**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №2**

**по дисциплине «Компьютерная графика»**

Тема: Примитивы OpenGL

|  |  |  |
| --- | --- | --- |
| Студенты гр. 6304 |  | Корытов П.В.  Пискунов Я.А. |
| Преподаватель |  | Герасимова Т.В. |

Санкт-Петербург

2018

**Цель работы.**

На базе предложенного шаблона разработать программу 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 тестов через вызов соответствующих элементов интерфейса пользователя

**Общие сведения**

Управление режимами работы в OpenGL осуществляется при помощи двух команд - glEnable и glDisable, одна из которых включает, а вторая выключает некоторый режим.

void glEnable(GLenum cap)

void glDisable(GLenum cap)

Обе команды имеют один аргумент – сар, который может принимать значения определяющие тот или иной режим, например, GL\_ALPHA\_TEST, GL\_BLEND, GL\_SCISSOR\_TEST и многие другие.

*Тест отсечения*

Режим GL\_SCISSOR\_TEST разрешает отсечение тех фрагментов объекта, которые находятся вне прямоугольника "вырезки".

Прямоугольник "вырезки" определяется функцией glScissor:

**void glScissor( GLint x, GLint y, GLsizei width, GLsizei height );**

где параметры

* x, y определяют координаты левого нижнего угла прямоугольника «вырезки», исходное значение - (0,0).
* width, height - ширина и высота прямоугольника «вырезки».

*Тест прозрачности*

Режим GL\_ALPHA\_TEST задает тестирование по цветовому параметру альфа.Функция glAlphaFunc устанавливает функцию тестирования параметра альфа.

**void glAlphaFunc( GLenum func, GLclampf ref )**

где параметр – func может принимать следующие значения:

* GL\_NEVER – никогда не пропускает GL\_LESS – пропускает, если входное значение альфа меньше, чем значение ref
* GL\_EQUAL – пропускает, если входное значение альфа равно значению ref
* GL\_LEQUAL – пропускает, если входное значение альфа меньше или равно значения ref
* GL\_GREATER – пропускает, если входное значение альфа больше, чем значение ref
* GL\_NOTEQUAL – пропускает, если входное значение альфа не равно значению ref
* GL\_GEQUAL – пропускает, если входное значение альфа больше или равно значения ref GL\_ALWAYS – всегда пропускается, по умолчанию,

а параметр ref – определяет значение, с которым сравнивается входное значение альфа. Он может принимать значение от 0 до 1, причем 0 представляет наименьшее возможное значение альфа, а 1 – наибольшее. По умолчанию ref равен 0.

*Тест смешения цветов*

Режим GL\_BLEND разрешает смешивание поступающих значений цветов RGBA со значениями, находящимися в буфере цветов.

Функция glBlendFunc устанавливает пиксельную арифметику.

**void glBlendFunc( GLenum sfactor, GLenum dfactor );**

где параметры

* sfactor устанавливает способ вычисления входящих факторов смешения RGBA. Может принимать одно из следующих значений – GL\_ZERO, GL\_ONE, GL\_DST\_COLOR, GL\_ONE\_MINUS\_DST\_COLOR, GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA, GL\_DST\_ALPHA, GL\_ONE\_MINUS\_DST\_ALPHA и GL\_SRC\_ALPHA\_SATURATE.

* dfactor устанавливает способ вычисления факторов смешения RGBA, уже находящихся в буфере кадра. Может принимать одно из следующих значений – GL\_ZERO, GL\_ONE, GL\_SRC\_COLOR, GL\_ONE\_MINUS\_SRC\_COLOR, GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA, GL\_DST\_ALPHA и GL\_ONE\_MINUS\_DST\_ALPHA.

Прозрачность лучше организовывать используя команду glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA). Такой же вызов применяют для устранения ступенчатости линий и точек. Для устранения ступенчатости многоугольников применяют вызов команды glBlendFunc(GL\_SRC\_ALPHA\_SATURATE, GL\_ONE).

