Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

Лабораторная работа № 4

по курсу «Алгоритмы компьютерной графики»

Студент группы ИУ9-41Б Горбунов А. Д.

Преподаватель Цалкович П. А.

Москва 2024

1 Задача

а. Реализовать алгоритм растровой развертки многоугольника построчного сканирования многоугольника со списком активных ребер

б. Реализовать алгоритм постфильтрация с равномерным усреднением области 3x3

в. Реализовать необходимые вспомогательные алгоритмы (растеризации отрезка) с модификациями, обеспечивающими корректную работу основного алгоритма.

г. Ввод исходных данных каждого из алгоритмов производится интерактивно с помощью клавиатуры и/или мыши. Предусмотреть также возможность очистки области вывода (отмены ввода).

д. Растеризацию производить в специально выделенном для этого буфере в памяти с последующим копированием результата в буфер кадра OpenGL. Предусмотреть возможность изменение размеров окна.

2 Теория

Построчное сканирование со списком активных ребер

• создается y-список ребер: ребра многоугольника упорядочиваются по возрастанию по координате y начальной вершины (начальной считается вершина с наименьшей координатой y);

• на каждой итерации рассматривается только текущая строка сканирования, для которой формируется список «активных» ребер (САР), в котором хранится информация только о ребрах многоугольника, пересекаемых текущей строкой;

• при переходе к очередной строке сканирования:

– из САР удаляются ребра, чья нижняя вершина оказалась выше текущей строки сканирования;

– из y-списка добавляются ребра, начинающиеся на данной строке сканирования;

– вычисляются новые точки пересечения строки сканирования с ребрами (используя свойство пространственной когерентности, выводятся рекуррентные соотношения)

• для каждой строки сканирования:

– список точек пересечения с ребрами упорядочивается по возрастанию x;

– заполняются все промежутки вида $[x_{2i-1}; x_{2i})$.

Алгоритм заполнения многоугольника по ребрам

• для каждой строки сканирования инвертируются все пиксели справа от точки пересечения данной строки сканирования с ребром многоугольника;

• порядок обработки ребер многоугольника не важен;

• каждый пиксель может обрабатываться многократно.

