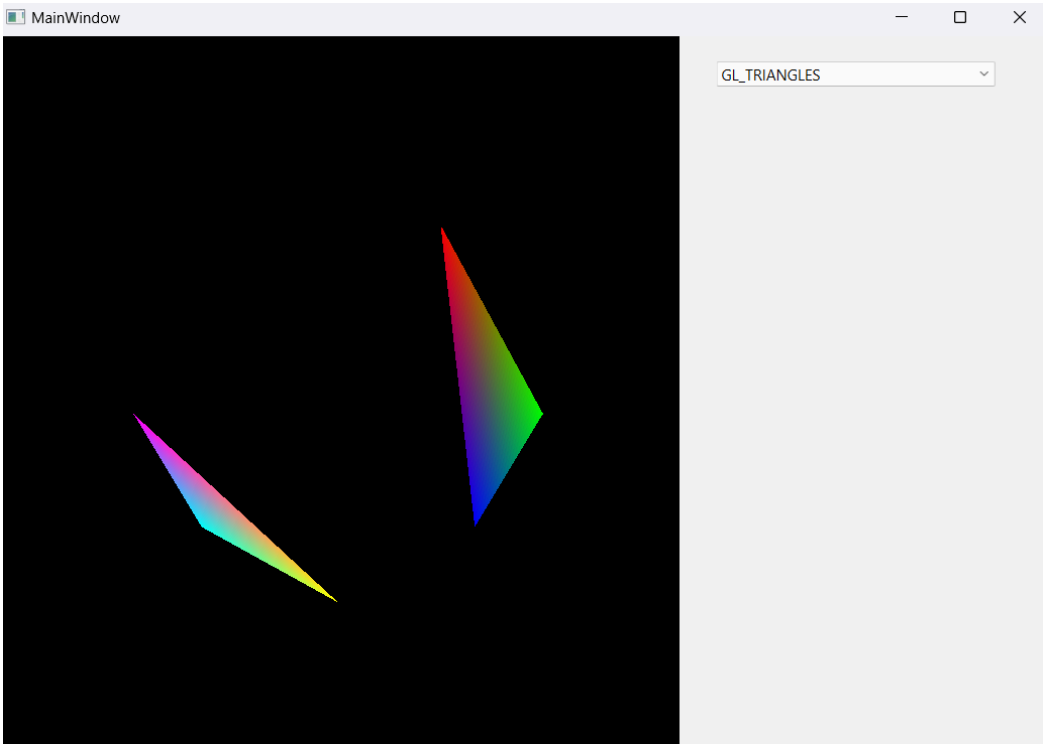


Рисунок 5 — GL_TRIANGLES

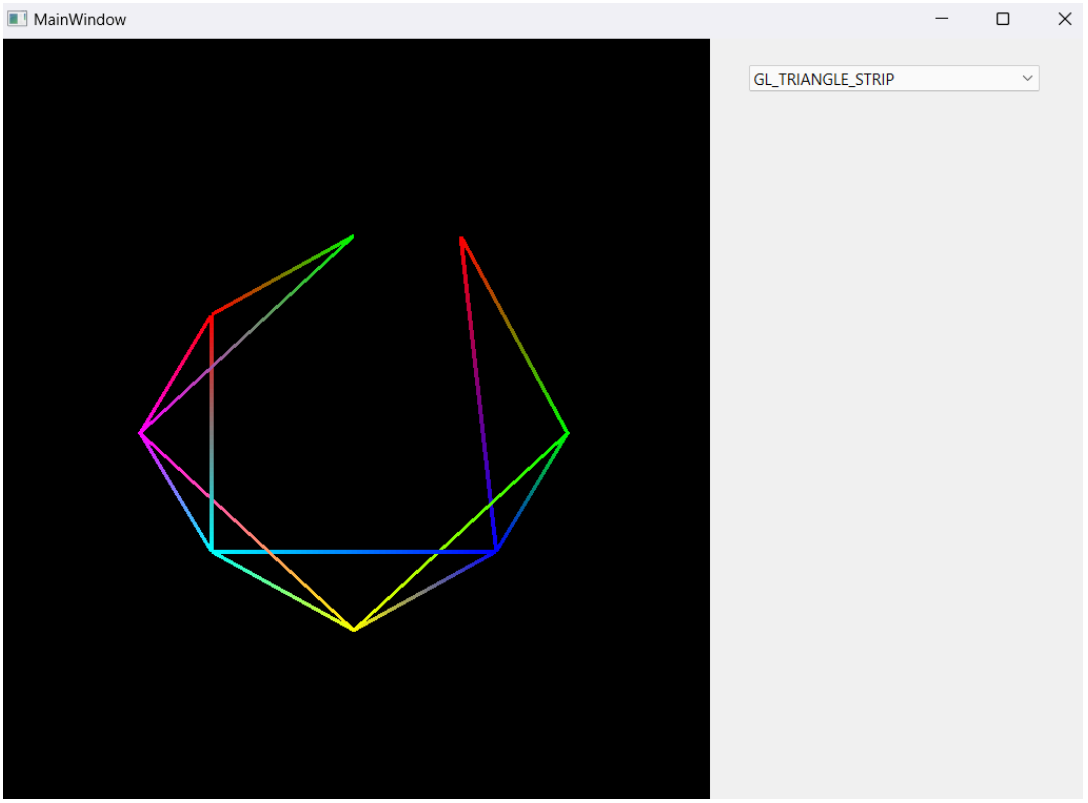