**Выполнение работы**

* Для разработки использована IDE PyCharm
* Используемый ЯП – Python, интерпретатор версии 3.6.7
* Для разработки интерфейса использован байдинг фреймворка Qt на Python, поддерживаемый сообществом – PyQt5
* Формы разрабатываются в Qt Designer и конвертируются в python-код с помощью утилиты pyuic5
* Для работы с OpenGL используется библиотека pyopengl
* Разработка проводится как на платформе Linux, так и на Windows

Основной класс программы – MainWindow, унаследованный от QMainWindow и Ui\_MainWindow. Первый суперкласс содержит логику отображения окна, второй – элементы ui вроде кнопок, виджетов и т.п.

В конструкторе класса происходит соединение сигналов от Ui со слотами. Некоторые слоты – методы объектов Ui:

self.blendSFactor.activated.connect(self.openGLWidget.update)

self.blendDFactor.activated.connect(self.openGLWidget.update)

Другие – методы класса:

self.primitiveComboBox.activated.connect(self.resetRandomAndUpdate)

В отличие от предыдущей лабораторной, теперь в настройке OpenGL обеспечиваются относительные координаты от 0 до 1:

def loadScene(self):

width, height = self.openGLWidget.width(), self.openGLWidget.height()

GL.glViewport(0, 0, width, height)

GL.glMatrixMode(GL.GL\_PROJECTION)

GL.glLoadIdentity()

GL.glOrtho(0, 1, 0, 1, -1, 1)

GL.glMatrixMode(GL.GL\_MODELVIEW)

GL.glLoadIdentity()

Точки иногда генерируются абсолютно случайно, как и цвета:

self.generated\_points = [(np.random.random(), np.random.random()) for \_ in range(number)]

self.generated\_colors = [random\_rgb() for \_ in range(number)]

Иногда используются полярные координаты:

for \_ in range(N):

acc\_angle += random.random() \* 360 / N

x, y = pol2cart(r, acc\_angle / 180 \* np.pi)

x += 0.5

y += 0.5

self.generated\_points.append((x, y))

Теперь при вызове методов точки генерируются только один раз, в дальнейшем – перерисовываются. Таким образом обеспечивается корректное поведение при масштабировании окна и т.п. Для того, чтобы сгенерировать новые точки, существует специальный метод.

Также теперь перед вызовом методов, оставшихся от предыдущей лабораторной, вызываются методы для обработки отсечения, прозрачности и смешивания

def paintGL(self):

self.loadScene()

try:

self.glScissorTest()

self.glAlphaTest()

self.glBlendTest()

self.actionsDict[self.primitiveComboBox.currentText()]()

except Exception as exp:

print(exp)

pass

Ниже приведены сами методы:

def glScissorTest(self):

GL.glEnable(GL.GL\_SCISSOR\_TEST)

x\_scissor = self.XScissorSlider.value() / 100

y\_scissor = self.YScissorSlider.value() / 100

GL.glScissor(int(x\_scissor \* self.openGLWidget.width()), int(y\_scissor \* self.openGLWidget.height()),

self.openGLWidget.width(), self.openGLWidget.height())

def glAlphaTest(self):

GL.glEnable(GL.GL\_ALPHA\_TEST)

alpha\_method = self.AlphaComboBox.currentText()

alpha\_value = self.AlphaSlider.value() / 100

GL.glAlphaFunc(getattr(GL, alpha\_method), alpha\_value)

def glBlendTest(self):

GL.glEnable(GL.GL\_BLEND)

sFactor = self.blendSFactor.currentText()

dFactor = self.blendDFactor.currentText()

GL.glBlendFunc(getattr(GL, sFactor), getattr(GL, dFactor))

Полный код находится в приложении A.

