Санкт-Петербургский Государственный
Электротехнический Университет

Кафедра МОЭВМ

Задание для лабораторной работы № 1
"Примитивы OpenGL"

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

Санкт-Петербург
2023 г.

Цель работы.

- ознакомление с основными примитивами OpenGL.

- освоение возможности подключения графической библиотеки в среду разработки.

- проанализировать полученное задание, выделить информационные объекты и действия;

- разработать программу с использованием требуемых примитивов и атрибутов.

Основные теоретические положения.

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.

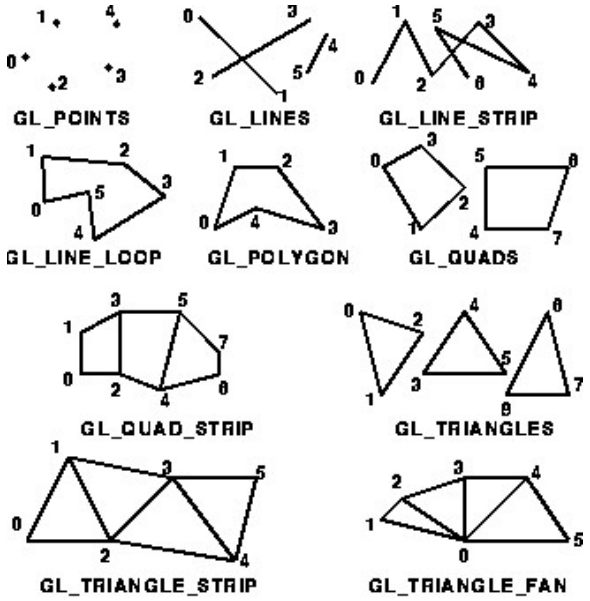
GL_TRIANGLE_STRIP - в этом режиме рисуется группа связанных треугольников, имеющих общую грань. Первые три вершины определяют первый треугольник, вторая, третья и четвертая – второй и т.д. для нечетного n вершины n, (n+1) и (n+2) определяют треугольник n. Для четного n треугольник определяют вершины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

Также можно указывать, какой тип граней отображать на экране. Для этого сначала на-до установить соответствующий режим вызовом команды glEnable

(GL_CULL_FACE), а за- тем выбрать тип отображаемых граней с помощью команды void glCullFace (GLenum mode).

Вызов с параметром GL_FRONT приводит к удалению из изображения всех лицевых граней, а с параметром GL_BACK – обратных (установка по умолчанию).

Кроме рассмотренных стандартных примитивов в библиотеках GLU и GLUT описаны более сложные фигуры, такие как сфера, цилиндр, диск (в GLU) и сфера, куб, конус, тор, тетраэдр, додекаэдр, икосаэдр, октаэдр и чайник (в GLUT). Автоматическое наложение текстуры предусмотрено только для фигур из библиотеки GLU.

Например, чтобы нарисовать сферу или цилиндр, надо сначала создать объект специального типа GLUquadricObj с помощью команды GLUquadricObj* gluNewQuadric(void);а затем вызвать соответствующую команду:

```
void gluSphere (GLUquadricObj * qobj, GLdouble radius,
        GLint slices, GLint stacks);


void gluCylinder (GLUquadricObj * qobj,
        GLdouble baseRadius,
        GLdouble topRadius,
        GLdouble height, GLint slices,
        GLint stacks);
```

где параметр slices задает число разбиений вокруг оси z, а stacks – вдоль оси z.

Задание.

Разработать программу, реализующую представление определенного набора примитивов (4) из имеющихся в 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 разработанного вами шаблона программа должна быть пополнена возможностями остановки интерактивно различных атрибутов примитивов рисования через вызов соответствующих элементов интерфейса пользователя.

Выполнение работы.

Работа выполнена на языке программирования Python с использованием PyQt5, PyOpenGL в среде разработки pyCharm. Для работы с PyOpenGL были установлены следующие элементы PyOpenGL-3.1.6-cp311-cp311-win_amd64.whl и PyOpenGL_accelerate-3.1.6-cp311-cp311-win_amd64.whl. Импортирование осуществлено с помощью следующих команд:

*from OpenGL.GL import **

*from OpenGL.GLU import**.

PyQt5 установлен с помощью Python Packages среды разработки pyCharm. Импортирование осуществлено с помощью следующих команд:

*from PyQt5.QtOpenGL import **

from PyQt5 import QtWidgets

from PyQt5.QtWidgets import (QWidget, QLabel,

QComboBox, QStackedWidget)

Создан класс окна программы mainWindow(QWidget), который наследуется от класса QWidget в PyQt5.QtWidgets. В конструкторе атрибут self.stack принадлежит классу QStackedWidget() – стеку виджетов, в котором виден только один виджет. С помощью метода addWidget добавлены виджеты всех примитивов. Создан слой buttonsLayout класса QVBoxLayout, который выстраивает виджеты вертикально.

Атрибут self.box принадлежит классу QComboBox - объединенная кнопка с всплывающим списком. Список заполнен с помощью метода addItems "GL_POINTS", "GL_LINES", "GL_LINE_STRIP", "GL_LINE_LOOP", "GL_TRIANGLES", "GL_TRIANGLE_STRIP", "GL_TRIANGLE_FAN", "GL_QUADS", "GL_QUAD_STRIP", "GL_POLYGON". self.box связан с self.stack с помощью метода activated, который посылает сигнал с индексом выбранного элемента в списке, с помощью метода connect осуществляется связь с методом setCurrentIndex у self.stack, так меняется виджет в self.stack. Атрибут self.lbl принадлежит классу QLabel отображает текст с выбранным примитивом. self.box связан с self.lbl с помощью метода activated, который посылает сигнал со строкой выбранного элемента в списке, с помощью метода connect осуществляется связь с методом activated_box у mainWindow, так меняется текст с выбранным примитивом. В методе activated_box используется метод setText у self.lbl. В слой buttonsLayout добавлены виджеты self.box, self.lbl с помощью метода addWidget, добавлено расширяемое пространство в конец в помощью метода addStretch. mainLayout – слой принадлежит классу QHBoxLayout, который выстраивает виджеты горизонтально. widgetLayout - слой принадлежит классу QHBoxLayout, который выстраивает виджеты горизонтально. В него добавлен виджет self.stack. В слой mainLayout добавлены слои widgetLayout и buttonsLayout. В окне mainWindow установлен слой mainLayout с помощью метода setLayout. Установлено название окна с помощью метода setWindowTitle.

