

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

ОТЧЕТ
по лабораторным работам №1-2
по дисциплине «Компьютерная графика»
Тема: Прimitives OpenGL

Студентка гр. 1304

Чернякова В.А.

Студентка гр. 1304

Ярусова Т.В.

Преподаватель

Герасимова Т.В.

Санкт-Петербург

2024

Цель работы.

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

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*), а также реализующую представление тестов отсечения (*glScissor*), прозрачности (*glAlphaFunc*), смешения цветов (*glBlendFunc*) в библиотеке *OpenGL* на базе разработанных вами в предыдущей работе примитивов.

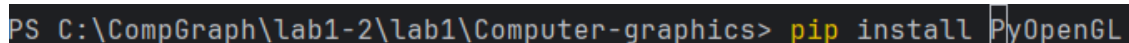
Разработанная на базе шаблона программа должна быть пополнена возможностями остановки интерактивно различных атрибутов примитивов рисования через вызов соответствующих элементов интерфейса пользователя.

Выполнение работы.

Работы была выполнена с помощью языка программирования *Python*. Использовалась библиотека *PyQt* для создания приложений с графическим интерфейсом с помощью инструментария *Qt*.

Для подключения *OpenGL* в *Python* необходимо в файле, где реализованы функции, импортировать модуль *OpenGL.GL*, который представляет собой библиотеку *OpenGL*. Прежде чем подключить модуль с помощью команды *pip install*, которая используется в среде *Python* для установки пакетов из *Python Package Index (PyPI)*, необходимо его скачать.

Процесс скачивания модуля представлен на рисунке 1.



```
PS C:\CompGraph\lab1-2\lab1\Computer-graphics> pip install PyOpenGL
```

Рисунок 1 – скачивание модуля *OpenGL.GL*

Процесс импорта представлен на рисунке 2.

```
from OpenGL.GL import *
```

Рисунок 2 – импорт модуля *OpenGL.GL*

Для отрисовки графики с помощью библиотеки *OpenGL* в *PyQt* используется *gl_widget* – экземпляр класса *GLScene*, который используется в качестве виджета *OpenGL* в главном окне (*QMainWindow*). Когда *gl_widget = GLScene()* вызывается в *MainWindow*, создается новый экземпляр *GLScene*, который становится частью пользовательского интерфейса этого окна.

Сам *GLScene* наследуется от *QGLWidget* – виджета, который интегрирует *OpenGL* в пользовательский интерфейс *PyQt*. *GLScene* представляет собой виджет *OpenGL*, который может быть встроен в пользовательский интерфейс.

Как виджет *OpenGL*, *GLScene* предоставляет методы для инициализации *OpenGL* (*initializeGL*), рендеринга графики (*paintGL*) и обновления сцены при изменении размеров виджета (*resizeGL*).

Метод *initializeGL* вызывается при создании виджета и используется для инициализации состояния *OpenGL* перед началом рисования. В листинге 1 представлено следующее:

- *glClearColor()*: устанавливает цвет очистки экрана и прозрачность.
- *glPointSize()*: устанавливает размер точек, которые будут рисоваться.
- *glLineWidth()*: устанавливает ширину линий, которые будут рисоваться.

Листинг 1. Код метода *initializeGL*

```
def initializeGL(self):  
    glClearColor(0.0, 0.0, 0.0, 1.0)  
    glPointSize(5.0)  
    glLineWidth(3.0)
```

Метод *resizeGL* вызывается при изменении размеров окна или виджета, и он используется для обновления параметров *OpenGL*, связанных с размером видимой области. В листинге 2 представлено следующее:

- *self.frameWidth = width* и *self.frameHeight = height*: сохранение размеров окна.

- `glViewport()`: устанавливает область вывода *OpenGL*, которая соответствует размеру окна. Это указывает *OpenGL*, что координаты, передаваемые в функции рисования, должны быть преобразованы таким образом, чтобы они попадали в заданный прямоугольник.
- `glMatrixMode(GL_PROJECTION)`: устанавливает текущую матрицу как матрицу проекции. Матрица проекции отвечает за преобразование координат в трехмерном пространстве в координаты экрана или изображения, которые будут видны на экране.
- `glLoadIdentity()`: загружает единичную матрицу в текущую матрицу проекции, чтобы очистить все предыдущие преобразования и начать с чистого листа.
- `glOrtho()`: устанавливает ортогональное проекционное преобразование для отображения. Если ширина меньше или равна высоте, то пространство ортогональной проекции будет выровнено по горизонтали, иначе - по вертикали.
- `glMatrixMode(GL_MODELVIEW)`: возвращает текущую матрицу в режим моделирования, которая используется для определения положения и ориентации объектов.
- `glLoadIdentity()`: снова загружает единичную матрицу в текущую матрицу моделирования, обнуляя все предыдущие преобразования.

Листинг 2. Код метода *resizeCL*

```
def resizeGL(self, width, height):
    self.frameWidth = width
    self.frameHeight = height

    glViewport(0, 0, width, height)
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    aspect_ratio = width / height
    if width <= height:
        glOrtho(-1.0, 1.0, -1.0 / aspect_ratio, 1.0 / aspect_ratio, -
1.0, 1.0)
```

```

else:
    glOrtho(-1.0 * aspect_ratio, 1.0 * aspect_ratio, -1.0, 1.0, -
1.0, 1.0)
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()

```

Метод *paintGL* используется для отрисовки. В листинге 3 представлено следующее:

