Министерство образования Республики Беларусь
Учреждение образования “Белорусский государственный университет информатики и радиоэлектроники”

Факультет информационных технологий и управления
Кафедра интеллектуальных информационных технологий
Дисциплина: Графический интерфейс интеллектуальных систем

Отчёт к лабораторной работе №6

Выполнил: Гафаров М.С.
Группа: 221702
Проверила: Жмырко А.В.

Минск 2024

Лабораторная работа №6
Заполнение полигонов

Цель: изучить основные группы методов заполнения полигонов: растровая развёртка, затравочное заполнение и комбинированные методы.

Задание: разработать элементарный графический редактор, реализующий построение полигонов и их заполнение, используя алгоритм растровой развёртки с упорядоченным списком рёбер; алгоритм растровой развёртки с упорядоченным списком рёбер, использующий список активных рёбер; простой алгоритм заполнения с затравкой; построчный алгоритм заполнения с затравкой. Выбор алгоритма задаётся из пункта меню и доступен через панель инструментов “Алгоритмы заполнения полигонов”. В редакторе должен быть предусмотрен режим отладки, где отображается пошаговое решение.

Теоретические сведения:
Генерация сплошных областей из простых описаний рёбер или вершин называется **растровой развёрткой сплошных областей** или **заполнением полигонов**. Для заполнения можно использовать несколько методов, которые подразделяются на 3 группы:

- Растровая развёртка;
- Затравочное заполнение;
- Комбинированные методы;

Суть методов **растровой развёртки** в нахождении точки пересечения полигона со сканирующими строками. Такие строки представляют собой линии, параллельные оси ординат. Такие методы обычно движутся от “верха” полигона к низу.
В методах **затравочного заполнения** предполагается, что известна некоторая точка внутри замкнутого контура (затравочный пиксель). В алгоритме ищут точки, соседние с затравочной и находящиеся внутри замкнутой области. Если найденная соседняя точка расположена внутри контура, она становится затравочной и поиск продолжается рекурсивно. Если же она расположена не внутри контура, то найдена граница контура.
Комбинированный метод сочетает в себе преимущества методов растровой развёртки и затравочного заполнения – в нём стек минимизируется за счёт хранения только затравочного пикселя для любого непрерывного интервала на сканирующей строке.

Программная реализация:

1. **Вспомогательные функции.**
   - **1.1. Нахождение точек пересечения отрезка со сканирующими строками:**

```python
def find_intersect(x_coords: list, y_coords: list) -> list:  1 usage
    x1, y1, x2, y2, result = x_coords[0], y_coords[0], x_coords[1], y_coords[1], list()
    if y1 == y2:
        return []
    for i in range(0, max(y1, y2) + 1):
        if min(y1,y2) <= i <= max(y1,y2):
            t = (i - y1) / (y2 - y1)
            if t < 0 or t > 1:
                continue
            x_intersect = x1 + (x2 - x1) * t
            result.append([x_intersect, i])
    return result
```

1.2. Нахождение точек пересечения отрезка с заданной сканирующей строкой.

```python
def find_intersect_with_concrete_line(x_coords: list, y_coords: list, line_y: int) -> list:  3 usages
    x1, y1, x2, y2 = x_coords[0], y_coords[0], x_coords[1], y_coords[1]
    if y1 == y2:
        return []
    t = (line_y - y1) / (y2 - y1)
    if t < 0 or t > 1:
        return []
    x_intersect = x1 + (x2 - x1) * t
    result = [line_y, x_intersect]
    return result
```

1.3. Разбивание списка точек на пары.

```python
def split_into_pairs(lst):  7 usages
    pairs = [lst[i:i+2] for i in range(0, len(lst), 2)]
    if pairs and len(pairs[-1]) == 1:
        pairs[-1].append(pairs[-1][0])
    return pairs
```

1.4. Проверка, находится ли заданная точка внутри замкнутого контура.

```python
def is_point_in_polygon(x, y, polygon):  12 usages
    inside = False
    n = len(polygon)
    p1x, p1y = polygon[0]
    for i in range(1, n):
        p2x, p2y = polygon[i]
        if (p1y > y) != (p2y > y):
            xinters = (y - p1y) * (p2x - p1x) / (p2y - p1y) + p1x
            if x < xinters:
                inside = not inside
        p1x, p1y = p2x, p2y
    return inside
```

2. Основная реализация.