Создан класс glWidget0(QGLWidget), который наследуется от PyQt5.QtOpenGL.QGLWidget. От этого класса будут наследоваться все примитивы. Определен метод initializeGL. Функция glClearColor(0, 0, 0, 0) устанавливает цвет для очистки буфера цвета. glClearDepth(1.0) устанавливает параметр 1.0 для

очистки буфера глубины. glDepthFunc(GL_LESS) указывает значение GL_LESS, используемое для сравнения буфера глубины. glEnable(GL_DEPTH_TEST) включение буфера глубины. glShadeModel(GL_SMOOTH) выбран режим затенения. glMatrixMode(GL_PROJECTION) установка текущей матрицы матрицу проекций. glLoadIdentity() – заменяет текущую матрицу на единичную. gluPerspective(45.0, 1, 0.1, 100.0) – настроена перспектива. glMatrixMode(GL_MODELVIEW) установка текущей матрицы матрицу модельно-видовой. Метод paintGL не определен, он будет переопределяться у примитивов.

Класс glWidget1 наследуется от класса glWidget0 – класс примитива GL_POINTS. glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT) – очистка буферов. glLoadIdentity() – заменяет текущую матрицу на единичную. glViewport(0, 0, 480, 480) – задание преобразования аффинных значений x и y из нормализованных координат устройства в координаты окна. glTranslatef(-2.5, 0.5, -5.0) – перемещение осей координат. glColor3f(0.5, 1.0, 0.5) – установка цвета. glBegin(GL_POINTS), glVertex3f(2.0, -1.2, 0.0), glVertex3f(2.6, 0.0, 0.0), glVertex3f(2.9, -1.2, 0.0), glEnd() – определение координат для примитива. glFlush() – очистка буферов и ускорение.

Класс glWidget2 наследуется от класса glWidget0 – класс примитива GL_LINES. Аналогичен классу glWidget1.

Класс glWidget3 наследуется от класса glWidget0 – класс примитива GL_LINE_STRIP. Аналогичен классу glWidget1. Включена прерывистая линия с помощью glEnable(GL_LINE_STIPPLE). С помощью glLineStipple(1, 0x0101) задан точечный пунктир.

Класс glWidget4 наследуется от класса glWidget0 – класс примитива GL_LINE_LOOP. Аналогичен классу glWidget1. Включена прерывистая линия с

помощью glEnable(GL_LINE_STIPPLE). С помощью glLineStipple(1, 0x00FF) задан штриховой пунктир.

Класс glWidget5 наследуется от класса glWidget0 – класс примитива GL_TRIANGLES. Аналогичен классу glWidget1. glPolygonMode(GL_FRONT, GL_FILL) – управление интерпретацией многоугольника для лицевых граней, включена заливка.

Класс glWidget6 наследуется от класса glWidget0 – класс примитива GL_TRIANGLE_STRIP. Аналогичен классу glWidget1. glPolygonMode(GL_FRONT_AND_BACK, GL_LINE) – управление интерпретацией многоугольника для всех граней, представление набором отрезков.

Класс glWidget7 наследуется от класса glWidget0 – класс примитива GL_TRIANGLE_FAN. Аналогичен классу glWidget6.

Класс glWidget8 наследуется от класса glWidget0 – класс примитива GL_QUADS. Аналогичен классу glWidget6.

Класс glWidget9 наследуется от класса glWidget0 – класс примитива GL_QUAD_STRIP. Аналогичен классу glWidget6.

Класс glWidget10 наследуется от класса glWidget0 – класс примитива GL_POLYGON. Аналогичен классу glWidget6.

app принадлежит классу QApplication в QtWidgets, переданы sys.argv – приложение с GUI. qWindow принадлежит классу QMainWindow в QtWidgets. window принадлежит классу mainWindow(qWindow) – главное окно. Показ окна осуществляется с помощью метода show. sys.exit(app.exec_()) – запуск цикла обработки событий с безопасным выходом.

ТЕСТИРОВАНИЕ.