- *glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT):* очищает буфер цвета и буфер глубины.
- *colors:* список цветов в формате *RGBA* (красный, зеленый, синий, альфа) для отрисовки точек.
- *num_points:* количество точек, которые нужно нарисовать.
- *radius:* радиус окружности, в пределах которой будут расположены точки.
- *glEnable(GL_ALPHA_TEST), glEnable(GL_BLEND), glEnable(GL_SCISSOR_TEST):* включают тесты и функции *OpenGL* для обработки прозрачности и обрезки изображения, смешения цветов.
- *glAlphaFunc(self.alpha, self.alpha_value):* устанавливает функцию сравнения для теста прозрачности. *self.alpha* – тип теста прозрачности, *self.alpha_value* – значение, используемое в тесте.
- *glBlendFunc(self.sfactor, self.dfactor):* устанавливает функцию смешивания цветов для операций смешивания. *self.sfactor* и *self.dfactor* – факторы источника и назначения для операции смешивания.
- *glScissor(self.x, self.y, self.width, self.height):* устанавливает область отсечения для обрезки вывода *OpenGL*. *self.x* и *self.y* – координаты левого нижнего угла прямоугольника, а *self.width* и *self.height* – его ширина и высота.
- *glBegin(self.primitiveMode):* начинает определение примитива *OpenGL*, который будет отрисован.
- В цикле *for* отрисовываются точки в форме круга. Для каждой точки вычисляются координаты *x* и *y*.
- *glEnd():* завершает определение примитива *OpenGL*.

- `glDisable(GL_SCISSOR_TEST), glDisable(GL_ALPHA_TEST), glDisable(GL_BLEND)`: отключают тесты и функции *OpenGL* после отрисовки, чтобы вернуть контекст *OpenGL* в исходное состояние.

Листинг 3. Код метода `paintGL`

```
def paintGL(self):
    glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    colors = [(0.0, 1.0, 1.0, 0.42), (1.0, 0.0, 1.0, 0.66), (0.0,
0.5, 0.0, 0.34), (0.5, 0.0, 0.0, 0.92), (0.0, 0.5, 0.5, 0.2), (0.5, 0.0,
0.5, 0.12), (0.0, 1.0, 0.0, 0.55), (1.0, 0.0, 0.0, 0.81), (1.0, 1.0, 0.0,
0.72), (0.0, 0.0, 1.0, 0.32)]
    num_points = 12
    radius = 0.7

    glEnable(GL_ALPHA_TEST)
    glEnable(GL_BLEND)
    glEnable(GL_SCISSOR_TEST)
    glAlphaFunc(self.alpha, self.alpha_value)
    glBlendFunc(self.sfactor, self.dfactor)
    glScissor(self.x, self.y, self.width, self.height)

    glBegin(self.primitiveMode)
    for point in range(num_points):
        glColor4f(colors[point % len(colors)][0], colors[point %
len(colors)][1], colors[point % len(colors)][2], colors[point %
len(colors)][3])
        angle = 2 * math.pi * point / num_points
        x = radius * math.cos(angle)
        y = radius * math.sin(angle)
        glVertex2f(x, y)
    glEnd()

    glDisable(GL_SCISSOR_TEST)
    glDisable(GL_ALPHA_TEST)
    glDisable(GL_BLEND)
```

Тестирование.

Программа была протестирована на нескольких типов примитивов, а также для различных настроек тестов прозрачности, смешения цветов и отсечения. Результаты выводов представлены на рисунках 3-18.