**Тестирование**

Тестирование отсечения:

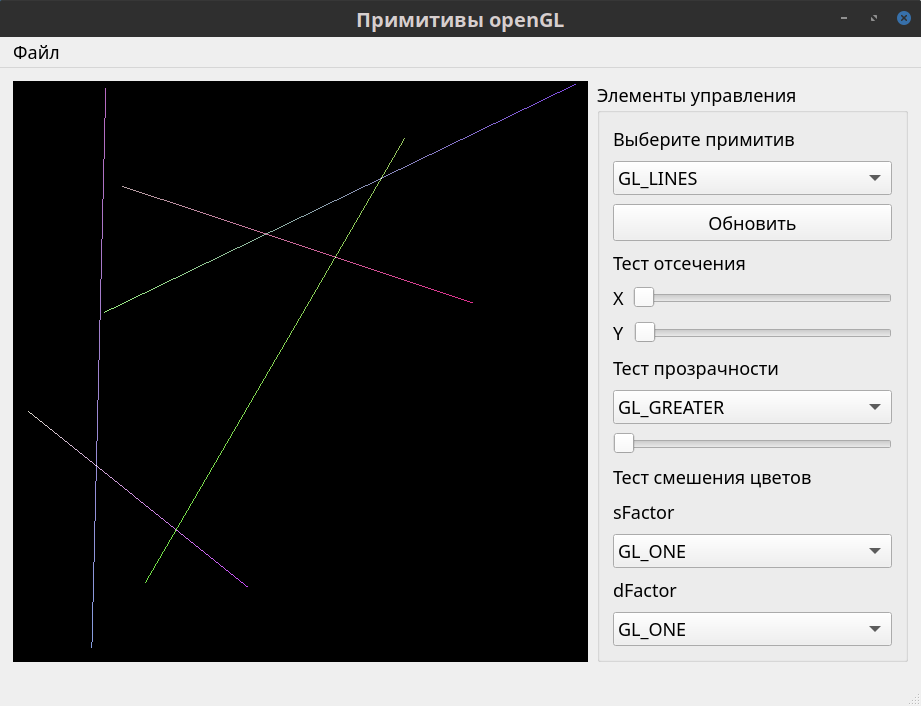


Рисунок - тестирование отсечения (1)

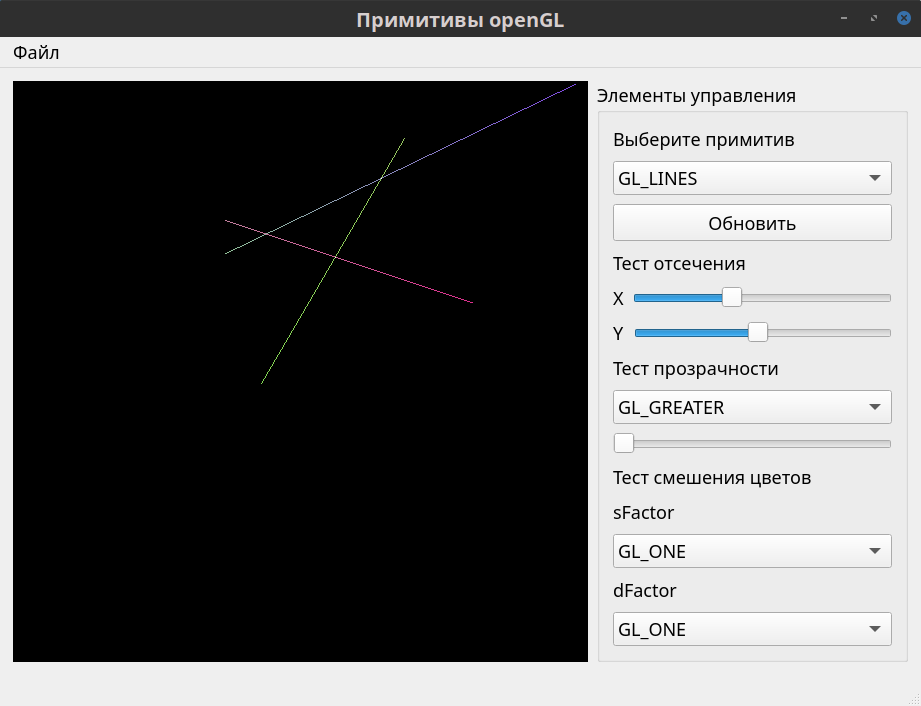


Рисунок - тестирование отсечения (2)

Тестирование прозрачности:

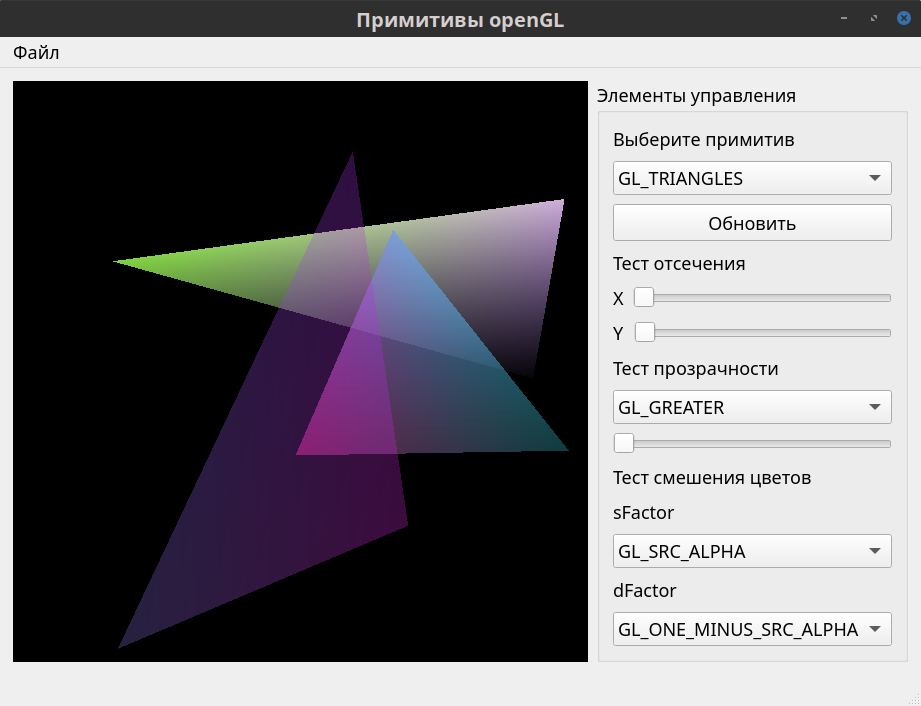


Рисунок - тестирование прозрачности (1)

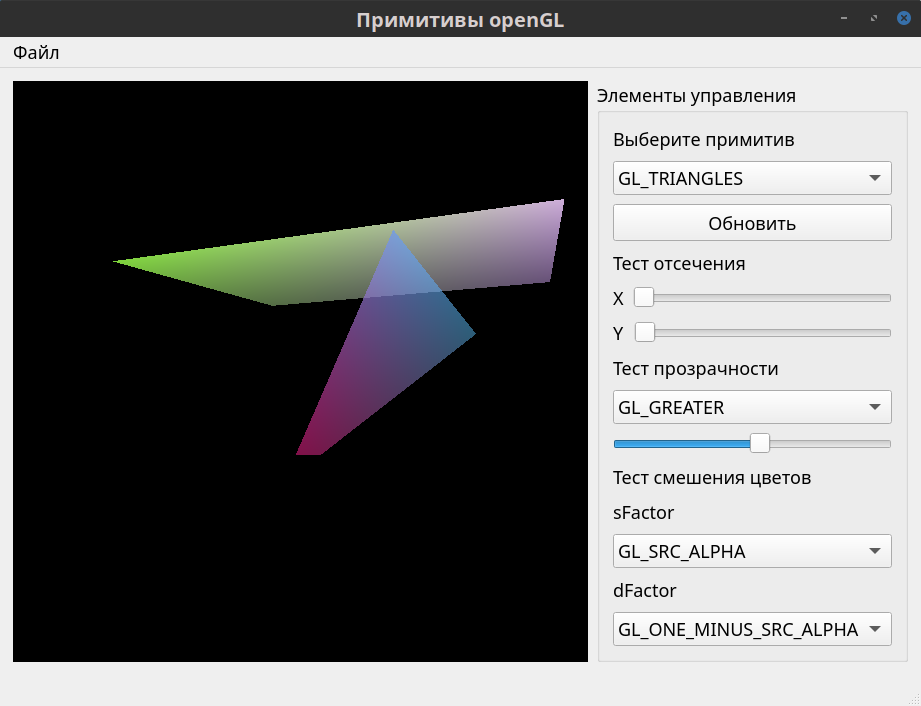


Рисунок - тестирование прозрачности (2)