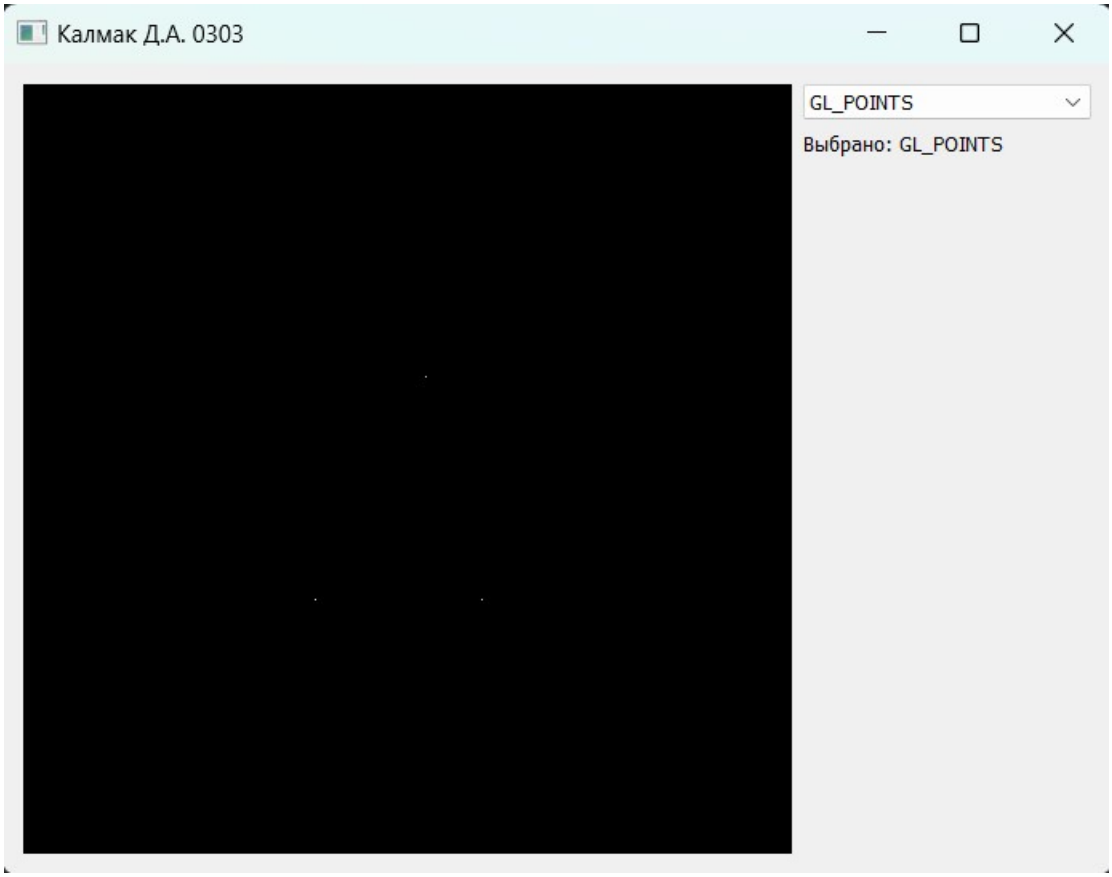


Рисунок 1 – Примитив GL_POINTS

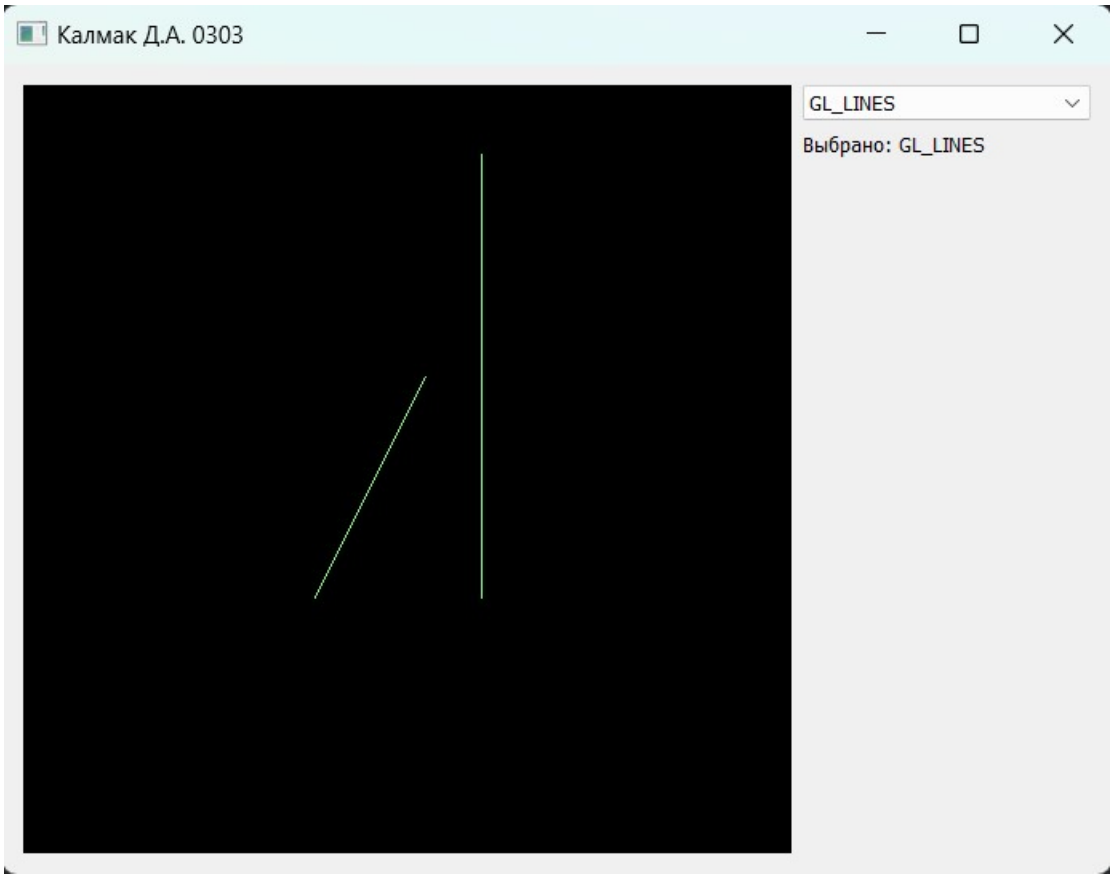


Рисунок 2 – Примитив GL_LINES

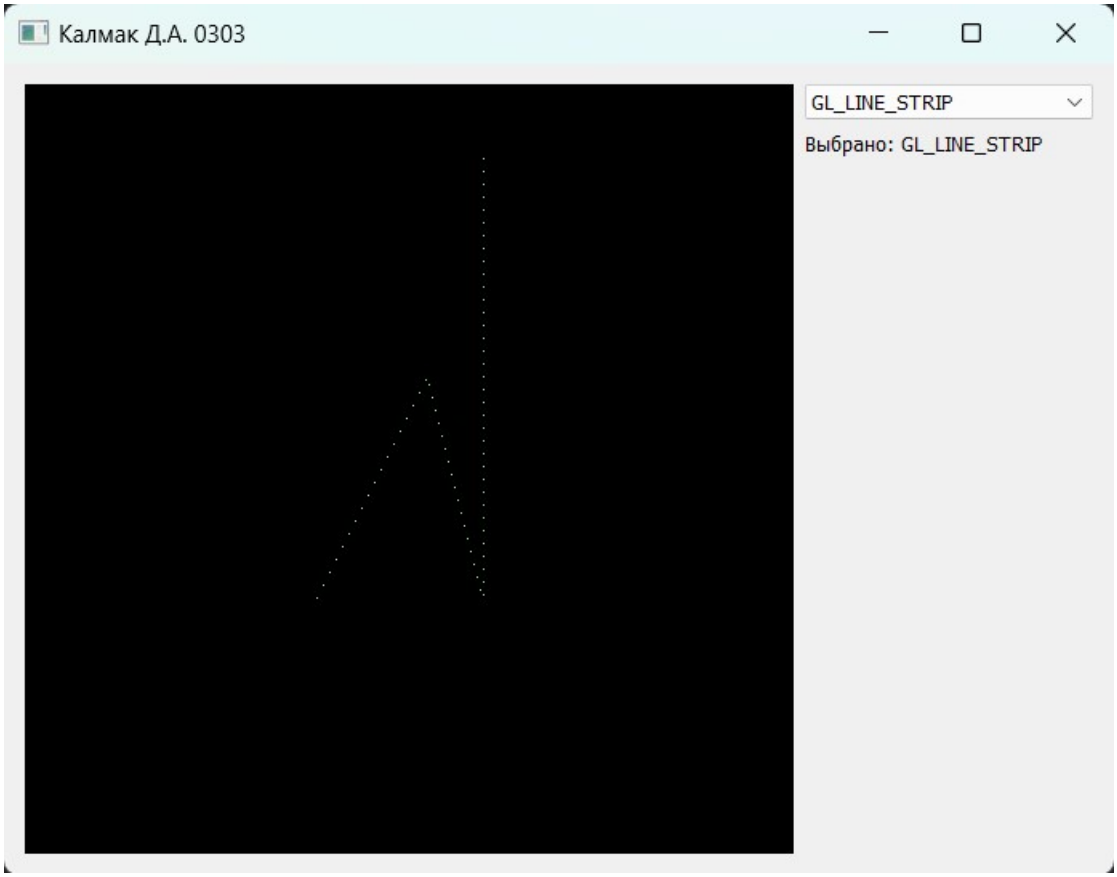


Рисунок 3 – Примитив GL_LINE_STRIP

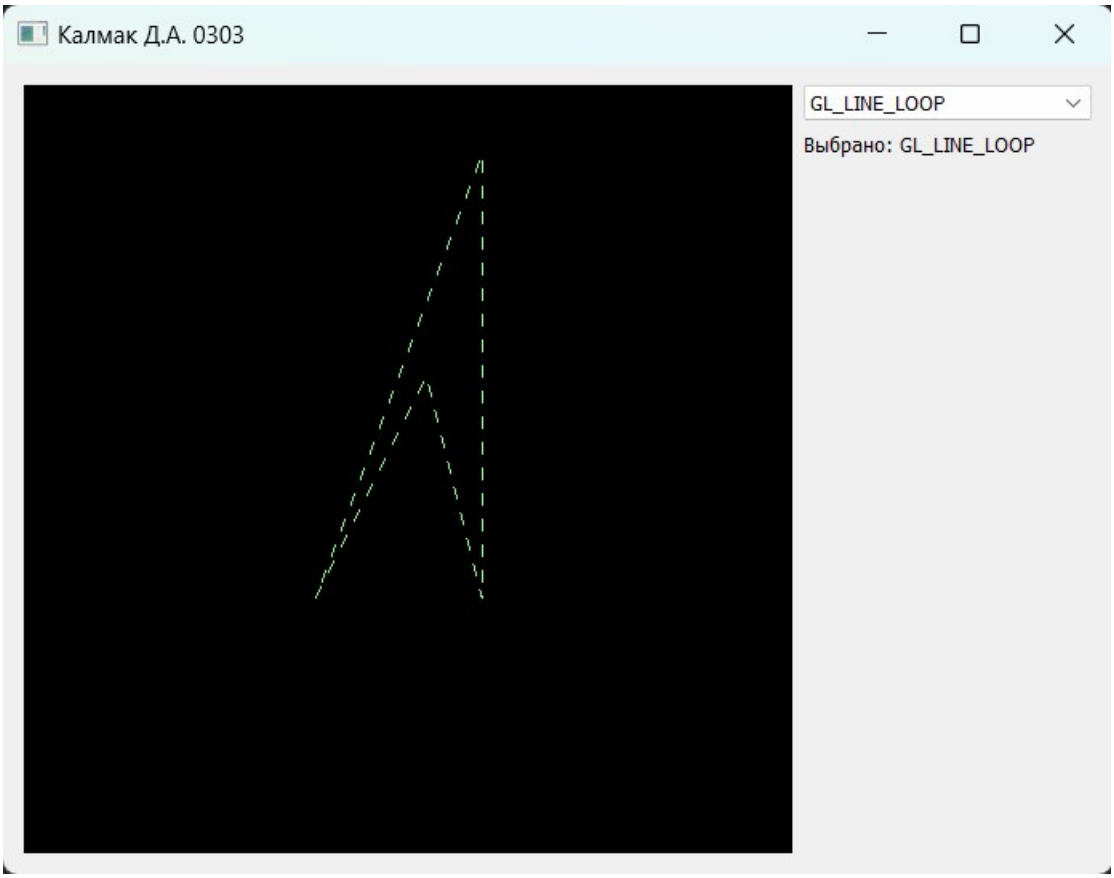


Рисунок 4 – Примитив GL_LINE_LOOP

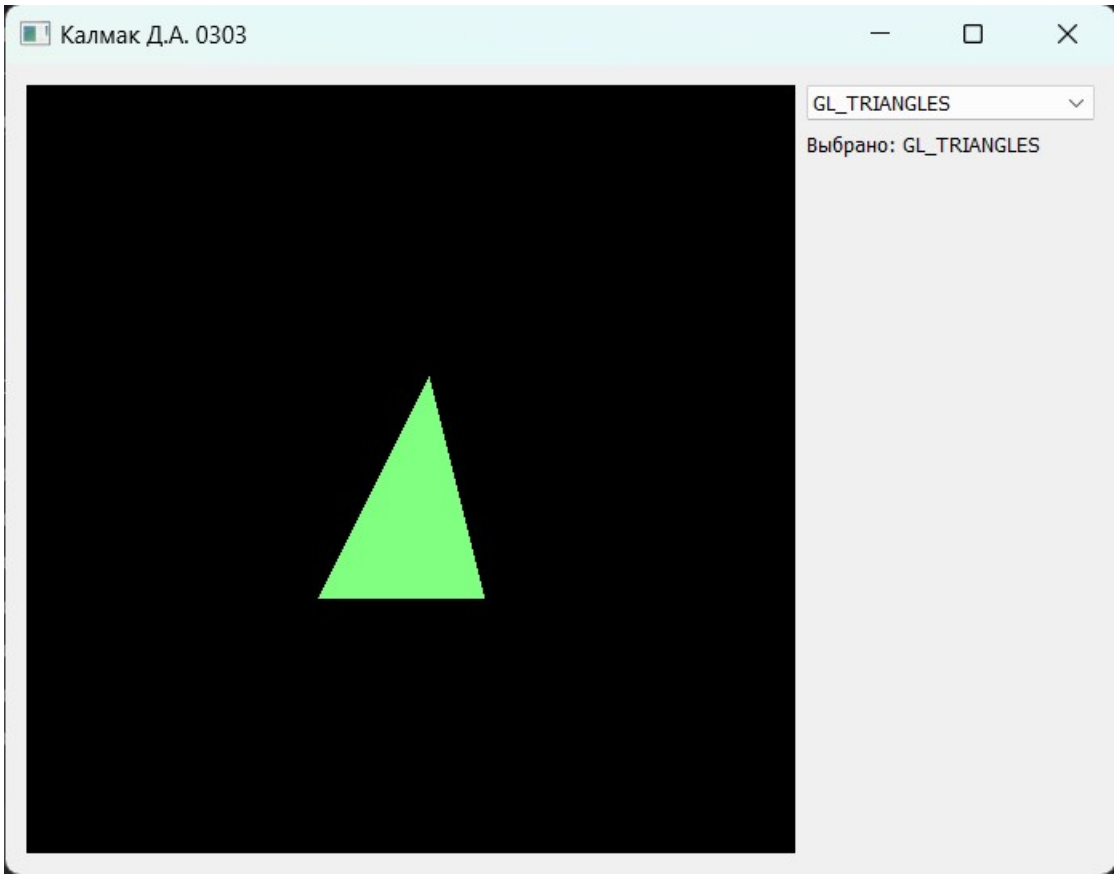


Рисунок 5 – Примитив GL_TRIANGLES

Рисунок 6 – Примитив GL_TRIANGLE_STRIP

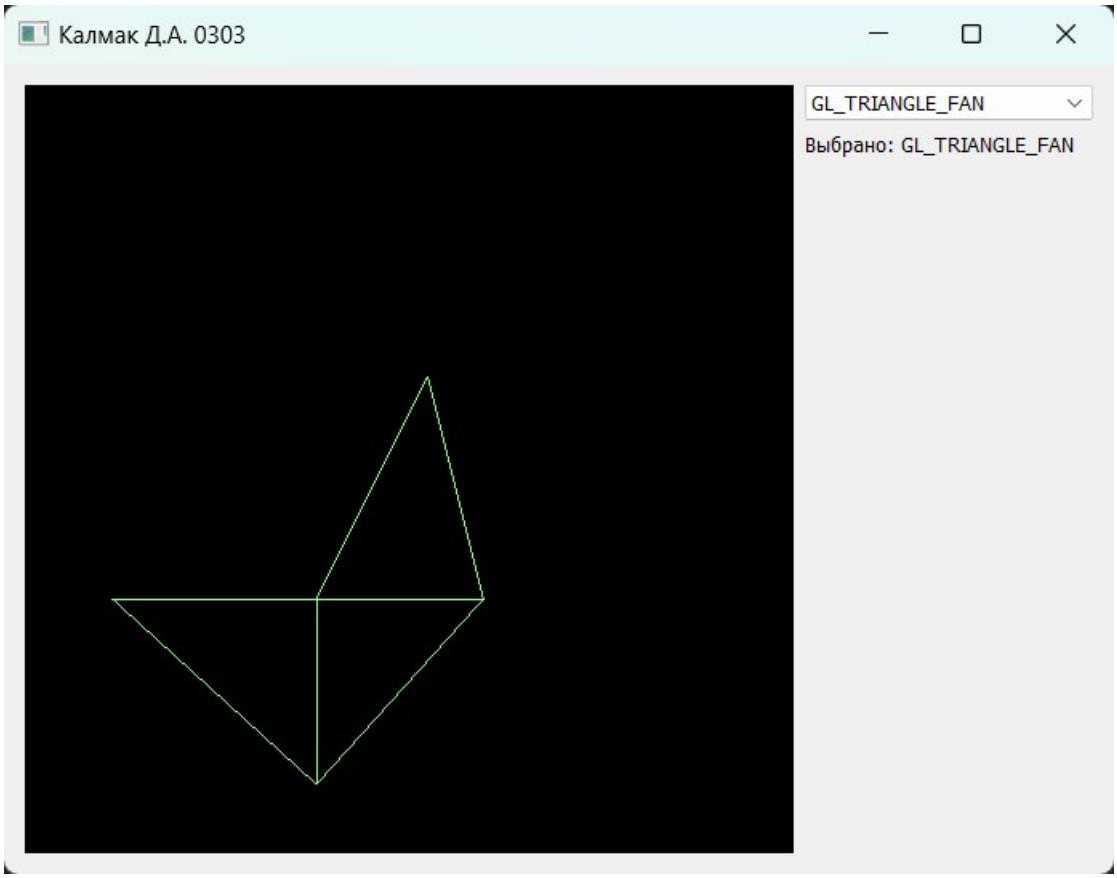


Рисунок 7 – Примитив GL_TRIANGLE_FAN

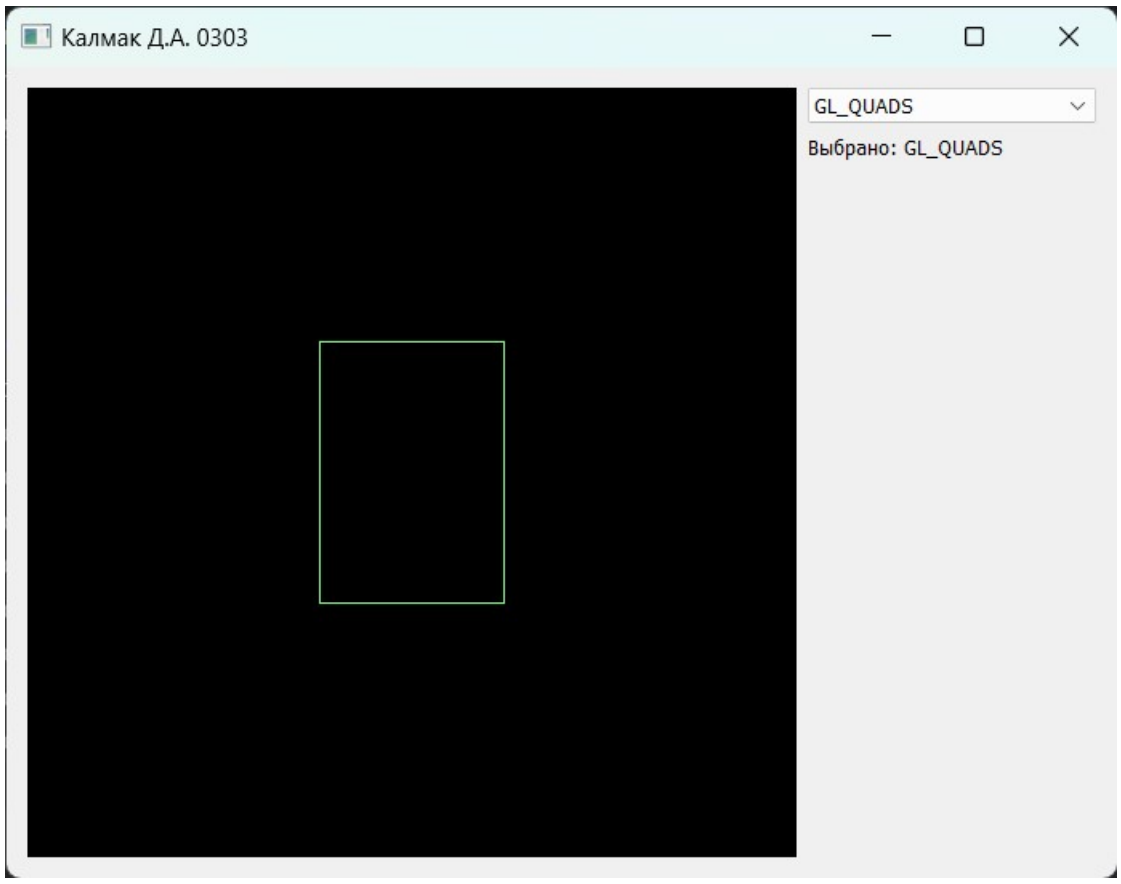


Рисунок 8 – Примитив GL_QUADS

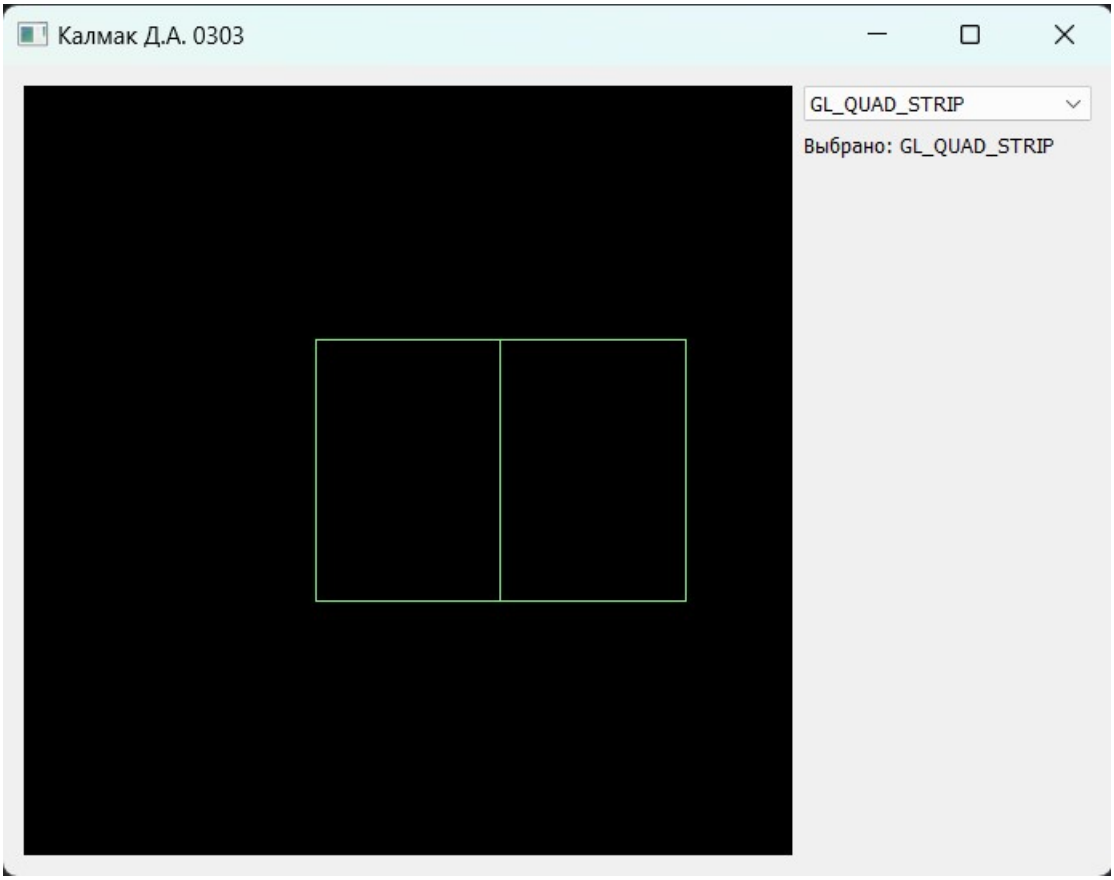


Рисунок 9 – Примитив GL_QUAD_STRIP

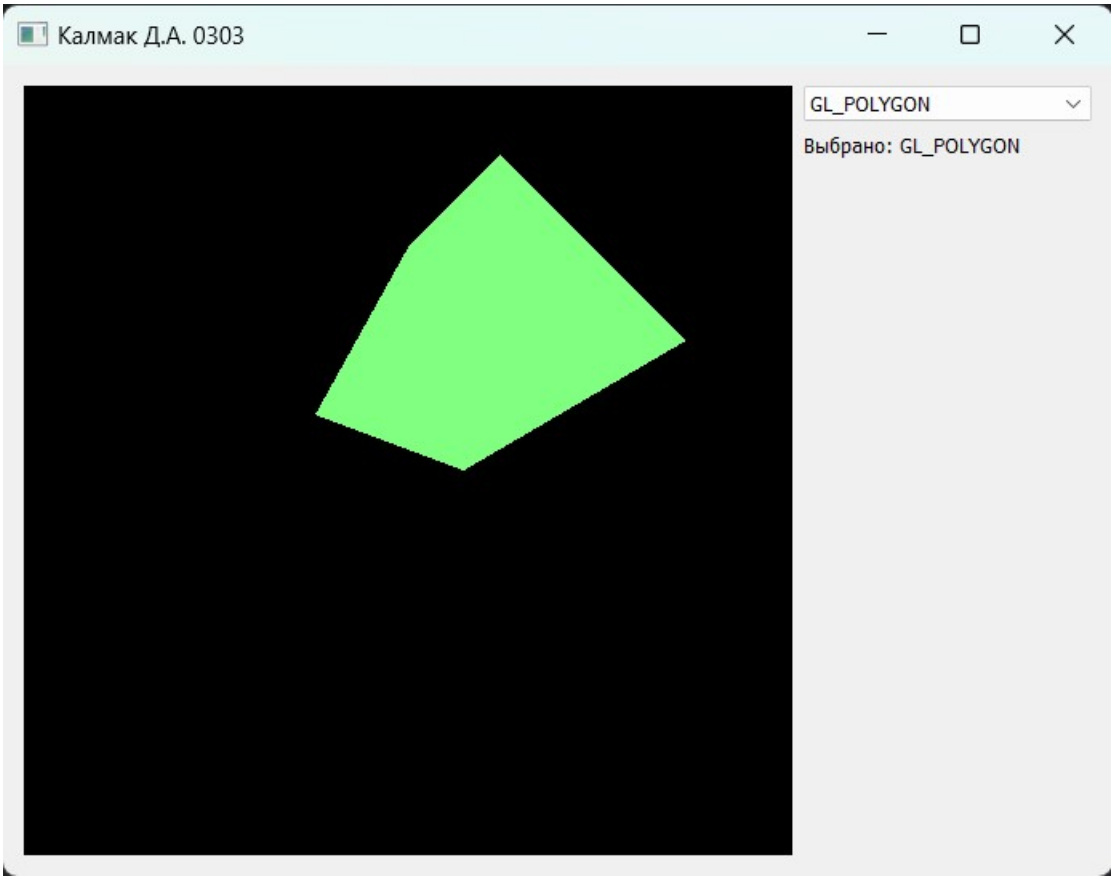


Рисунок 10 – Примитив GL_POLYGON

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

```
import sys
from OpenGL.GL import *
from OpenGL.GLU import *
from PyQt5.QtOpenGL import *
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import (QWidget, QLabel,
                             QComboBox, QStackedWidget)


class mainWindow(QWidget):
    def __init__(self, parent=None):
        super(mainWindow, self).__init__()
        self.stack = QStackedWidget()
        self.stack.addWidget(glWidget1())
        self.stack.addWidget(glWidget2())
        self.stack.addWidget(glWidget3())
