Санкт-Петербургский Государственный
Электротехнический Университет

Кафедра МОЭВМ

Задание для лабораторной работы № 2
"Примитивы OpenGL"

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

Санкт-Петербург
2023 г.

Цель работы.

- ознакомление с отсечением, прозрачностью и смешением цветов.

- освоение возможностей режимов работы в OpenGL.

- проанализировать полученное задание, выделить информационные объекты и действия;

- разработать программу с использованием требуемых режимов.

Основные теоретические положения.

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 – cap, 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».

В приведенном ниже фрагменте программы реализуется тест отсечения. Сначала изображается группа связных отрезков не используя режим отсечения, а затем включается этот режим.

```
glEnable(GL_SCISSOR_TEST);
InitViewport(0, windH*2/3, vpW, vpH);
glScissor(0,windH*2/3,vpW/2,vpH/2);
Triangles();
Quads();
glDisable(GL_SCISSOR_TEST);
InitViewport(windW/3, windH*2/3, vpW, vpH);
glScissor(windW/3,windH*2/3,vpW/2,vpH/2);
Triangles();
Quads();
```

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.

В приведенном ниже фрагменте программы реализуется тест прозрачности

```
glEnable(GL_ALPHA_TEST);
InitViewport(windW*2/3, windH*2/3, vpW, vpH);
glAlphaFunc(GL_LESS, 0.7f);
Triangles();
Quads();
InitViewport(0, windH/3, vpW, vpH);
glAlphaFunc(GL_GREATER, 0.7f);
Triangles();
Quads();
glDisable(GL_ALPHA_TEST);
```

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.

В приведенном ниже фрагменте программы реализуется тест смешения

```
glEnable(GL_BLEND);
InitViewport(windW/3, windH/3, vpW, vpH);
glBlendFunc(GL_ONE, GL_ZERO);
Triangles();
Quads();
InitViewport(windW*2/3, windH/3, vpW, vpH);
glBlendFunc(GL_ONE, GL_ONE);
Triangles();
Quads();
InitViewport(0, 0, vpW, vpH);
glBlendFunc(GL_ONE, GL_SRC_COLOR);
Triangles();
Quads();
```

```
InitViewport(windW/3, 0, vpW, vpH);
glBlendFunc(GL_ONE, GL_ONE_MINUS_SRC_COLOR);
Triangles();
Quads();
InitViewport(windW*2/3, 0, vpW, vpH);
glBlendFunc(GL_ZERO, GL_ONE_MINUS_SRC_COLOR);
Triangles();
Quads();
```

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).

Задание.

На базе разработанной вами оболочки из 1 работы 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

Выполнение работы.

В классе mainWindow добавлен атрибут self.lbltests, который принадлежит классу QLabel. Он содержит текст для заголовка теста перед тестом отсечения.

Атрибут self.boxscissor пренадлежит классу QCheckBox. Он необходим для управления режимом отсечения: включен и выключен. Когда в нем меняется состояние, через метод stateChanged у self.boxscissor передается с помощью метода connect состояние в метод self.set_scissor. Если активно, то есть равно Qt.Checked, то для всех виджетов в self.stack, обращение к которым осуществляется с помощью метода widget, меняется значение атрибута scissor_flag на True. Атрибут self.scissor_flag был добавлен в класс glWidget0, от которого наследуются все примитивы. Иначе атрибуту присваивается значение False. В обоих случаях происходит обновление виджетов с помощью метода updateGL. Атрибут self.lblx подписывает ползунок X и принадлежит классу QLabel. Ползунок self.sliderx принадлежит классу QSlider в горизонтальном виде с помощью параметра Qt.Orientation.Horizontal. Двигая ползунок, с помощью метода valueChanged у self.sliderx, который передает значение ползунка, и метода connect, который связывает ползунок с методом update_scissorsx, метод update_scissorx в классе mainWindow получает значения с ползунка. В методе для всех виджетов в self.stack, обращение к которым осуществляется с помощью метода widget, обновляется значение атрибута self.x, который добавлен в класс glWidget0, от которого наследуются все примитивы. Происходит обновление виджетов с помощью метода updateGL. Ползунок может принимать значения от 0 до 240. В слой buttonsLayout добавлены виджеты self.lbltests, self.boxscissor, self.lblx, self.sliderx с помощью метода addWidget. Аналогично ползунку X создан ползунок Y. Аналогично методу update_scissorsx и атрибуту self.x созданы метод update_scissorsy и атрибут self.y. В классе glWidget0 добавлен метод resizeGL. Он устанавливает область вывода, проекции, а именно определяет область отрисовки в зависимости от размера окна. Добавлены в класс glWidget0 атрибуты self.w и self.h, которым присваиваются

значения ширины и высоты виджета примитива. В glViewport передаются текущие размеры виджета w и h. glMatrixMode(GL_PROJECTION) установка текущей матрицы матрицу проекций. glLoadIdentity() – заменяет текущую матрицу на единичную. Вычисляется aspect, равный отношению ширины к высоте. Он используется в gluPerspective. glMatrixMode(GL_MODELVIEW) установка текущей матрицы матрицу модельно-видовой. Таким образом, метод по текущим размерам виджета устанавливает параметры отрисовки. glViewport из метода paintGL примитивов был удален. Если self.scissor_flag переведен в True, то в виджетах примитивов в методе paintGL включается режим отсечения с помощью glEnable(GL_SCISSOR_TEST). Отсечение задается функцией glScissor (self.x, self.y, int(self.w / 2), int(self.h / 2)), в нее передаются координаты x, y левого нижнего угла, которые задаются ползунками, размеры области: ширина, высота. Они зависят от текущей ширины и высоты виджета примитива, которые хранятся в атрибутах self.w и self.h. После отрисовки режим отсечения выключается с помощью glDisable(GL_SCISSOR_TEST).

В классе mainWindow добавлен атрибут self.lbltesta, который принадлежит классу QLabel. Он содержит текст для заголовка теста перед тестом прозрачности. Атрибут self.boxalpha принадлежит классу QComboBox - объединенная кнопка с всплывающим списком. Список заполнен с помощью метода addItems "GL_NEVER", "GL_LESS", "GL_EQUAL", "GL_LEQUAL", "GL_GREATER", "GL_NOTEQUAL", "GL_GEQUAL", "GL_ALWAYS". Первоначальное состояние установлено на "GL_ALWAYS" с помощью метода setCurrentIndex. self.boxalpha связан с добавленным в класс glWidget0 атрибутом alphafunc, параметром для функции glAlphaFunc. Связь осуществлена с помощью метода activated, который посылает сигнал со строкой выбранного элемента в списке, с помощью метода

connect осуществляется связь с методом activated_boxalpha у mainWindow. В методе activated_boxalpha для всех виджетов в self.stack, обращение к которым осуществляется с помощью метода widget, меняется значение атрибута alphafunc на текущее значение элемента в списке, замененное на символьную константу вместо строки. Происходит обновление виджетов с помощью метода updateGL. Атрибут self.lblalpha подписывает ползунок значением прозрачности, которое нужно для сравнения в функции glAlphaFunc, и принадлежит классу QLabel. Ползунок self.slideralpha принадлежит классу QSlider в горизонтальном виде с помощью параметра Qt.Orientation.Horizontal. Двигая ползунок, с помощью метода valueChanged у self.slideralpha, который передает значение ползунка, и метода connect, который связывает ползунок с методом update_alpha, метод update_alpha в классе mainWindow получает значения с ползунка. В методе с помощью метода setText у self.lblalpha меняется текст на “Alpha: ” и после идет текущее значение прозрачности с ползунка (поделено на 100), для всех виджетов в self.stack, обращение к которым осуществляется с помощью метода widget, обновляется значение атрибута self.alpharef, который добавлен в класс glWidget0, от которого наследуются все примитивы. Происходит обновление виджетов с помощью метода updateGL. Виджеты self.lbltesta, self.boxalpha, self.lblalpha, self.slideralpha добавлены в слой buttonsLayout с помощью метода addWidget. В виджетах примитивов в методе paintGL была добавлен параметр прозрачности в цвет, включается тест прозрачности с помощью glEnable(GL_ALPHA_TEST). Тест прозрачности задается функцией glAlphaFunc(self.alphafunc, self.alpharef / 100), в нее передаются func для сравнения, которая выбирается в всплывающем списке, и ref значение для сравнения, которое выбирается ползунком (поделено на 100). После отрисовки режим прозрачности выключается с помощью glDisable(GL_ALPHA_TEST).

В классе mainWindow добавлен атрибут self.lbltestb, который принадлежит классу QLabel. Он содержит текст для заголовка теста перед тестом смешения цветов. Атрибут self.lblsfactor подписывает выпадающий список с выбором sfactor и принадлежит классу QLabel. Атрибут self.boxsfactor принадлежит классу QComboBox - объединенная кнопка с всплывающим списком. Список заполнен с помощью метода addItems "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", "GL_SRC_ALPHA_SATURATE". Первоначальное состояние установлено на "GL_ONE" с помощью метода setCurrentIndex. self.boxsfactor связан с добавленным в класс glWidget0 атрибутом sfact, параметром для функции glBlendFunc. Связь осуществлена с помощью метода activated, который посылает сигнал со строкой выбранного элемента в списке, с помощью метода connect осуществляется связь с методом activated_boxsfactor у mainWindow. В методе activated_boxsfactor для всех виджетов в self.stack, обращение к которым осуществляется с помощью метода widget, меняется значение атрибута sfact на текущее значение элемента в списке, замененное на символьную константу вместо строки. Происходит обновление виджетов с помощью метода updateGL. Атрибут self.lbldfactor подписывает выпадающий список с выбором dfactor и принадлежит классу QLabel. Атрибут self.boxdfactor принадлежит классу QComboBox - объединенная кнопка с всплывающим списком. Список заполнен с помощью метода addItems "GL_ZERO", "GL_ONE", "GL_SRC_COLOR", "GL_ONE_MINUS_SRC_COLOR", "GL_SRC_ALPHA", "GL_ONE_MINUS_SRC_ALPHA", "GL_DST_ALPHA", "GL_ONE_MINUS_DST_ALPHA". self.boxdfactor связан с добавленным в класс

glWidget0 атрибутом dfact, параметром для функции glBlendFunc. Связь осуществлена с помощью метода activated, который посылает сигнал со строкой выбранного элемента в списке, с помощью метода connect осуществляется связь с методом activated_boxdfactor у mainWindow. В методе activated_boxdfactor для всех виджетов в self.stack, обращение к которым осуществляется с помощью метода widget, меняется значение атрибута dfact на текущее значение элемента в списке, замененное на символьную константу вместо строки. Происходит обновление виджетов с помощью метода updateGL. Виджеты self.lbltestb, self.lblsfactor, self.boxsfactor, self.lbldfactor, self.boxdfactor добавлены в слой buttonsLayout с помощью метода addWidget. В виджетах примитивов в методе paintGL отключается запись в буфер глубины для корректности полупрозрачности с помощью функции glDepthMask(GL_FALSE), включается режим смешения с помощью glEnable(GL_BLEND). Смешение задается функцией glBlendFunc(self.sfact, self.dfact), где параметры sfact и dfact выбираются в всплывающем списке. После отрисовки запись в буфер глубины включается с помощью glDepthMask(GL_TRUE), режим смешения выключается с помощью glDisable(GL_BLEND).

ТЕСТИРОВАНИЕ.

