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 «Компьютерная графика»

«Алгоритмы растровой графики»

Студент группы ИУ9-42Б Волохов А. В.

Преподаватель Цалкович П. А.

Москва 2024

1 Задача

Необходимо:

а) Реализовать алгоритм растровой развертки многоугольника: заполнения многоугольника со списком ребер и флагом;

б) Реализовать алгоритм фильтрации: целочисленный алгоритм Брезенхема с устранением ступенчатости;

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 Основная теория

В данном коде реализован алгоритм растровой развертки многоугольника. Пользователь взаимодействует с приложением, добавляя вершины многоугольника с помощью мыши. После добавления каждой вершины происходит перерисовка многоугольника с учетом новой вершины. Для растровой развертки используется алгоритм Брезенхема для рисования линий с учетом интенсивности цвета пикселя (для сглаживания). Каждая вершина многоугольника сохраняется в списке точек, а затем используется для построения ребер многоугольника.

После добавления всех вершин многоугольника пользователь может нажать клавишу "Enter чтобы заполнить многоугольник. Для этого используется алгоритм заливки по координатам точки. Происходит определение внутренних точек многоугольника, которые необходимо заполнить.

3 Практическая реализация

Код представлен в Листинге 1.

Листинг 1 - lab4.py

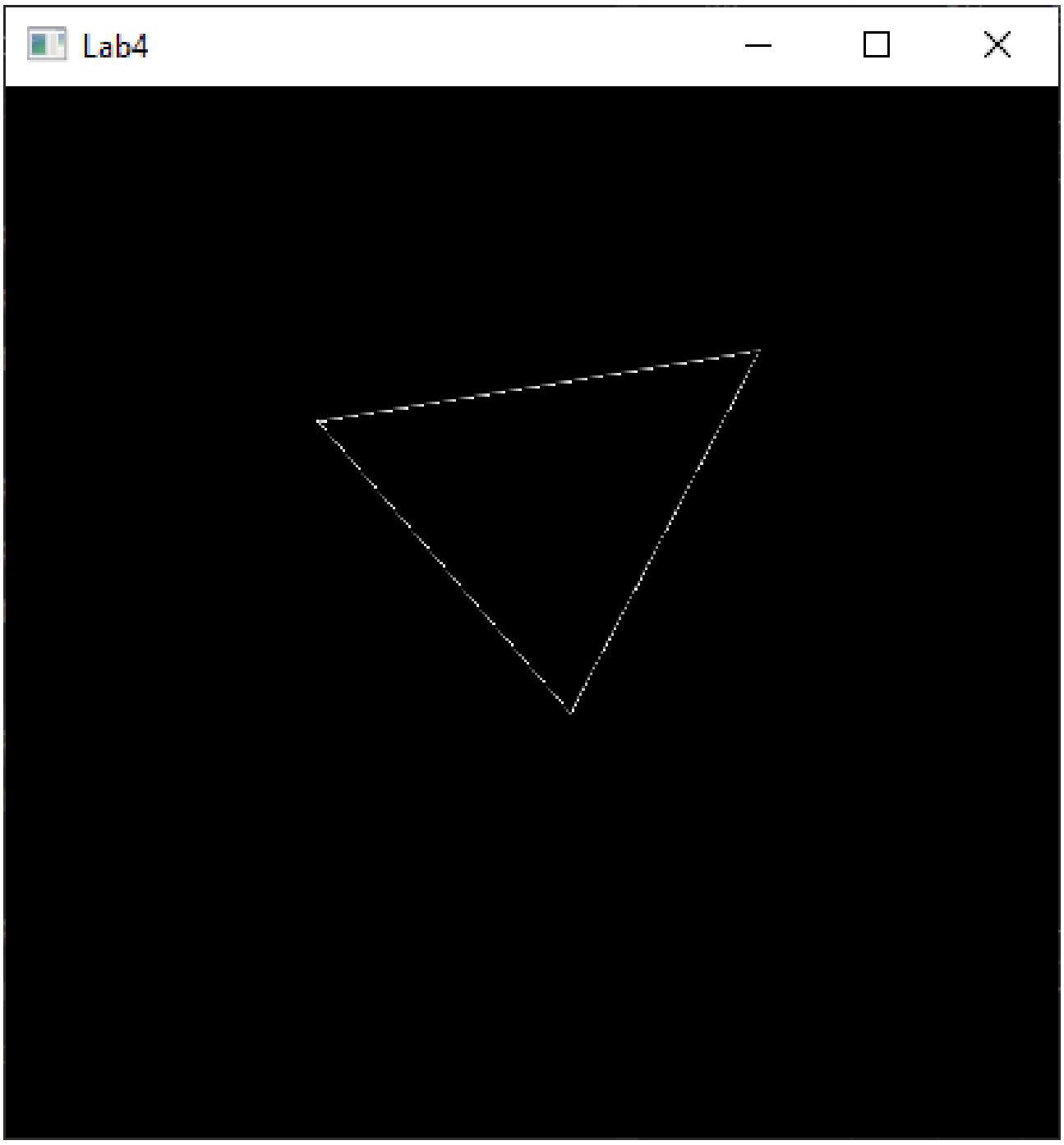
```
from OpenGL.GL import *
import glfw
import collections

size = 400
pixels = [0] * (size * size * 3)
points = []
edges = []


def sign(a, b):
    if a > b:
        return -1
    return 1


def bresenham(x1, y1, x2, y2):
    dx = abs(x2 - x1)
    dy = abs(y2 - y1)
    x, y = x1, y1
    sx = sign(x1, x2)
    sy = sign(y1, y2)
    if dx > dy:
        err = dx / 2
        while x != x2:
            if 0 <= x < size and 0 <= y < size:
                position = (x + y * size) * 3
                #
                pixels[position] = int(255 * (1 - err / dx))
                pixels[position + 1] = int(255 * (1 - err / dx))
                pixels[position + 2] = int(255 * (1 - err / dx))
            err -= dy
            if err < 0:
                y += sy
                err += dx
            x += sx
    else:
        err = dy / 2
        while y != y2:
            if 0 <= x < size and 0 <= y < size:
```

