Санкт-Петербургский Государственный
Электротехнический Университет
Кафедра МОЭВМ

Задание для лабораторной работы № 2
"Примитивы OpenGL"

Студенты гр. 1384 _____ Усачева Д.В.
Пчелинцева К.Р.

Преподаватель _____ Герасимова Т.В.

Санкт-Петербург
2024 г.

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 установки 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 – 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).

Выполнение работы

Работа выполнена на операционной системе Windows 10. Приложение было создано на Python 3.10 с применением библиотеки OpenGL, в среде разработки VSCode. Для создания интерфейса использовалась библиотека PyQt6.

Пользовательский интерфейс, был взят из 1 работы и был расширен: добавлены тесты отсечения, прозрачности, смешения цветов.

Тестирование

Результаты тестирования представлены на снимках экрана.

Тест прозрачности

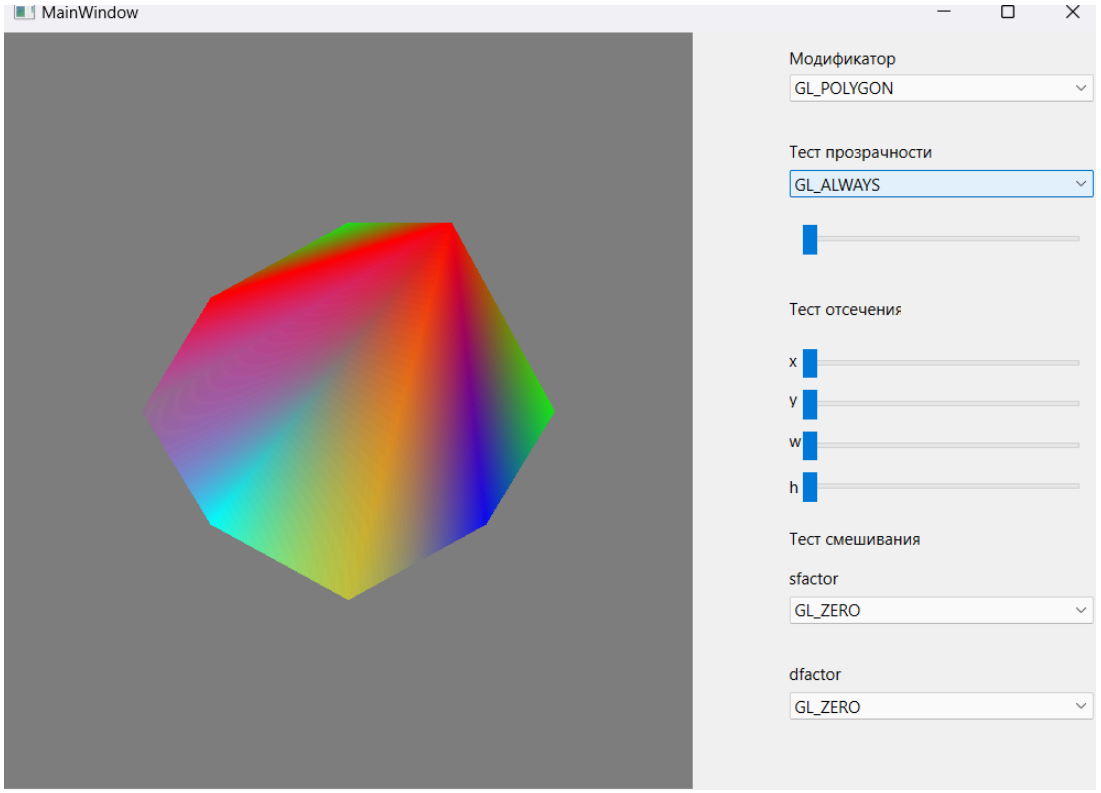

Рисунок 1 — Пропускает цвет всегда

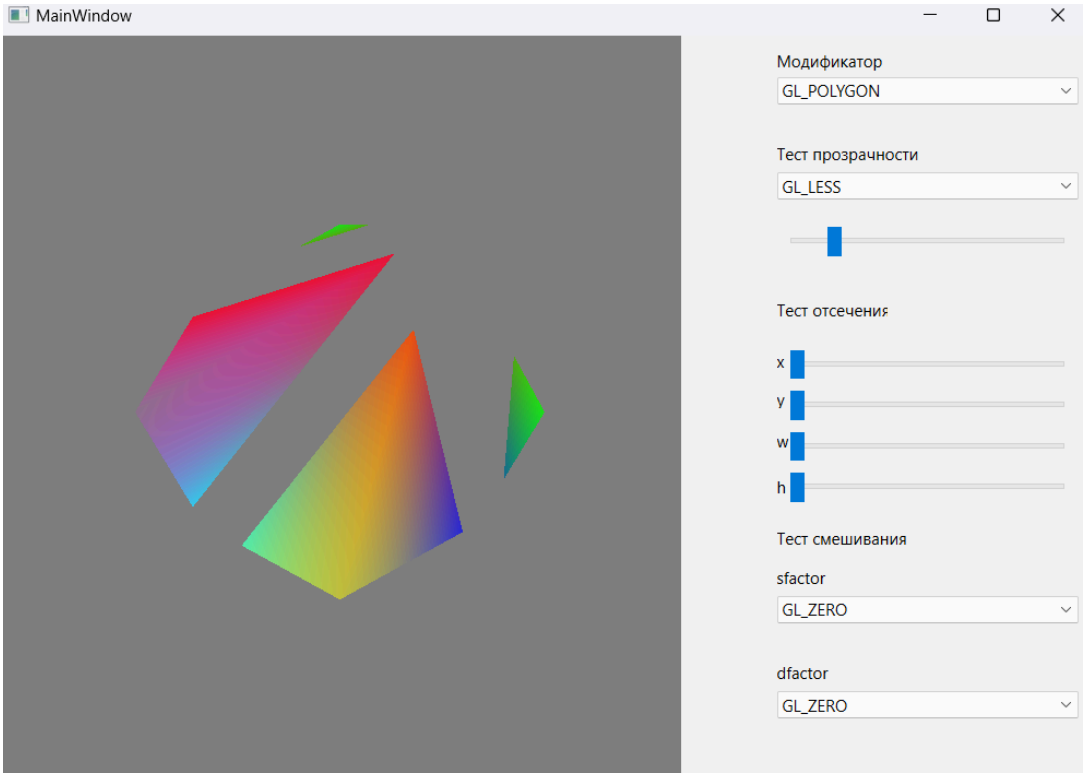

