МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №1
по дисциплине «Компьютерная графика»
Тема: Примитивы OpenGL

Студент гр. 0382 Корсунов А.А.

Преподаватель Герасимова Т.В.

Санкт-Петербург
2023

Цель работы.

Ознакомиться с основными примитивами OpenGL, освоить возможности подключений графической библиотеки в среду разработки, разработать программу с использованием требуемых примитивов и атрибутов.

Теоретические положения.

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.

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

Ход работы.

1. Выбор среды разработки и подключение графической библиотеки.

Средой разработки был выбран «PyCharm», с пакетами «pyopengltk», «OpenGL» для прорисовки примитивов и пакетом «tkinter» для создание интерфейса пользователя.

В установленной IDE для установки требуемых пакетов необходимо зайти в настройки проекта в секцию «Python Interpreter», найти пакет «pyopengltk» и установить его для проекта (с этим пакетом будет установлен и PyOpenGL. Если tkinter не подключен к проекту, то его таким же образом можно установить. Более подробный алгоритм представлен на скриншотах ниже.

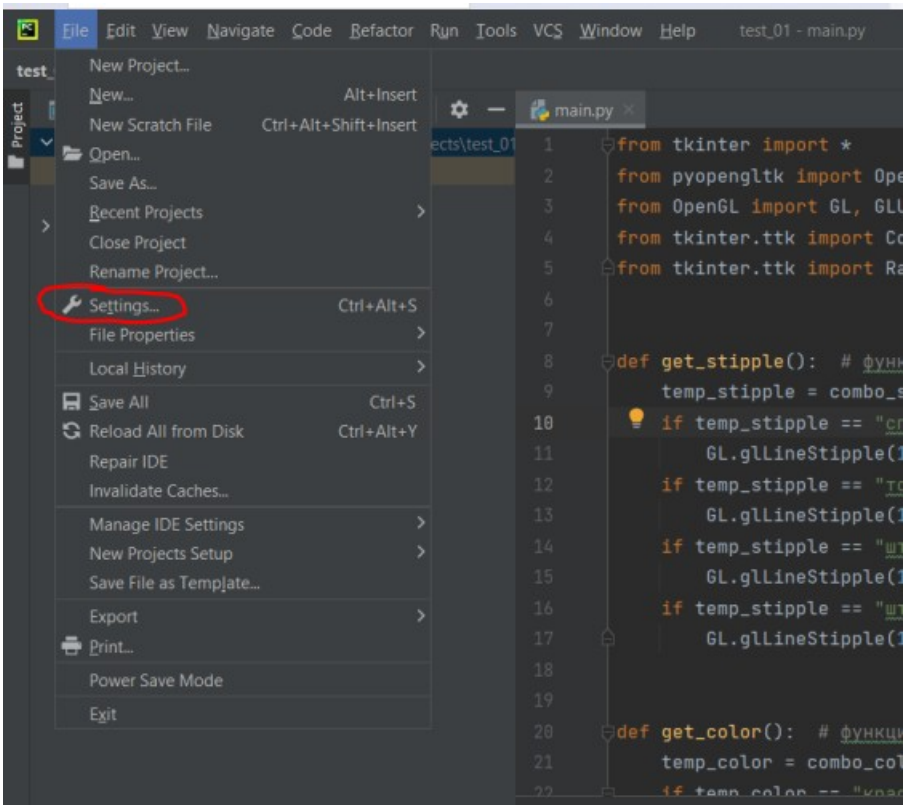

Рисунок 1 — Подключение пакетов, настройки

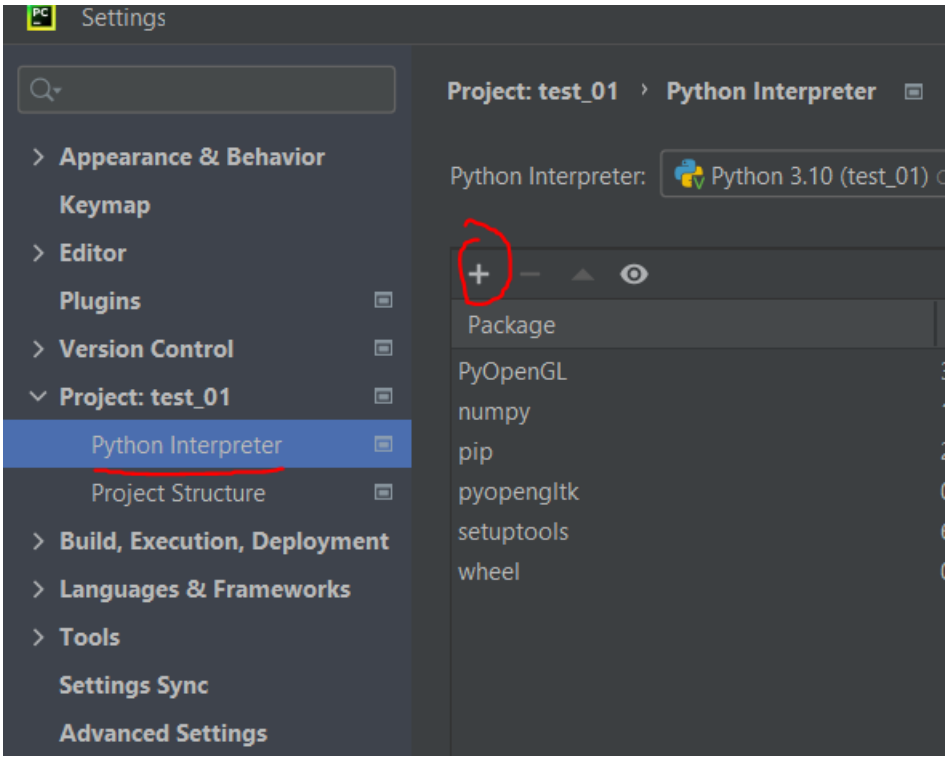

Рисунок 2 — Подключение пакетов, добавление пакета

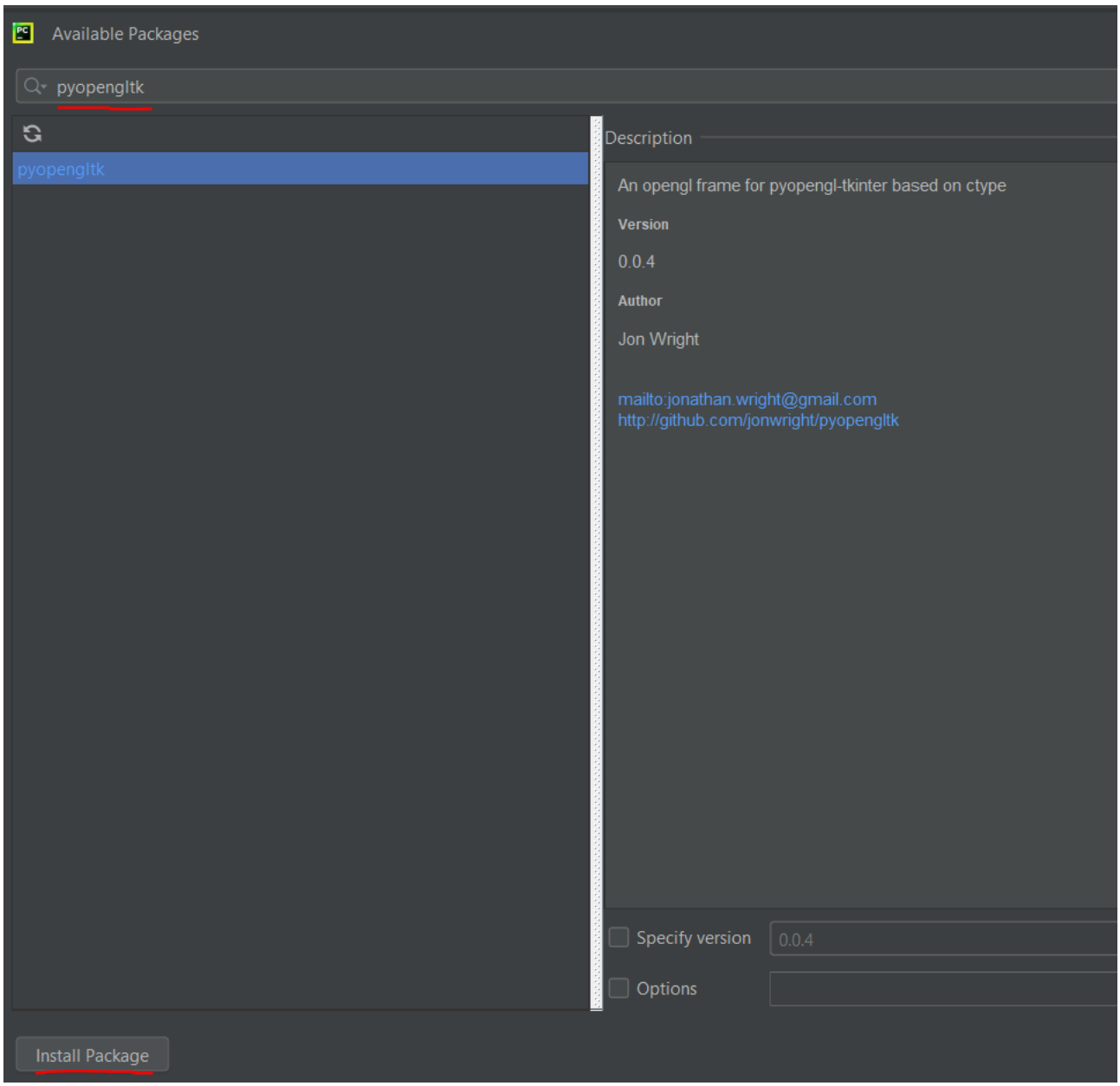

Рисунок 3 — Подключение пакетов, установка

После успешной установки можно будет воспользоваться пакетами так же, как и любыми другими на питоне:

```
from tkinter import *
from pyopengltk import OpenGLFrame
from OpenGL import GL, GLU
from tkinter.ttk import Combobox
from tkinter.ttk import Radiobutton
```

Рисунок 4 — Подключенные пакеты

2. Написание программы, реализующая представление примитивов и изменение их атрибутов.

Интерфейс программы и взаимодействие с ним описаны ниже:

