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 Утебаева М. Б.

Преподаватель Цалкович П. А.

Москва 2024

1 Задача

- Реализовать алгоритм растровой развертки многоугольника с помощью построчного сканирования многоугольника со списком активных ребер.
- Реализовать алгоритм постфильтрации с взвешенным усреднением области 3x3.
- Реализовать необходимые вспомогательные алгоритмы растеризации отрезка с модификациями, обеспечивающими корректную работу основного алгоритма.
- Реализовать ввод исходных данных каждого из алгоритмов производится интерактивно с помощью клавиатуры и/или мыши. Предусмотреть также возможность очистки области вывода, то есть отмены ввода.
- Растеризацию производить в специально выделенном для этого буфере в памяти с последующим копированием результата в буфер кадра OpenGL.

2 Основная теория

Растровое изображение состоит из отдельных точек, которые могут занимать строго определенные места, и при этом каждая точка может иметь собственные значения атрибутов.

Алгоритм Брезенхэма – это алгоритм, который определяет, какие точки двумерного растра нужно закрасить, чтобы получить близкое приближение прямой линии между двумя заданными точками.

Фильтрация – это свёртка изображения с ядром свёртки, которая позволяет усреднить значение в некоторой области, опираясь на значения смежных областей.

```
glRasterPos2d(-1, -1) #задаёт текущую позицию в растреё
glDrawPixels(sz1, sz2, GL_GREEN, GL_UNSIGNED_BYTE, pixels) #определяет буфер кадра для записи пикселей
glfw.get_cursor_pos(window) #возвращает координаты позиции курсора
```

3 Практическая реализация

```
import glfw
from OpenGL.GL import *
from math import ceil

window_size = (800, 800)
background_color = 255
color = 0
pixels = [0] * window_size[0] * window_size[1]
points = []
edges = []
cnt = 0

def add_point(x, y):
    global cnt
    #если выходим за границы или пиксель уже покрашен
    if x * window_size[0] + y >= len(pixels) or pixels[x * window_size[0] + y] != 0:
        return
    points.append((x, y))
    cnt += 1
```

```
    if cnt > 1:
        if cnt == 3:
            edges.append((0, 1)) #исключительный случай
        edges.append((cnt - 2, cnt - 1))
        if cnt > 2:
            bresenham(points[edges[0][0]], points[edges[0][1]], 0) #удаляем последнее ребро
            edges[0] = (0, cnt - 1) #соединяем первую и последнюю точки

def bresenham(p1, p2, color): #алгоритм брезенхема без лишних слов
    x1, y1 = p1
    x2, y2 = p2
    x, y = x1, y1
    dx, dy = x2 - x1, y2 - y1
    dist_x = abs(dx)
    dist_y = abs(dy)
    dist = dist_x
    if dist_y > dist:
        dist = dist_y
    step_x, step_y = 1, 1
    if dx < 0:
        step_x = -1
    if dy < 0:
        step_y = -1
    err_y, err_x = 0, 0
    dst = dist + 1
    while (dst):
        dst -= 1
        pixels[x * window_size[0] + y] = color
        err_x += dist_x
        err_y += dist_y
        if err_x >= dist:
            err_x -= dist
            x += step_x
        if err_y >= dist:
            err_y -= dist
            y += step_y

def draw(): #рисуем все рёбра
    for edge in edges:
        bresenham(points[edge[0]], points[edge[1]], 255)

def fill(start, end): #закрашивание области
    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 = (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):
                        pixels[x * window_size[0] + y] = color
            ind += 1

def weight_sum(i, j): #усреднение по соседям
    sum = 0
    for k in range(i - 1, i - 1 + 3):
        for l in range(j - 1, j - 1 + 3):
            if k >= 0 and k < window_size[1] and l >= 0 and l < window_size[0]:
                sum += pixels[k * window_size[0] + l]
    return sum

def postfilter(): #постфильтрация (растеризация)
    global pixels
    pixels2 = [0] * window_size[0] * window_size[1]
    for i in range(0, window_size[1]):
        for j in range(0, window_size[0]):
            color = weight_sum(i, j)
            pixels2[i * window_size[0]+ j] = color
    pixels = pixels2
    print("postfilter completed")

def display(window):
    global pixel_sz, view_sz
    glClearColor(0, 0, 0, 1)
    glClear(GL_COLOR_BUFFER_BIT)
    glRasterPos2d(-1, -1) #откуда начать отрисовку
    glDrawPixels(window_size[0], window_size[1], GL_GREEN, GL_UNSIGNED_BYTE, pixels)
    glfw.swap_buffers(window)
    glfw.poll_events()

def key_callback(window, key, scancode, action, mods):
    global color, cnt, background_color, points, pixels, edges
    if action == glfw.PRESS:
```

```
        if key == glfw.KEY_F:
            color = background_color
            miny = min([y for x, y in points])
            maxy = max([y for x, y in points])
            fill(miny + 1, maxy - 1)
        elif key == glfw.KEY_R:
            postfilter()
        elif key == glfw.KEY_C:
            pixels = [0] * window_size[0] * window_size[1]
            points = []
            edges = []
            cnt = 0

def mouse_button(window, button, action, mods):
    y, x = glfw.get_cursor_pos(window)
    x, y = round(x), round(y)
    if action == glfw.PRESS:
        if button == glfw.MOUSE_BUTTON_LEFT:
            add_point(x, y)
            if len(edges) > 0:
                draw()

def main():
    if not glfw.init():
        return
    window = glfw.create_window(window_size[0], window_size[1], "laba4", 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)
    while not glfw.window_should_close(window):
        display(window)
    glfw.destroy_window(window)
    glfw.terminate()

main()
```