2.1. Алгоритм растровой развёртки с упорядоченным списком рёбер.

```python
def with_ordered_list_of_edges(self) -> list:  4 usages
    intersections, intervals = [], []
    self.pol.append(self.pol[0])
    for i in range(len(self.pol) - 1):
        is_list = False
        temp_results = find_intersect( x_coords: [self.pol[i][0], self.pol[i + 1][0]],  y_coords: [self.pol[i][1], self.pol[i + 1][1]])
        for j in range(len(temp_results)):
            if isinstance(temp_results[i], list):
                is_list = True
        if is_list:
            intersections.extend(temp_results)
        else:
            intersections.append(temp_results)
    intersections = [el for el in intersections if el]
    sorted_intersections = sorted(intersections, key=lambda x: (x[1], x[0]))
    intervals = split_into_pairs(sorted_intersections)
    return intervals
```

2.2. Алгоритм растровой развёртки с упорядоченным списком рёбер, использующий список активных рёбер.

```python
def ordered_list_using_active_edges(self) -> list:  3 usages
    self.pol.append(self.pol[0])
    sap, upgraded_sap = dict(), dict()
    intervals = []
    x_values, y_values = [el[0] for el in self.pol], [el[1] for el in self.pol]
    max_x, max_y, max_index = find_max_point(x_values, y_values)
    for i in range(1, max_y + 1):
        temp_list = []
        for j in range(len(self.pol) - 1):
            if self.pol[j][1] == self.pol[j+1][1]:
                continue
            intersection_points = find_intersect_with_concrete_line( x_coords: [self.pol[j][0], self.pol[j+1][0]],
                                                                     y_coords: [self.pol[j][1], self.pol[j+1][1]], i)
            temp_list.append(intersection_points[::-1])
        sap[i] = temp_list
    upgraded_sap = {key: [item for item in value if item] for key, value in sap.items()}
    sorted_sap = dict(sorted(upgraded_sap.items(), reverse=True))
    for key in sorted_sap:
        sorted_sap[key].sort(key=lambda x: (x[0], x[1]))
    values = list(sorted_sap.values())
    for i in range(len(values)):
        if len(values[i]) <= 2:
            intervals.append(values[i])
        else:
            pairs = split_into_pairs(values[i])
            for j in range(len(pairs)):
                intervals.append(pairs[j])
    return intervals
```

2.3. Простой алгоритм заполнения с затравкой.

```python
def simple_seeded_filling(self, pixel: list):  3 usages
    filled_pixels, stack, visited = [], [(pixel[0], pixel[1])], set()
    self.pol.append(self.pol[0])
    xs, ys = [el[0] for el in self.pol], [el[1] for el in self.pol]
    max_x, min_x, max_y, min_y = int(max(xs)), int(min(xs)), int(max(ys)), int(min(ys))
    if not is_point_in_polygon(pixel[0], pixel[1], self.pol):
        print("Затравочная точка вне полигона!")
        return []
    while stack:
        x, y = stack.pop()
        if (x, y) in visited:
            continue
        visited.add((x, y))
        if x < min_x or x > max_x or y < min_y or y > max_y:
            continue
        if is_point_in_polygon(x, y, self.pol):
            filled_pixels.append((x, y))
            stack.append((x, y + 1))
            stack.append((x - 1, y))
            stack.append((x, y - 1))
            stack.append((x + 1, y))
    return filled_pixels
```

2.4. Комбинированный метод. Построчный алгоритм заполнения с затравкой.

```python
def seeded_filling_line_by_line(self, pixel: list):  3 usages
    filled_pixels, stack = set(), [(pixel[0], pixel[1])]
    self.pol.append(self.pol[0])
    xs, ys = [el[0] for el in self.pol], [el[1] for el in self.pol]
    max_x, min_x, max_y, min_y = int(max(xs)), int(min(xs)), int(max(ys)), int(min(ys))
    while stack:
        x, y = stack.pop()
        if (x, y) in filled_pixels:
            continue
        x_left = x
        while x_left > min_x and is_point_in_polygon(x_left - 1, y, self.pol) and ((x_left - 1, y)
                                                                                   not in filled_pixels):
            x_left -= 1
        x_right = x
        while x_right < max_x and is_point_in_polygon(x_right + 1, y, self.pol) and ((x_right + 1, y)
                                                                                     not in filled_pixels):
            x_right += 1
        for xi in range(x_left, x_right + 1):
            filled_pixels.add((xi, y))
        for ny in [y - 1, y + 1]:
            if ny < min_y or ny > max_y:
                continue
            xi = x_left
            while xi <= x_right:
                if is_point_in_polygon(xi, ny, self.pol) and ((xi, ny) not in filled_pixels):
                    stack.append((xi, ny))
                    xi_temp = xi + 1
                    while xi_temp <= x_right and is_point_in_polygon(xi_temp, ny, self.pol) and (
                            (xi_temp, ny) not in filled_pixels):
                        xi_temp += 1
                    xi = xi_temp
                else:
                    xi += 1
    return list(filled_pixels)
```

Вывод к лабораторной работе: в результате выполнения лабораторной работы были изучены и реализованы основные алгоритмы заполнения

полигонов: алгоритмы растровой развёртки, алгоритмы затравочного заполнения и комбинированные алгоритмы. Результаты лабораторной работы были объединены с результатами лабораторной работы №5 с целью получения полноценного элементарного графического редактора, способного выполнять функции построения и заполнения полигонов.