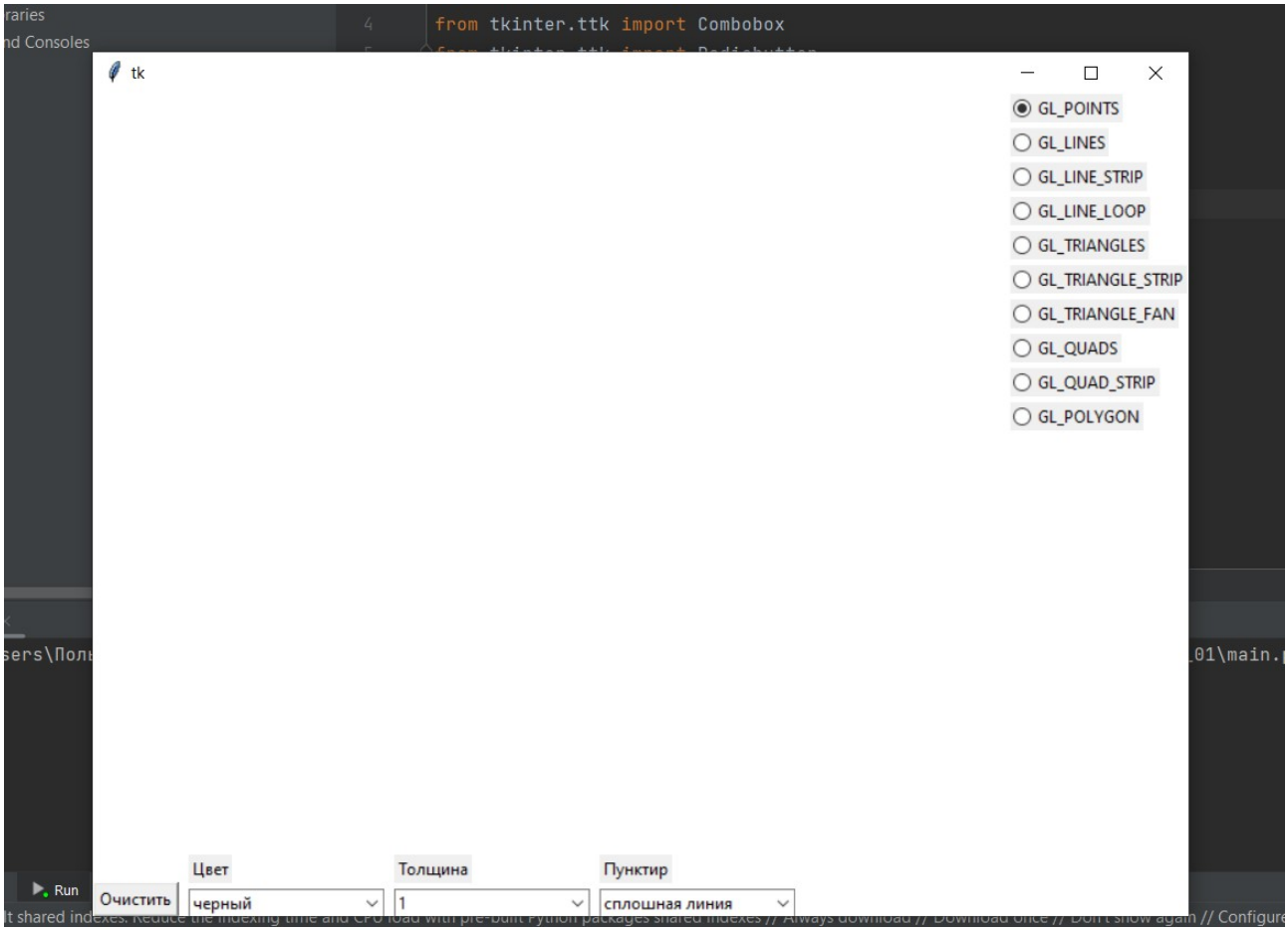

Рисунок 5 — Интерфейс пользователя

Как можно увидеть, итоговым приложением является окно с виджетами, справа находятся режимы рисования примитивов (все, что требовались по заданию). Для создания вершины нужно нажать правую кнопку мыши на часть окна, в котором пользователь хочет ее задать.

Снизу есть:

- кнопка «Очистить», с помощью которой можно очистить окно от созданных примитивов
- виджет цвета, с помощью которого можно изменить цвет вершин

- виджет толщины, с помощью которого можно изменить толщину линий у соответствующих примитивов
- виджет пунктира, с помощью которого можно изменить пунктир линий

а) Код, отвечающих за прорисовку примитивов:

```
class DrawingWindow(OpenGLFrame):  # создание класса на основе пакета pyopengltk

   def initgl(self):  # инициализация
      GL.glClear(GL.GL_COLOR_BUFFER_BIT)  # очистка буфферов от цветов ~ очищение экарана
      GL.glClearColor(1, 1, 1, 0)  # задает цвет, в который окно будет окрашиваться при его
очистке ~ очищение
      # цветопередачи
      GL.glMatrixMode(GL.GL_PROJECTION)  # матрица проекции (для проецирования 3D
прсостранства в 2D)
      GL.glLoadIdentity()  # единичная матрица ~ очистика
      GLU.gluOrtho2D(0, window_width, window_height, 0)  # смещецение оси координат
(чтобы не возникало искажения при
      # отрисовке

   def redraw(self):  # перерисовка
      GL.glClear(GL.GL_COLOR_BUFFER_BIT)
      draw()  # функция для отрисовки
```

Был создан специальный класс DrawingWindow, инициализирующий начальную настройку окна (метод initgl): очистка буффера, выбор режима матрицы, смещение оси координат) и метод для перерисовки (redraw), использующий функцию draw

часть функции draw:

```
...
GL.glBegin(primitive)  # непосредстенно отрисова виджета
get_color()  # выбранный цвет вершин
GL.glColor3f(color[0], color[1], color[2])  # установка цвета
for i in range(len(coordinates)):  # в цикле отрисовка вершин по координатам из списка
   GL.glVertex3f(coordinates[i][0], coordinates[i][1], 0.0)
GL.glEnd()
...
```

В представленной части используется конструкция для задания примитива (glBegin(<Название примитива>)...glEnd()), на вход этой конструкции подается примитив через переменную primitive, которую задает пользователь через специальные виджеты (рис. 5, справа-сверху).

б) Часть кода, отвечающего за пользовательский интерфейс:

```
…
label_thickness = Label(text="Толщина")  # виджеты для выбора толщины линий
label_thickness.place(x=220, y=555)
combo_thickness = Combobox(app)
combo_thickness['values'] = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
combo_thickness['state'] = 'readonly'
combo_thickness.current(0)
combo_thickness.place(x=220, y=580)
…
```

Как уже было упоминалось выше, для создания пользовательсного интерфейса была использована библиотека «tkinter». В коде выше создается combobox (при нажатии на него, будет представлен выбор из нескольких вариантов с возможностью выбора одного) и label (надпись над прошлым элементом). Так создается виджет, с помощью которого пользователь может изменить толщину линий. Таким же образом был написан код для остальных виджетов.