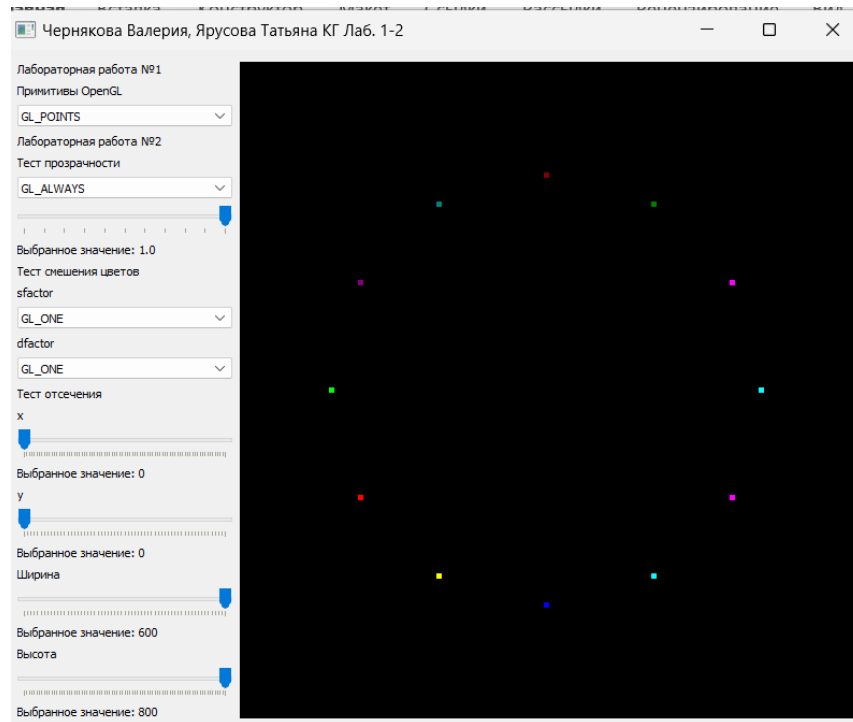


Рисунок 3 – GL_POINTS

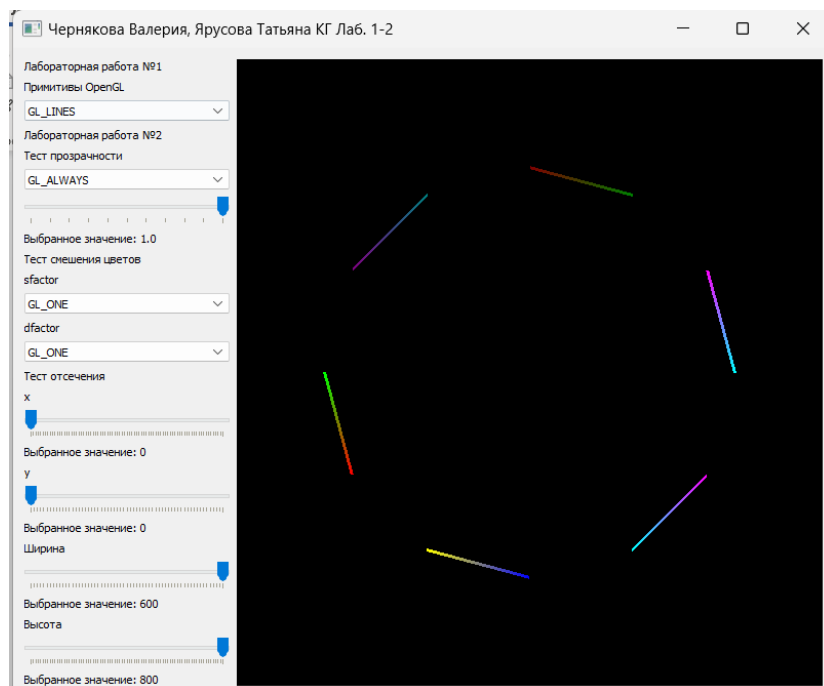


Рисунок 4 – GL_LINES

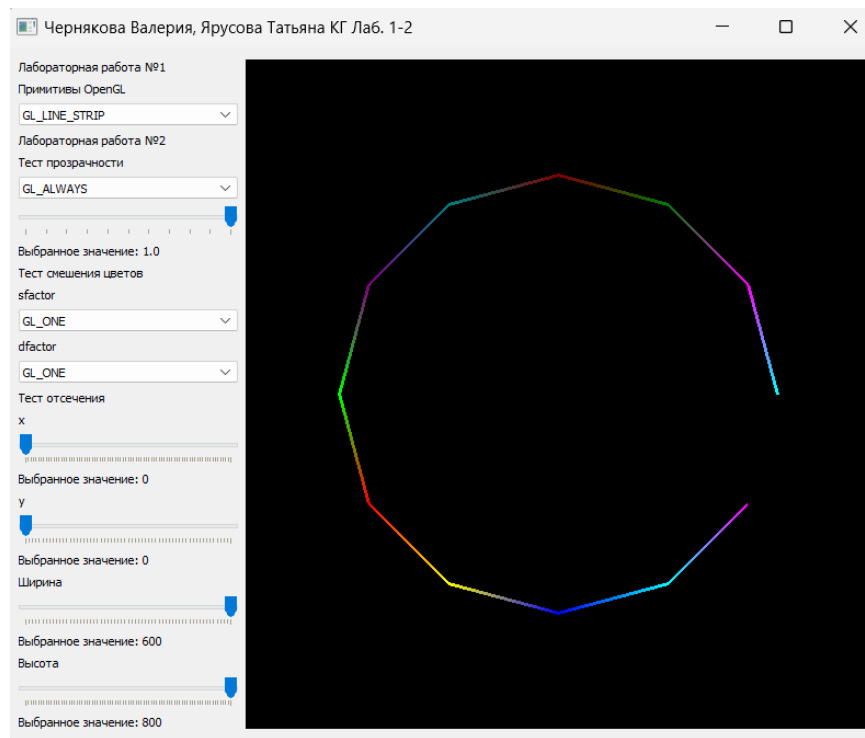


Рисунок 5 – GL_LINE_STRIP

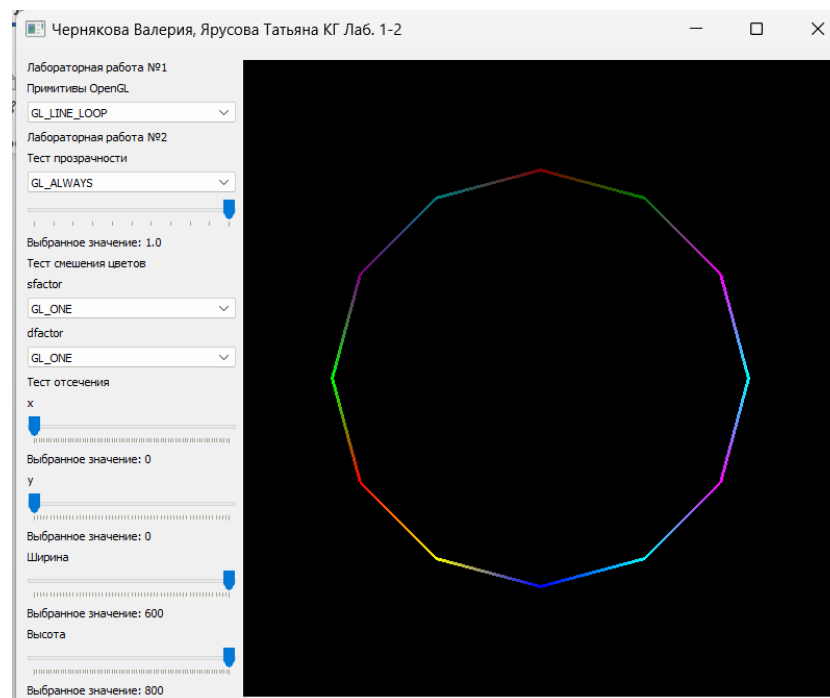