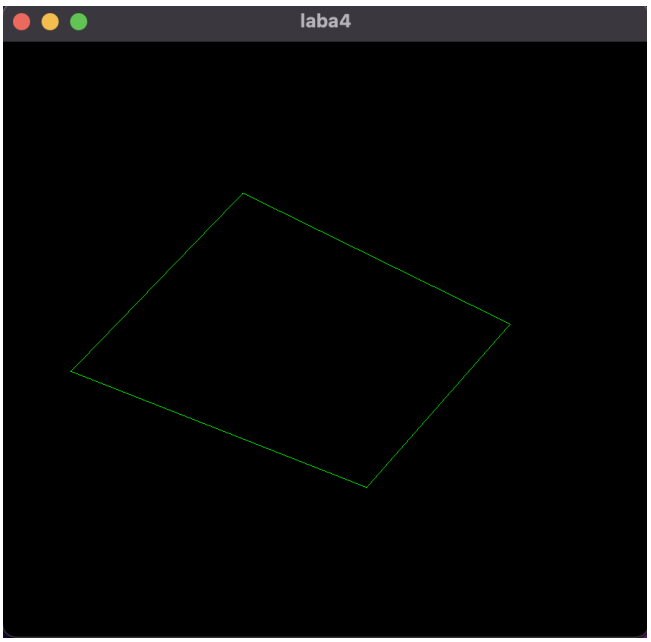

Рис. 1 — Добавлены первые точки

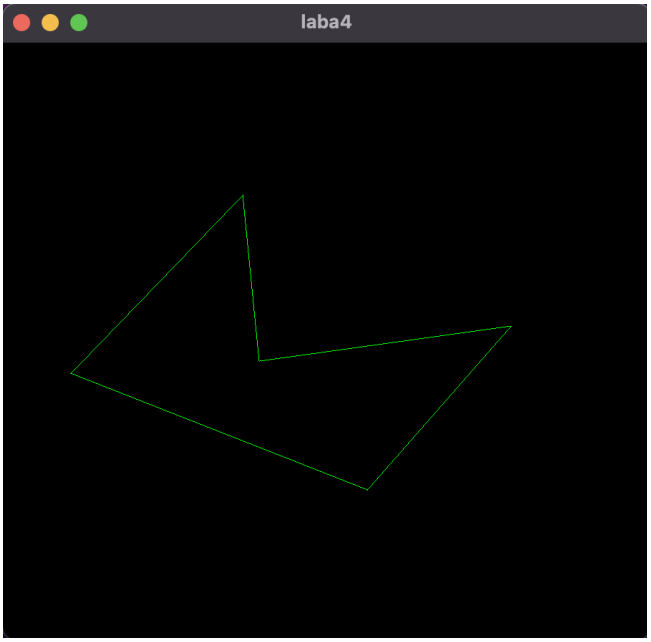

Рис. 2 — Добавлена ещё одна точка

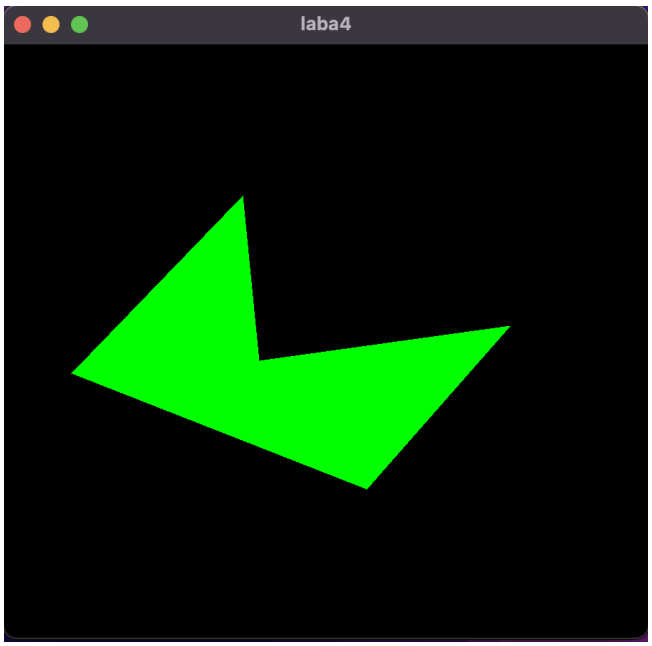

Рис. 3 — Фигура после заливки

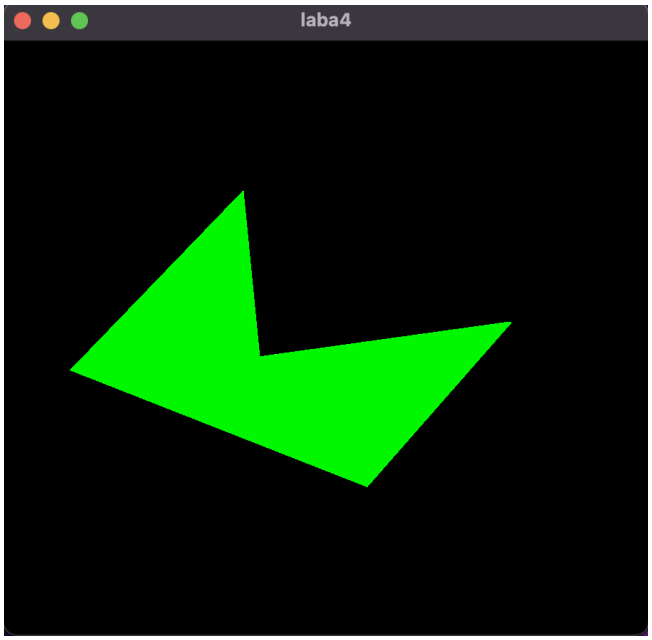

Рис. 4 — Фигура после постфильтрации

4 Заключение

Я узнала как устроено растровое изображение, познакомилась с алгоритмом Брезенхэма, который определяет какие точки двумерного растра нужно закрасить, основываясь на значении ошибки, и реализовала его целочисленную версию, познакомилась с алгоритмами растровой развёртки, реализовала один из них – с построчным сканированием активных рёбер, также я реализовала алгоритм постфильтрации со взвешенным усреднением области 3x3. Помимо этого в ходе лабораторной работы я познакомилась с библиотечными функциями OpenGL: glRasterPos2d(x, y), glDrawPixels(...), glReadPixels(...) и другими.