Тестирование смешения цветов:

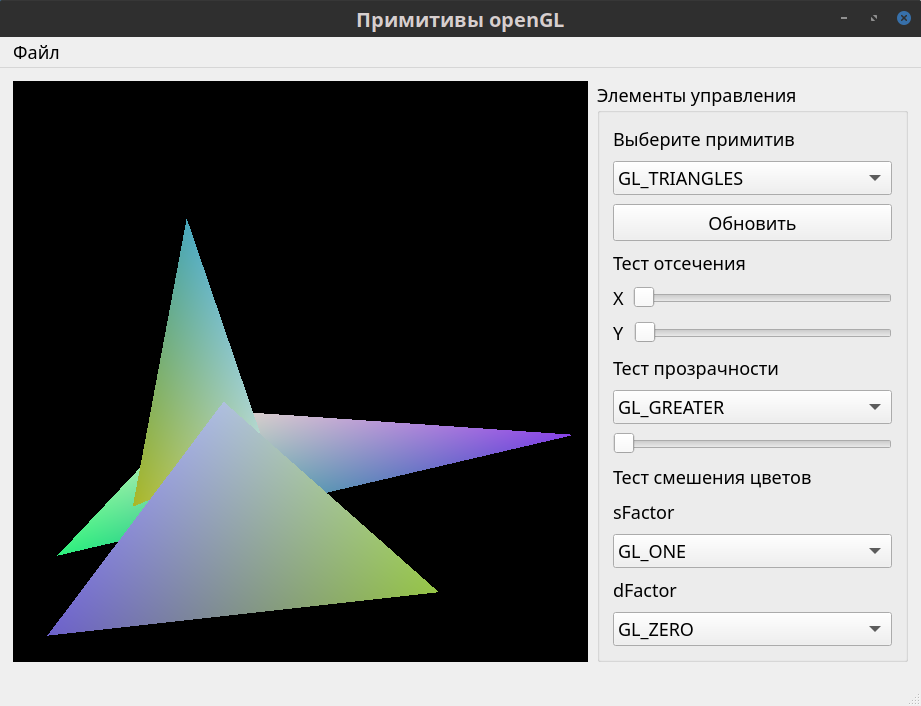


Рисунок 5 - GL\_ONE / GL\_ZERO

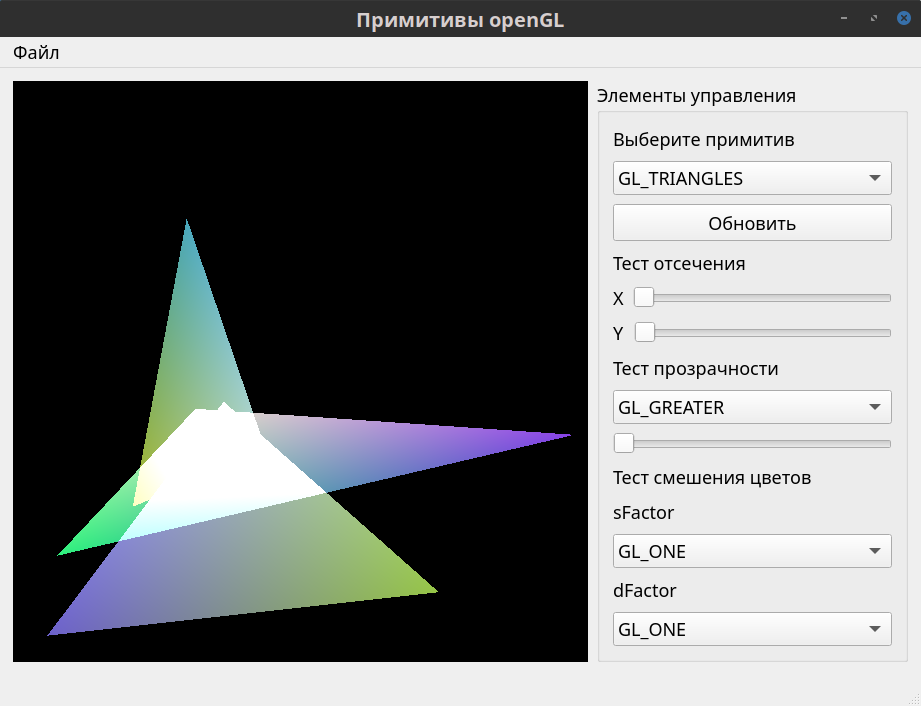


Рисунок 6 - GL\_ONE / GL\_ONE

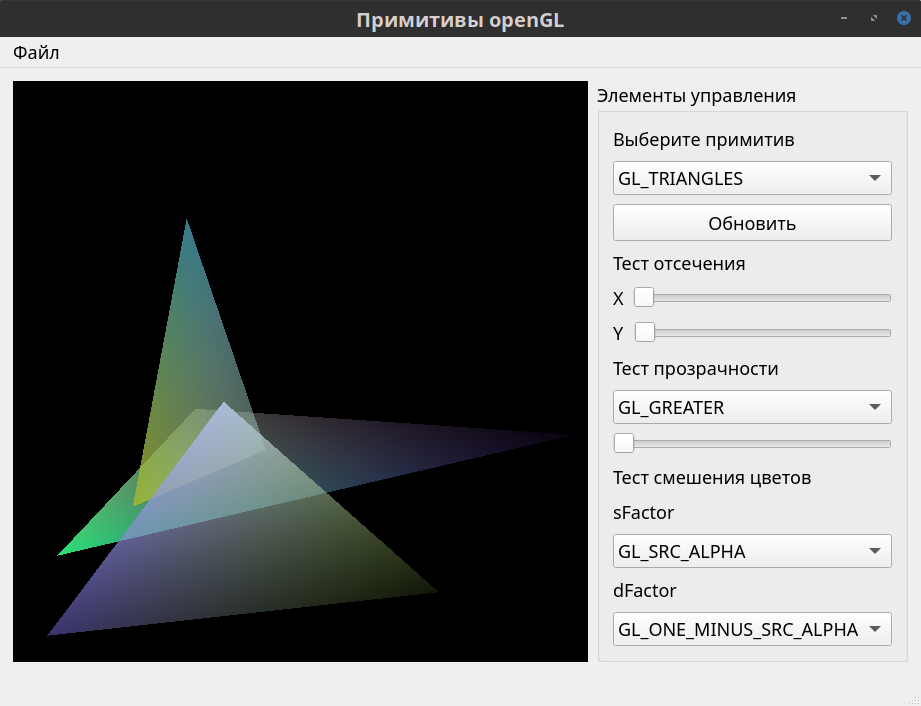


Рисунок 7 - GL\_SRC\_ALPHA / GL\_ONE\_MINUS\_SRC\_ALPHA

**Выводы**

В результате выполнения лабораторной работы была разработана программа, реализующая представление тестов смешивания цветов, отсечения и прозрачности для графических примитивов OpenGL, разработанных в лабораторной работе № 1. Программа работает корректно. При выполнении работы были приобретены навыки работы с графической библиотекой OpenGL.

**ПРИЛОЖЕНИЕ A**

**Код main.py**

import random

import numpy as np

from PyQt5.QtWidgets import QMainWindow, QApplication

from OpenGL import GL, GLU, GLUT

import logging

from ui.mainwindow import Ui\_MainWindow

def random\_rgb():

"""

Get random bright enough color

"""

color = [random.random() for \_ in range(4)]

if sum(color[:3]) / 3 < 0.5:

return random\_rgb()

else:

return color

def cart2pol(x, y):

r = np.sqrt(x \*\* 2 + y \*\* 2)