Рисунок 6 – GL_LINE_LOOP

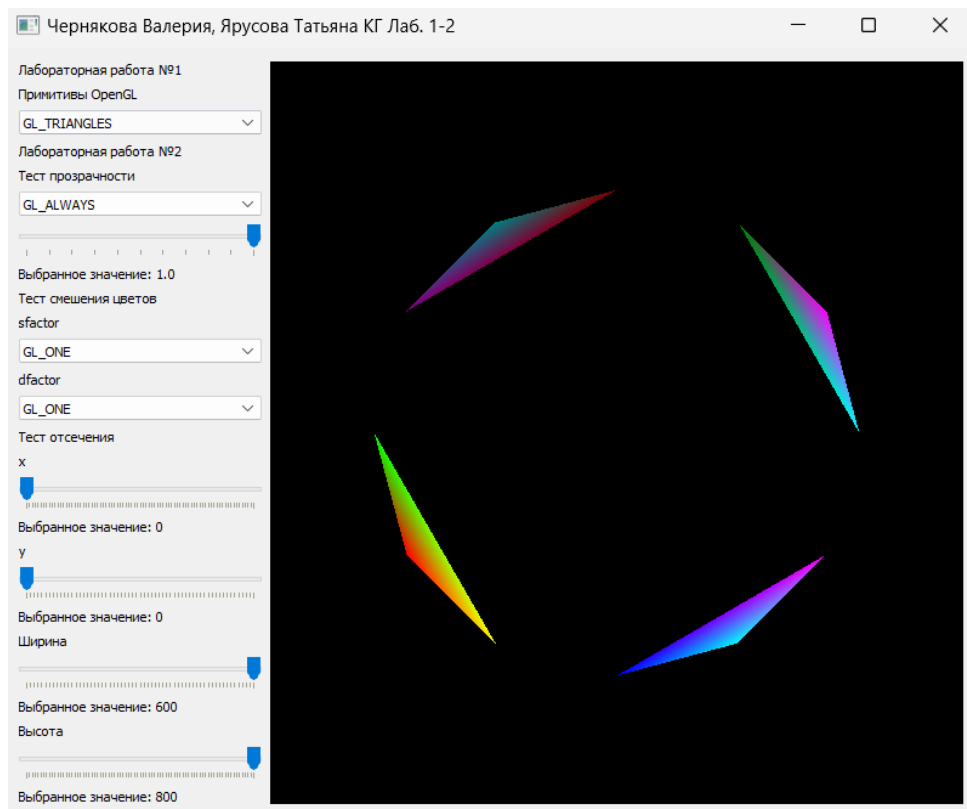


Рисунок 7 – GL_TRIANGLES

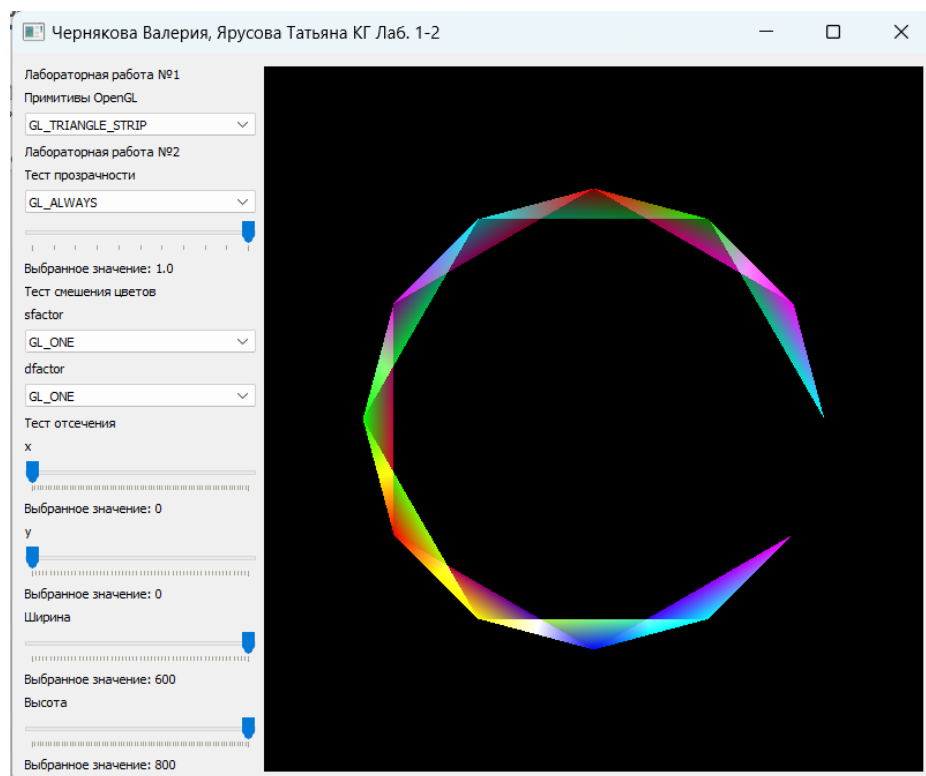


Рисунок 8 – GL_TRIANGLE_STRIP

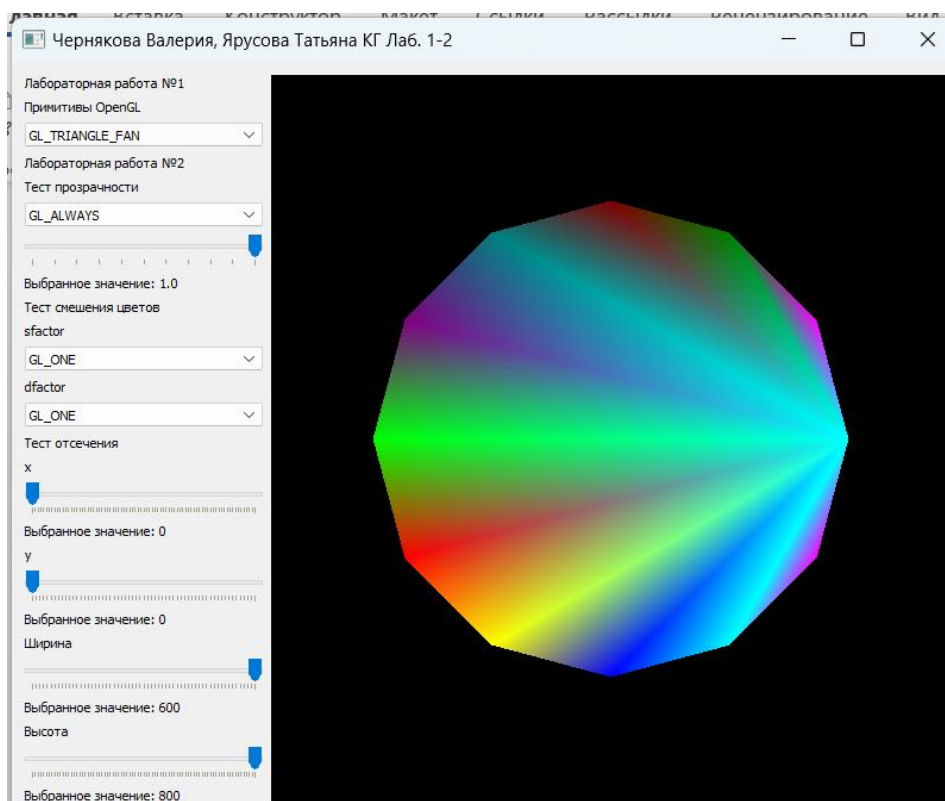


Рисунок 9 – GL_TRIANGLE_FAN