Рисунок 2 — Пропускает, если входное значение альфа меньше, чем значение ref

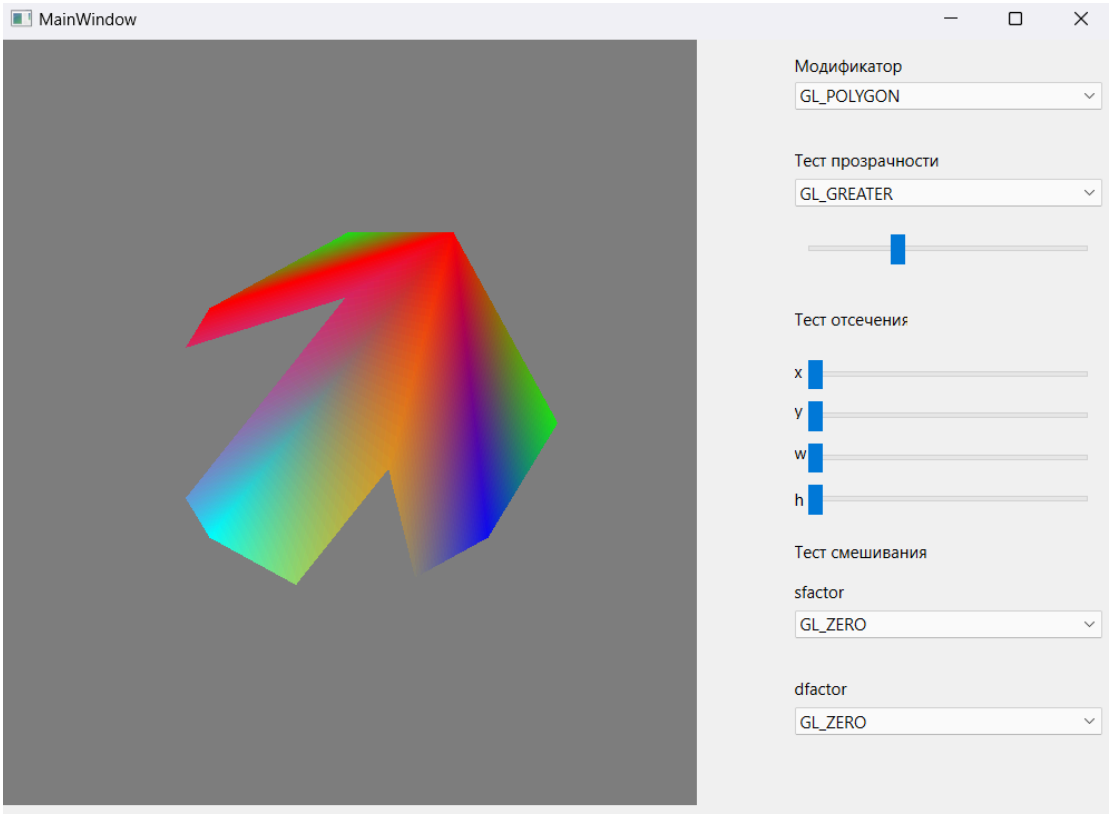

Рисунок 3 — Пропускает, если входное значение альфа больше, чем значение ref

Тест смешивания цветов

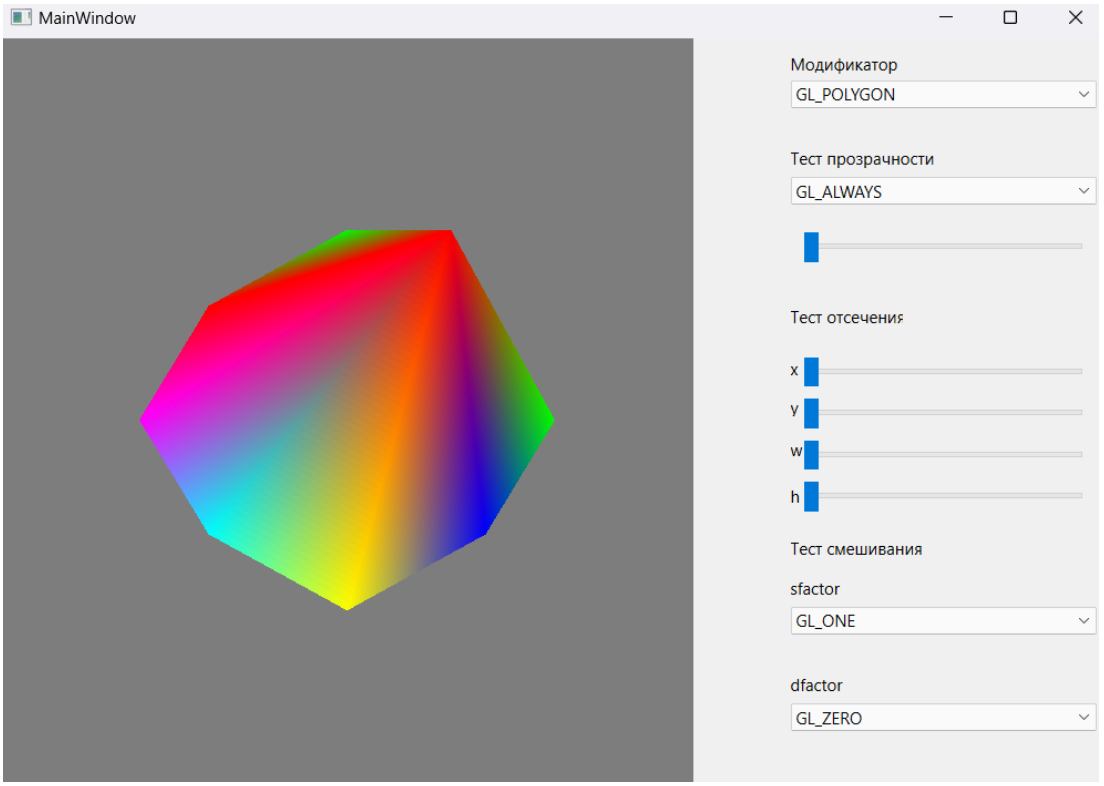

Рисунок 4 — Цвет на переднем фоне отображается, на заднем фоне нет

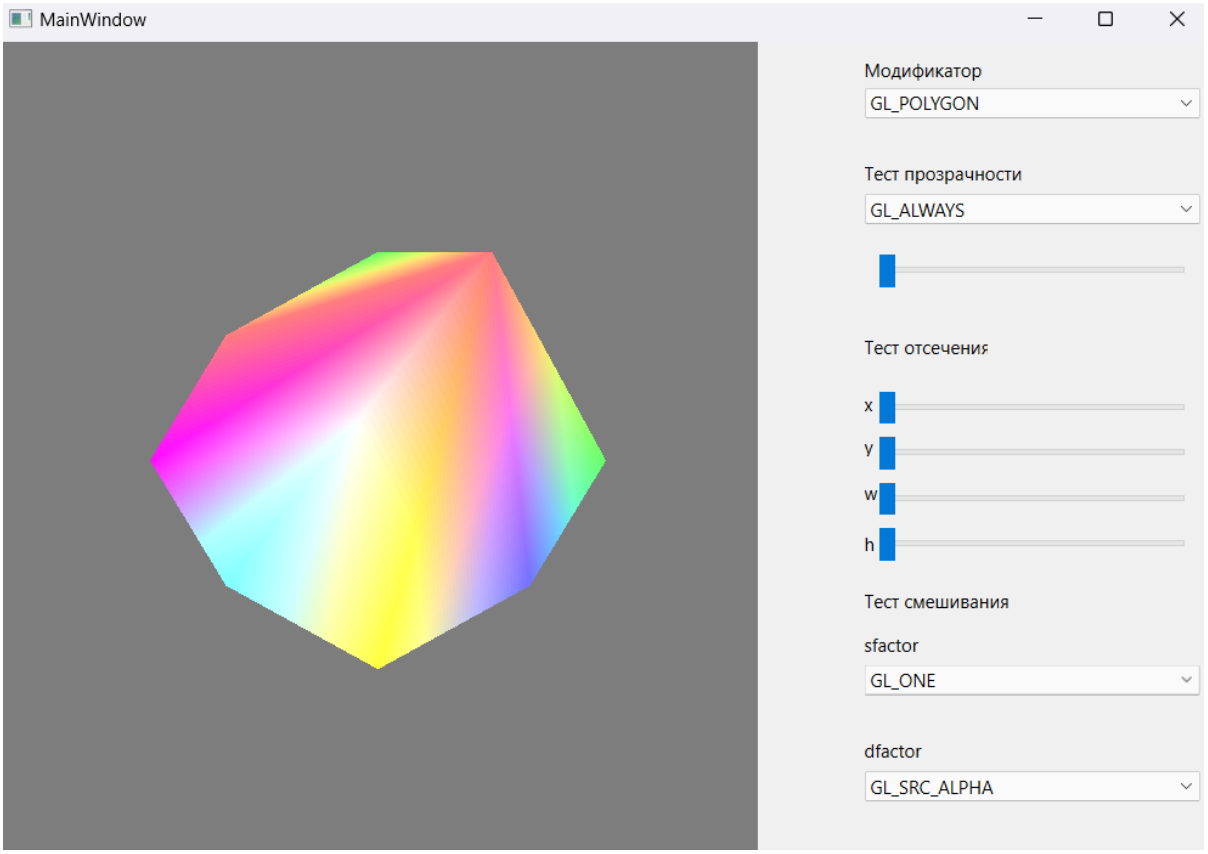

Рисунок 5 — Цвет на переднем фоне отображается, на заднем фоне отображен согласно своей прозрачности