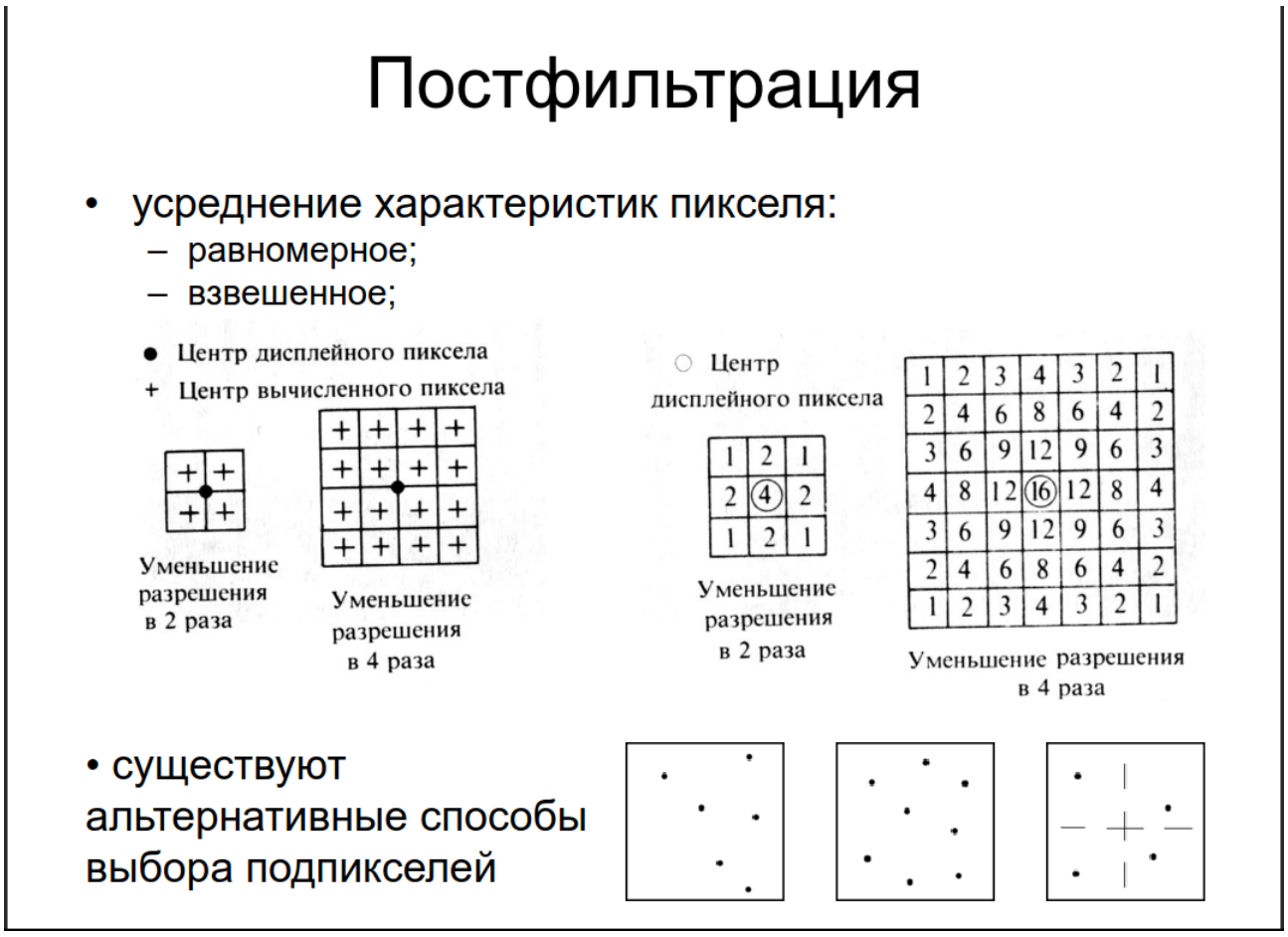

Рис. 1 — Постфильтрация

3 Код решения

Файл main.py

```
import glfw
from OpenGL.GL import *
from math import ceil
sizeX = 1000
sizeY = 1000
data = [[255] * sizeX for i in range(sizeY)]
points = []
edges = []
cnt = 0

def key_callback(window, key, scancode, action, mods):
    global cnt, data, edges, points
```

```
    if key == glfw.KEY_SPACE and action == glfw.PRESS:
        drawLine(points[-1][0], points[-1][1], points[0][0], points[0][1])
        add_point(points[0][0], points[0][1])
    if key == glfw.KEY_1 and action == glfw.PRESS:
        miny = min([y for _, y in points])
        maxy = max([y for _, y in points])
        fill(miny + 1, maxy - 1)
    if key == glfw.KEY_2 and action == glfw.PRESS:
        filtration()
    if key == glfw.KEY_0 and action == glfw.PRESS:
        data = [[255] * sizeX for i in range(sizeY)]
        points = []
        edges = []
        cnt = 0
        print("Clear all points")
    if key == glfw.KEY_ESCAPE and action == glfw.PRESS:
        glfw.set_window_should_close(window, True)


def mouse_button_callback(window, button, action, mods):
    global cnt, data, edges, points
    if button == glfw.MOUSE_BUTTON_LEFT and action == glfw.PRESS:
        t = list(glfw.get_cursor_pos(window))
        t[0] = int(t[0])
        t[1] = int(-t[1])
        print(f"Ox = {t[0]}, Oy = {t[1]}")
        add_point(t[0], t[1])
        if len(edges) > 0:
            for edge in edges:
                drawLine(points[edge[0]][0], points[edge[0]][1], points[edge[1]][0],  points[ed


def add_point(x,y):
```

```
    global cnt, points
    points.append((x, y))
    cnt += 1
    add_edge()

def add_edge():
    global cnt, edges
    if cnt > 1:
        if cnt == 3:
            edges.append((0, 1))
        edges.append((cnt - 2, cnt - 1))

def drawLine(x0, y0, x1, y1):
    if x0 == x1:
        m = 2 ** 32
    else:
        m = ((y1 - y0) / (x1 - x0))
    e = -.5
    x = x0
    y = y0
    isSharp = True
    if x <= x1 and y <= y1:
        if m > 1:
            isSharp = False
            m **= -1
        while x <= x1 and y <= y1:
            data[y][x] = 0
            if isSharp:
                x += 1
            else:
                y += 1
            e += m
            if e >= 0:
```

```
            if isSharp:
                y += 1
            else:
                x += 1
            e -= 1
    elif x >= x1 and y <= y1:
        m = -m
        if m > 1:
            isSharp = False
            m **= -1
        while x >= x1 and y <= y1:
            data[y][x] = 0
            if isSharp:
                x -= 1
            else:
                y += 1
            e += m
            if e >= 0:
                if isSharp:
                    y += 1
                else:
                    x -= 1
                e -= 1
    elif x >= x1 and y >= y1:
        if m > 1:
            isSharp = False
            m **= -1
        while x >= x1 and y >= y1:
            data[y][x] = 0
            if isSharp:
                x -= 1
            else:
                y -= 1
```

```
            e += m
            if e >= 0:
                if isSharp:
                    y -= 1
                else:
                    x -= 1
                e -= 1
        elif x <= x1 and y >= y1:
            m = -m
            if m > 1:
                m **= -1
                isSharp = False
            while x <= x1 and y >= y1:
                data[y][x] = 0
                if isSharp:
                    x += 1
                else:
                    y -= 1
                e += m
                if e >= 0:
                    if isSharp:
                        y -= 1
                    else:
                        x += 1
                    e -= 1

def filtration():
    global data
    mask = [[1, 2, 1],
            [2, 4, 2],
            [1, 2, 1]]
    for i in range(1, sizeY - 1):
        for j in range(1, sizeX - 1):
```

```
            if zeroChek(data, i, j):
                data[i][j] = int(
                    (mask[0][0] * data[i + 1][j - 1] + mask[0][1] * data[i + 1][j] + mask[0][
                     mask[1][0] * data[i][j - 1]   +   mask[1][1] * data[i][j]   +   mask[1][2] *
                     mask[2][0] * data[i - 1][j - 1] + mask[2][1] * data[i - 1][j] + mask[2][2]
                    / 16)
            else:
                data[i][j] = 0
    print("filtration = True")

def zeroChek(data, i, j):
    return 0 < (data[i + 1][j - 1] + data[i + 1][j] + data[i + 1][j + 1] + data[i][j - 1] +
                + data[i - 1][j - 1] + data[i - 1][j] + data[i - 1][j + 1])

def fill(start, end):
    global data
    for y in range(start, end):
        active_edge = []

        for edge in edges:
            x1, y1 = points[edge[0]]
            x2, y2 = points[edge[1]]
            if (y1 >= y and y2 <= y) or (y1 <= y and y2 >= y):
                dx = 1
                if (y2 - y1 != 0):
                    dx = (x2 - x1) / (y2 - y1)
                x = int(ceil(((y - y1) * dx) + x1))
                active_edge.append(x)

        active_edge.sort()
        ind, e1 = 0, 0
        for e2 in active_edge:
```

```
            if ind % 2 == 0:
                e1 = e2
            else:
                if e1 == e2:
                    ind += 1
                else:
                    for x in range(e1, e2):
                        if(data[y][x] != 0):
                            data[y][x] = 150
            ind += 1
        print("fill = True")

def display(window):
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glLoadIdentity()
    glClearColor(0, 0, 0, 0)
    glRasterPos(-1, -1)
    #glPixelZoom(2, 2)
    glDrawPixels(sizeX, sizeY, GL_LUMINANCE, GL_UNSIGNED_BYTE, data)
    glfw.swap_buffers(window)
    glfw.poll_events()

def main():

    if not glfw.init():
        return
    window = glfw.create_window(sizeX, sizeY, "lab_4", None, None)
    if not window:
        glfw.terminate()
        return
    glfw.make_context_current(window)
    glfw.set_key_callback(window, key_callback)
    glfw.set_mouse_button_callback(window, mouse_button_callback)
```

```
    while not glfw.window_should_close(window):
        display(window)
    glfw.destroy_window(window)
    glfw.terminate()

if __name__ == '__main__':
    main()
```