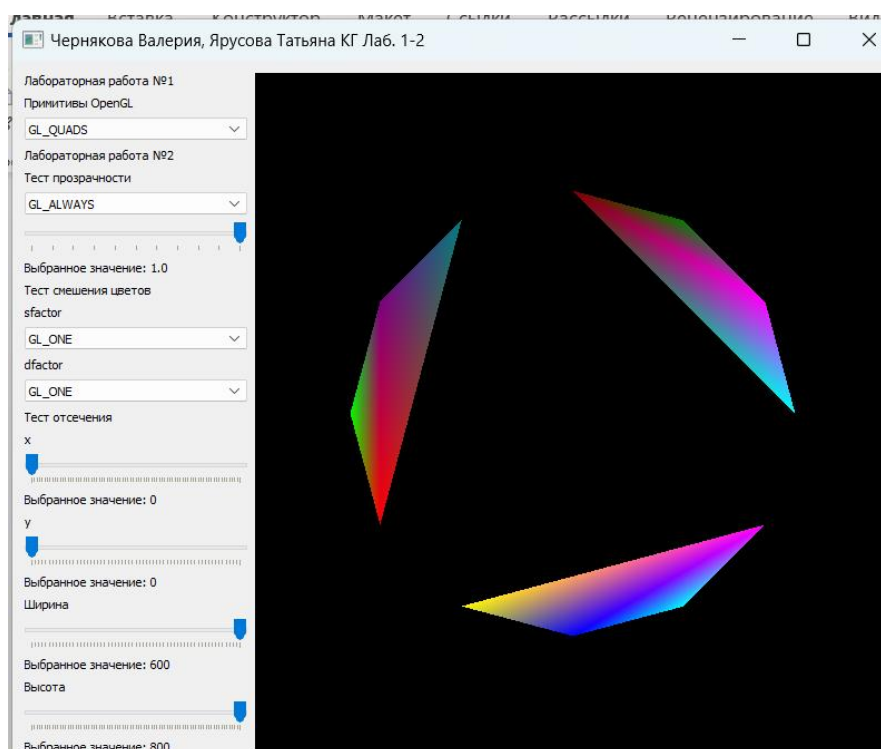


Рисунок 10 – GL_QUADS

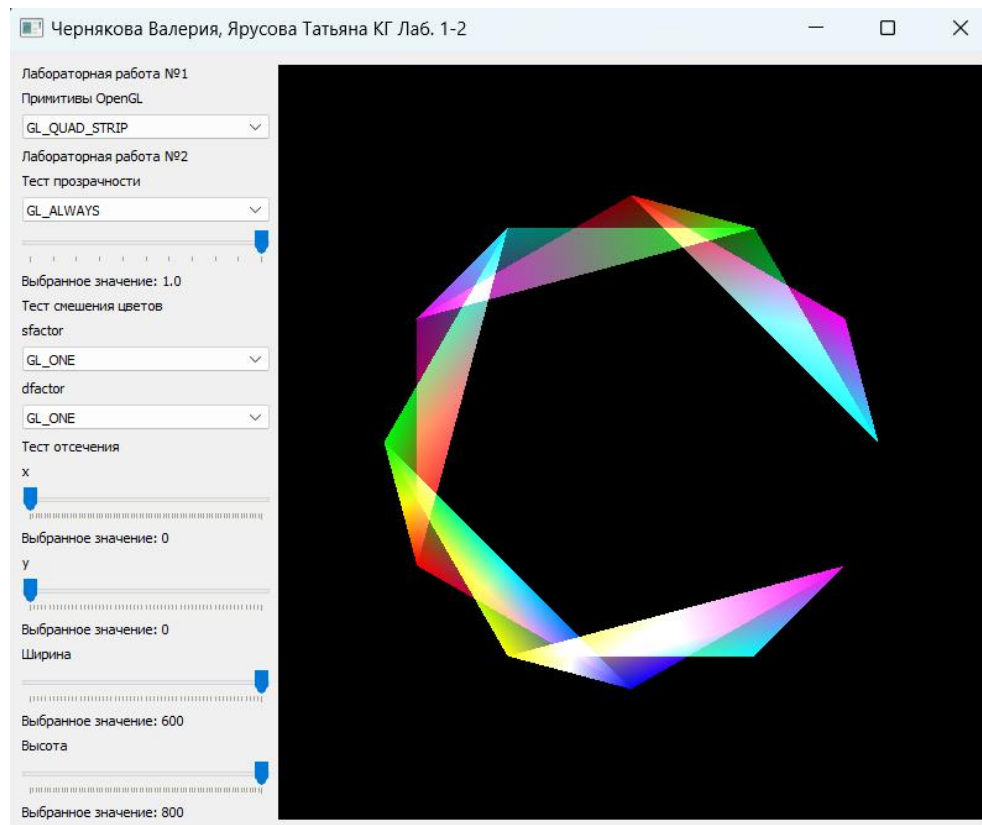


Рисунок 11 – GL_QUAD_STRIP

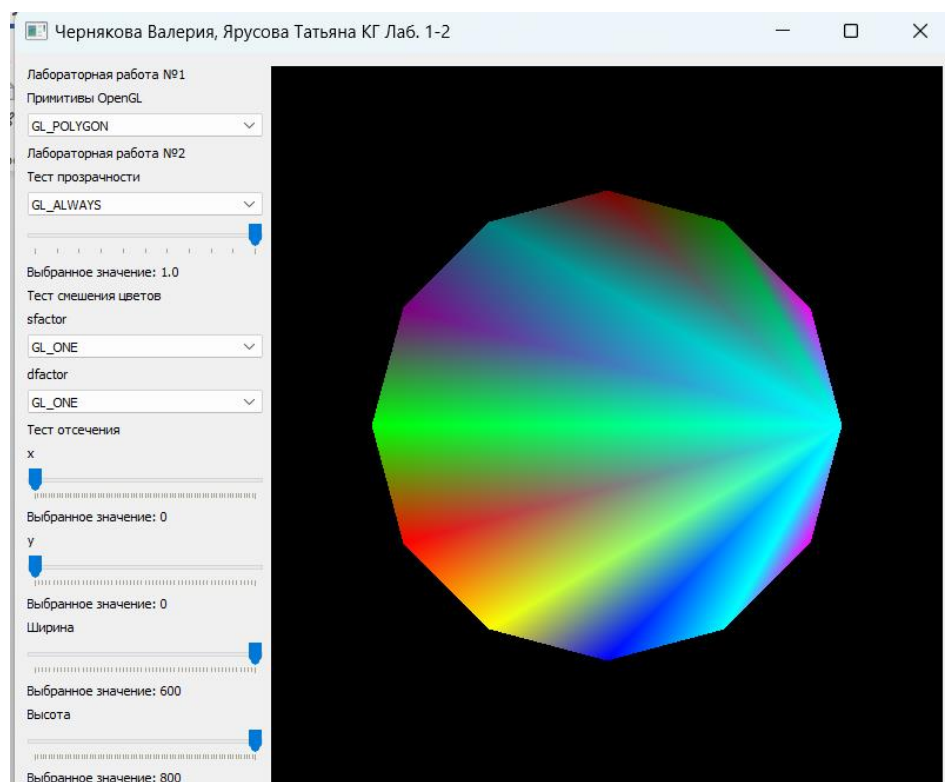


Рисунок 12 – GL_POLYGON