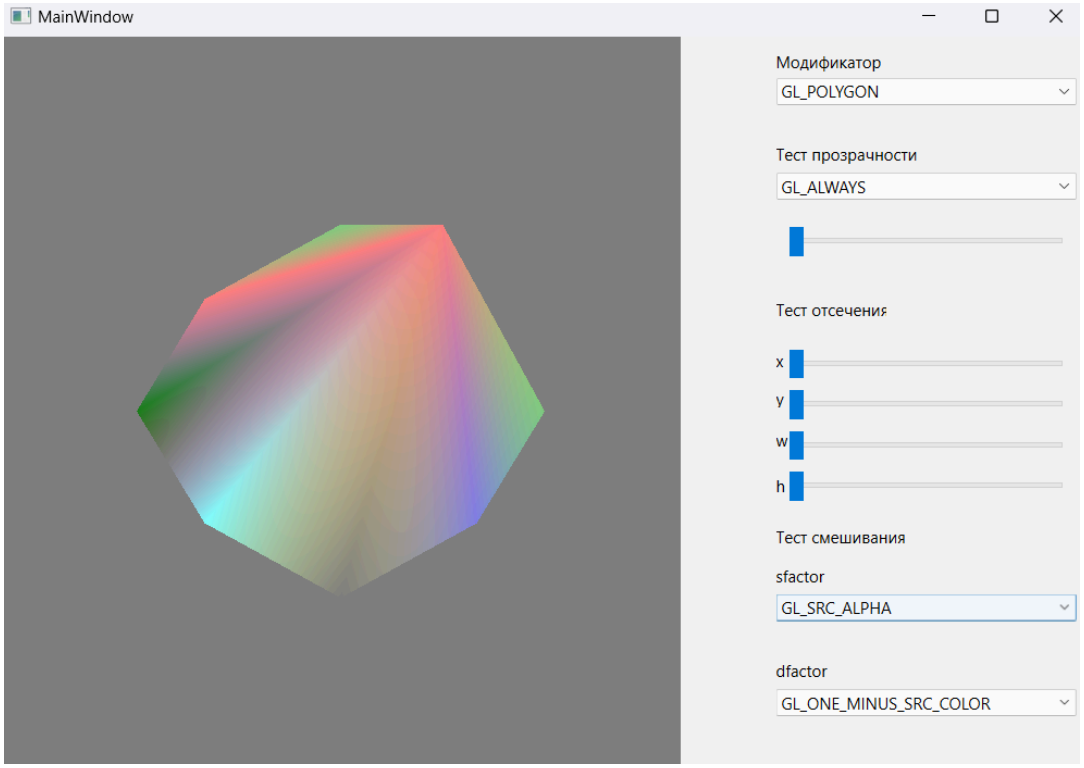

Рисунок 6 — Цвет на переднем плане отображен согласно своей прозрачности, цвет на заднем фоне отображается как цвет, обратный цвету переднего фона