        self.stack.addWidget(glWidget4())
        self.stack.addWidget(glWidget5())
        self.stack.addWidget(glWidget6())
        self.stack.addWidget(glWidget7())
        self.stack.addWidget(glWidget8())
        self.stack.addWidget(glWidget9())
        self.stack.addWidget(glWidget10())

        buttonsLayout = QtWidgets.QVBoxLayout()
        self.box = QComboBox()
        self.box.setMinimumSize(180, 20)
        self.box.addItems(["GL_POINTS", "GL_LINES", "GL_LINE_STRIP", "GL_LINE_LOOP",
"GL_TRIANGLES", "GL_TRIANGLE_STRIP",
                   "GL_TRIANGLE_FAN", "GL_QUADS", "GL_QUAD_STRIP", "GL_POLYGON"])
        self.box.activated[int].connect(self.stack.setCurrentIndex)
        self.lbl = QLabel("Выбрано: GL_POINTS", self)
        self.box.activated[str].connect(self.activated_box)
        buttonsLayout.addWidget(self.box)
        buttonsLayout.addWidget(self.lbl)
        buttonsLayout.addStretch()

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

    def activated_box(self, text):
        self.lbl.setText("Выбрано: " + text)


class glWidget0(QGLWidget):
    def __init__(self, parent=None):
```

```
        QGLWidget.__init__(self, parent)
        self.setMinimumSize(480, 480)

    def initializeGL(self):
        glClearColor(0, 0, 0, 0)
        glClearDepth(1.0)