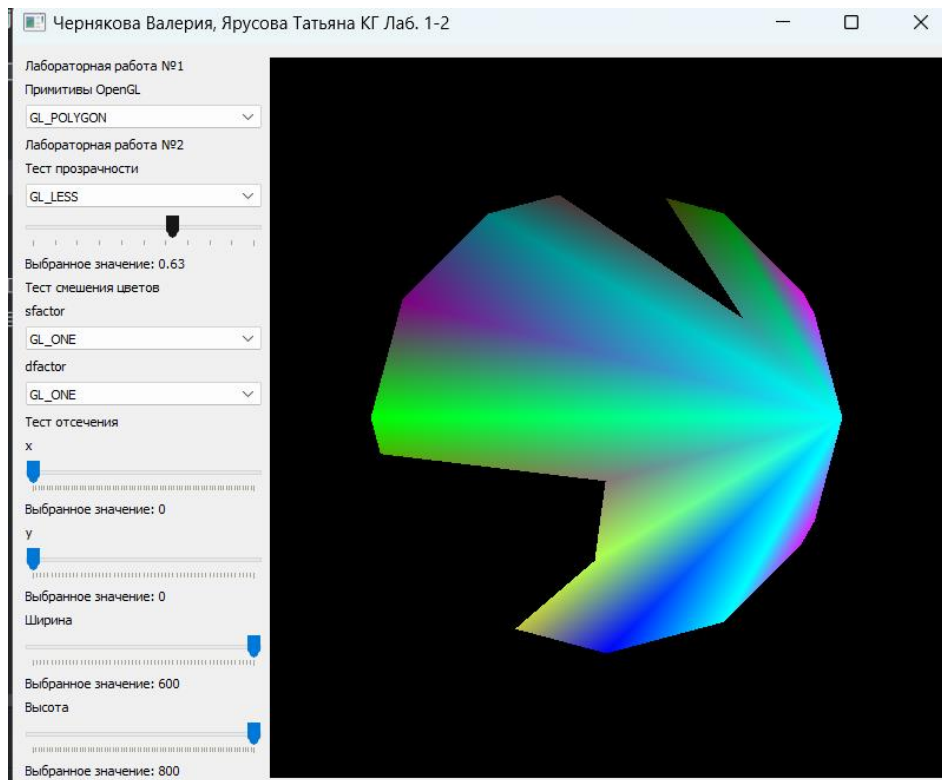


Рисунок 13 – тест прозрачности GL_LESS со значением сравнения 0.63

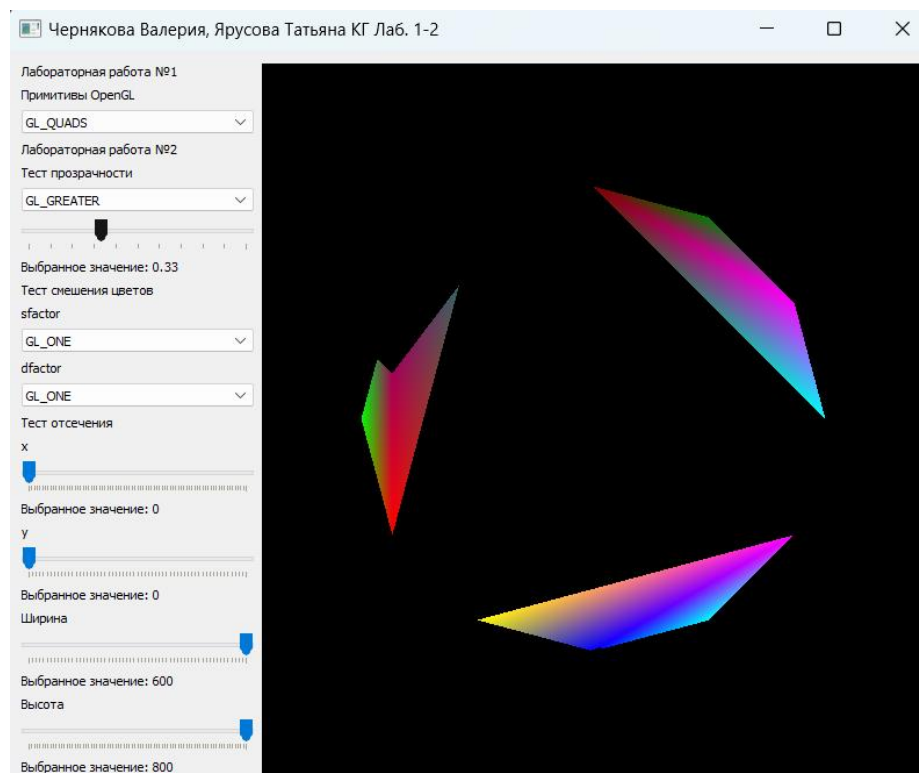


Рисунок 14 – тест прозрачности GL_GREATER со значением сравнения 0.33

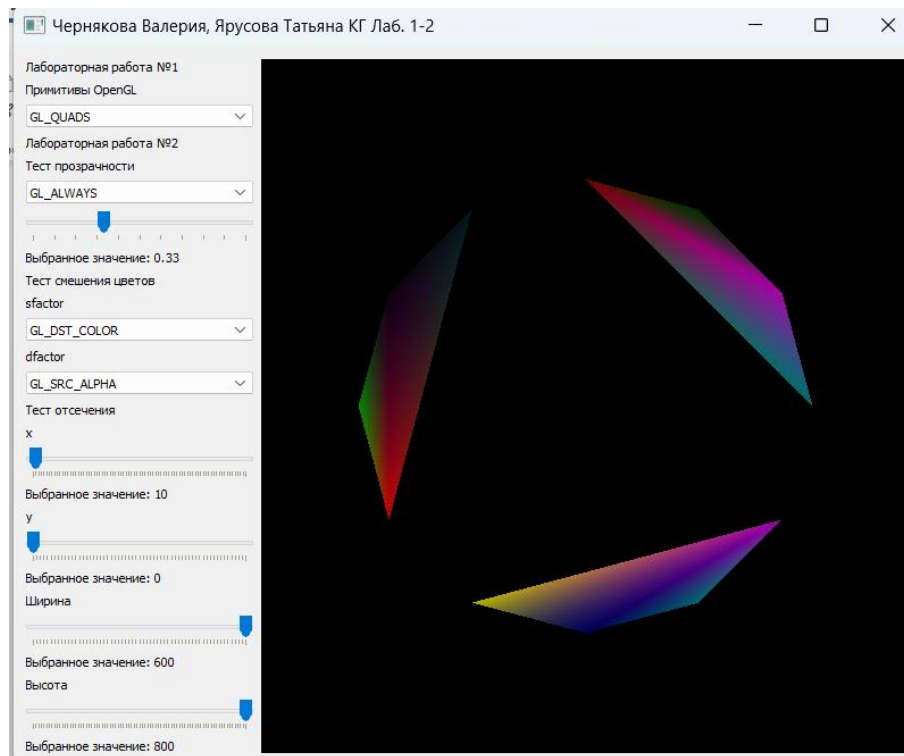