Тест отсечения

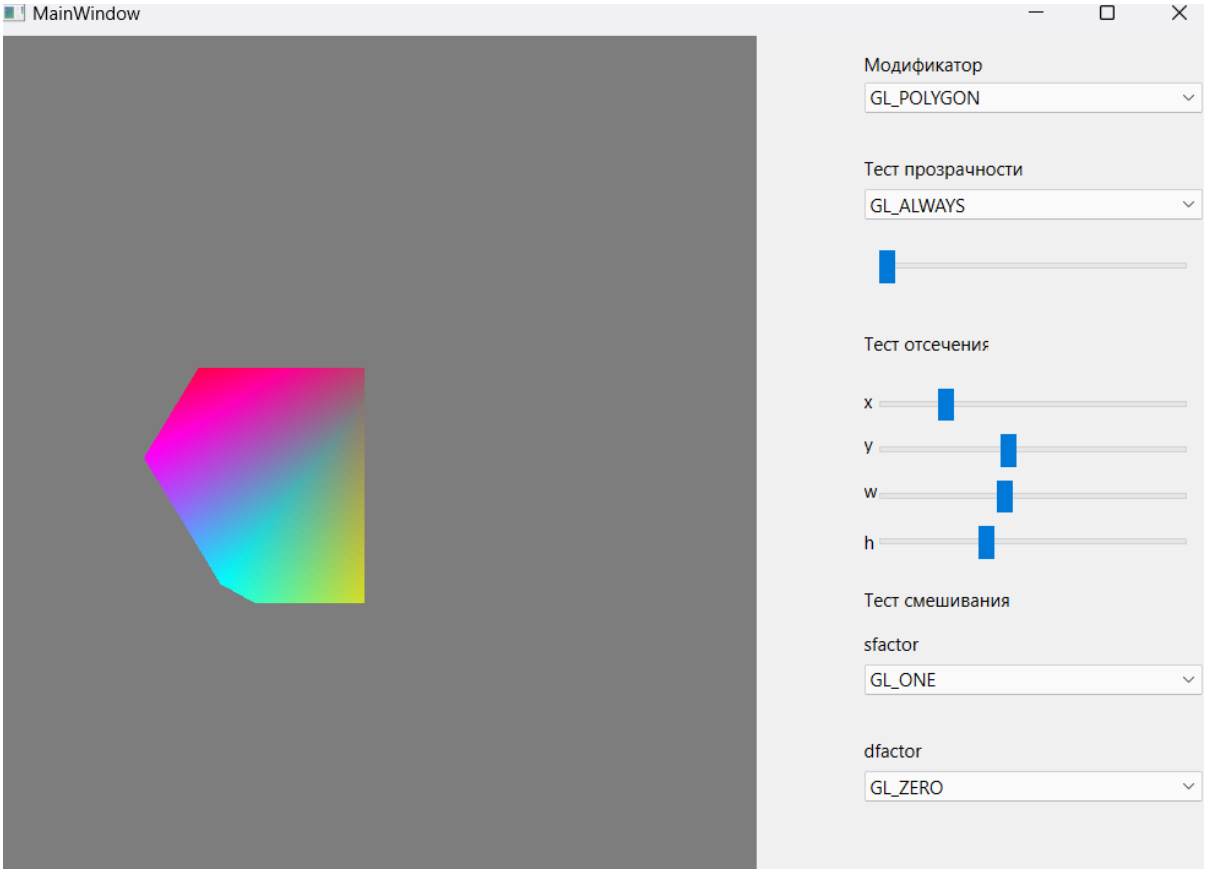

Рисунок 7 — Тест отсечения

Вывод

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 А

ИСХОДНЫЙ КОД ПРОГРАММЫ

main.py

```
from PyQt6 import QtCore, QtWidgets
import sys
from OpenGL.GL import *
from OpenGL.GLU import *
from OpenGL.GLUT import *
from    PyQt6.QtWidgets    import    QApplication,    QMainWindow,
QVBoxLayout, QComboBox, QWidget
from PyQt6.QtOpenGLWidgets import QOpenGLWidget
from commands import commands, func, sfactor, dfactor

from PyQt6 import QtCore, QtGui, QtWidgets


class MyGLWidget(QOpenGLWidget):
    def __init__(self, parent):
        super(MyGLWidget, self).__init__(parent)
        self.x_cut = 0
        self.y_cut = 0
        self.width = 99
        self.height = 99
        self.ref = 0
        self.mode = "GL_POINTS"
        self.func = "GL_ALWAYS"
        self.sfactor = "GL_SRC_ALPHA"
        self.dfactor = "GL_ONE_MINUS_SRC_ALPHA"

    def paintGL(self):
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glClearColor(0.5, 0.5, 0.5, 1.0)
        glEnable(GL_ALPHA_TEST)
        glEnable(GL_SCISSOR_TEST)
        glEnable(GL_BLEND)
        glBlendFunc(sfactor[self.sfactor], dfactor[self.dfactor])
        commands[self.mode]()
        glAlphaFunc(func[self.func], self.ref)
        glScissor(
            int(((self.x_cut) / 100) * 500),
            int(((self.y_cut) / 100) * 550),
            int(((self.width + 1) / 100) * 500),
            int(((self.height + 1) / 100) * 550),
        )
        glDisable(GL_SCISSOR_TEST)
        glDisable(GL_ALPHA_TEST)
        glDisable(GL_BLEND)
        self.update()


class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
```

```
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(800, 585)
        self.centralwidget = QtWidgets.QWidget(parent=MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.glWidget = MyGLWidget(parent=self.centralwidget)
        self.glWidget.setGeometry(QtCore.QRect(0, 0, 500, 550))
        self.mode = QtWidgets.QComboBox(parent=self.centralwidget)
        self.mode.setGeometry(QtCore.QRect(570, 30, 221, 20))
        self.mode.setObjectName("mode")
        self.mode.addItem("")
        self.mode.addItem("")
        self.mode.addItem("")
        self.mode.addItem("")
        self.mode.addItem("")
        self.mode.addItem("")
        self.mode.addItem("")
        self.mode.addItem("")
        self.mode.addItem("")
        self.mode.addItem("")
        self.mode.currentIndexChanged.connect(self.modeChanged)
        self.label = QtWidgets.QLabel(parent=self.centralwidget)
        self.label.setGeometry(QtCore.QRect(570, 10, 81, 16))
        self.label.setObjectName("label")
        self.func = QtWidgets.QComboBox(parent=self.centralwidget)
        self.func.setGeometry(QtCore.QRect(570, 100, 221, 20))
        self.func.setObjectName("func")
        self.func.addItem("")
        self.func.addItem("")
        self.func.addItem("")
        self.func.addItem("")
        self.func.addItem("")
        self.func.addItem("")
        self.func.addItem("")
        self.func.addItem("")
        self.func.currentIndexChanged.connect(self.funcChanged)
                                                self.sfactor      =
QtWidgets.QComboBox(parent=self.centralwidget)
        self.sfactor.setGeometry(QtCore.QRect(570, 410, 221, 20))
        self.sfactor.setObjectName("sfactor")
        self.sfactor.addItem("")
        self.sfactor.addItem("")
        self.sfactor.addItem("")
        self.sfactor.addItem("")
        self.sfactor.addItem("")
        self.sfactor.addItem("")
        self.sfactor.addItem("")
        self.sfactor.addItem("")
        self.sfactor.addItem("")

self.sfactor.currentIndexChanged.connect(self.sfactorChanged)
                                                self.dfactor      =
QtWidgets.QComboBox(parent=self.centralwidget)
        self.dfactor.setGeometry(QtCore.QRect(570, 480, 221, 20))
        self.dfactor.setObjectName("dfactor")
        self.dfactor.addItem("")
        self.dfactor.addItem("")
```

```
        self.dfactor.addItem("")
        self.dfactor.addItem("")
        self.dfactor.addItem("")
        self.dfactor.addItem("")
        self.dfactor.addItem("")
        self.dfactor.addItem("")

self.dfactor.currentIndexChanged.connect(self.dfactorChanged)
        self.x_cut = QtWidgets.QSlider(parent=self.centralwidget)
        self.x_cut.setGeometry(QtCore.QRect(580, 230, 201, 22))