Рисунок 6 — GL_TRIANGLE_STRIP

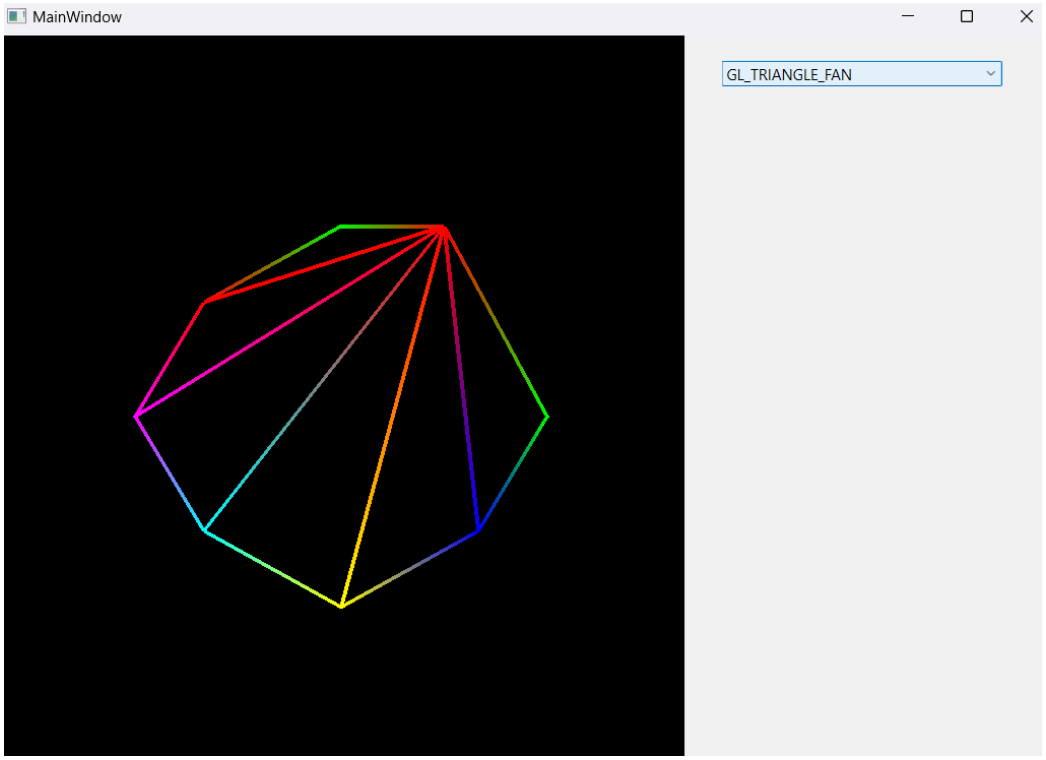


Рисунок 7 — GL_TRIANGLE_FAN

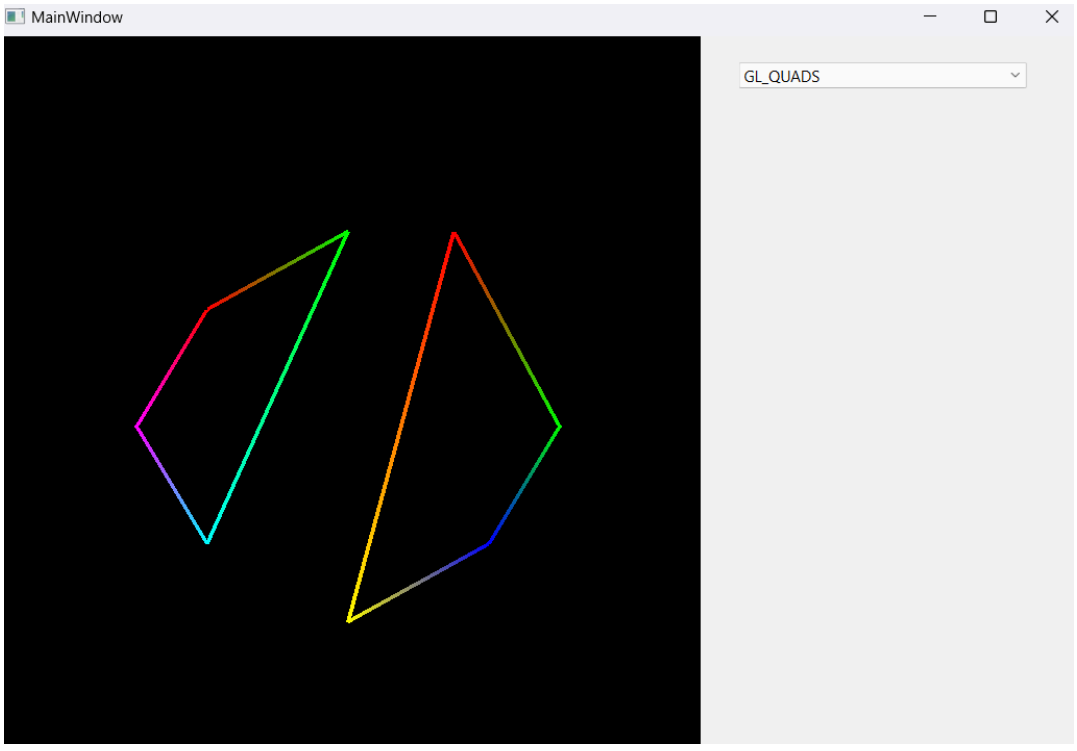


