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 № 5

по курсу «Алгоритмы компьютерной графики»

Студент группы ИУ9-41Б Утебаева М. Б.

Преподаватель Цалкович П. А.

Москва 2024

1 Задача

Реализовать алгоритм Сазерленда-Ходжмена (алгоритм внутреннего отсечения) в двумерном пространстве. Ввод исходных данных производится интерактивно с помощью клавиатуры и/или мыши.

2 Основная теория

Алгоритм Сазерленда-Ходжмена реализует последовательное отсечение произвольного многоугольника выпуклым отсекателем.

Алгоритм реализуется при помощи двух циклов: первый идёт по рёбрам отсекателя, а второй, внутренний, перебирает все рёбра отсекаемого многоугольника и формирует новый список вершин.

Для каждого ребра SP осуществляется ряд проверок

- полная видимость: добавляем точку P
- полная невидимость: ничего не добавляем
- выход ребра из области видимости: добавляем точку пересечения с ребром отсекателя
- вход ребра в область видимости: добавляем точку пересечения с ребром отсекателя и точку, лежащую в отсекателе

3 Практическая реализация

```
import glfw
from OpenGL.GL import *

window_size = (800, 800)
background_color = 255
color, cnt, cnt_2 = 0, 0, 0
pixels = [0] * window_size[0] * window_size[1]
points, points_2 = [], []
edges, edges_2 = [], []
otsek = 0 #0 - обычный, 1 - отсекатель

def is_inside(p1, p2, p):
    is_in = (p2[0] - p1[0]) * (p[1] - p1[1]) - (p2[1] - p1[1]) * (p[0] - p1[0])
    if is_in <= 0:
        return True
    else:
        return False

def intersect(p1, p2, p3, p4):
    if p2[0] - p1[0] == 0: #первая линия вертикальная
        x = p1[0]
        m2 = (p4[1] - p3[1]) / (p4[0] - p3[0])
        b2 = p3[1] - m2 * p3[0]
        y = m2 * x + b2
    elif p4[0] - p3[0] == 0: #вторая линия вертикальная
        x = p3[0]
```

```
        m1 = (p2[1] - p1[1]) / (p2[0] - p1[0])
        b1 = p1[1] - m1 * p1[0]
        y = m1 * x + b1
    else: #обе линии не вертикальные
        m1 = (p2[1] - p1[1]) / (p2[0] - p1[0])
        b1 = p1[1] - m1 * p1[0]
        m2 = (p4[1] - p3[1]) / (p4[0] - p3[0])
        b2 = p3[1] - m2 * p3[0]
        x = (b2 - b1) / (m1 - m2)
        y = m1 * x + b1
    intersection = (x, y) #точка пересечения
    return intersection

def cut(poly, otsek_poly):
    res = poly.copy()
    for i in range(len(otsek_poly)):#цикл по рёбрам отсекателя
        cur = res.copy()
        res = []
        otsek_start = otsek_poly[i - 1]
        otsek_end = otsek_poly[i]
        for j in range(len(cur)): #цикл по рёбрам фигуры
            poly_start = cur[j - 1]
            poly_end = cur[j]
            if is_inside(otsek_start, otsek_end, poly_end):
                if not is_inside(otsek_start, otsek_end, poly_start):
                    intersection = intersect(poly_start, poly_end, otsek_start, otsek_end)
                    res.append(intersection)
                res.append(tuple(poly_end))
            elif is_inside(otsek_start, otsek_end, poly_start):
                intersection = intersect(poly_start, poly_end, otsek_start, otsek_end)
                res.append(intersection)
    return res

def add_point(x, y):
    global color, cnt, cnt_2, points, points_2, pixels, edges, edges_2, otsek
    cur_cnt = cnt
    cur_points, cur_edges = points, edges
    if otsek == 1:
        cur_cnt = cnt_2
        cur_points, cur_edges = points_2, edges_2
    #если выходим за границы или пиксель уже покрашен
    if x * window_size[0] + y >= len(pixels):
        return
    cur_points.append((x, y))
    cur_cnt += 1
    if cur_cnt > 1:
        if cur_cnt == 3:
```

```
            cur_edges.append((0, 1)) #исключительный случай
        cur_edges.append((cur_cnt - 2, cur_cnt - 1))
        if cur_cnt > 2:
            cur_edges[0] = (0, cur_cnt - 1) #соединяем первую и последнюю точки
    if otsek:
        points_2, edges_2, cnt_2 = cur_points, cur_edges, cur_cnt
    else:
        points, edges, cnt = cur_points, cur_edges, cur_cnt

def bresenham(p1, p2, color=255): #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(color=255): #рисуем все рёбра
    global cnt, cnt_2, points, points_2, pixels, edges, edges_2, otsek
    cur_points, cur_edges = points, edges
    if otsek:
        cur_points, cur_edges = points_2, edges_2
    for edge in cur_edges:
        bresenham(cur_points[edge[0]], cur_points[edge[1]], color)

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, cnt_2, background_color, points, points_2, pixels, edges, edges_2, otsek
    if action == glfw.PRESS:
        if key == glfw.KEY_D: #рисуем фигуру
            if otsek == 1 and len(edges_2) > 0:
                draw()
            elif otsek == 0 and len(edges) > 0:
                draw()
        elif key == glfw.KEY_C: #очищаем пространство
            pixels = [0] * window_size[0] * window_size[1]
            points, points_2 = [], []
            edges, edges_2 = [], []
            cnt, cnt_2 = 0, 0
            otsek = 0
        elif key == glfw.KEY_O: #режим отсекателя
            otsek = 1 - otsek
        elif key == glfw.KEY_M: #процедура отсечения
            poly = points.copy()
            otsek_poly = points_2.copy()
            result_poly = cut(poly, otsek_poly)
            otsek = 0
            draw(0)
            points, edges = [], []
            cnt = 0
            for p in result_poly:
                add_point(int(p[0]), int(p[1]))
            if len(edges) > 0:
                draw()

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)

def main():
    if not glfw.init():
        return
```

```
    window = glfw.create_window(window_size[0], window_size[1], "laba5", 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()
```