φ = np.arctan2(y, x)

return r, φ

def pol2cart(r, φ):

x = r \* np.cos(φ)

y = r \* np.sin(φ)

return x, y

# noinspection PyPep8Naming

class MainWindow(QMainWindow, Ui\_MainWindow):

generated\_points = []

generated\_colors = []

def \_\_init\_\_(self, parent=None):

super().\_\_init\_\_(parent)

self.setupUi(self)

self.primitiveComboBox.activated.connect(self.resetRandomAndUpdate)

self.AlphaSlider.valueChanged.connect(self.openGLWidget.update)

self.AlphaComboBox.activated.connect(self.openGLWidget.update)

self.blendSFactor.activated.connect(self.openGLWidget.update)

self.blendDFactor.activated.connect(self.openGLWidget.update)

self.updateButton.clicked.connect(self.resetRandomAndUpdate)

self.XScissorSlider.valueChanged.connect(self.openGLWidget.update)

self.YScissorSlider.valueChanged.connect(self.openGLWidget.update)

self.openGLWidget.initializeGL()

self.openGLWidget.paintGL = self.paintGL

self.actionsDict = {

"GL\_POINTS": self.paintGL\_random,

"GL\_LINES": self.paintGL\_random,

"GL\_LINE\_STRIP": self.paintGL\_random,

"GL\_LINE\_LOOP": self.paintGL\_random,

"GL\_TRIANGLES": self.paintGL\_random,

"GL\_TRIANGLE\_STRIP": self.paintGL\_circular\_random,

"GL\_TRIANGLE\_FAN": self.paintGL\_circular\_random,

"GL\_QUADS": self.paintGL\_quads,

"GL\_QUAD\_STRIP": self.paintGL\_quad\_strip,

"GL\_POLYGON": self.paintGL\_polygon

}

def resetRandomAndUpdate(self):

self.generated\_points = []

self.generated\_colors = []

self.openGLWidget.update()

def glScissorTest(self):

GL.glEnable(GL.GL\_SCISSOR\_TEST)

x\_scissor = self.XScissorSlider.value() / 100

y\_scissor = self.YScissorSlider.value() / 100

GL.glScissor(int(x\_scissor \* self.openGLWidget.width()), int(y\_scissor \* self.openGLWidget.height()),

self.openGLWidget.width(), self.openGLWidget.height())

def glAlphaTest(self):

GL.glEnable(GL.GL\_ALPHA\_TEST)

alpha\_method = self.AlphaComboBox.currentText()

alpha\_value = self.AlphaSlider.value() / 100

GL.glAlphaFunc(getattr(GL, alpha\_method), alpha\_value)

def glBlendTest(self):

GL.glEnable(GL.GL\_BLEND)

sFactor = self.blendSFactor.currentText()

dFactor = self.blendDFactor.currentText()

GL.glBlendFunc(getattr(GL, sFactor), getattr(GL, dFactor))

def loadScene(self):

width, height = self.openGLWidget.width(), self.openGLWidget.height()

GL.glViewport(0, 0, width, height)

GL.glMatrixMode(GL.GL\_PROJECTION)

GL.glLoadIdentity()

GL.glOrtho(0, 1, 0, 1, -1, 1)

GL.glMatrixMode(GL.GL\_MODELVIEW)

GL.glLoadIdentity()

def paintGL(self):

self.loadScene()

try:

self.glScissorTest()

self.glAlphaTest()

self.glBlendTest()

self.actionsDict[self.primitiveComboBox.currentText()]()

except Exception as exp:

print(exp)

pass

def paintGL\_random(self):

GL.glPointSize(2)

random\_dict = {

"GL\_POINTS": {

"GL\_MODE": GL.GL\_POINTS,

"POINTS\_NUM": 50

},

"GL\_LINES": {

"GL\_MODE": GL.GL\_LINES,

"POINTS\_NUM": 5 \* 2

},

"GL\_LINE\_STRIP": {

"GL\_MODE": GL.GL\_LINE\_STRIP,

"POINTS\_NUM": 5

},

"GL\_LINE\_LOOP": {

"GL\_MODE": GL.GL\_LINE\_LOOP,

"POINTS\_NUM": 5

},

"GL\_TRIANGLES": {

"GL\_MODE": GL.GL\_TRIANGLES,

"POINTS\_NUM": 3 \* 3

}

}

GL.glBegin(random\_dict[self.primitiveComboBox.currentText()]['GL\_MODE'])

self.drawRandomPoints(random\_dict[self.primitiveComboBox.currentText()]['POINTS\_NUM'])