Рисунок 8 — GL_QUADS

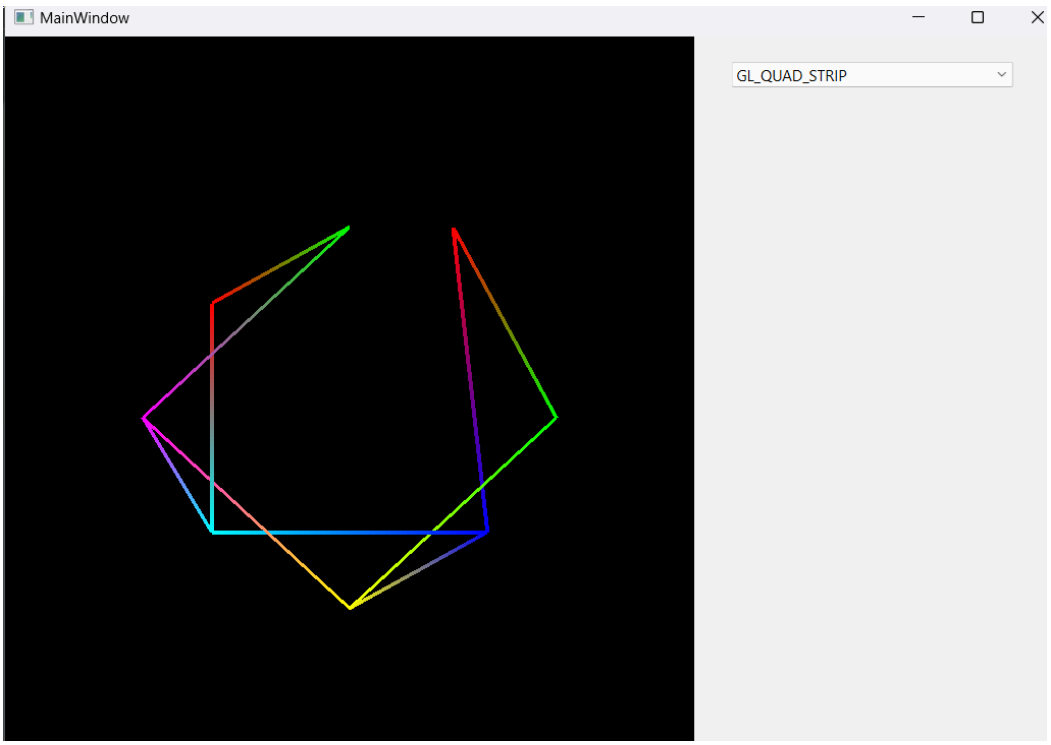


Рисунок 9 — GL_QUAD_STRIP