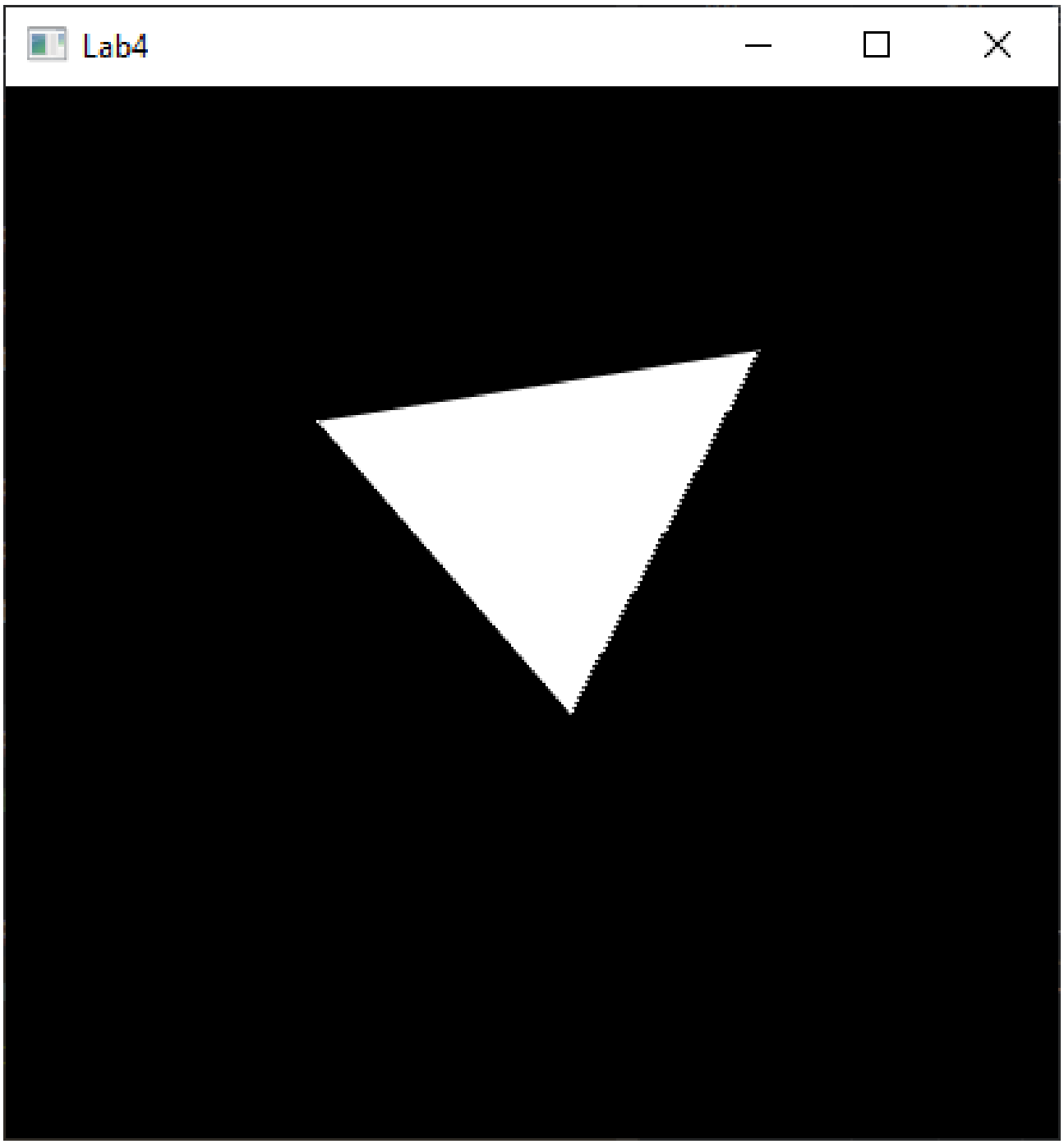
```
                position = (x + y * size) * 3
                #
                pixels[position] = int(255 * (1 - err / dy))
                pixels[position + 1] = int(255 * (1 - err / dy))
                pixels[position + 2] = int(255 * (1 - err / dy))
            err -= dx
            if err < 0:
                x += sx
                err += dy
            y += sy
    if 0 <= x < size and 0 <= y < size:
        position = (x + y * size) * 3
        pixels[position] = 255
        pixels[position + 1] = 255
        pixels[position + 2] = 255


def mouse_button_callback(window, button, action, mods):
    global points, edges, size
    if action == glfw.PRESS and button == glfw.MOUSE_BUTTON_LEFT:
        x, y = glfw.get_cursor_pos(window)
        y = size - y
        points.append((int(x), int(y)))

        redraw_polygon()


def redraw_polygon():
    global pixels, points, edges, size
    pixels = [0] * (size * size * 3)
    edges = create_edge_list()

    for i in range(1, len(points)):
        bresenham(points[i - 1][0], points[i - 1][1], points[i][0], points[i
            ][1])

    if len(points) > 2:
        bresenham(points[-1][0], points[-1][1], points[0][0], points[0][1])


def create_edge_list():
    edge_list = []
    for i in range(len(points)):
        p1 = points[i]
        p2 = points[(i + 1) % len(points)]
        edge_list.append((p1, p2))
```

```
    return edge_list


def fill(x, y):
    q = collections.deque([(x, y)])
    while q:
        x, y = q.popleft()
        current_pos = (x + y * size) * 3
        if x < 0 or x >= size or y < 0 or y >= size:
            continue
        if pixels[current_pos] == pixels[current_pos + 1] == pixels[
            current_pos + 2] == 0:
            if 0 <= x < size and 0 <= y < size:
                position = (x + y * size) * 3
                pixels[position] = 255
                pixels[position + 1] = 255
                pixels[position + 2] = 255
            q.append((x + 1, y))
            q.append((x - 1, y))