в) Демонстрация создания примитивов с помощью созданного приложения:

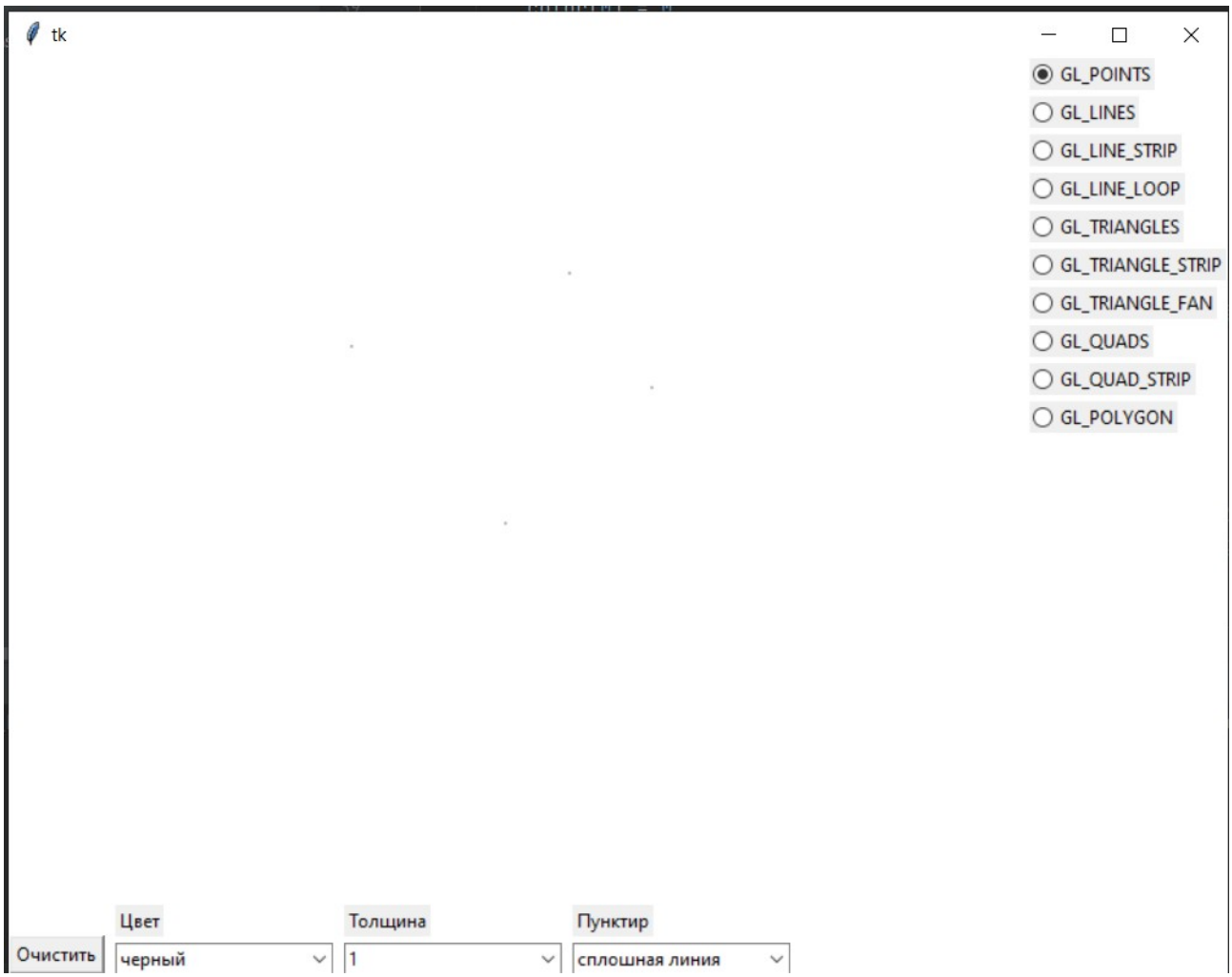

Рисунок 6 - GL_POTNS

Щелкнув левой кнопкой мышью по четырем частям окна были получены четыре точки. Далее можно поменять режим на любой другой:

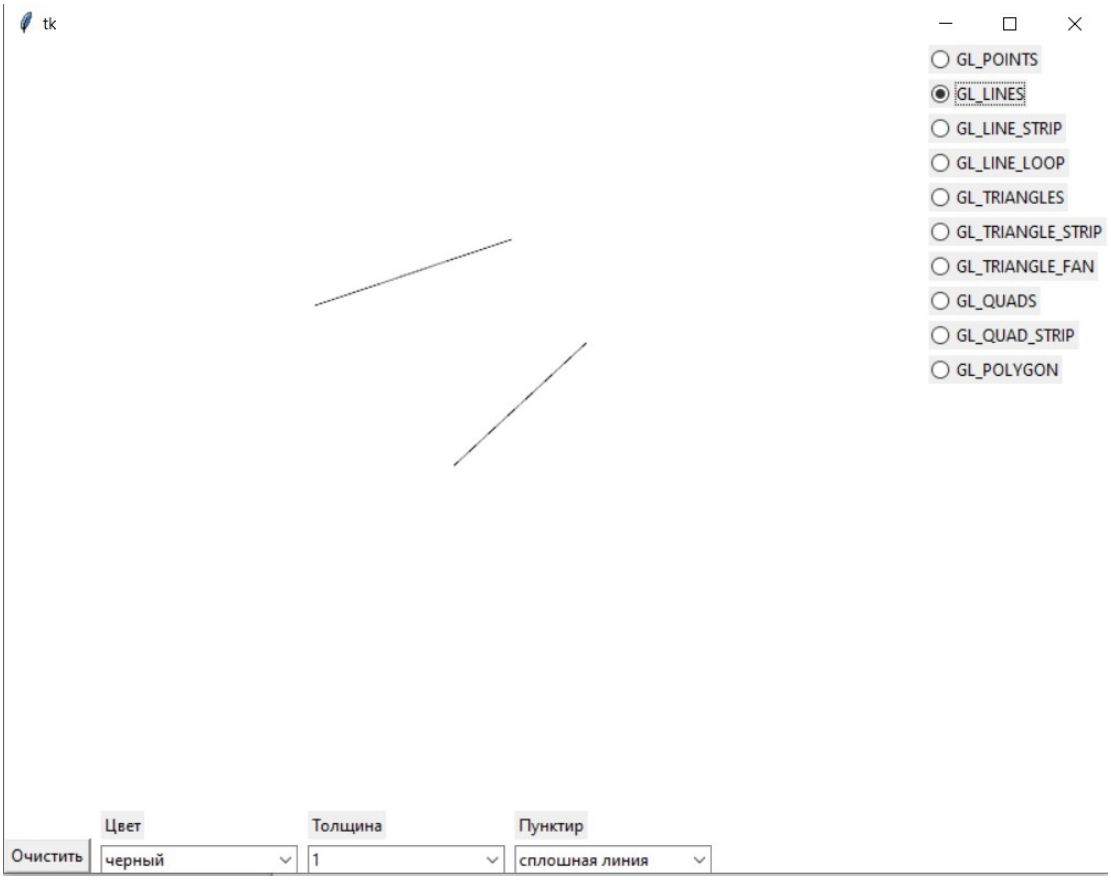

Рисунок 7 - GL_LINES

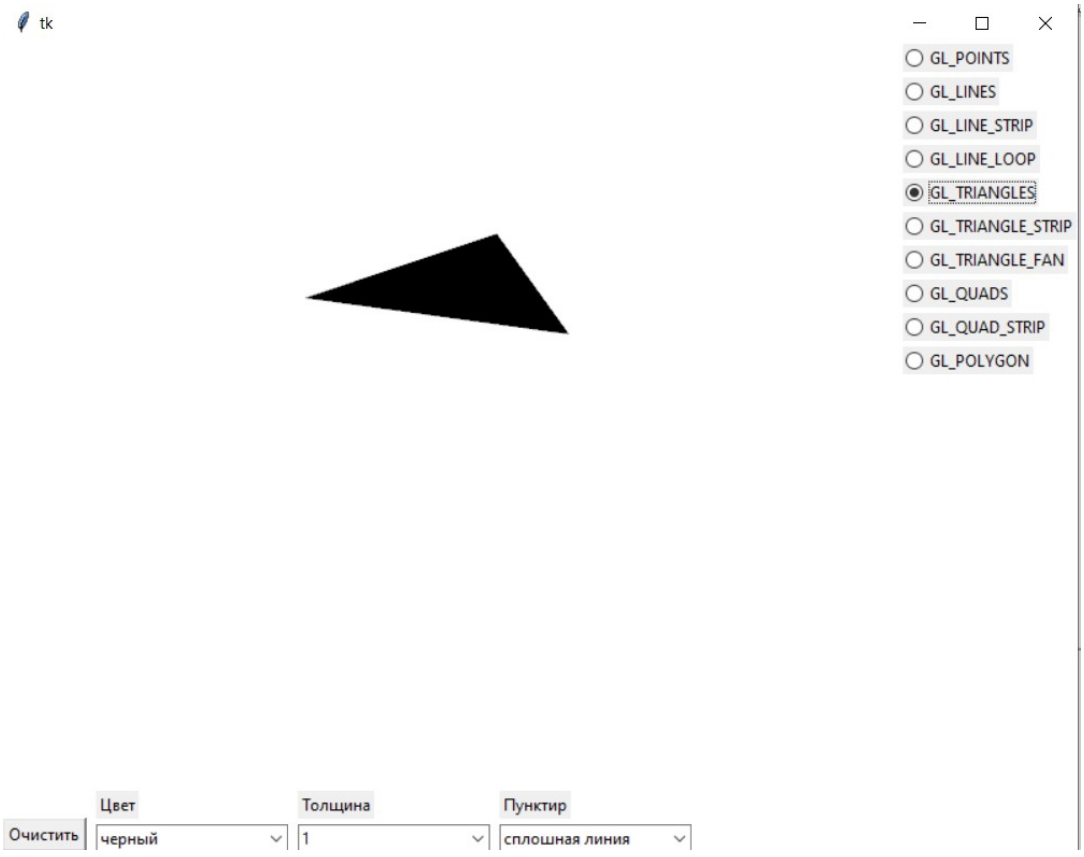

Рисунок 8 — GL_TRIANGLES

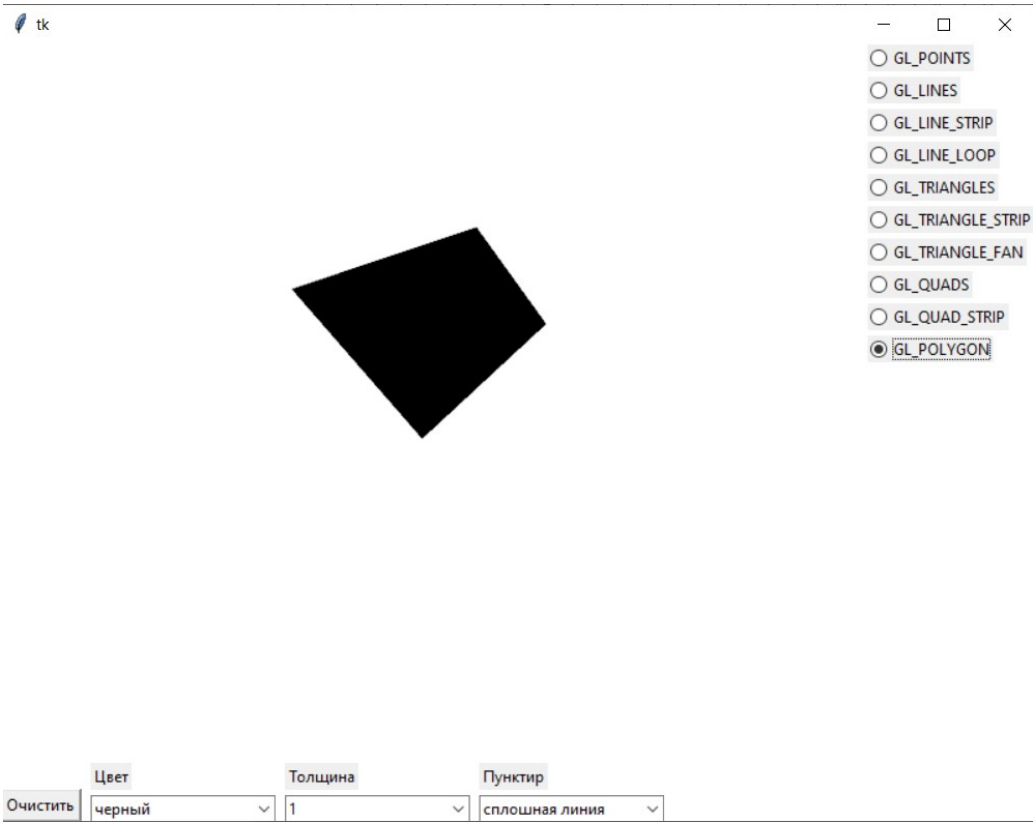

Рисунок 9 — GL_POLYGON

Далее можно добавить новые вершины и сменить цвет на красный:

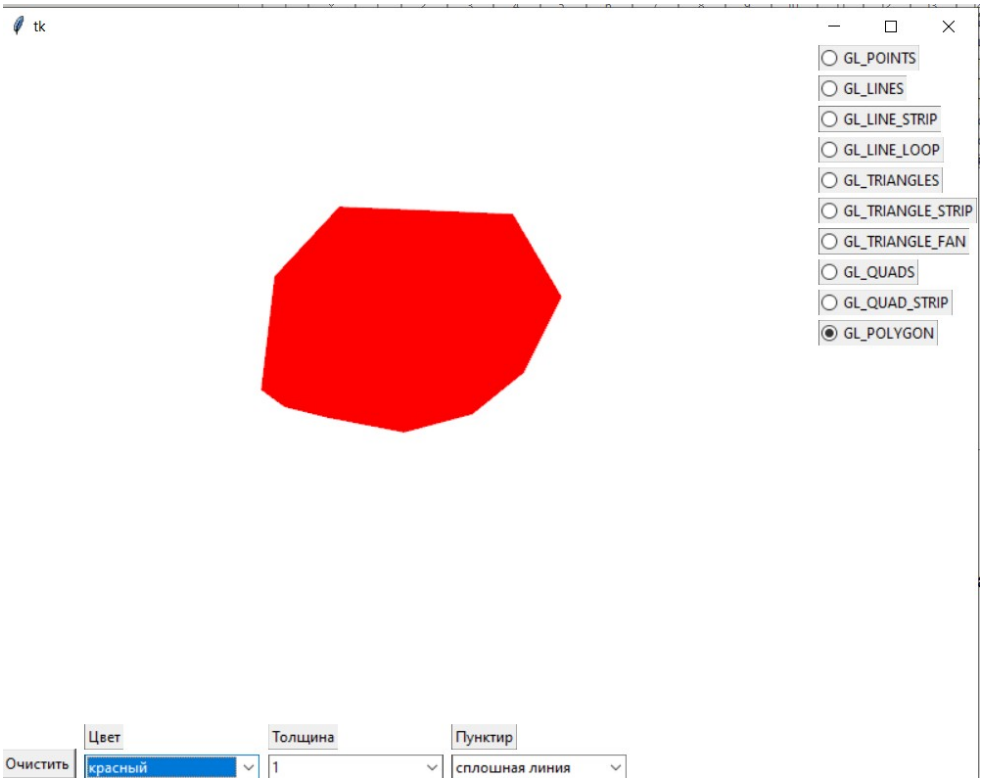

Рисунок 10 — Добавление точек и смена цвета

Далее можно выбрать режим на GL_LINE_LOOP увеличить толщину линий и поменять пунктир:

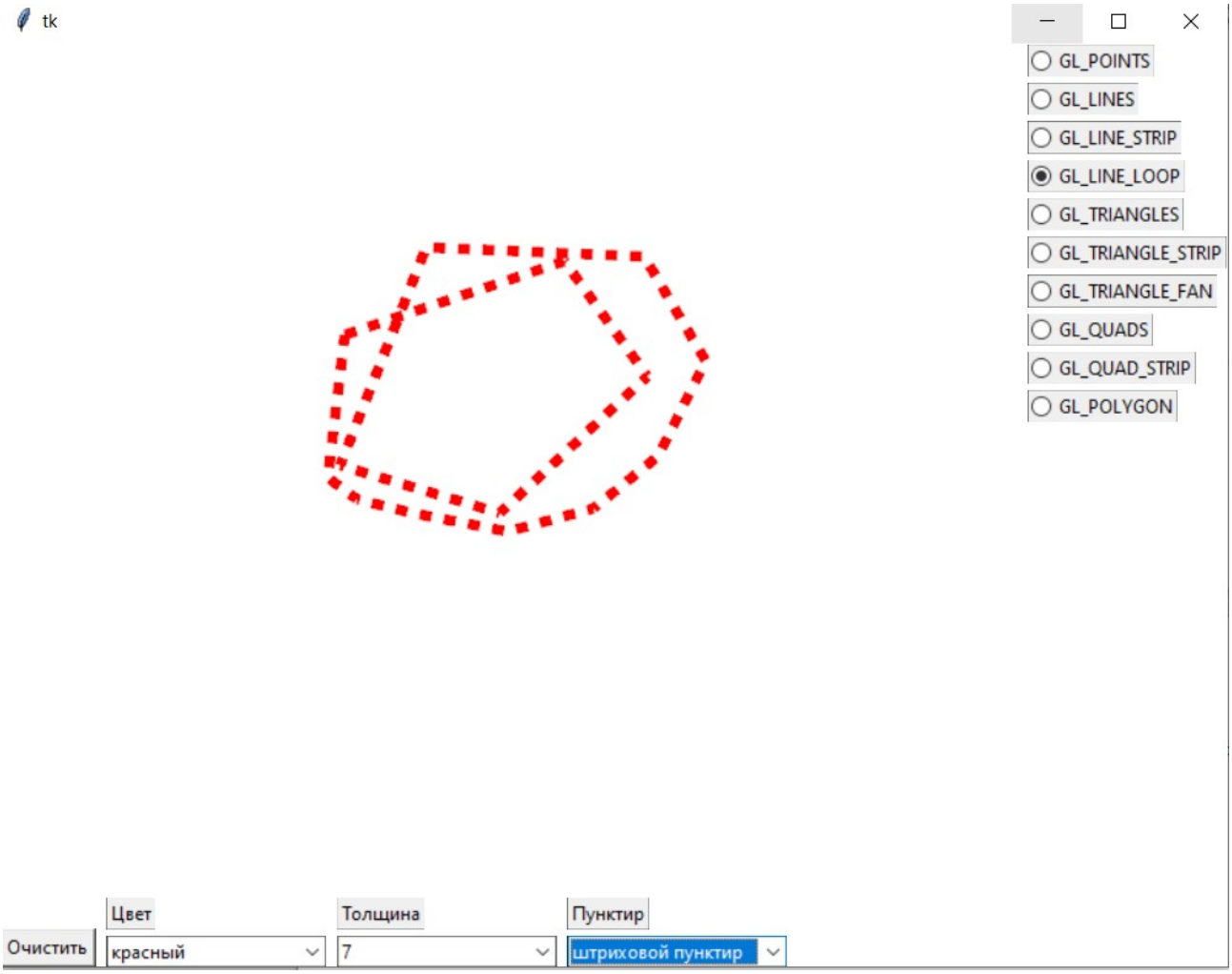

Рисунок 11 — Смена режима, толщины и пунктира

Если необходим другой набор вершин, то пользователь может нажать на кнопку «Очистить» и заново задать вершины с помощью мыши:

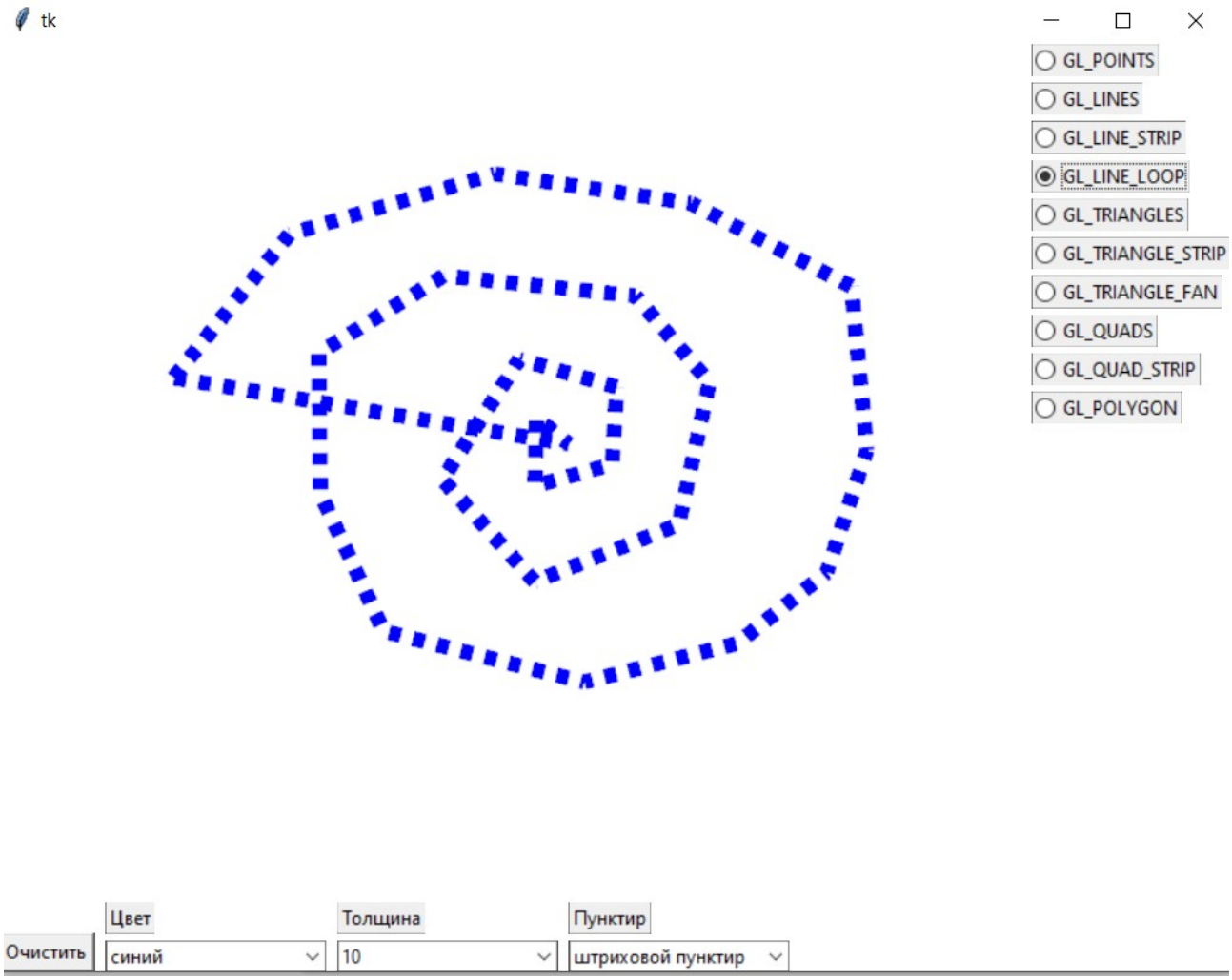

Рисунок 12 — Очистка и задание новых вершин с другими атрибутами

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.