self.x_cut.setOrientation(QtCore.Qt.Orientation.Horizontal)
        self.x_cut.setObjectName("x_cut")
        self.x_cut.valueChanged.connect(self.xChanged)
        self.width = QtWidgets.QSlider(parent=self.centralwidget)
        self.width.setGeometry(QtCore.QRect(580, 290, 201, 22))

self.width.setOrientation(QtCore.Qt.Orientation.Horizontal)
        self.width.setObjectName("width")
        self.width.valueChanged.connect(self.wChanged)
        self.label_2 = QtWidgets.QLabel(parent=self.centralwidget)
        self.label_2.setGeometry(QtCore.QRect(570, 76, 131, 20))
        self.label_2.setObjectName("label_2")
        self.label_3 = QtWidgets.QLabel(parent=self.centralwidget)
        self.label_3.setGeometry(QtCore.QRect(570, 190, 81, 21))
        self.label_3.setObjectName("label_3")
        self.label_4 = QtWidgets.QLabel(parent=self.centralwidget)
        self.label_4.setGeometry(QtCore.QRect(570, 360, 101, 16))
        self.label_4.setObjectName("label_4")
        self.label_5 = QtWidgets.QLabel(parent=self.centralwidget)
        self.label_5.setGeometry(QtCore.QRect(570, 230, 16, 16))
        self.label_5.setObjectName("label_5")
        self.label_6 = QtWidgets.QLabel(parent=self.centralwidget)
        self.label_6.setGeometry(QtCore.QRect(570, 320, 41, 20))
        self.label_6.setObjectName("label_6")
        self.label_7 = QtWidgets.QLabel(parent=self.centralwidget)
        self.label_7.setGeometry(QtCore.QRect(570, 390, 47, 13))
        self.label_7.setObjectName("label_7")
        self.label_8 = QtWidgets.QLabel(parent=self.centralwidget)
        self.label_8.setGeometry(QtCore.QRect(570, 460, 47, 13))
        self.label_8.setObjectName("label_8")
        self.y_cut = QtWidgets.QSlider(parent=self.centralwidget)
        self.y_cut.setGeometry(QtCore.QRect(580, 260, 201, 22))

self.y_cut.setOrientation(QtCore.Qt.Orientation.Horizontal)
        self.y_cut.setObjectName("y_cut")
        self.y_cut.valueChanged.connect(self.yChanged)
        self.label_9 = QtWidgets.QLabel(parent=self.centralwidget)
        self.label_9.setGeometry(QtCore.QRect(570, 290, 47, 13))
        self.label_9.setObjectName("label_9")
                                                self.label_10     =
QtWidgets.QLabel(parent=self.centralwidget)
        self.label_10.setGeometry(QtCore.QRect(570, 260, 47, 13))
        self.label_10.setObjectName("label_10")
        self.height = QtWidgets.QSlider(parent=self.centralwidget)
        self.height.setGeometry(QtCore.QRect(580, 320, 201, 22))
```

```
self.height.setOrientation(QtCore.Qt.Orientation.Horizontal)
        self.height.setObjectName("height")
        self.height.valueChanged.connect(self.hChanged)
        self.ref = QtWidgets.QSlider(parent=self.centralwidget)
        self.ref.setGeometry(QtCore.QRect(580, 140, 201, 22))
        self.ref.setOrientation(QtCore.Qt.Orientation.Horizontal)
        self.ref.setObjectName("ref")
        self.ref.valueChanged.connect(self.refChanged)
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(parent=MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 800, 26))
        self.menubar.setObjectName("menubar")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(parent=MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
                MainWindow.setWindowTitle(_translate("MainWindow",
"MainWindow"))
                 self.mode.setCurrentText(_translate("MainWindow",
"GL_POINTS"))
                self.mode.setItemText(0,  _translate("MainWindow",
"GL_POINTS"))
                self.mode.setItemText(1,  _translate("MainWindow",
"GL_LINES"))
                self.mode.setItemText(2,  _translate("MainWindow",
"GL_LINE_STRIP"))
                self.mode.setItemText(3,  _translate("MainWindow",
"GL_LINE_LOOP"))
                self.mode.setItemText(4,  _translate("MainWindow",
"GL_TRIANGLES"))
                self.mode.setItemText(5,  _translate("MainWindow",
"GL_TRIANGLE_STRIP"))
                self.mode.setItemText(6,  _translate("MainWindow",
"GL_TRIANGLE_FAN"))
                self.mode.setItemText(7,  _translate("MainWindow",
"GL_QUADS"))
                self.mode.setItemText(8,  _translate("MainWindow",
"GL_QUAD_STRIP"))
                self.mode.setItemText(9,  _translate("MainWindow",
"GL_POLYGON"))
                       self.label.setText(_translate("MainWindow",
"Модификатор"))
                 self.func.setCurrentText(_translate("MainWindow",