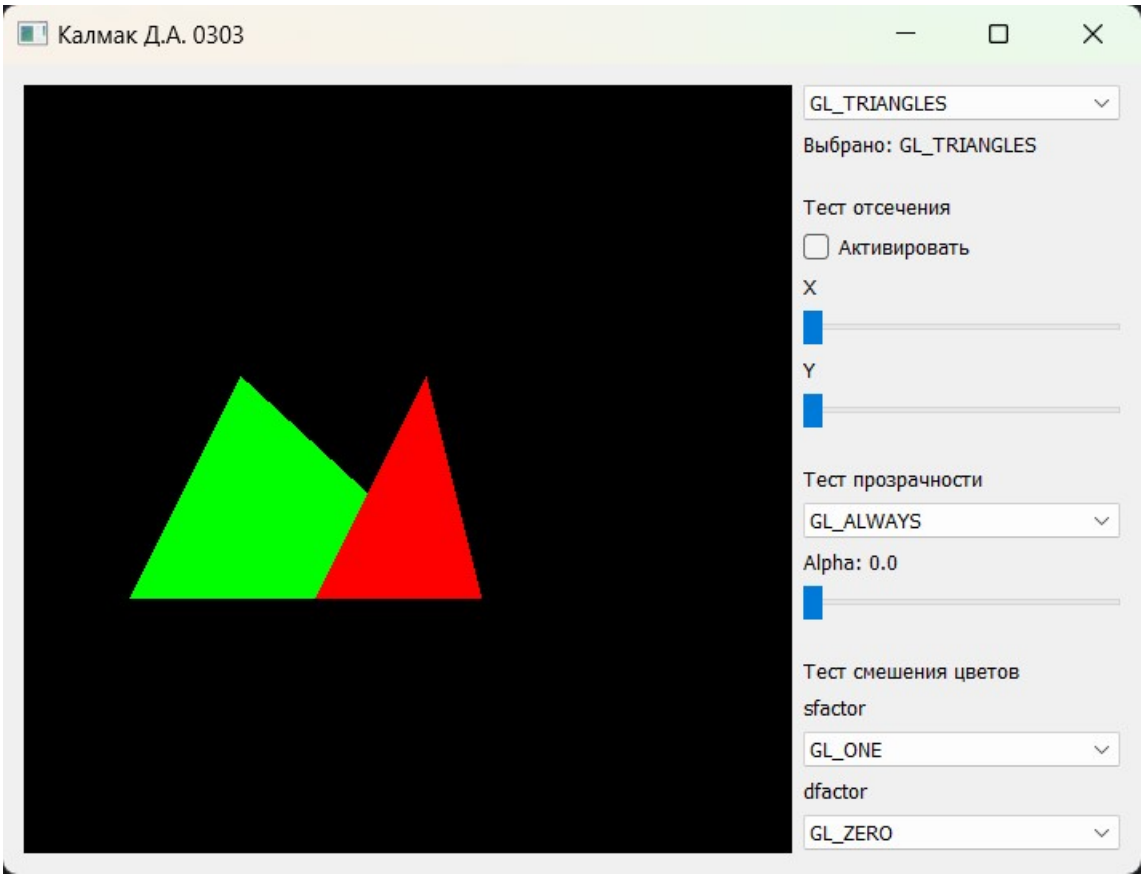

Рисунок 1 – Режим отсечения отключен

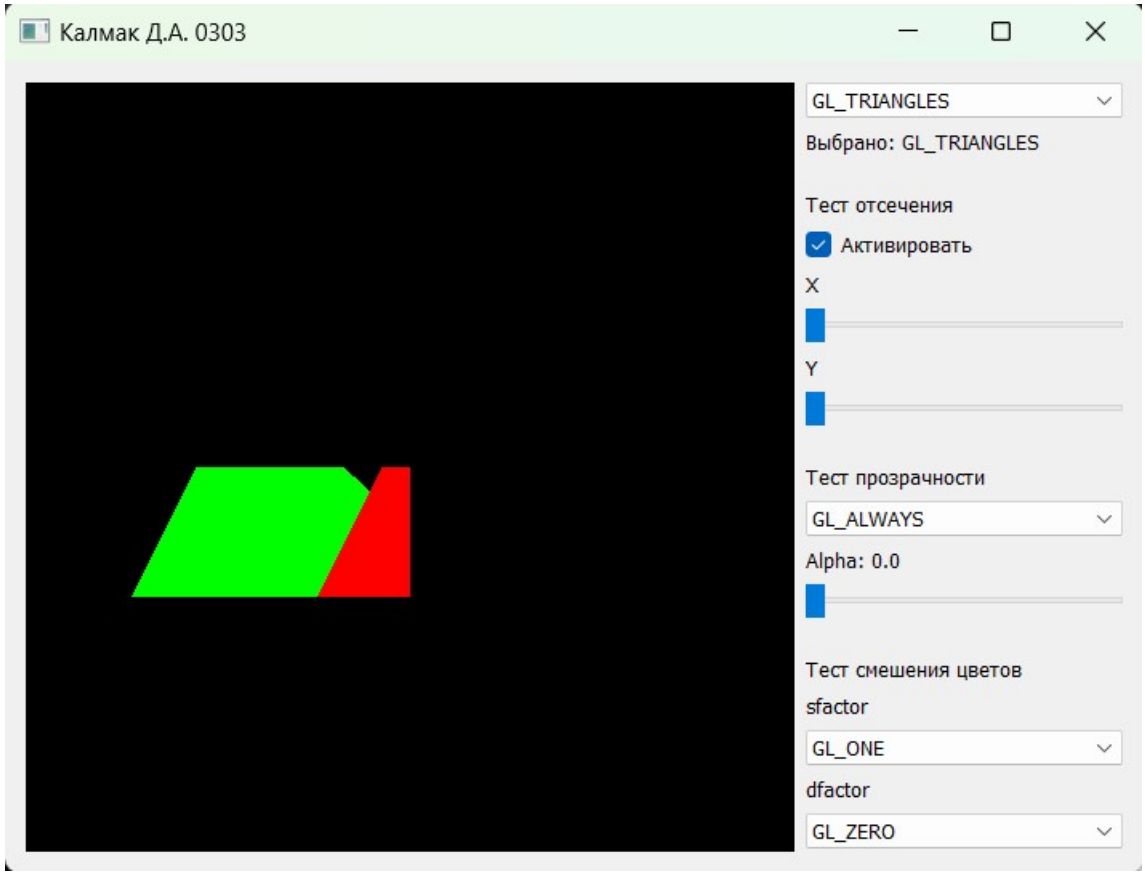

Рисунок 2 – Режим отсечения включен

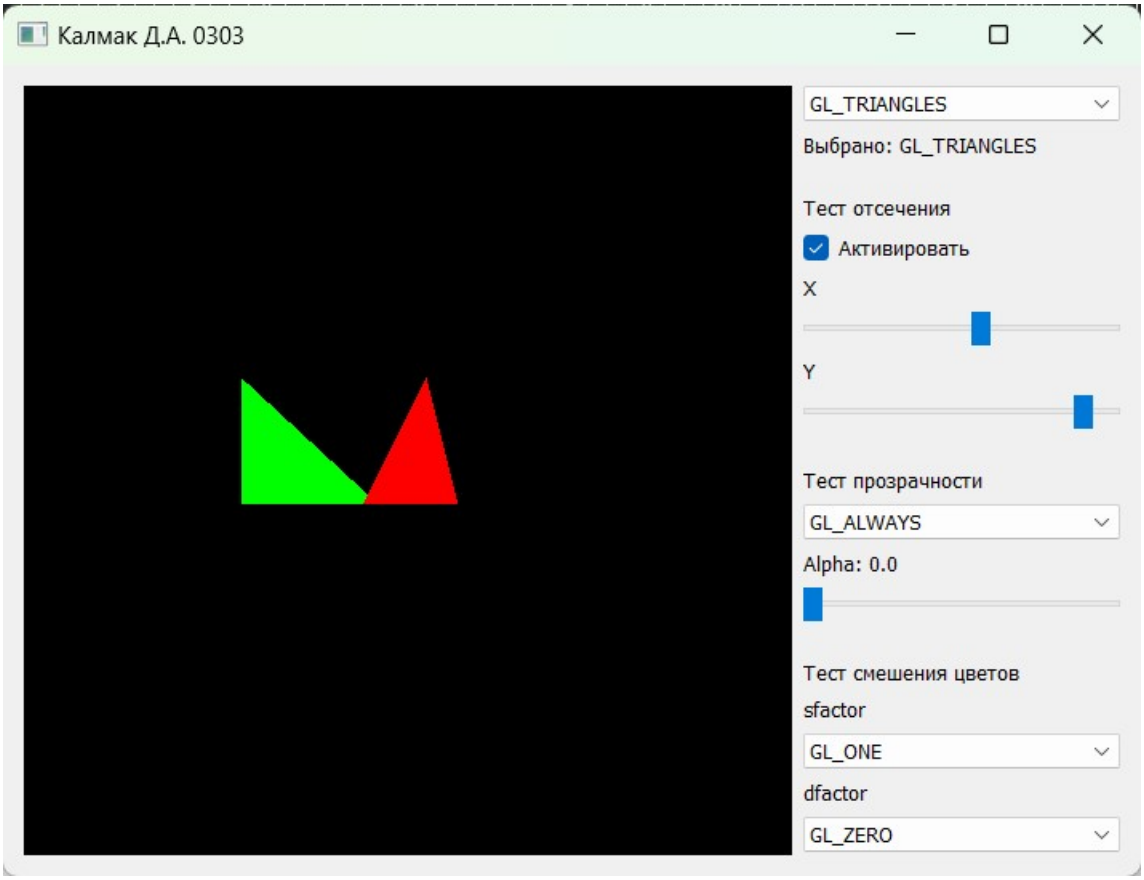

Рисунок 3 – Режим отсечения включен с измененными x, y

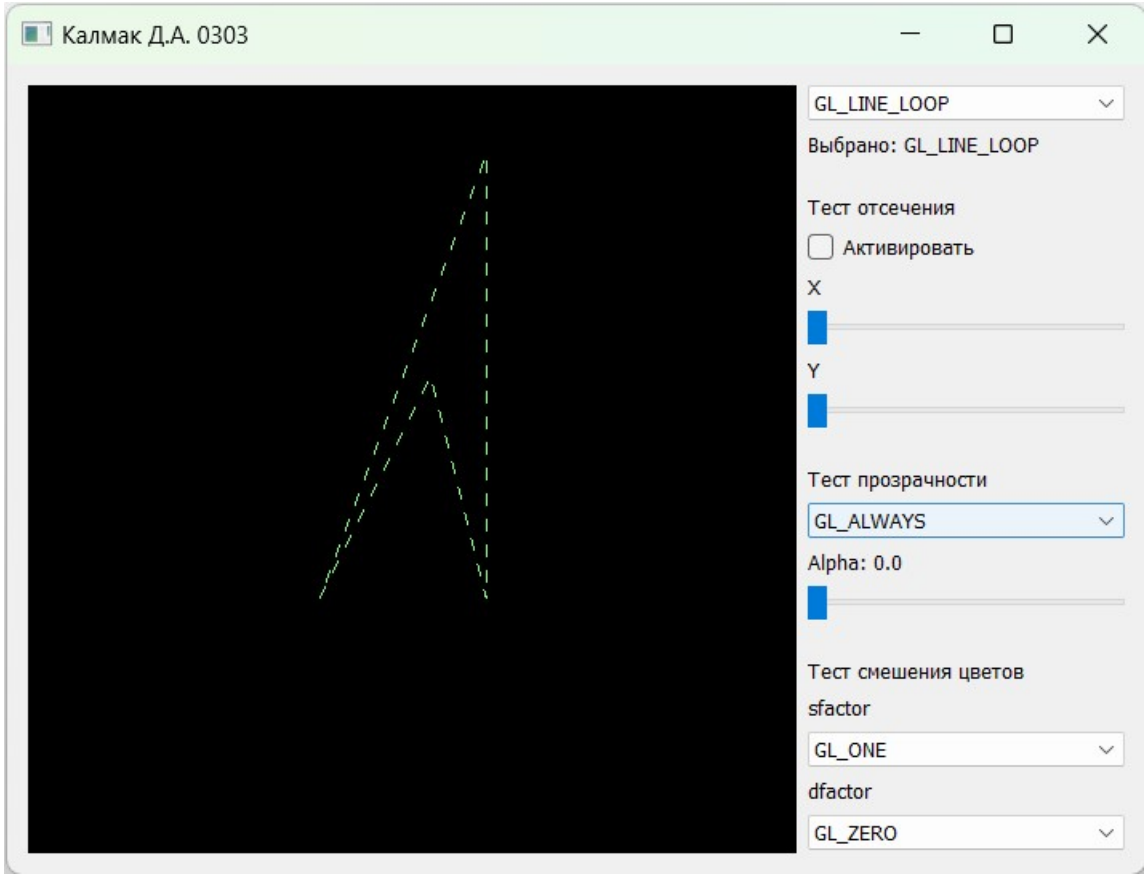

Рисунок 4 – Тест прозрачности с параметром GL_ALWAYS

Пропускает всегда.