        glDepthFunc(GL_LESS)
        glEnable(GL_DEPTH_TEST)
        glShadeModel(GL_SMOOTH)
        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()
        gluPerspective(45.0, 1, 0.1, 100.0)
        glMatrixMode(GL_MODELVIEW)

    def paintGL(self):
        pass


class glWidget1(glWidget0):
    def paintGL(self):
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glLoadIdentity()
        glViewport(0, 0, 480, 480)
        glTranslatef(-2.5, 0.5, -5.0)
        glColor3f(0.5, 1.0, 0.5)
        glBegin(GL_POINTS)
        glVertex3f(2.0, -1.2, 0.0)
        glVertex3f(2.6, 0.0, 0.0)
        glVertex3f(2.9, -1.2, 0.0)
        glEnd()
        glFlush()


class glWidget2(glWidget0):
    def paintGL(self):
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glLoadIdentity()
        glViewport(0, 0, 480, 480)
        glTranslatef(-2.5, 0.5, -5.0)
        glColor3f(0.5, 1.0, 0.5)
        glBegin(GL_LINES)
        glVertex3f(2.0, -1.2, 0.0)
        glVertex3f(2.6, 0.0, 0.0)
        glVertex3f(2.9, -1.2, 0.0)
        glVertex3f(2.9, 1.2, 0.0)
        glEnd()
        glFlush()


class glWidget3(glWidget0):
    def paintGL(self):
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glLoadIdentity()
        glViewport(0, 0, 480, 480)
        glTranslatef(-2.5, 0.5, -5.0)