"GL_ALWAYS"))
                self.func.setItemText(0,  _translate("MainWindow",
"GL_ALWAYS"))
                self.func.setItemText(1,  _translate("MainWindow",
"GL_LESS"))
                self.func.setItemText(2,  _translate("MainWindow",
```

```
"GL_EQUAL"))
                self.func.setItemText(3,  _translate("MainWindow",
"GL_LEQUAL"))
                self.func.setItemText(4,  _translate("MainWindow",
"GL_GREATER"))
                self.func.setItemText(5,  _translate("MainWindow",
"GL_NOTEQUAL"))
                self.func.setItemText(6,  _translate("MainWindow",
"GL_GEQUAL"))
                self.func.setItemText(7,  _translate("MainWindow",
"GL_NEVER"))
             self.sfactor.setCurrentText(_translate("MainWindow",
"GL_ZERO"))
            self.sfactor.setItemText(0,  _translate("MainWindow",
"GL_ZERO"))
            self.sfactor.setItemText(1,  _translate("MainWindow",
"GL_ONE"))
            self.sfactor.setItemText(2,  _translate("MainWindow",
"GL_DST_COLOR"))
            self.sfactor.setItemText(3,  _translate("MainWindow",
"GL_ONE_MINUS_DST_COLOR"))
            self.sfactor.setItemText(4,  _translate("MainWindow",
"GL_SRC_ALPHA"))
            self.sfactor.setItemText(5,  _translate("MainWindow",
"GL_ONE_MINUS_SRC_ALPHA"))
            self.sfactor.setItemText(6,  _translate("MainWindow",
"GL_DST_ALPHA"))
            self.sfactor.setItemText(7,  _translate("MainWindow",
"GL_ONE_MINUS_DST_ALPHA"))
            self.sfactor.setItemText(8,  _translate("MainWindow",
"GL_SRC_ALPHA_SATURATE"))
             self.dfactor.setCurrentText(_translate("MainWindow",
"GL_ZERO"))
            self.dfactor.setItemText(0,  _translate("MainWindow",
"GL_ZERO"))
            self.dfactor.setItemText(1,  _translate("MainWindow",
"GL_ONE"))
            self.dfactor.setItemText(2,  _translate("MainWindow",
"GL_SRC_COLOR"))
            self.dfactor.setItemText(3,  _translate("MainWindow",
"GL_ONE_MINUS_SRC_COLOR"))
            self.dfactor.setItemText(4,  _translate("MainWindow",
"GL_SRC_ALPHA"))
            self.dfactor.setItemText(5,  _translate("MainWindow",
"GL_ONE_MINUS_SRC_ALPHA"))
            self.dfactor.setItemText(6,  _translate("MainWindow",
"GL_DST_ALPHA"))
            self.dfactor.setItemText(7,  _translate("MainWindow",
"GL_ONE_MINUS_DST_ALPHA"))
             self.label_2.setText(_translate("MainWindow",  "Тест
прозрачности"))
             self.label_3.setText(_translate("MainWindow",  "Тест
отсечения"))
             self.label_4.setText(_translate("MainWindow",  "Тест
смешивания"))
        self.label_5.setText(_translate("MainWindow", "x"))
```

```
        self.label_6.setText(_translate("MainWindow", "h"))
        self.label_7.setText(_translate("MainWindow", "sfactor"))
        self.label_8.setText(_translate("MainWindow", "dfactor"))
        self.label_9.setText(_translate("MainWindow", "w"))
        self.label_10.setText(_translate("MainWindow", "y"))

    def modeChanged(self):
        print(self.mode.currentText())
        self.glWidget.mode = self.mode.currentText()
        self.glWidget.update()

    def xChanged(self):
        print(self.x_cut.value())
        self.glWidget.x_cut = self.x_cut.value()
        self.glWidget.update()

    def yChanged(self):
        print(self.y_cut.value())
        self.glWidget.y_cut = self.y_cut.value()
        self.glWidget.update()

    def wChanged(self):
        print(self.width.value())
        self.glWidget.width = self.width.value()
        self.glWidget.update()

    def hChanged(self):
        print(self.height.value())
        self.glWidget.height = self.height.value()
        self.glWidget.update()

    def refChanged(self):
        print(1 - self.ref.value() / 100)
        self.glWidget.ref = 1 - self.ref.value() / 100
        self.glWidget.update()

    def funcChanged(self):
        print(self.func.currentText())
        self.glWidget.func = self.func.currentText()
        self.glWidget.update()

    def sfactorChanged(self):
        print(self.sfactor.currentText())
        self.glWidget.sfactor = self.sfactor.currentText()
        self.glWidget.update()

    def dfactorChanged(self):
        print(self.dfactor.currentText())
        self.glWidget.dfactor = self.dfactor.currentText()
        self.glWidget.update()


if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
```

```
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec())
```