Рисунок 15 – тест смешения цветов с sfactor `GL_DST_COLOR` и dfactor `GL_SRC_ALPHA`

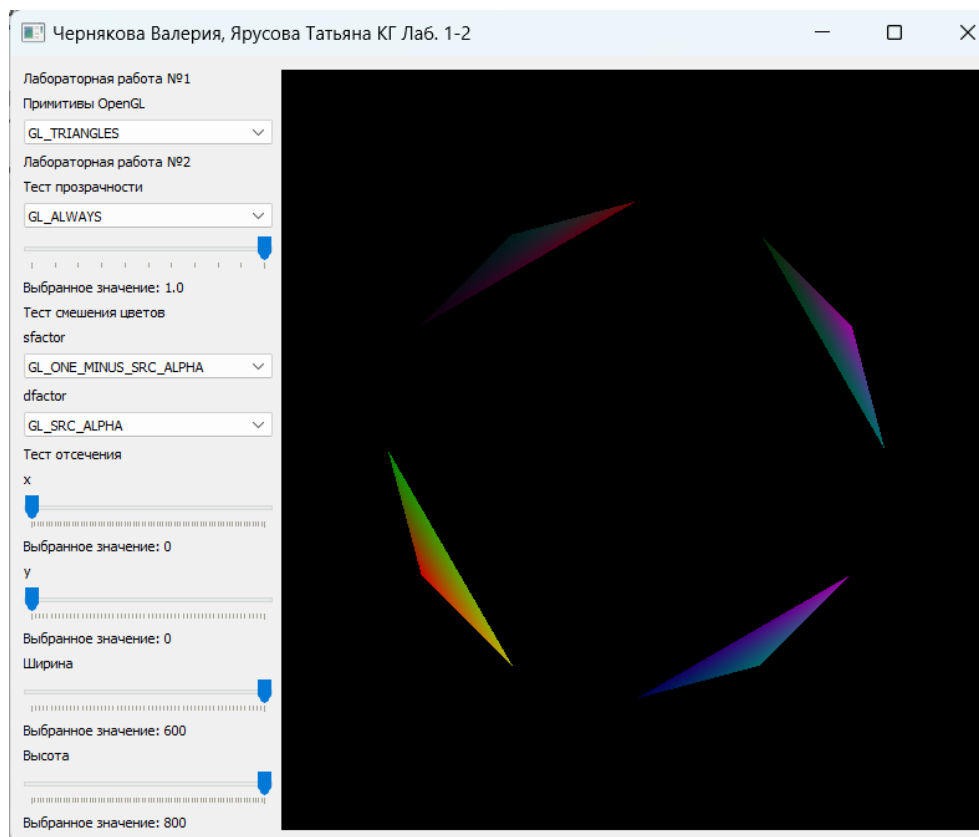


Рисунок 16 – тест смешения цветов с sfactor `GL_ONE_MINUS_SRC_ALPHA` и dfactor `GL_SRC_ALPHA`