        glColor3f(0.5, 1.0, 0.5)
        glEnable(GL_LINE_STIPPLE)
```

```
        glLineStipple(1, 0x0101)
        glBegin(GL_LINE_STRIP)
        glVertex3f(2.0, -1.2, 0.0)
        glVertex3f(2.6, 0.0, 0.0)
        glVertex3f(2.9, -1.2, 0.0)
        glVertex3f(2.9, 1.2, 0.0)
        glEnd()
        glFlush()


class glWidget4(glWidget0):
    def paintGL(self):
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glLoadIdentity()
        glViewport(0, 0, 480, 480)
        glTranslatef(-2.5, 0.5, -5.0)
        glColor3f(0.5, 1.0, 0.5)
        glEnable(GL_LINE_STIPPLE)
        glLineStipple(1, 0x00FF)
        glBegin(GL_LINE_LOOP)
        glVertex3f(2.0, -1.2, 0.0)
        glVertex3f(2.6, 0.0, 0.0)
        glVertex3f(2.9, -1.2, 0.0)
        glVertex3f(2.9, 1.2, 0.0)
        glEnd()
        glFlush()


class glWidget5(glWidget0):
    def paintGL(self):
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glLoadIdentity()
        glViewport(0, 0, 480, 480)
        glTranslatef(-2.5, 0.5, -5.0)
        glColor3f(0.5, 1.0, 0.5)
        glPolygonMode(GL_FRONT, GL_FILL)
        glBegin(GL_TRIANGLES)
        glVertex3f(2.0, -1.2, 0.0)
        glVertex3f(2.6, 0.0, 0.0)
        glVertex3f(2.9, -1.2, 0.0)
        glEnd()
        glFlush()


class glWidget6(glWidget0):
