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 «Компьютерная графика»

«Алгоритмы отсечения»

Студент группы ИУ9-42Б Волохов А. В.

Преподаватель Цалкович П. А.

Москва 2024

1 Задача

Необходимо:

а) Реализовать алгоритм отсечения средней точкой, тип отсечения - внешний.

б) Ввод исходных данных каждого из алгоритмов производится интерактивно с помощью клавиатуры и/или мыши.

2 Основная теория

Задача отсечения отрезков: Отсечение отрезков - это процесс определения видимых частей отрезков на экране компьютерного изображения или в графическом приложении. Он используется для оптимизации процесса рендеринга и уменьшения количества избыточной графики, которая не будет видна пользователю.

Алгоритм отсечения средней точкой: Алгоритм отсечения средней точкой является одним из методов отсечения отрезков. Он работает следующим образом:

Сначала определяются координаты окна отсечения, которые определяют область, видимую на экране. Затем для каждого отрезка проверяется его положение относительно окна отсечения. Если отрезок полностью находится вне этой области, то он отбрасывается. Если он полностью видим, то оставляется без изменений.

Если отрезок пересекает границы окна отсечения, то он обрезается до видимой части. Тип отсечения - внешний: Тип отсечения определяет, какие части отрезков будут отбрасываться или оставаться в результате. В случае внешнего типа отсечения отрезок будет отбрасываться только в том случае, если он полностью находится за пределами окна отсечения.

Принцип работы интерактивного ввода данных: Интерактивный ввод данных с помощью клавиатуры и/или мыши позволяет пользователю вводить координаты начальной и конечной точек отрезка, а также координаты границ окна отсечения. Это делает процесс ввода более удобным и понятным для пользователя, так как он может непосредственно взаимодействовать с изображением.

3 Практическая реализация

Код представлен в Листинге 1.

Листинг 1 - lab5.py

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

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


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


def bresenham(x1, y1, x2, y2, color):
    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] = color[0]
                pixels[position + 1] = color[1]
                pixels[position + 2] = color[2]
            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] = color[0]
                pixels[position + 1] = color[1]
                pixels[position + 2] = color[2]
            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] = color[0]
        pixels[position + 1] = color[1]
        pixels[position + 2] = color[2]


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], color=(255, 255, 255))

    if len(points) > 2:
        bresenham(points[-1][0], points[-1][1], points[0][0], points[0][1],
            color=(255, 255, 255))


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 clip_line(xmin, xmax, ymin, ymax, point1, point2):
    #
    def compute_code(point):
        code = 0
        if point[0] < xmin:
            code |= 1
        elif point[0] > xmax:
            code |= 2
        if point[1] < ymin:
            code |= 4
        elif point[1] > ymax:
            code |= 8
        return code

    #
    new_point1 = list(point1)
    new_point2 = list(point2)

    #
    code1 = compute_code(new_point1)
    code2 = compute_code(new_point2)

    while True:
```

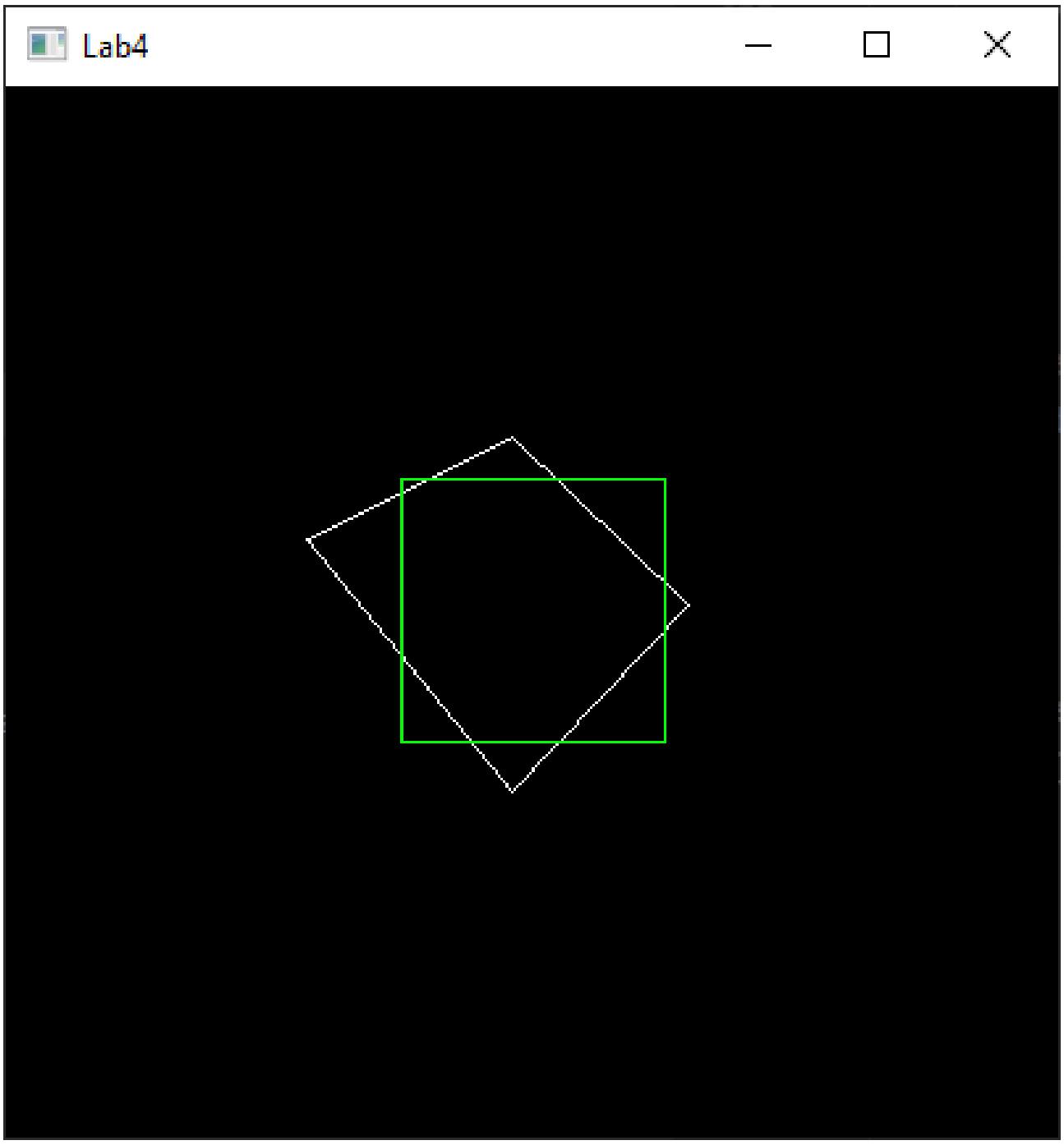
```
        #                    ,