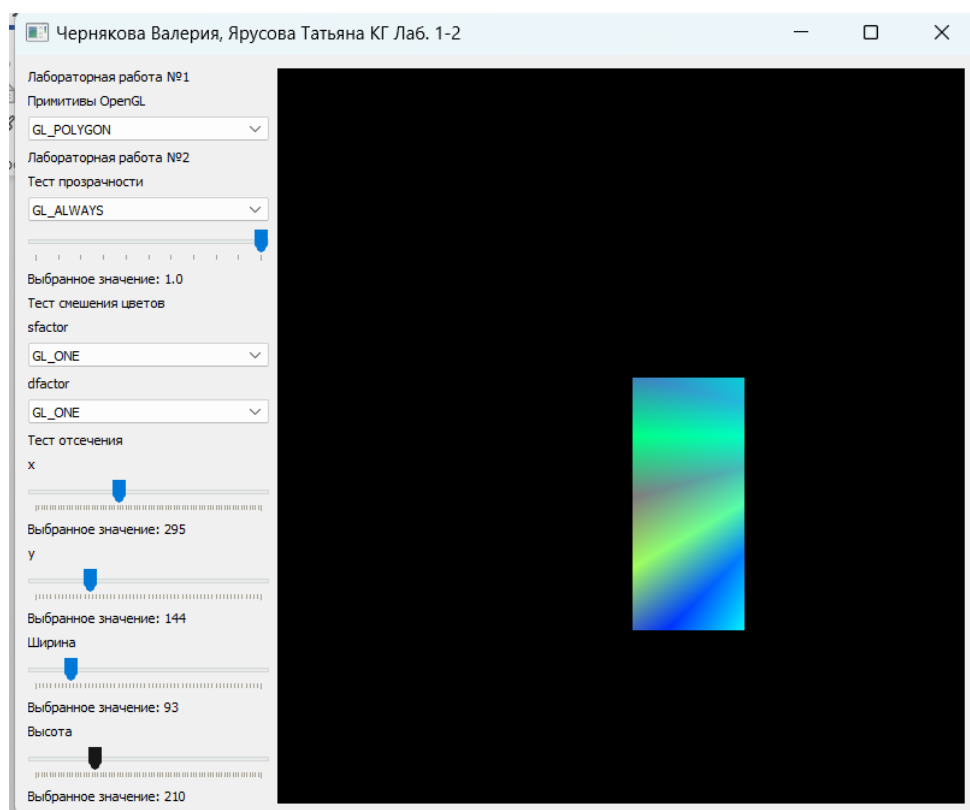


Рисунок 17 – тест отсечения из координат (295, 144) области высотой 210 и шириной 93

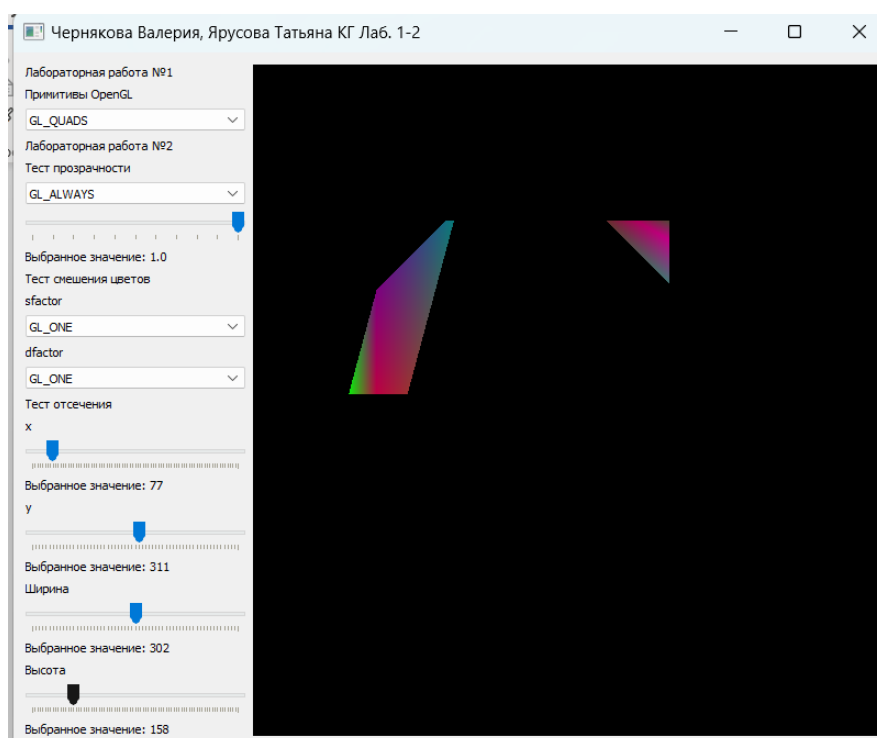


Рисунок 18 – тест отсечения из координат (77, 311) области высотой 158 и шириной 158

Выводы.

В результате выполнения работы были изучены основы работы с *OpenGL*.

Были освоены возможности подключения библиотеки *OpenGL* и *Qt* в *Python*.

С помощью языка программирования *Python* и библиотеки *PyQt* была написана программа с пользовательским интерфейсом для отрисовки примитивов, а также реализующая представление тестов смешивания цветов, отсечения и прозрачности для графических примитивов.

Программа работает корректно.

При выполнении работы были приобретены навыки работы с графической библиотекой *OpenGL*.