            q.append((x, y + 1))
            q.append((x, y - 1))


def key_callback(window, key, scancode, action, mods):
    global pixels, points, edges
    if action == glfw.PRESS:
        if key == glfw.KEY_ENTER:
            if len(points) > 2:
                bresenham(points[-1][0], points[-1][1], points[0][0], points
                    [0][1])
                min_x = min(points, key=lambda x: x[0])[0]
                max_x = max(points, key=lambda x: x[0])[0]
                min_y = min(points, key=lambda x: x[1])[1]
                max_y = max(points, key=lambda x: x[1])[1]
                x = (min_x + max_x) // 2
                y = (min_y + max_y) // 2
                if x and y:
                    fill(x, y)
        elif key == glfw.KEY_BACKSPACE:
            pixels = [0 for _ in range(size * size * 3)]
            points.clear()
            edges.clear()


def display(window):
    glDrawPixels(size, size, GL_RGB, GL_UNSIGNED_BYTE, pixels)
```

```
    glBegin(GL_LINES)
    for edge in edges:
        for p in edge:
            glVertex2f(*p)
    glEnd()

    glfw.swap_buffers(window)
    glfw.poll_events()


def main():
    if not glfw.init():
        return
    window = glfw.create_window(size, size, "Lab4", 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.terminate()


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

В результате работы программы получился следующий вывод:


Lab4


Lab4

4 Заключение

В результате выполнения лабораторной работы были реализованы алгоритмы растровой развертки многоугольника и фильтрации в контексте двумерной графики с использованием библиотеки OpenGL. Пользователь может интерактивно добавлять вершины многоугольника с помощью мыши, а затем заполнять его, что позволяет визуализировать исследуемую область. Я ознакомился с такими алгоритмами растровой графики, как алгоритм Брезенхема для построения прямой и устранения ступенчатости. Эти алгоритмы могут быть полезны при разработке графических приложений и программ для работы с графикой.