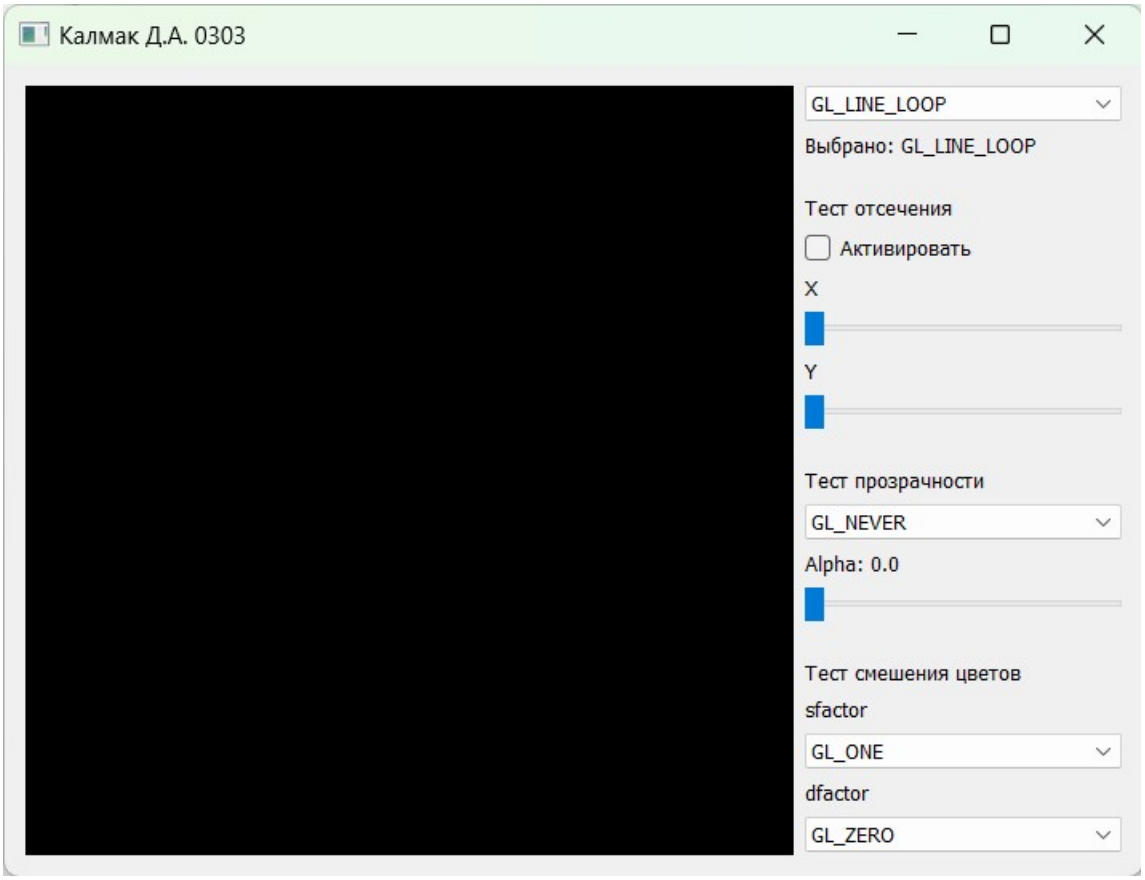

Рисунок 5 – Тест прозрачности с параметром GL_NEVER

Никогда не пропускает.

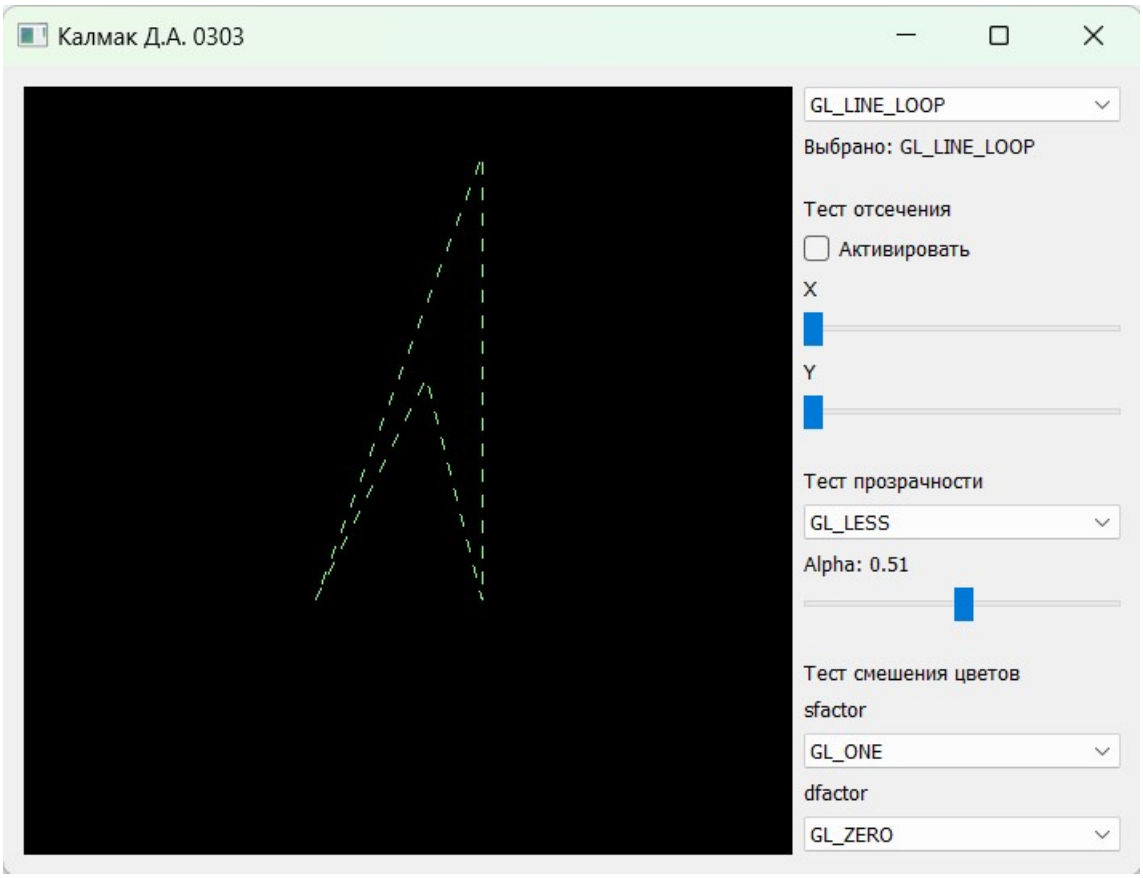

Рисунок 6 – Тест прозрачности с параметром GL_LESS

Пропускает, когда прозрачность фигуры меньше значения alpha на ползунке.

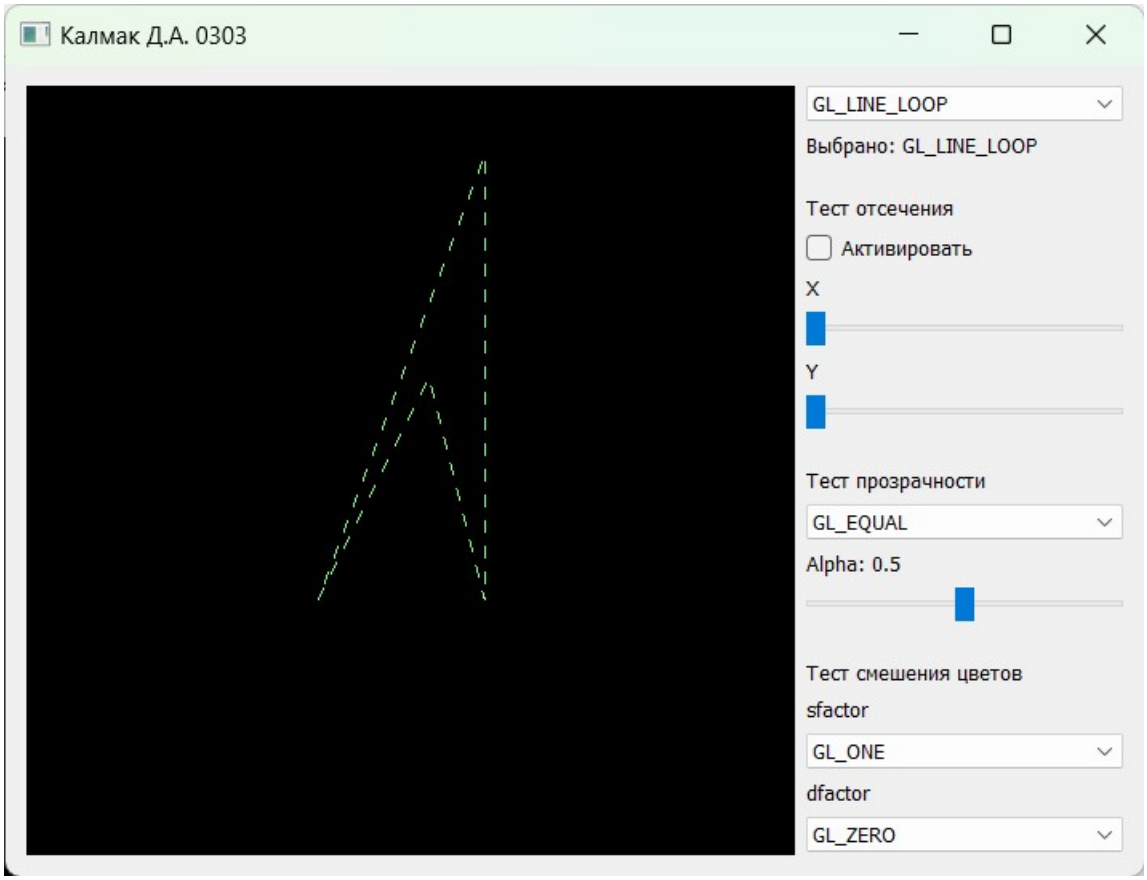

Рисунок 7 – Тест прозрачности с параметром GL_EQUAL

Пропускает, когда прозрачность фигуры равна значению alpha на ползунке.