GL.glEnd()

GL.glFinish()

def drawRandomPoints(self, number):

if len(self.generated\_points) == 0:

self.generated\_points = [(np.random.random(), np.random.random()) for \_ in range(number)]

self.generated\_colors = [random\_rgb() for \_ in range(number)]

self.placeGeneratedPoints()

def placeGeneratedPoints(self):

for point, color in zip(self.generated\_points, self.generated\_colors):

GL.glColor4d(\*color)

GL.glVertex2d(\*point)

def paintGL\_circular\_random(self):

random\_dict = {

"GL\_TRIANGLE\_STRIP": GL.GL\_TRIANGLE\_STRIP,

"GL\_TRIANGLE\_FAN": GL.GL\_TRIANGLE\_FAN,

}

acc\_angle = 0

N = 5

if len(self.generated\_points) == 0:

self.generated\_colors.append(random\_rgb())

self.generated\_points.append((0.5, 0.5))

max\_rad = 0.5

self.generated\_colors.extend([random\_rgb() for \_ in range(N)])

for \_ in range(N):

r = np.random.random() \* max\_rad

acc\_angle += random.random() \* 360 / N

x, y = pol2cart(r, acc\_angle / 180 \* np.pi)

x += 0.5

y += 0.5

self.generated\_points.append((x, y))

GL.glPointSize(2)

GL.glBegin(random\_dict[self.primitiveComboBox.currentText()])

self.placeGeneratedPoints()

GL.glEnd()

GL.glFinish()

def paintGL\_quads(self):

N = 4

if len(self.generated\_points) == 0:

for \_ in range(N):

c\_x, c\_y = np.random.random() \* 0.98 + 0.01, np.random.random() \* 0.98 + 0.01

max\_rad = min(c\_x, 1 - c\_x, c\_y, 1 - c\_y)

acc\_angle = 0

r = np.random.random() \* max\_rad

self.generated\_colors.extend([random\_rgb() for \_ in range(4)])

for \_ in range(4):

acc\_angle += random.random() \* 360 / 4

x, y = pol2cart(r, acc\_angle / 180 \* np.pi)

x += c\_x

y += c\_y

self.generated\_points.append((x, y))

GL.glPointSize(2)

GL.glBegin(GL.GL\_QUADS)

self.placeGeneratedPoints()

GL.glEnd()

GL.glFinish()

def paintGL\_quad\_strip(self):

N = 4

if len(self.generated\_points) == 0:

y\_s = [np.random.random() for \_ in range(N)]

y\_s.sort()

x\_s = [[np.random.random() for \_ in range(2)] for \_ in range(N)]

[l.sort() for l in x\_s]

self.generated\_colors = [random\_rgb() for \_ in range(N \* 2)]

for i in range(N):

self.generated\_points.append((x\_s[i][0], y\_s[i]))

self.generated\_points.append((x\_s[i][1], y\_s[i]))

GL.glPointSize(2)

GL.glBegin(GL.GL\_QUAD\_STRIP)

self.placeGeneratedPoints()

GL.glEnd()

GL.glFinish()

def paintGL\_polygon(self):

N = 8

if len(self.generated\_points) == 0:

acc\_angle = 0

r = np.random.random() \* 0.5

self.generated\_colors = [random\_rgb() for \_ in range(N)]

for \_ in range(N):

acc\_angle += random.random() \* 360 / N

x, y = pol2cart(r, acc\_angle / 180 \* np.pi)

x += 0.5

y += 0.5

self.generated\_points.append((x, y))

GL.glPointSize(2)

GL.glBegin(GL.GL\_POLYGON)

self.placeGeneratedPoints()

GL.glEnd()

GL.glFinish()

if \_\_name\_\_ == "\_\_main\_\_":

import sys

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec\_())