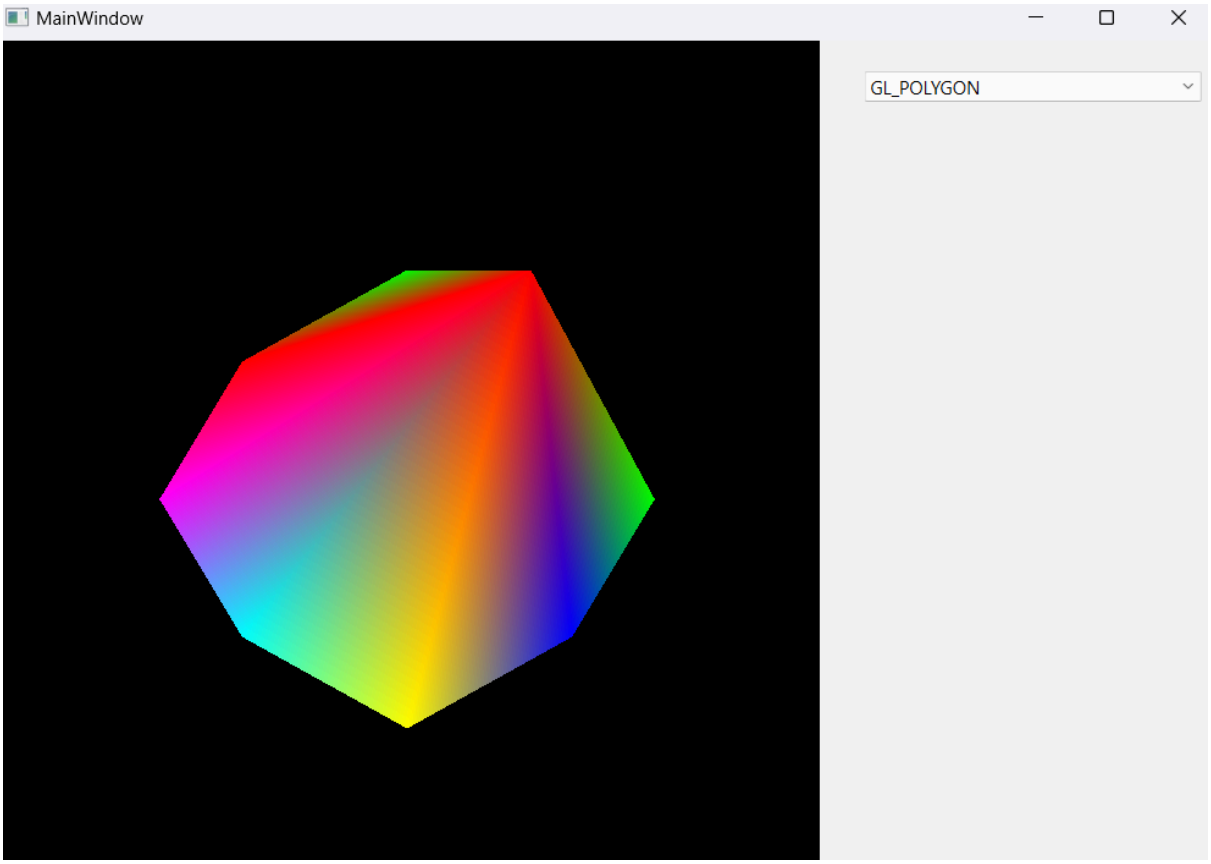


Рисунок 10 — GL_POLYGON

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
}
```