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 №7

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

Минск 2025

Лабораторная работа №7
Триангуляция. Построение диаграммы Вороного

Цель: изучить методы построения триангуляции Делоне и диаграммы Вороного, а также реализовать их на практике.

Задание: разработать графическую программу, выполняющую триангуляцию Делоне и построение диаграммы Вороного по заданному набору точек.

Теоретические сведения:
Триангуляция Делоне и **диаграмма Вороного** – фундаментальные инструменты вычислительной геометрии, которые применяются как в теории, так и на практике для решения различных типов задач машинной и компьютерной графики.
Триангуляцией называется планарный граф, все внутренние области которого являются треугольниками.
Триангуляция для некоторого набора точек S называется **триангуляцией Делоне**, если описанная окружность для каждого треугольника будет свободна от точек.
Диаграмма Вороного – геометрическое разбиение области на многоугольники (полигоны), обладающее следующим свойством: для любого центра системы {A} можно указать область пространства, все точки которой ближе к данному центру, чем к любому другому центру системы. Такая область называется многогранником/областью/ячейкой Вороного.

Программная реализация:

1. **Триангуляция Делоне.**
   - **1.1. Построение области (треугольника), охватывающего все точки множества.**

```
def big_triangle(points):  1 usage
    minx = np.min(points[:, 0])
    maxx = np.max(points[:, 0])
    miny = np.min(points[:, 1])
    maxy = np.max(points[:, 1])
    dx = maxx - minx
    dy = maxy - miny
    dxy = max(dx, dy)
    midx = dx * 0.5 + minx
    midy = dy * 0.5 + miny
    return np.array([
        [midx - 10 * dxy, midy - 10 * dxy],
        [midx, midy + 10 * dxy],
        [midx + 10 * dxy, midy - 10 * dxy]
    ])
```

1.2. Нахождение описанной окружности треугольника.

```
def circumcircle_of_triangle(points, v1, v2, v3):  2 usages
    x1, y1 = points[v1]
    x2, y2 = points[v2]
    x3, y3 = points[v3]
    dy12 = abs(y1 - y2)
    dy23 = abs(y2 - y3)
    if dy12 < 1e-7:
        m2 = -((x3 - x2) / (y3 - y2))
        mx2, my2 = (x2 + x3) / 2, (y2 + y3) / 2
        xc = (x1 + x2) / 2
        yc = m2 * (xc - mx2) + my2
    elif dy23 < 1e-7:
        m1 = -((x2 - x1) / (y2 - y1))
        mx1, my1 = (x1 + x2) / 2, (y1 + y2) / 2
        xc = (x2 + x3) / 2
        yc = m1 * (xc - mx1) + my1
    else:
        m1 = -((x2 - x1) / (y2 - y1))
        m2 = -((x3 - x2) / (y3 - y2))
        mx1, my1 = (x1 + x2) / 2, (y1 + y2) / 2
        mx2, my2 = (x2 + x3) / 2, (y2 + y3) / 2
        xc = (m1 * mx1 - m2 * mx2 + my2 - my1) / (m1 - m2)
        if dy12 > dy23:
            yc = m1 * (xc - mx1) + my1
        else:
            yc = m2 * (xc - mx2) + my2

    dx = x2 - xc
    dy = y2 - yc
    r = dx * dx + dy * dy
    return {'a': v1, 'b': v2, 'c': v3, 'x': xc, 'y': yc, 'r': r}
```

1.3. Построение триангуляции.

```python
def triangulate(points):  3 usages
    n = len(points)
    if n < 3:
        return []
    points = points.copy()
    ind = np.argsort(points[:, 0])
    big = big_triangle(points)
    points = np.vstack([points, big])
    cur_points = [circumcircle_of_triangle(points, n, n + 1, n + 2)]
    ans = []
    edges = []

    for i in range(len(ind) - 1, -1, -1):
        for j in range(len(cur_points) - 1, -1, -1):
            dx = points[ind[i], 0] - cur_points[j]['x']
            if dx > 0 and dx * dx > cur_points[j]['r']:
                ans.append(cur_points[j])
                cur_points.pop(j)
                continue
            dy = points[ind[i], 1] - cur_points[j]['y']
            if dx * dx + dy * dy - cur_points[j]['r'] > 1e-7:
                continue
            edges.extend(
                [cur_points[j]['a'], cur_points[j]['b'], cur_points[j]['b'],
                 cur_points[j]['c'], cur_points[j]['c'],
                 cur_points[j]['a']])
            cur_points.pop(j)
        edges = delete_multiples_edges(edges)
        for j in range(0, len(edges), 2):
            a, b = edges[j], edges[j + 1]
            cur_points.append(circumcircle_of_triangle(points, a, b, ind[i]))
        edges = []
    for triangle in cur_points:
        ans.append(triangle)
    tr = []
    for triangle in ans:
        if triangle['a'] < n and triangle['b'] < n and triangle['c'] < n:
            tr.extend([triangle['a'], triangle['b'], triangle['c']])

    return tr
```

2. Диаграмма Вороного.