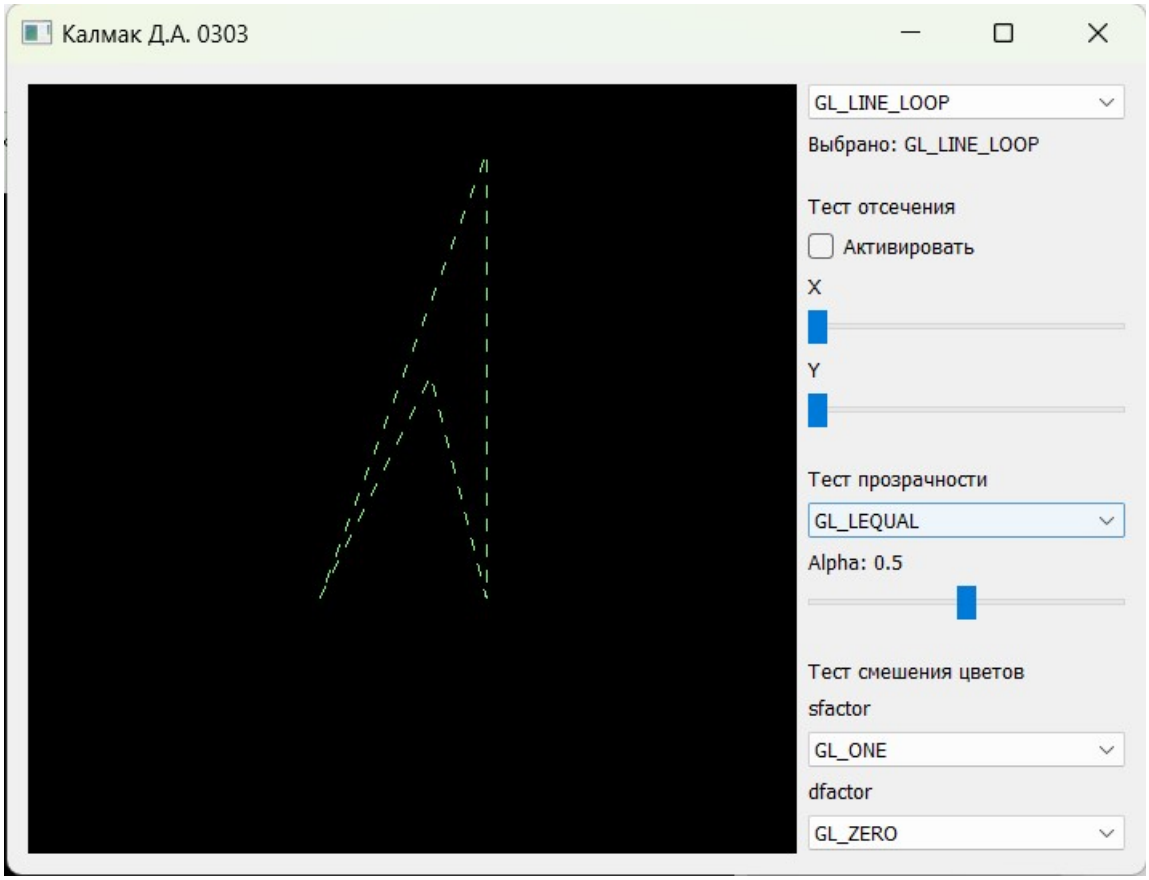

Рисунок 8 – Тест прозрачности с параметром GL_LEQUAL

Пропускает, когда прозрачность фигуры меньше или равна значению alpha на ползунке.

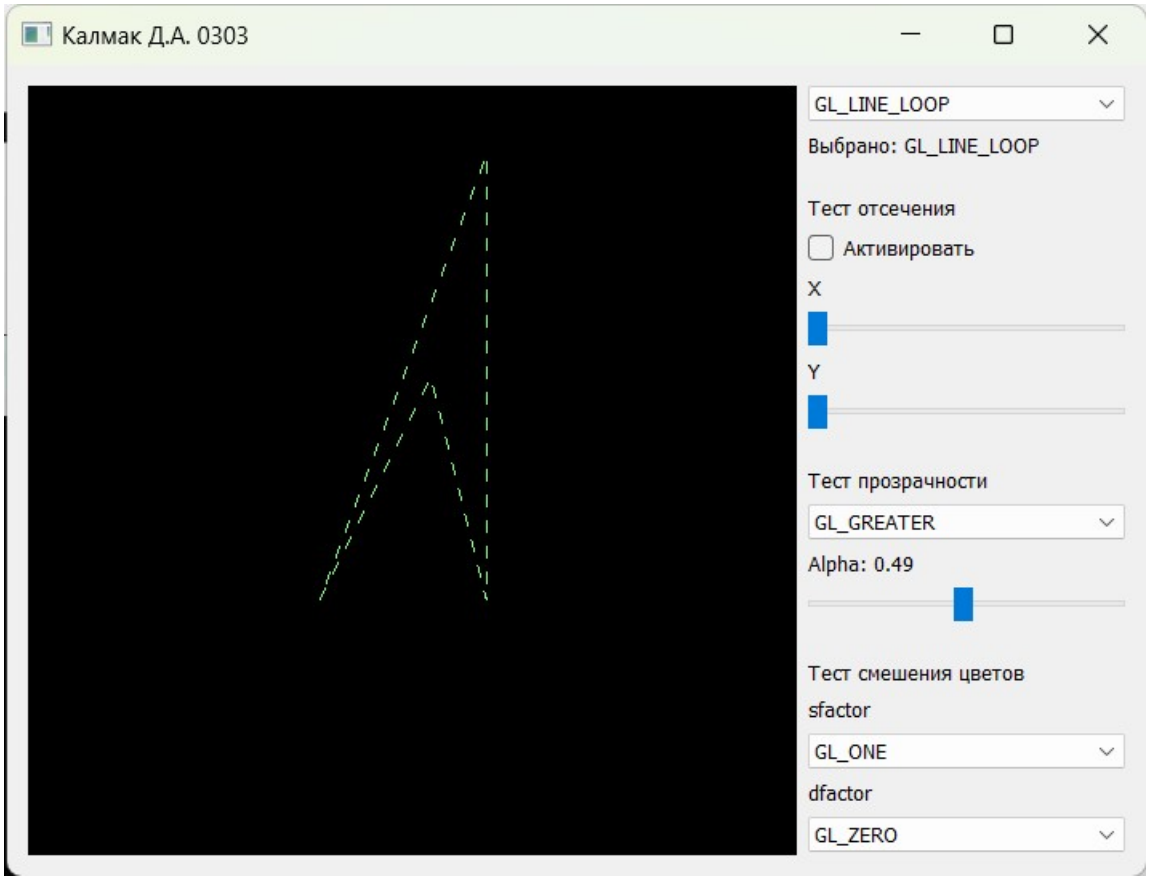

Рисунок 9 – Тест прозрачности с параметром GL_GREATER

Пропускает, когда прозрачность фигуры больше значения alpha на ползунке.

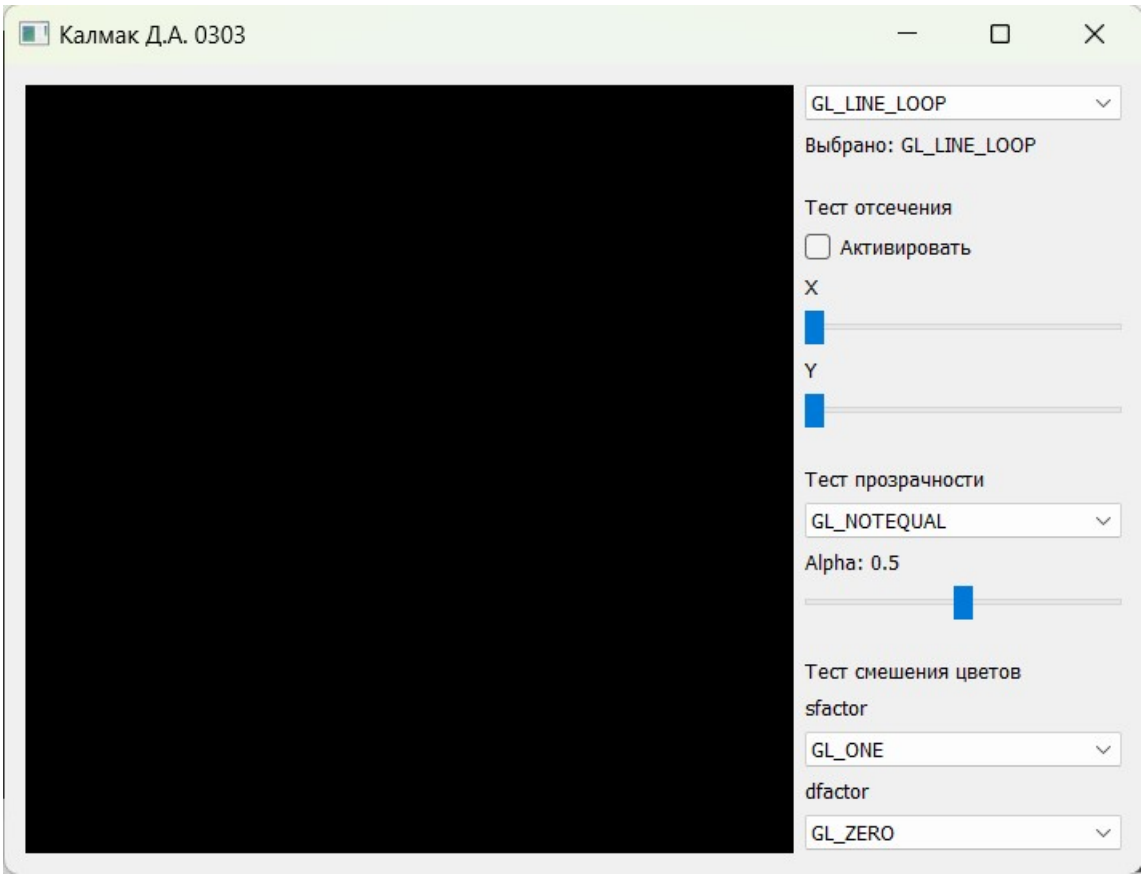

Рисунок 10 – Тест прозрачности с параметром GL_NOTEQUAL

Не пропускает, когда прозрачность фигуры равна значению alpha на ползунке, и пропускает в ином случае.

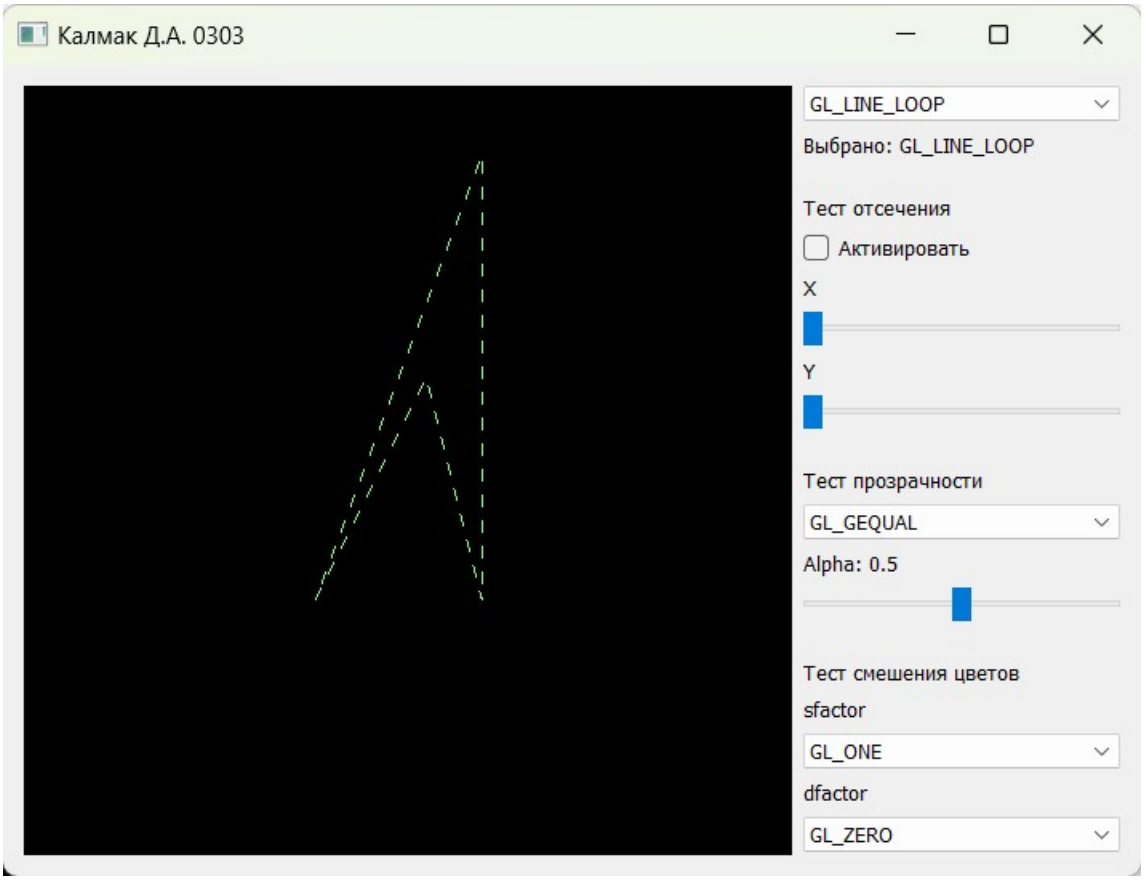

Рисунок 11 – Тест прозрачности с параметром GL_GEQUAL

Пропускает, когда прозрачность фигуры больше или равна значению alpha на ползунке.

sfactor устанавливает способ вычисления входящих факторов смешения RGBA, а dfactor устанавливает способ вычисления факторов смешения RGBA уже находящихся в буфере кадра.