Файл main.pi

```
from tkinter import *
from pyopengltk import OpenGLFrame
from OpenGL import GL, GLU
from tkinter.ttk import Combobox
from tkinter.ttk import Radiobutton


def get_stipple():  # функция для установки пунктира
    temp_stipple = combo_stipple.get()
    if temp_stipple == "сплошная линия":
        GL.glLineStipple(1, 0xFFFF)
    if temp_stipple == "точечный пунктир":
        GL.glLineStipple(1, 0x0101)
    if temp_stipple == "штриховой пунктир":
        GL.glLineStipple(1, 0x00FF)
    if temp_stipple == "штрих точка штрих":
        GL.glLineStipple(1, 0x1C47)


def get_color():  # функция для установки цвета
    temp_color = combo_color.get()
    if temp_color == "красный":
        color[0] = 255
        color[1] = 0
        color[2] = 0
    elif temp_color == "синий":
        color[0] = 0
        color[1] = 0
        color[2] = 255
    elif temp_color == "зеленый":
        color[0] = 0
        color[1] = 255
        color[2] = 0
    elif temp_color == "желтый":
        color[0] = 255
        color[1] = 255
        color[2] = 0
    elif temp_color == "черный":
        color[0] = 0
        color[1] = 0
        color[2] = 0


def clear():  # очистка списка координат вершин
    coordinates.clear()
```

```
def click(event):  # добавление координаты вершины по клику в окне
    coordinates.append((event.x, event.y))


def draw():  # отрисовка
    primitive = GL.GL_POINT  # примитив по умолчанию
    line_width = combo_thickness.get()  # получение значения из виджета толщины
    temp = selected.get()  # получение примитива из выбранного виджета

    if temp == 0:  # установка примитивов для рисования (+толщины и типа линий для
примитивов вида линий)
        primitive = GL.GL_POINTS
    elif temp == 1:
        primitive = GL.GL_LINES
        GL.glLineWidth(int(line_width))
        GL.glEnable(GL.GL_LINE_STIPPLE)
        get_stipple()
    elif temp == 2:
        primitive = GL.GL_LINE_STRIP
        GL.glLineWidth(int(line_width))
        GL.glEnable(GL.GL_LINE_STIPPLE)
        get_stipple()
    elif temp == 3:
        primitive = GL.GL_LINE_LOOP
        GL.glLineWidth(int(line_width))
        GL.glEnable(GL.GL_LINE_STIPPLE)
        get_stipple()
    elif temp == 4:
        primitive = GL.GL_TRIANGLES
    elif temp == 5:
        primitive = GL.GL_TRIANGLE_STRIP
    elif temp == 6:
        primitive = GL.GL_TRIANGLE_FAN
    elif temp == 7:
        primitive = GL.GL_QUADS
    elif temp == 8:
        primitive = GL.GL_QUAD_STRIP
    elif temp == 9:
        primitive = GL.GL_POLYGON

    GL.glBegin(primitive)  # непосредстенно отрисова виджета
    get_color()  # выбранный цвет вершин
    GL.glColor3f(color[0], color[1], color[2])  # установка цвета
    for i in range(len(coordinates)):  # в цикле отрисовка вершин по координатам из списка
        GL.glVertex3f(coordinates[i][0], coordinates[i][1], 0.0)
    GL.glEnd()
    GL.glFlush()
```

```
class DrawingWindow(OpenGLFrame):  # создание класса на основе пакета pyopengltk

    def initgl(self):  # инициализация
        GL.glClear(GL.GL_COLOR_BUFFER_BIT)  # очистка буфферов от цветов ~ очищение экарана
        GL.glClearColor(1, 1, 1, 0)  # задает цвет, в который окно будет окрашиваться при его очистке ~ очищение
        # цветопередачи
        GL.glMatrixMode(GL.GL_PROJECTION)  # матрица проекции (для проецирования 3D прсостранства в 2D)
        GL.glLoadIdentity()  # единичная матрица ~ очистика
        GLU.gluOrtho2D(0, window_width, window_height, 0)  # смещецение оси координат (чтобы не возникало искажения при
        # отрисовке

    def redraw(self):  # перерисовка
        GL.glClear(GL.GL_COLOR_BUFFER_BIT)
        draw()  # функция для отрисовки


root = Tk()  # главное окно

window_width = 800  # размеры окна (ширина и высота)
window_height = 600
coordinates = []  # (координаты вершин)
color = [0, 0, 0]  # (цвет  вершин)

app = DrawingWindow(root, width=window_width, height=window_height)  # создание окна для отрисови
app.bind('<Button 1>', click)  # биндим ПКМ по созданному окну, чтобы запоминать координаты клика
app.pack(fill=BOTH, expand=YES)  # отобразить
app.animate = 1  # для отрисовки в реальном времени

selected = IntVar()  # сюда помещается значение выбранного примитива
rad1 = Radiobutton(app, text='GL_POINTS', value=0, variable=selected)  #радиокнопки для выбора примитива
rad2 = Radiobutton(app, text='GL_LINES', value=1, variable=selected)
rad3 = Radiobutton(app, text='GL_LINE_STRIP', value=2, variable=selected)
rad4 = Radiobutton(app, text='GL_LINE_LOOP', value=3, variable=selected)
rad5 = Radiobutton(app, text='GL_TRIANGLES', value=4, variable=selected)
rad6 = Radiobutton(app, text='GL_TRIANGLE_STRIP', value=5, variable=selected)
rad7 = Radiobutton(app, text='GL_TRIANGLE_FAN', value=6, variable=selected)
rad8 = Radiobutton(app, text='GL_QUADS', value=7, variable=selected)
rad9 = Radiobutton(app, text='GL_QUAD_STRIP', value=8, variable=selected)
rad10 = Radiobutton(app, text='GL_POLYGON', value=9, variable=selected)

rad1.place(x=670, y=0)  # их отрбражение
```

```
rad2.place(x=670, y=25)
rad3.place(x=670, y=50)
rad4.place(x=670, y=75)
rad5.place(x=670, y=100)
rad6.place(x=670, y=125)
rad7.place(x=670, y=150)
rad8.place(x=670, y=175)
rad9.place(x=670, y=200)
rad10.place(x=670, y=225)

btn = Button(app, text="Очистить", command=clear)  # кнопка для очистки
btn.place(x=0, y=575)

label_color = Label(text="Цвет")  # виджеты для выбора цвета вершин
label_color.place(x=70, y=555)
combo_color = Combobox(app)
combo_color['values'] = ("красный", "синий", "зеленый", "желтый", "черный")
combo_color['state'] = 'readonly'
combo_color.current(4)
combo_color.place(x=70, y=580)

label_thickness = Label(text="Толщина")  # виджеты для выбора толщины линий
label_thickness.place(x=220, y=555)
combo_thickness = Combobox(app)
combo_thickness['values'] = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
combo_thickness['state'] = 'readonly'
combo_thickness.current(0)
combo_thickness.place(x=220, y=580)

label_stipple = Label(text="Пунктир")  # виджеты для выбора типа пунктира
label_stipple.place(x=370, y=555)
combo_stipple = Combobox(app)
combo_stipple['values'] = ("сплошная линия", "точечный пунктир", "штриховой пунктир",
"штрих точка штрих")
combo_stipple['state'] = 'readonly'
combo_stipple.current(0)
combo_stipple.place(x=370, y=580)

app.mainloop()  # запуск главного цикла
```