        if not (code1 | code2):
            return [int(new_point1[0]), int(new_point1[1])], [int(new_point2
                [0]), int(new_point2[1])]  #


        #                    ,

        elif code1 & code2:
            return None  #

        else:
            #
            if code1:
                code = code1
            else:
                code = code2

            #

            if code & 1:  #
                x = xmin
                y = new_point1[1] + (new_point2[1] - new_point1[1]) * (xmin -
                    new_point1[0]) / (new_point2[0] - new_point1[0])
            elif code & 2:  #
                x = xmax
                y = new_point1[1] + (new_point2[1] - new_point1[1]) * (xmax -
                    new_point1[0]) / (new_point2[0] - new_point1[0])
            elif code & 4:  #
                y = ymin
                x = new_point1[0] + (new_point2[0] - new_point1[0]) * (ymin -
                    new_point1[1]) / (new_point2[1] - new_point1[1])
            elif code & 8:  #
                y = ymax
                x = new_point1[0] + (new_point2[0] - new_point1[0]) * (ymax -
                    new_point1[1]) / (new_point2[1] - new_point1[1])

            #
            if code == code1:
                new_point1[0], new_point1[1] = int(x), int(y)
                code1 = compute_code(new_point1)
            else:
                new_point2[0], new_point2[1] = int(x), int(y)
                code2 = compute_code(new_point2)
```

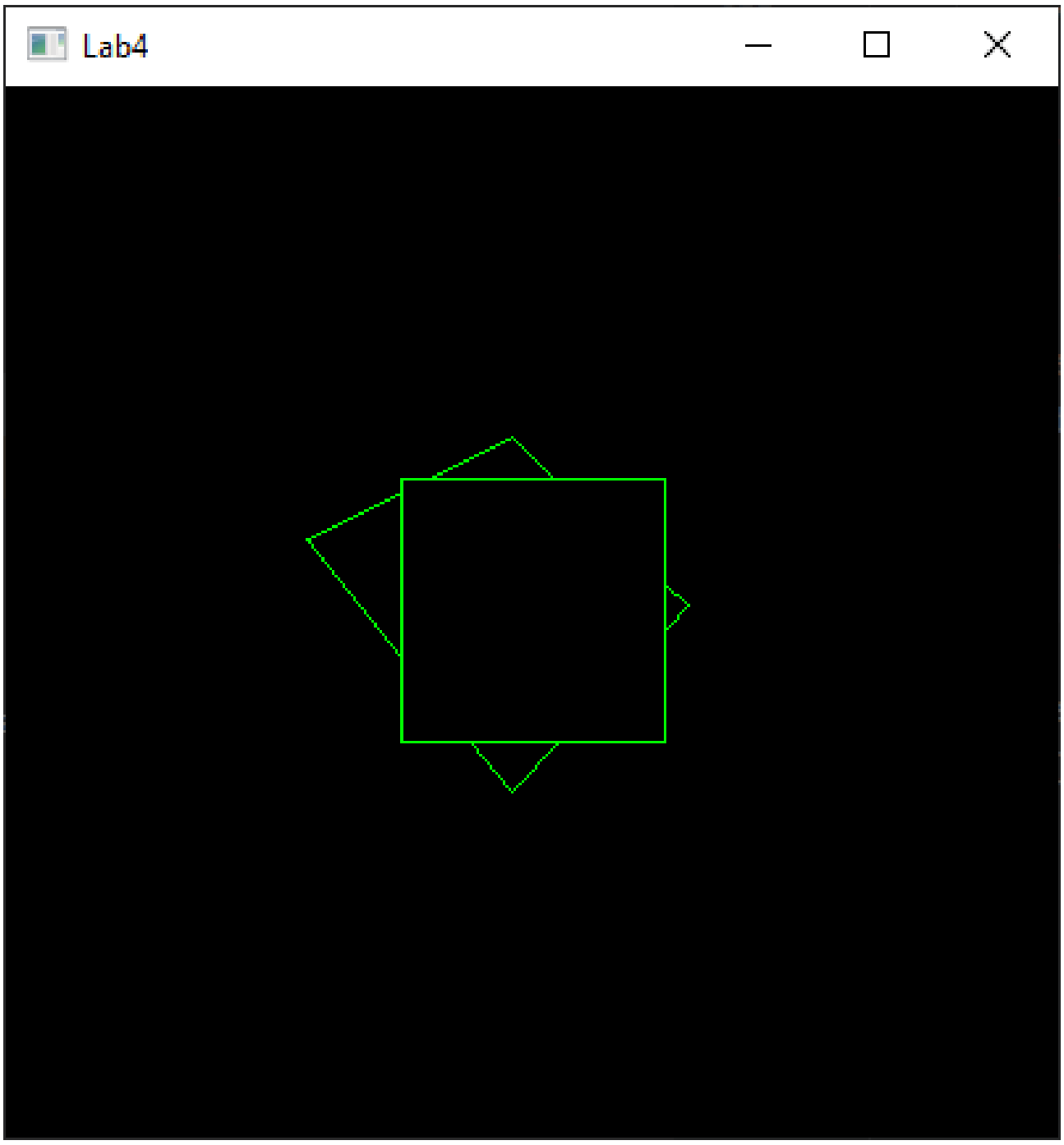
```
def clip_edges():
    global edges, clipped_edges
    clipped_edges.clear()
    min_x_rect = 150
    max_x_rect = 250
    min_y_rect = 150
    max_y_rect = 250
    edges.append([edges[0][0], edges[-1][1]])
    for edge in edges:
        p1, p2 = edge
        clipped = clip_line(min_x_rect, max_x_rect, min_y_rect, max_y_rect,
            p1, p2)
        if clipped is None:
            clipped_edges.append(edge)
        elif p1 != clipped[0] and p1 != clipped[1] and p2 != clipped[0] and
            p2 != clipped[1]:
            clipped_edges.append([p1, clipped[0]])
            clipped_edges.append([clipped[1], p2])
        elif p1 == clipped[0]:
            clipped_edges.append([clipped[1], p2])