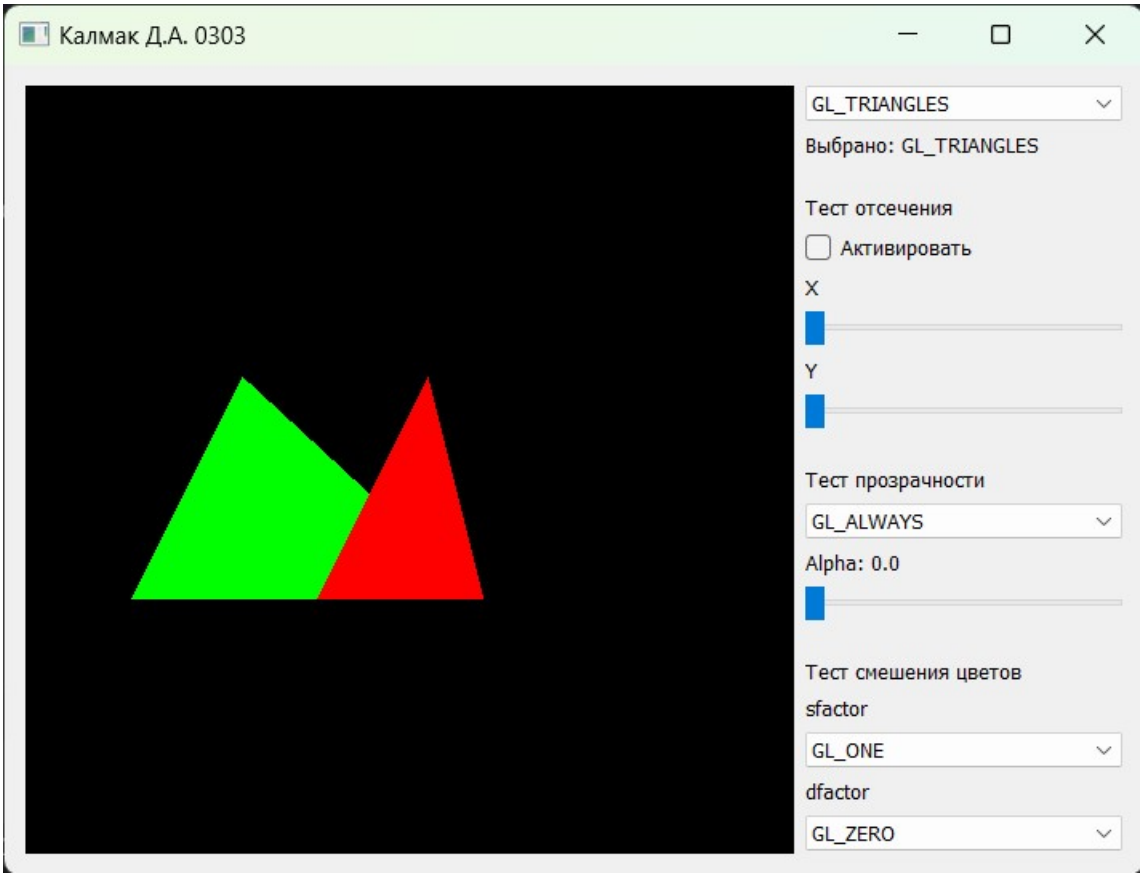

Рисунок 12 – Тест смешения цветов с sfactor GL_ONE и dfactor GL_ZERO

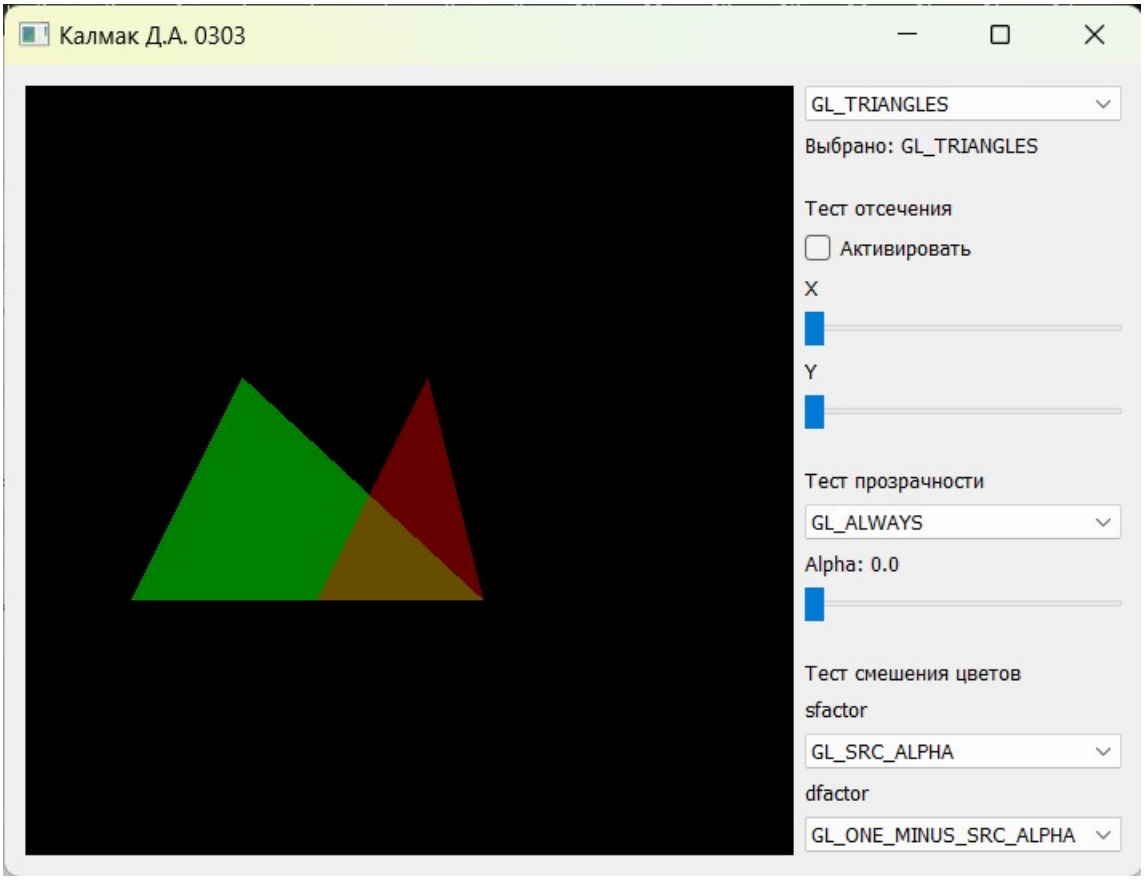

Рисунок 13 – Тест смешения цветов с sfactor GL_SRC_ALPHA и dfactor GL_ONE_MINUS_SRC_ALPHA

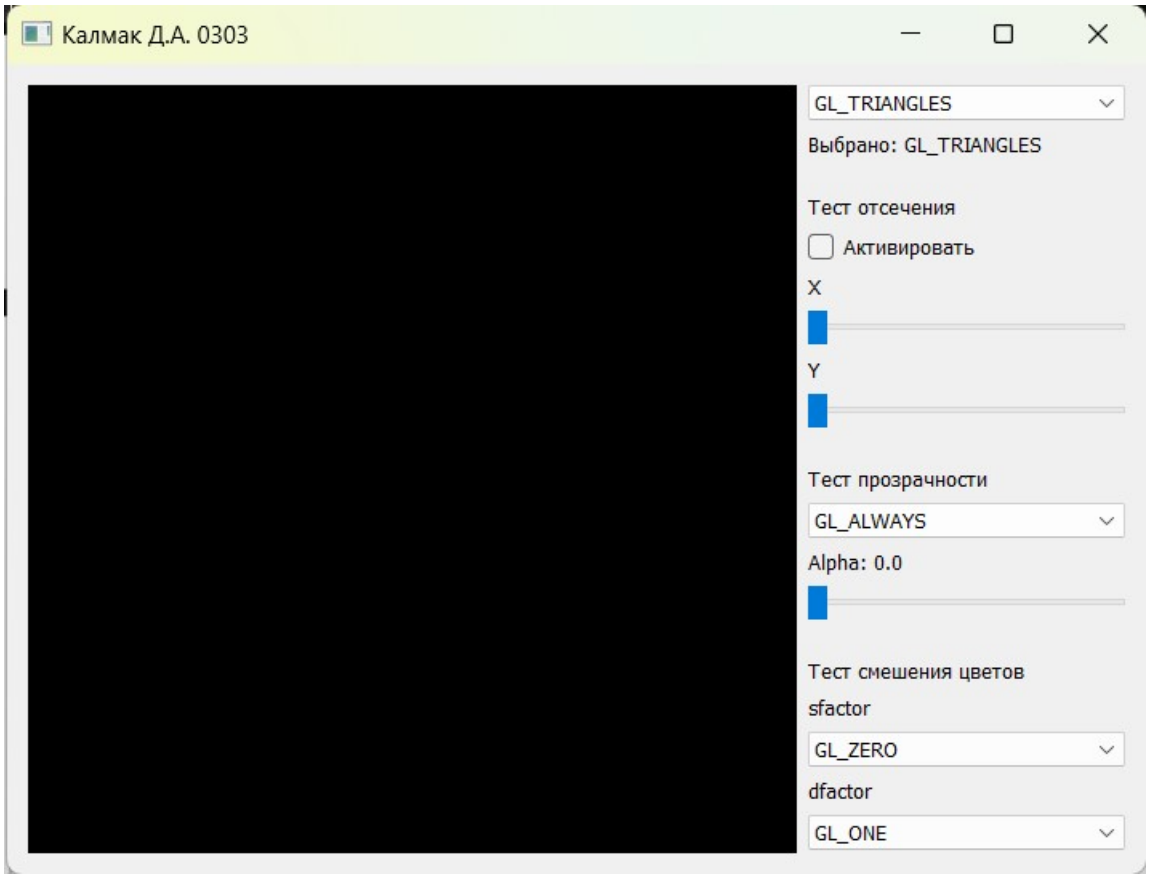

Рисунок 14 – Тест смешения цветов с sfactor GL_ZERO и dfactor GL_ONE

При sfactor GL_ZERO и любом dfactor примитивов не видно.