    def paintGL(self):
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glLoadIdentity()
        glViewport(0, 0, 480, 480)
        glTranslatef(-2.5, 0.5, -5.0)
        glColor3f(0.5, 1.0, 0.5)
        glPolygonMode(GL_FRONT_AND_BACK, GL_LINE)
        glBegin(GL_TRIANGLE_STRIP)
        glVertex3f(2.0, -1.2, 0.0)
        glVertex3f(2.6, 0.0, 0.0)
        glVertex3f(2.9, -1.2, 0.0)
        glVertex3f(3.9, 0.2, 0.0)
```

```
        glVertex3f(3.9, -1.2, 0.0)
        glEnd()
        glFlush()


class glWidget7(glWidget0):
    def paintGL(self):
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glLoadIdentity()
        glViewport(0, 0, 480, 480)
        glTranslatef(-2.5, 0.5, -5.0)
        glColor3f(0.5, 1.0, 0.5)
        glPolygonMode(GL_FRONT_AND_BACK, GL_LINE)
        glBegin(GL_TRIANGLE_FAN)
        glVertex3f(2.0, -1.2, 0.0)
        glVertex3f(2.6, 0.0, 0.0)
        glVertex3f(2.9, -1.2, 0.0)
        glVertex3f(2.0, -2.2, 0.0)
        glVertex3f(0.9, -1.2, 0.0)
        glEnd()
        glFlush()


class glWidget8(glWidget0):
    def paintGL(self):
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glLoadIdentity()
        glViewport(0, 0, 480, 480)
        glTranslatef(-2.5, 0.5, -5.0)
        glColor3f(0.5, 1.0, 0.5)
        glPolygonMode(GL_FRONT_AND_BACK, GL_LINE)