def key_callback(window, key, scancode, action, mods):
    global pixels, points, edges
    if action == glfw.PRESS:
        if key == glfw.KEY_F:
            if len(points) > 2:
                bresenham(points[-1][0], points[-1][1], points[0][0], points
                    [0][1], color=(255, 255, 255))
            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_ENTER:
            if len(points) > 2:
                #

                clip_edges()
                redraw_clipped_polygon()

        elif key == glfw.KEY_BACKSPACE:
```

```
            pixels = [0 for _ in range(size * size * 3)]
            points.clear()
            edges.clear()


def redraw_clipped_polygon():
    global pixels, points, edges, clipped_edges, size
    pixels = [0] * (size * size * 3)
    # for i in range(1, len(points)):
    #     bresenham(points[i - 1][0], points[i - 1][1], points[i][0], points[
        i][1], color=(255, 255, 255))
    #     if len(points) > 2:
    #         bresenham(points[-1][0], points[-1][1], points[0][0], points
        [0][1], color=(255, 255, 255))
    print(clipped_edges)
    for edge in clipped_edges:
        p1, p2 = edge
        if p1[0] != p2[0] and p1[1] != p2[1]:
            bresenham(int(p1[0]), int(p1[1]), int(p2[0]), int(p2[1]), color
                =(0, 255, 0))


def display(window):
    min_x_rect = 150
    max_x_rect = 250
    min_y_rect = 150
    max_y_rect = 250
    bresenham(min_x_rect, min_y_rect, max_x_rect, min_y_rect, color=(0, 255,
        0))
    bresenham(min_x_rect, min_y_rect, min_x_rect, max_y_rect, color=(0, 255,
        0))
    bresenham(min_x_rect, max_y_rect, max_x_rect, max_y_rect, color=(0, 255,
        0))
    bresenham(max_x_rect, min_y_rect, max_x_rect, max_y_rect, color=(0, 255,
        0))
    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

Таким образом, алгоритм отсечения средней точкой является важным инструментом в области компьютерной графики, который позволяет оптимизировать процесс рендеринга изображений. Его использование позволяет улучшить производительность графических приложений и обеспечить более эффективное использование ресурсов компьютера. Интерактивный ввод данных делает процесс работы с алгоритмом более удобным и интуитивно понятным для пользователей.