2.1. Построение перпендикуляра к заданному отрезку.

```python
def find_bisector(point1: list, point2: list) -> list:
    x0, y0 = ((float(point1[0]) + float(point2[0])) / 2,
              (float(point1[1]) + float(point2[1])) / 2)
    a, b = point2[0] - point1[0], point2[1] - point1[1]
    c = a * x0 * (-1) - b * y0
    return [a, b, c]
```

2.2. Нахождение точек пересечения заданной прямой с рёбрами заданного многоугольника.

```python
def find_intersection_with_polygon(polygon: list, line_coeffs: list) -> list:
    a, b, c = line_coeffs
    intersections = []
    n = len(polygon)
    eps = 1e-9
    def already_exists(p, points_list):
        for pt in points_list:
            if (math.isclose(pt[0], p[0], abs_tol=eps) and
                    math.isclose(pt[1], p[1], abs_tol=eps)):
                return True
        return False
    for i in range(n):
        p1 = polygon[i]
        p2 = polygon[(i + 1) % n]
        x1, y1 = p1
        x2, y2 = p2
        dx = x2 - x1
        dy = y2 - y1
        denom = a * dx + b * dy
        if abs(denom) < eps:
            if math.isclose(a * x1 + b * y1 + c, b: 0, abs_tol=eps):
                if not already_exists( p: [x1, y1], intersections):
                    intersections.append([x1, y1])
                if not already_exists( p: [x2, y2], intersections):
                    intersections.append([x2, y2])
            continue
        t = - (a * x1 + b * y1 + c) / denom
        if 0 <= t <= 1:
            x_inter = x1 + t * dx
            y_inter = y1 + t * dy
            pt = [x_inter, y_inter]
            if not already_exists(pt, intersections):
                intersections.append(pt)
    return intersections
```

2.3. Проверка, лежит ли заданная точка на заданном отрезке.

```python
def point_on_segment(p, a, b, eps=1e-9):  1 usage
    ax, ay = a
    bx, by = b
    px, py = p
    cross = (px - ax) * (by - ay) - (py - ay) * (bx - ax)
    if abs(cross) > eps:
        return False
    dot = (px - ax) * (bx - ax) + (py - ay) * (by - ay)
    if dot < -eps:
        return False
    sq_len = (bx - ax) ** 2 + (by - ay) ** 2
    if dot - sq_len > eps:
        return False
    return True
```

2.4. Нахождение позиции точки вдоль периметра многоугольника.

```python
def find_point_position(point, polygon):  2 usages
    n = len(polygon)
    for i in range(n):
        a = polygon[i]
        b = polygon[(i + 1) % n]
        if point_on_segment(point, a, b):
            if abs(point[0] - a[0]) < 1e-9 and abs(point[1] - a[1]) < 1e-9:
                return i, 0.0
            dx = b[0] - a[0]
            dy = b[1] - a[1]
            if abs(dx) >= abs(dy):
                t = (point[0] - a[0]) / dx if dx != 0 else 0.0
            else:
                t = (point[1] - a[1]) / dy if dy != 0 else 0.0
            return i, t
    raise ValueError("Точка {} не принадлежит границе многоугольника".format(point))
```

2.5. Отсечение от многоугольника части по пути от одной заданной точки до другой вдоль границы.

```python
def cut_polygon(poly, pt1, pt2):  3 usages
    i1, t1 = find_point_position(pt1, poly)
    i2, t2 = find_point_position(pt2, poly)
    n = len(poly)
    result = [pt1]
    if i1 == i2:
        if t2 > t1:
            result.append(pt2)
        else:
            current_index = (i1 + 1) % n
            while current_index != i1:
                result.append(poly[current_index])
                current_index = (current_index + 1) % n
                 if current_index == i2:
                     break
             result.append(pt2)
         return result
    current_index = (i1 + 1) % n
    while current_index != i2:
        result.append(poly[current_index])
        current_index = (current_index + 1) % n
    if not math.isclose(t2,  b: 0.0, abs_tol=1e-9):
        result.append(poly[i2])
    result.append(pt2)
    return result
```

2.6. Нахождение ячеек Вороного.

```python
def diagram(points: list):  4 usages
    points_edges = points
    points_edges.append(points_edges[0])
    cells = list()
    xs, ys = [int(el[0]) for el in points], [int(el[1]) for el in points]
    padding = 0.5
    x_min, x_max, y_min, y_max = min(xs), max(xs), min(ys), max(ys)
    box = [[x_min - padding, y_max + padding], [x_max + padding, y_max + padding],
           [x_max + padding, y_min - padding], [x_min - padding, y_min - padding]]
    for point in points:
        cell = box
        for point2 in points:
            if point2 != point:
                b = find_bisector(point, point2)
                print(b)
                intersections = find_intersection_with_polygon(cell, b)
                print(intersections)
                if len(intersections) == 2:
                    newCell = cut_polygon(cell, intersections[0], intersections[1])
                    print(newCell)
                    if not Path(newCell).contains_point(point):
                        newCell = cut_polygon(cell, intersections[1], intersections[0])
                    cell = newCell
            else:
                continue
        cells.append(cell)
    return cells, box
```

Вывод к лабораторной работе: в результате выполнения лабораторной работы были изучены и программно реализованы алгоритмы построения триангуляции Делоне и диаграммы Вороного – основных фундаментальных инструментов вычислительной геометрии.