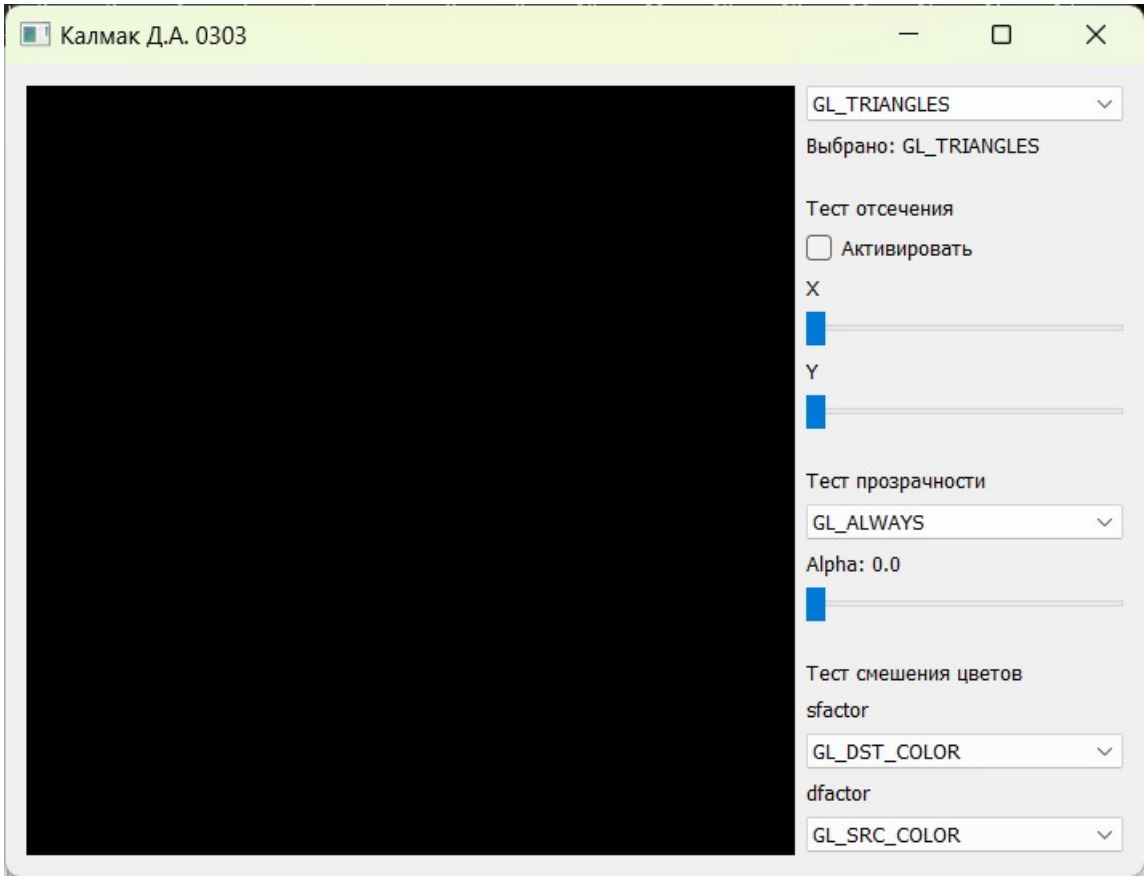

Рисунок 15 – Тест смешения цветов с sfactor GL_DST_COLOR и dfactor GL_SRC_COLOR

При sfactor GL_DST_COLOR и любом dfactor примитивов не видно.

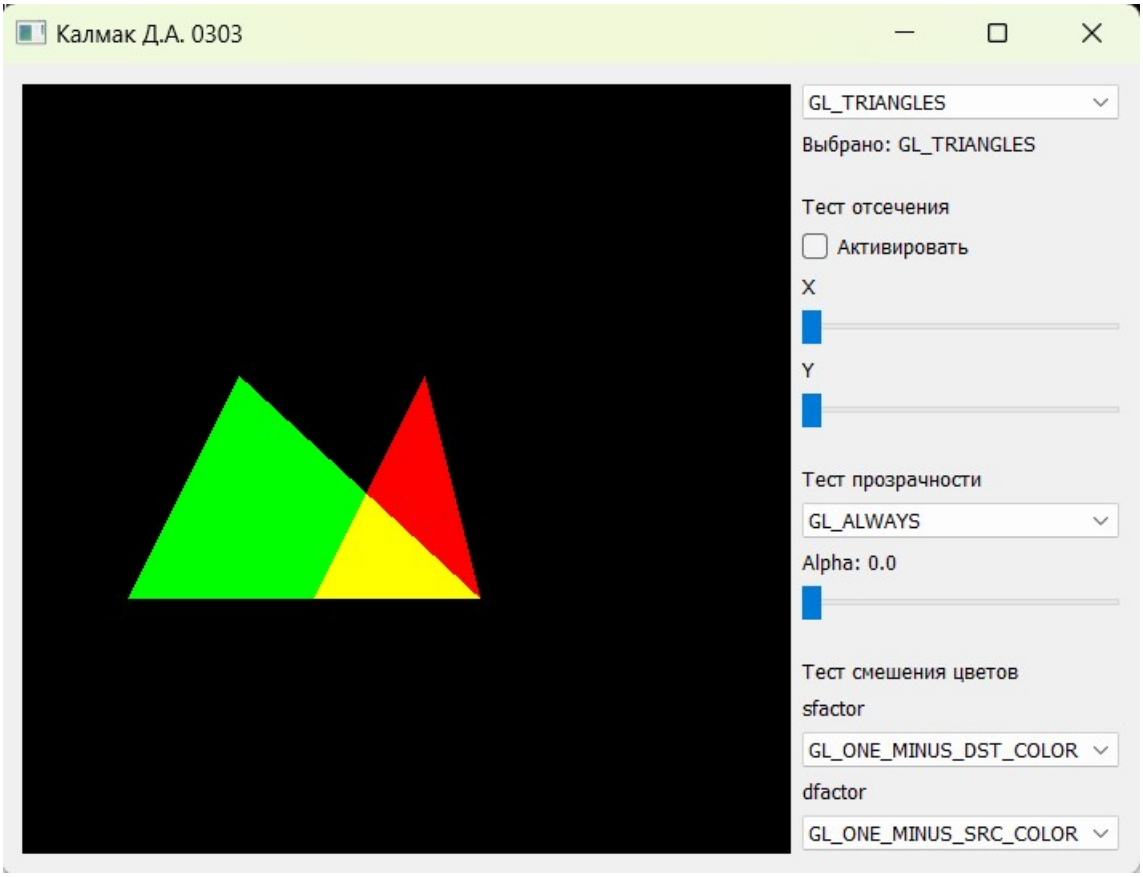

Рисунок 16 – Тест смешения с sfactor GL_ONE_MINUS_DST_COLOR и dfactor GL_ONE_MINUS_SRC_COLOR

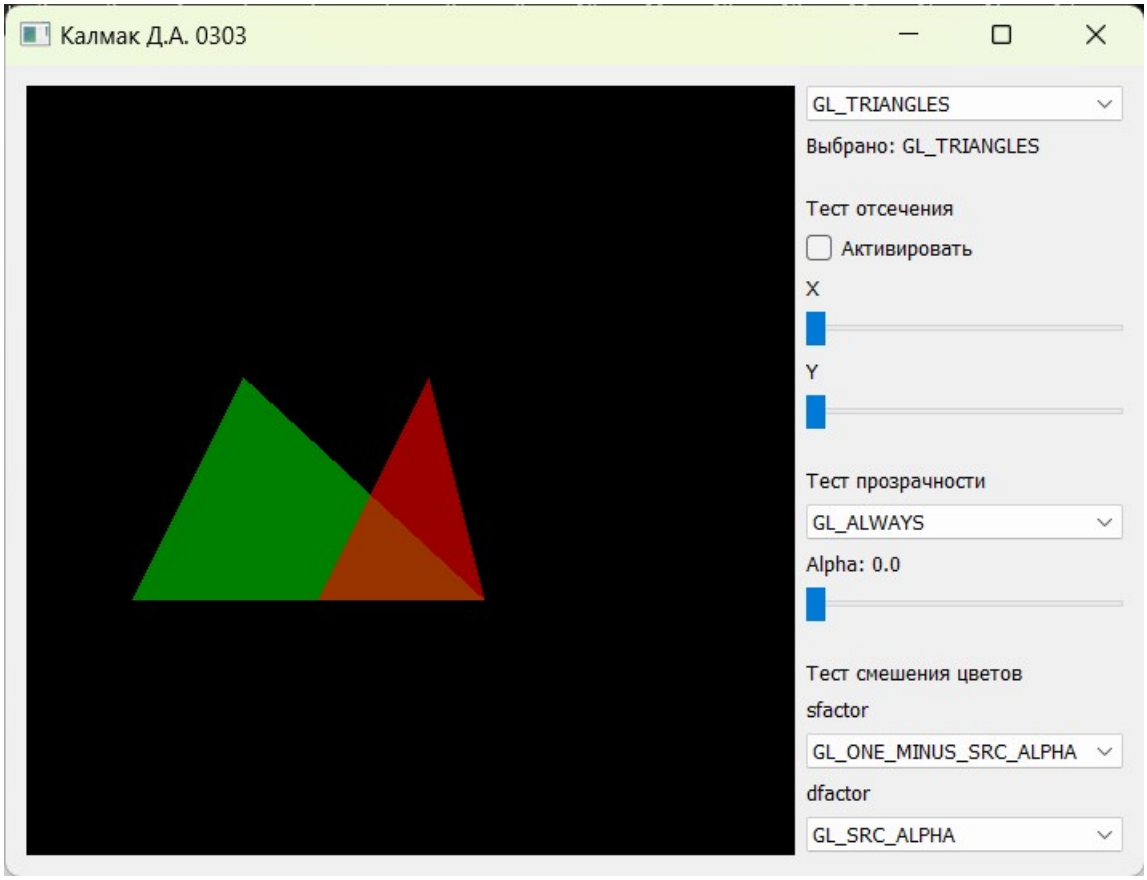

Рисунок 17 – Тест смешения с sfactor GL_ONE_MINUS_SRC_ALPHA и dfactor GL_SRC_ALPHA