4 Заключение

В данной работе я изучил возможности языка python в работе с библиотекой OpenGL, а именно научился ввод с мышки и применять алгоритмы заполнения и пост фильтрации с помощь функций: glfw.get cursor pos(window), glLoadIdentity(), glRasterPos(-1, -1), glDrawPixels(sizeX, sizeY, GL LUMINANCE, GL UNSIGNED BYTE, data), glfw.swap buffers(window), glfw.poll events(), glfw.set mouse button callback(window, mouse button callback).

5 Результат запуска

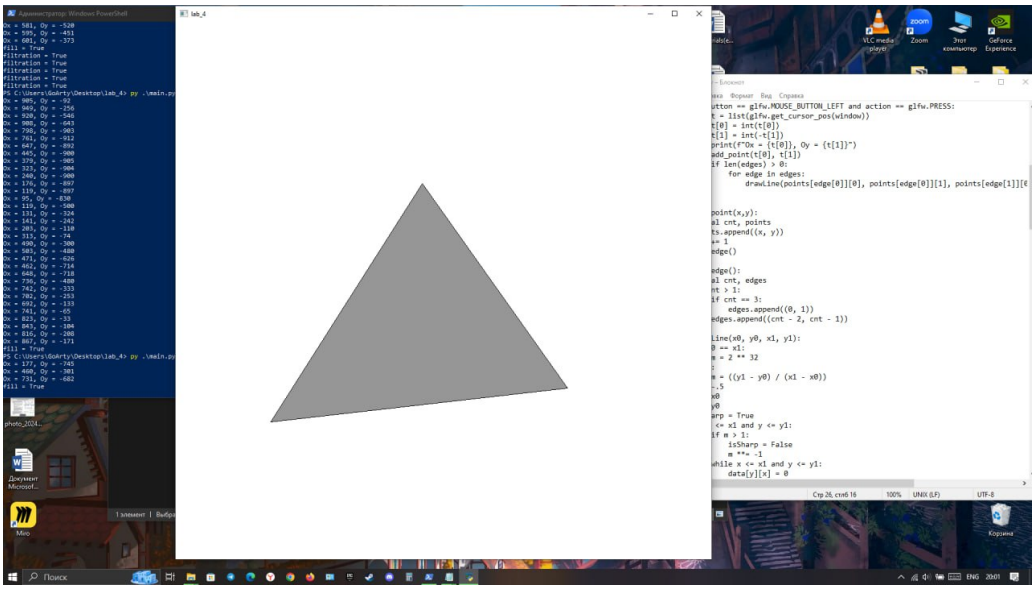

Рис. 2 — Заполненный треугольник

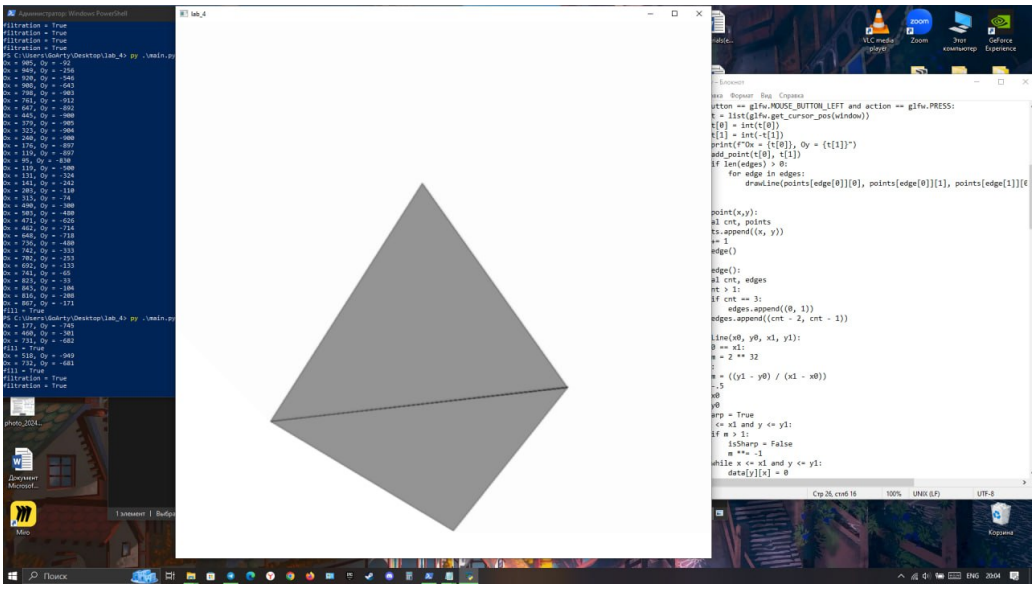

Рис. 3 — Заполненный многоугольник

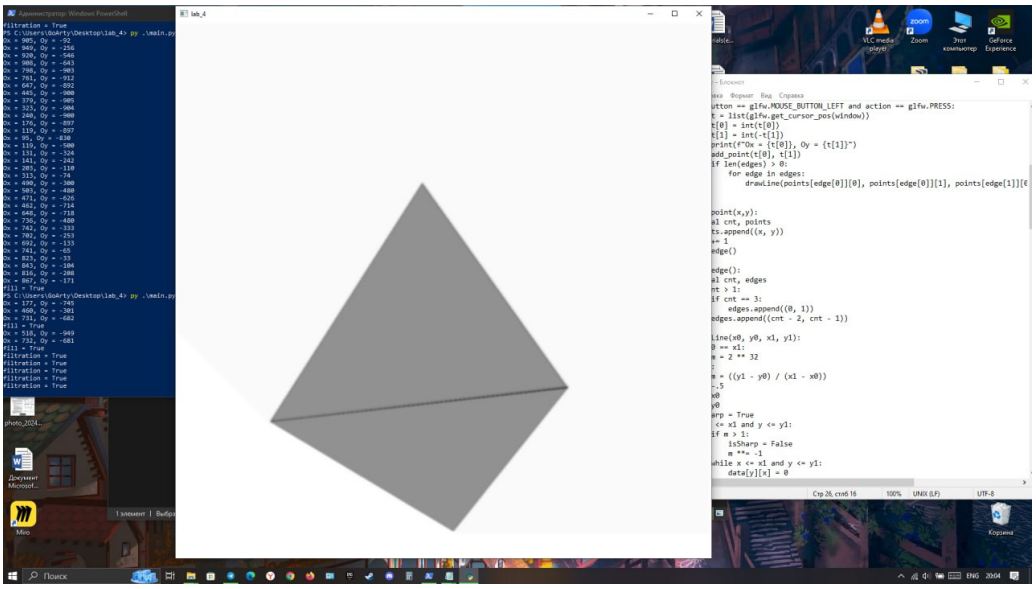

Рис. 4 — Многоугольник с постфильтрацией