        glBegin(GL_QUADS)
        glVertex3f(2.0, -1.2, 0.0)
        glVertex3f(3.0, -1.2, 0.0)
        glVertex3f(3.0, 0.2, 0.0)
        glVertex3f(2.0, 0.2, 0.0)
        glEnd()
        glFlush()


class glWidget9(glWidget0):
    def paintGL(self):
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glLoadIdentity()
        glViewport(0, 0, 480, 480)
        glTranslatef(-2.5, 0.5, -5.0)
        glColor3f(0.5, 1.0, 0.5)
        glPolygonMode(GL_FRONT_AND_BACK, GL_LINE)
        glBegin(GL_QUAD_STRIP)
        glVertex3f(2.0, -1.2, 0.0)
        glVertex3f(2.0, 0.2, 0.0)
        glVertex3f(3.0, -1.2, 0.0)
        glVertex3f(3.0, 0.2, 0.0)
        glVertex3f(4.0, -1.2, 0.0)
        glVertex3f(4.0, 0.2, 0.0)
        glEnd()
        glFlush()
```

```
class glWidget10(glWidget0):
    def paintGL(self):
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glLoadIdentity()
        glViewport(0, 0, 480, 480)
        glTranslatef(-2.5, 0.5, -5.0)
        glColor3f(0.5, 1.0, 0.5)
        glPolygonMode(GL_FRONT, GL_FILL)
        glBegin(GL_POLYGON)
        glVertex2f(2.0, -0.2)
        glVertex2f(2.5, 0.7)
        glVertex2f(3.0, 1.2)
        glVertex2f(4.0, 0.2)
        glVertex2f(2.8, -0.5)
        glEnd()
        glFlush()


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