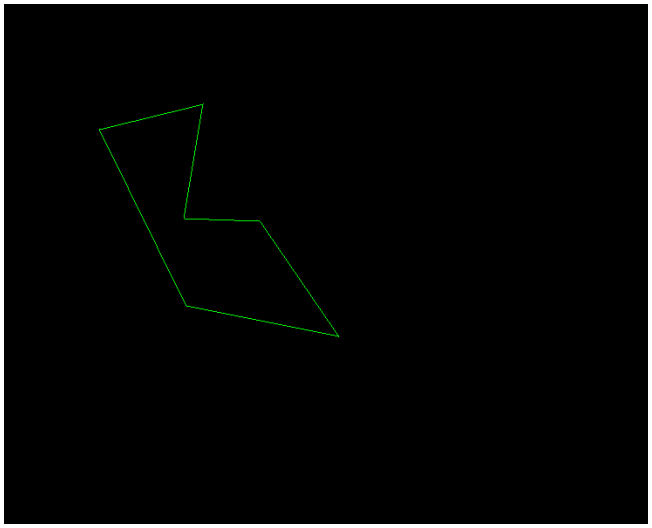

Рис. 1 — Многоугольник

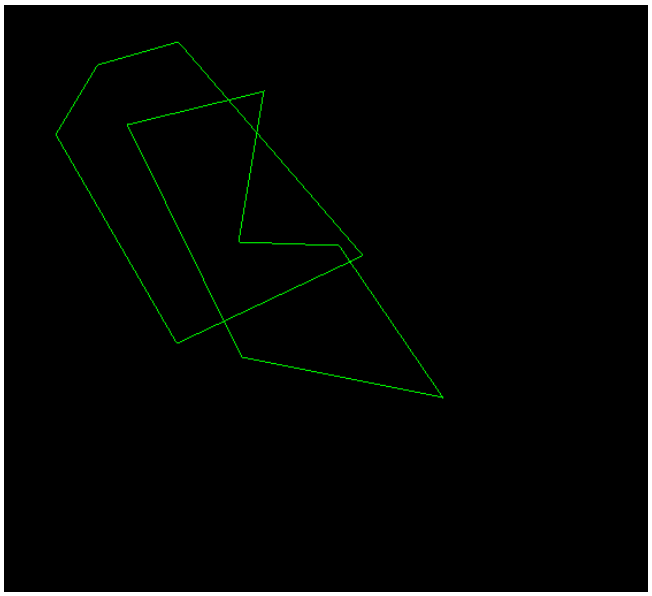

Рис. 2 — Многоугольник с добавленным отсекателем

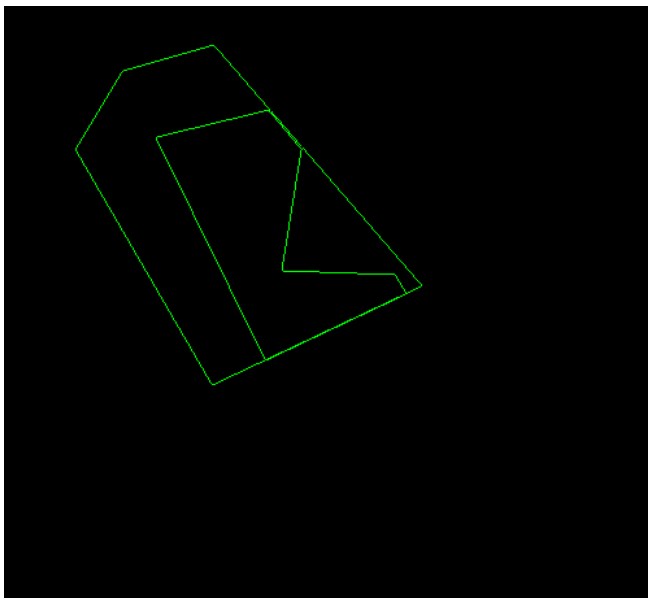

Рис. 3 — Фигура после внутреннего отсечения

4 Заключение

В данной лабораторной работе я познакомилась поближе с алгоритмами отсечения, их классификацией (по типу обрабатываемых объектов: отсечение точки, отсечение прямой, ...; по размерности: двумерное отсечение, трёхмерное отсечение; по расположению результата: внутреннее отсечение, внешнее отсечение), вспомнила как определять положение точки относительно прямой (через уравнение ориентированной прямой, через скалярное произведение, через векторное произведение). Также я реализовала алгоритм Сазерленда-Ходжмена, который осуществляет последовательное отсечение произвольного многоугольника выпуклым отсекателем.