commands.py

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

colors = [[1.0, 0.0, 0.0, 1.0],
          [0.0, 1.0, 0.0, 0.8],
          [0.0, 0.0, 1.0, 0.9],
          [1.0, 1.0, 0.0, 0.5],
          [0.0, 1.0, 1.0, 1.0],
          [1.0, 0.0, 1.0, 0.1]]

vertex = [[0.3, 0.5],
          [0.6, 0],
          [0.4, -0.3],
          [0, -0.5],
          [-0.4, -0.3],
          [-0.6, 0],
          [-0.4, 0.3],
          [0, 0.5]]

def points():
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glPointSize(3)
    glBegin(GL_POINTS)
    for i in range(len(vertex)):
        glColor4f(*colors[i%len(colors)])
        glVertex2f(vertex[i][0], vertex[i][1])
    glEnd()
```

```
def lines():
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glLineWidth(3)
    glBegin(GL_LINES)
    for i in range(len(vertex)):
        glColor4f(*colors[i%len(colors)])
        glVertex2f(vertex[i][0], vertex[i][1])
    glEnd()

def lineStrip():
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glLineWidth(3)
    glBegin(GL_LINE_STRIP)
    for i in range(len(vertex)):
        glColor4f(*colors[i%len(colors)])
        glVertex2f(vertex[i][0], vertex[i][1])
    glEnd()

def lineLoop():
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glLineWidth(2)
    glBegin(GL_LINE_LOOP)
    for i in range(len(vertex)):
        glColor4f(*colors[i%len(colors)])
        glVertex2f(vertex[i][0], vertex[i][1])
    glEnd()

def triangles():
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glLineWidth(4)
    glBegin(GL_TRIANGLES)
    for i in range(len(vertex)):
        glColor4f(*colors[i%len(colors)])
```

```
            glVertex2f(vertex[i][0], vertex[i][1])
        glEnd()

    def triangleStrip():
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glLineWidth(4)
        glPolygonMode(GL_FRONT_AND_BACK, GL_LINE)
        glBegin(GL_TRIANGLE_STRIP)
        for i in range(len(vertex)):
            glColor4f(*colors[i%len(colors)])
            glVertex2f(vertex[i][0], vertex[i][1])
        glEnd()

    def triangleFan():
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glLineWidth(4)
        glPolygonMode(GL_FRONT_AND_BACK, GL_LINE)
        glBegin(GL_TRIANGLE_FAN)
        for i in range(len(vertex)):
            glColor4f(*colors[i%len(colors)])
            glVertex2f(vertex[i][0], vertex[i][1])
        glEnd()

    def quads():
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glLineWidth(4)
        glBegin(GL_QUADS)
        for i in range(len(vertex)):
            glColor4f(*colors[i%len(colors)])
            glVertex2f(vertex[i][0], vertex[i][1])
        glEnd()

    def quadStrip():
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
```

```
    glLineWidth(4)
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE)
    glBegin(GL_QUAD_STRIP)
    for i in range(len(vertex)):
        glColor4f(*colors[i%len(colors)])
        glVertex2f(vertex[i][0], vertex[i][1])
    glEnd()

def polygon():
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glLineWidth(4)
    glBegin(GL_POLYGON)
    for i in range(len(vertex)):
        glColor4f(*colors[i%len(colors)])
        glVertex2f(vertex[i][0], vertex[i][1])
    glEnd()

commands = {
    "GL_POINTS" : points,
    "GL_LINES" : lines,
    "GL_LINE_STRIP" : lineStrip,
    "GL_LINE_LOOP" : lineLoop,
    "GL_TRIANGLES": triangles,
    "GL_TRIANGLE_STRIP": triangleStrip,
    "GL_TRIANGLE_FAN" : triangleFan,
    "GL_QUADS" : quads,
    "GL_QUAD_STRIP" : quadStrip,
    "GL_POLYGON" : polygon
}

func = {
    'GL_ALWAYS': GL_ALWAYS,
    'GL_NEVER': GL_NEVER,
    'GL_LESS': GL_LESS,
```

```
    'GL_EQUAL': GL_EQUAL,
    'GL_LEQUAL': GL_LEQUAL,
    'GL_GREATER': GL_GREATER,
    'GL_NOTEQUAL': GL_NOTEQUAL,
    'GL_GEQUAL': GL_GEQUAL,
    'GL_NEVER': GL_NEVER
}

sfactor = {
    'GL_ONE': GL_ONE,
    'GL_ZERO': GL_ZERO,
    'GL_DST_COLOR': GL_DST_COLOR,
    'GL_ONE_MINUS_DST_COLOR': GL_ONE_MINUS_DST_COLOR,
    'GL_SRC_ALPHA': GL_SRC_ALPHA,
    'GL_ONE_MINUS_SRC_ALPHA': GL_ONE_MINUS_SRC_ALPHA,
    'GL_DST_ALPHA': GL_DST_ALPHA,
    'GL_ONE_MINUS_DST_ALPHA': GL_ONE_MINUS_DST_ALPHA,
    'GL_SRC_ALPHA_SATURATE': GL_SRC_ALPHA_SATURATE
}

dfactor = {
    'GL_ZERO': GL_ZERO,
    'GL_ONE': GL_ONE,
    'GL_SRC_COLOR': GL_SRC_COLOR,
    'GL_ONE_MINUS_SRC_COLOR': GL_ONE_MINUS_SRC_COLOR,
    'GL_SRC_ALPHA': GL_SRC_ALPHA,
    'GL_ONE_MINUS_SRC_ALPHA': GL_ONE_MINUS_SRC_ALPHA,
    'GL_DST_ALPHA': GL_DST_ALPHA,
    'GL_ONE_MINUS_DST_ALPHA': GL_ONE_MINUS_DST_ALPHA
}
```