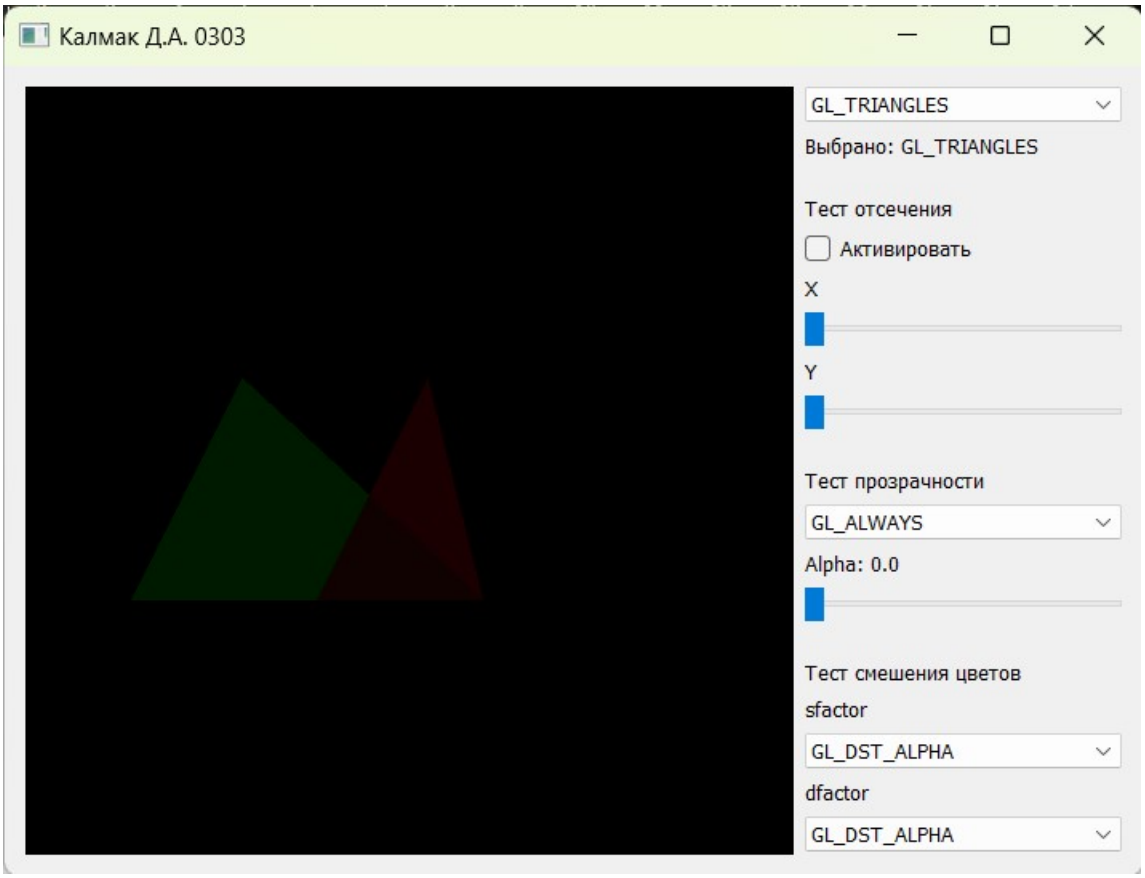

Рисунок 18 – Тест смешения с sfactor GL_DST_ALPHA и dfactor GL_DST_ALPHA

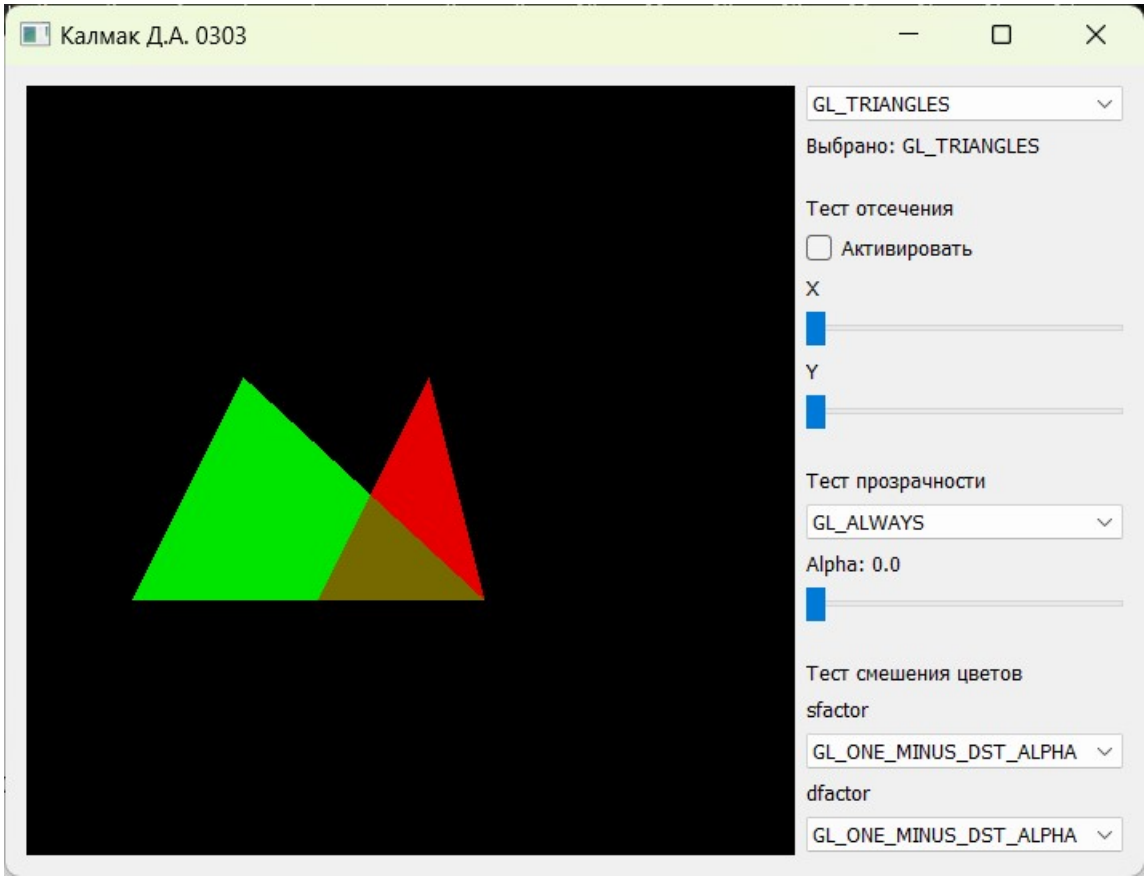

Рисунок 19 – Тест смешения с sfactor GL_ONE_MINUS_DST_ALPHA и dfactor GL_ONE_MINUS_DST_ALPHA

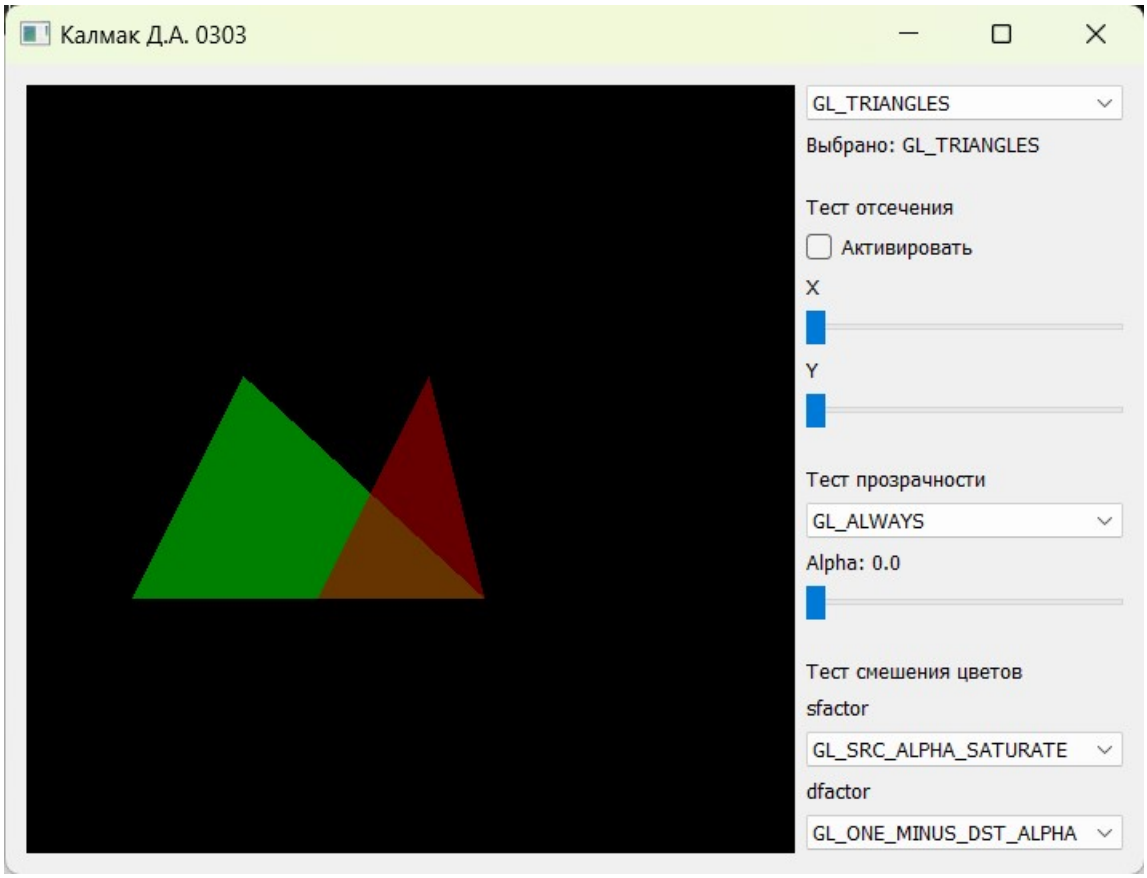

Рисунок 20 – Тест смешения с sfactor GL_SRC_ALPHA_SATURATE и dfactor GL_ONE_MINUS_DST_ALPHA

Вывод.

В результате выполнения лабораторной работы была разработана программа, реализующая представление тестов отсечения, прозрачности и смешивания цветов для графических примитивов OpenGL, разработанных в лабораторной работе № 1. 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.QtCore import Qt
from PyQt5.QtOpenGL import *
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import (QWidget, QLabel,
                             QComboBox, QStackedWidget, QSlider, QCheckBox)


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)

        self.lbltests = QLabel("\nТест отсечения", self)
        self.boxscissor = QCheckBox("Активировать", self)
        self.boxscissor.stateChanged.connect(self.set_scissor)
        self.lblx = QLabel("X", self)
        self.sliderx = QSlider(Qt.Orientation.Horizontal, self)
        self.sliderx.setRange(0, 240)
        self.sliderx.valueChanged.connect(self.update_scissorsx)
        buttonsLayout.addWidget(self.lbltests)
        buttonsLayout.addWidget(self.boxscissor)
        buttonsLayout.addWidget(self.lblx)
        buttonsLayout.addWidget(self.sliderx)
        self.lbly = QLabel("Y", self)
        self.slidery = QSlider(Qt.Orientation.Horizontal, self)
        self.slidery.setRange(0, 240)
```

```
        self.slidery.valueChanged.connect(self.update_scissorsy)
        buttonsLayout.addWidget(self.lbly)
        buttonsLayout.addWidget(self.slidery)

        self.lbltesta = QLabel("\nТест прозрачности", self)
        self.boxalpha = QComboBox()
        self.boxalpha.setMinimumSize(180, 20)
        self.boxalpha.addItems(
            ["GL_NEVER", "GL_LESS", "GL_EQUAL", "GL_LEQUAL", "GL_GREATER",
"GL_NOTEQUAL",
             "GL_GEQUAL", "GL_ALWAYS"])
        self.boxalpha.setCurrentIndex(7)
        self.boxalpha.activated[str].connect(self.activated_boxalpha)
        self.lblalpha = QLabel("Alpha: 0.0", self)
        self.slideralpha = QSlider(Qt.Orientation.Horizontal, self)
        self.slideralpha.setRange(0, 100)
        self.slideralpha.valueChanged.connect(self.update_alpha)
        buttonsLayout.addWidget(self.lbltesta)
        buttonsLayout.addWidget(self.boxalpha)
        buttonsLayout.addWidget(self.lblalpha)
        buttonsLayout.addWidget(self.slideralpha)

        self.lbltestb = QLabel("\nТест смешения цветов", self)
        self.lblsfactor = QLabel("sfactor", self)
        self.boxsfactor = QComboBox()
        self.boxsfactor.setMinimumSize(180, 20)
        self.boxsfactor.addItems(
            ["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", "GL_SRC_ALPHA_SATURATE"])
        self.boxsfactor.setCurrentIndex(1)
        self.boxsfactor.activated[str].connect(self.activated_boxsfactor)
        self.lbldfactor = QLabel("dfactor", self)
        self.boxdfactor = QComboBox()
        self.boxdfactor.setMinimumSize(180, 20)
        self.boxdfactor.addItems(
            ["GL_ZERO", "GL_ONE", "GL_SRC_COLOR", "GL_ONE_MINUS_SRC_COLOR",
"GL_SRC_ALPHA", "GL_ONE_MINUS_SRC_ALPHA",
             "GL_DST_ALPHA", "GL_ONE_MINUS_DST_ALPHA"])
        self.boxdfactor.activated[str].connect(self.activated_boxdfactor)
        buttonsLayout.addWidget(self.lbltestb)
        buttonsLayout.addWidget(self.lblsfactor)
        buttonsLayout.addWidget(self.boxsfactor)
        buttonsLayout.addWidget(self.lbldfactor)
        buttonsLayout.addWidget(self.boxdfactor)
        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)
```

```
def set_scissor(self, state):
    if state == Qt.Checked:
        for i in range(self.stack.__len__()):
            self.stack.widget(i).scissor_flag = True
            self.stack.widget(i).updateGL()
    else:
        for i in range(self.stack.__len__()):
            self.stack.widget(i).scissor_flag = False
            self.stack.widget(i).updateGL()

def update_scissorsx(self, value):
    for i in range(self.stack.__len__()):
        self.stack.widget(i).x = value
        self.stack.widget(i).updateGL()

def update_scissorsy(self, value):
    for i in range(self.stack.__len__()):
        self.stack.widget(i).y = value
        self.stack.widget(i).updateGL()

def activated_boxalpha(self, text):
    for i in range(self.stack.__len__()):
        if text == "GL_NEVER":
            text = GL_NEVER
        if text == "GL_LESS":
            text = GL_LESS
        if text == "GL_EQUAL":
            text = GL_EQUAL
        if text == "GL_LEQUAL":
            text = GL_LEQUAL
        if text == "GL_GREATER":
            text = GL_GREATER
        if text == "GL_NOTEQUAL":
            text = GL_NOTEQUAL
        if text == "GL_GEQUAL":
            text = GL_GEQUAL
        if text == "GL_ALWAYS":
            text = GL_ALWAYS
        self.stack.widget(i).alphafunc = text
        self.stack.widget(i).updateGL()

def update_alpha(self, value):
    self.lblalpha.setText("Alpha: " + str(value / 100))
    for i in range(self.stack.__len__()):
        self.stack.widget(i).alpharef = value
        self.stack.widget(i).updateGL()

def activated_boxsfactor(self, text):
    for i in range(self.stack.__len__()):
        if text == "GL_ZERO":
            text = GL_ZERO
        if text == "GL_ONE":
            text = GL_ONE
        if text == "GL_DST_COLOR":
            text = GL_DST_COLOR
        if text == "GL_ONE_MINUS_DST_COLOR":
```

```
                text = GL_ONE_MINUS_DST_COLOR
            if text == "GL_SRC_ALPHA":
                text = GL_SRC_ALPHA
            if text == "GL_ONE_MINUS_SRC_ALPHA":
                text = GL_ONE_MINUS_SRC_ALPHA
            if text == "GL_DST_ALPHA":
                text = GL_DST_ALPHA
            if text == "GL_ONE_MINUS_DST_ALPHA":
                text = GL_ONE_MINUS_DST_ALPHA
            if text == "GL_SRC_ALPHA_SATURATE":
                text = GL_SRC_ALPHA_SATURATE
            self.stack.widget(i).sfact = text
            self.stack.widget(i).updateGL()

    def activated_boxdfactor(self, text):
        for i in range(self.stack.__len__()):
            if text == "GL_ZERO":
                text = GL_ZERO
            if text == "GL_ONE":
                text = GL_ONE
            if text == "GL_SRC_COLOR":
                text = GL_SRC_COLOR
            if text == "GL_ONE_MINUS_SRC_COLOR":
                text = GL_ONE_MINUS_SRC_COLOR
            if text == "GL_SRC_ALPHA":
                text = GL_SRC_ALPHA
            if text == "GL_ONE_MINUS_SRC_ALPHA":
                text = GL_ONE_MINUS_SRC_ALPHA
            if text == "GL_DST_ALPHA":
                text = GL_DST_ALPHA
            if text == "GL_ONE_MINUS_DST_ALPHA":
                text = GL_ONE_MINUS_DST_ALPHA
            self.stack.widget(i).dfact = text
            self.stack.widget(i).updateGL()


class glWidget0(QGLWidget):
    def __init__(self, parent=None):
        QGLWidget.__init__(self, parent)
        self.setMinimumSize(480, 480)
        self.w = 480
        self.h = 480
        self.scissor_flag = False
        self.x = 0
        self.y = 0
        self.alphafunc = GL_ALWAYS
        self.alpharef = 0
        self.sfact = GL_ONE
        self.dfact = GL_ZERO

    def initializeGL(self):
        glClearColor(0, 0, 0, 0.1)
        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

    def resizeGL(self, w, h):
        self.w = w
        self.h = h
        glViewport(0, 0, w, h)
        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()
        aspect = w / h
        gluPerspective(45.0, aspect, 0.1, 100)
        glMatrixMode(GL_MODELVIEW)


class glWidget1(glWidget0):
    def paintGL(self):
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glLoadIdentity()
        glTranslatef(-2.5, 0.5, -5.0)
        glColor4f(0.5, 1.0, 0.5, 0.5)
        glDepthMask(GL_FALSE)
        glEnable(GL_BLEND)
        glBlendFunc(self.sfact, self.dfact)
        glEnable(GL_ALPHA_TEST)
        glAlphaFunc(self.alphafunc, self.alpharef / 100)
        if self.scissor_flag:
            glEnable(GL_SCISSOR_TEST)
            glScissor(self.x, self.y, int(self.w / 2), int(self.h / 2))
        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()
        glDisable(GL_SCISSOR_TEST)
        glDisable(GL_ALPHA_TEST)
        glDepthMask(GL_TRUE)
        glDisable(GL_BLEND)
        glFlush()


class glWidget2(glWidget0):
    def paintGL(self):
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glLoadIdentity()
        glTranslatef(-2.5, 0.5, -5.0)
        glColor4f(0.5, 1.0, 0.5, 0.5)
        glDepthMask(GL_FALSE)
        glEnable(GL_BLEND)
        glBlendFunc(self.sfact, self.dfact)
        glEnable(GL_ALPHA_TEST)
        glAlphaFunc(self.alphafunc, self.alpharef / 100)
        if self.scissor_flag:
            glEnable(GL_SCISSOR_TEST)
```

```
            glScissor(self.x, self.y, int(self.w / 2), int(self.h / 2))
        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()
        glDisable(GL_SCISSOR_TEST)
        glDisable(GL_ALPHA_TEST)
        glDepthMask(GL_TRUE)
        glDisable(GL_BLEND)
        glFlush()


class glWidget3(glWidget0):
    def paintGL(self):
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glLoadIdentity()
        glTranslatef(-2.5, 0.5, -5.0)
        glColor4f(0.5, 1.0, 0.5, 0.5)
        glEnable(GL_LINE_STIPPLE)
        glLineStipple(1, 0x0101)
        glDepthMask(GL_FALSE)
        glEnable(GL_BLEND)
        glBlendFunc(self.sfact, self.dfact)
        glEnable(GL_ALPHA_TEST)
        glAlphaFunc(self.alphafunc, self.alpharef / 100)
        if self.scissor_flag:
            glEnable(GL_SCISSOR_TEST)
            glScissor(self.x, self.y, int(self.w / 2), int(self.h / 2))
        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()
        glDisable(GL_SCISSOR_TEST)
        glDisable(GL_ALPHA_TEST)
        glDepthMask(GL_TRUE)
        glDisable(GL_BLEND)
        glFlush()


class glWidget4(glWidget0):
    def paintGL(self):
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glLoadIdentity()
        glTranslatef(-2.5, 0.5, -5.0)
        glColor4f(0.5, 1.0, 0.5, 0.5)
        glEnable(GL_LINE_STIPPLE)
        glLineStipple(1, 0x00FF)
        glDepthMask(GL_FALSE)
        glEnable(GL_BLEND)
        glBlendFunc(self.sfact, self.dfact)
        glEnable(GL_ALPHA_TEST)
        glAlphaFunc(self.alphafunc, self.alpharef / 100)
        if self.scissor_flag:
```

```
            glEnable(GL_SCISSOR_TEST)
            glScissor(self.x, self.y, int(self.w / 2), int(self.h / 2))
        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()
        glDisable(GL_SCISSOR_TEST)
        glDisable(GL_ALPHA_TEST)
        glDepthMask(GL_TRUE)
        glDisable(GL_BLEND)
        glFlush()


class glWidget5(glWidget0):
    def paintGL(self):
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glLoadIdentity()
        glTranslatef(-2.5, 0.5, -5.0)
        glColor4f(0, 1.0, 0, 0.5)
        glPolygonMode(GL_FRONT, GL_FILL)
        glDepthMask(GL_FALSE)
        glEnable(GL_BLEND)
        glBlendFunc(self.sfact, self.dfact)
        glEnable(GL_ALPHA_TEST)
        glAlphaFunc(self.alphafunc, self.alpharef / 100)
        if self.scissor_flag:
            glEnable(GL_SCISSOR_TEST)
            glScissor(self.x, self.y, int(self.w / 2), int(self.h / 2))
        glBegin(GL_TRIANGLES)
        glVertex3f(1.0, -1.2, 0.0)
        glVertex3f(1.6, 0.0, 0.0)
        glVertex3f(2.9, -1.2, 0.0)
        glEnd()
        glColor4f(1.0, 0, 0, 0.4)
        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()
        glDisable(GL_SCISSOR_TEST)
        glDisable(GL_ALPHA_TEST)
        glDepthMask(GL_TRUE)
        glDisable(GL_BLEND)
        glFlush()


class glWidget6(glWidget0):
    def paintGL(self):
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glLoadIdentity()
        glTranslatef(-2.5, 0.5, -5.0)
        glColor4f(0.5, 1.0, 0.5, 0.5)
        glPolygonMode(GL_FRONT_AND_BACK, GL_LINE)
        glDepthMask(GL_FALSE)
        glEnable(GL_BLEND)
```

```
        glBlendFunc(self.sfact, self.dfact)
        glEnable(GL_ALPHA_TEST)
        glAlphaFunc(self.alphafunc, self.alpharef / 100)
        if self.scissor_flag:
            glEnable(GL_SCISSOR_TEST)
            glScissor(self.x, self.y, int(self.w / 2), int(self.h / 2))
        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()
        glDisable(GL_SCISSOR_TEST)
        glDisable(GL_ALPHA_TEST)
        glDepthMask(GL_TRUE)
        glDisable(GL_BLEND)
        glFlush()


class glWidget7(glWidget0):
    def paintGL(self):
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glLoadIdentity()
        glTranslatef(-2.5, 0.5, -5.0)
        glColor4f(0.5, 1.0, 0.5, 0.5)
        glPolygonMode(GL_FRONT_AND_BACK, GL_LINE)
        glDepthMask(GL_FALSE)
        glEnable(GL_BLEND)
        glBlendFunc(self.sfact, self.dfact)
        glEnable(GL_ALPHA_TEST)
        glAlphaFunc(self.alphafunc, self.alpharef / 100)
        if self.scissor_flag:
            glEnable(GL_SCISSOR_TEST)
            glScissor(self.x, self.y, int(self.w / 2), int(self.h / 2))
        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()
        glDisable(GL_SCISSOR_TEST)
        glDisable(GL_ALPHA_TEST)
        glDepthMask(GL_TRUE)
        glDisable(GL_BLEND)
        glFlush()


class glWidget8(glWidget0):
    def paintGL(self):
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glLoadIdentity()
        glTranslatef(-2.5, 0.5, -5.0)
        glColor4f(0.5, 1.0, 0.5, 0.5)
        glPolygonMode(GL_FRONT_AND_BACK, GL_LINE)
        glDepthMask(GL_FALSE)
```

```
        glEnable(GL_BLEND)
        glBlendFunc(self.sfact, self.dfact)
        glEnable(GL_ALPHA_TEST)
        glAlphaFunc(self.alphafunc, self.alpharef / 100)
        if self.scissor_flag:
            glEnable(GL_SCISSOR_TEST)
            glScissor(self.x, self.y, int(self.w / 2), int(self.h / 2))
        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()
        glDisable(GL_SCISSOR_TEST)
        glDisable(GL_ALPHA_TEST)
        glDepthMask(GL_TRUE)
        glDisable(GL_BLEND)
        glFlush()


class glWidget9(glWidget0):
    def paintGL(self):
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glLoadIdentity()
        glTranslatef(-2.5, 0.5, -5.0)
        glColor4f(0.5, 1.0, 0.5, 0.5)
        glPolygonMode(GL_FRONT_AND_BACK, GL_LINE)
        glDepthMask(GL_FALSE)
        glEnable(GL_BLEND)
        glBlendFunc(self.sfact, self.dfact)
        glEnable(GL_ALPHA_TEST)
        glAlphaFunc(self.alphafunc, self.alpharef / 100)
        if self.scissor_flag:
            glEnable(GL_SCISSOR_TEST)
            glScissor(self.x, self.y, int(self.w / 2), int(self.h / 2))
        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()
        glDisable(GL_SCISSOR_TEST)
        glDisable(GL_ALPHA_TEST)
        glDepthMask(GL_TRUE)
        glDisable(GL_BLEND)
        glFlush()


class glWidget10(glWidget0):
    def paintGL(self):
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glLoadIdentity()
        glTranslatef(-2.5, 0.5, -5.0)
        glColor4f(0.5, 1.0, 0.5, 0.5)
        glPolygonMode(GL_FRONT, GL_FILL)
```

```
        glDepthMask(GL_FALSE)
        glEnable(GL_BLEND)
        glBlendFunc(self.sfact, self.dfact)
        glEnable(GL_ALPHA_TEST)
        glAlphaFunc(self.alphafunc, self.alpharef / 100)
        if self.scissor_flag:
            glEnable(GL_SCISSOR_TEST)
            glScissor(self.x, self.y, int(self.w / 2), int(self.h / 2))
        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()
        glDisable(GL_SCISSOR_TEST)
        glDisable(GL_ALPHA_TEST)
        glDepthMask(GL_TRUE)
        glDisable(GL_BLEND)
        glFlush()


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