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 №5

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

Минск 2024

Лабораторная работа №5
Предварительная обработка полигонов

Цель: изучить основные методы построения выпуклых оболочек полигонов и их обработки.

Задание: разработать элементарный графический редактор, реализующий построение полигонов. Реализованная программа должна уметь проверять полигон на выпуклость, находить его внутренние нормали. Программа должна выполнять построения выпуклых оболочек методом обхода Грэхема и методом Джарвиса. Выбор метода задаётся из пункта меню и должен быть доступен через панель инструментов “Построение полигонов”. Графический редактор должен позволять рисовать линии первого порядка (лабораторная работа №1) и определять точки пересечения отрезка со стороной полигона, также программа должна определять принадлежность введённой точки полигону.

Теоретические сведения:
Полигон(многоугольник) **–** замкнутая кривая на плоскости, образуемая отрезками прямых линий. Отрезки называются рёбрами или сторонами полигона. Концевые точки отрезков, совпадающие для двух соседних рёбер, называются вершинами.
Полигон называется ***простым***, если он не пересекает самого себя.
Вершины полигона подразделяются на ***выпуклые*** и ***вогнутые***. Вершина зазывается ***выпуклой***, если внутренний угол при этой вершине меньше или равен 180^{o}. В противном случае вершина считается ***вогнутой***.
Отрезок прямой линии между двумя не соседними вершинами называется ***диагональю***. ***Диагональ*** полигона называется ***хордой***, если она лежит внутри полигона и не затрагивает его внешнюю область.
К основным задачам, решаемым над полигонами, можно отнести:

1. Проверка полигона на выпуклость.
2. Определение принадлежности точки полигону.
3. Построение выпуклой оболочки.
4. Заполнение многоугольников.
5. Операции над многоугольниками (объединение, пересечение и др.)
6. Триангуляция и построение диаграммы Вороного.

В данной лабораторной работе рассматриваются и будут реализованы операции 1 – 3.

Программная реализация:

1. **Вспомогательные функции:**
   - **1.1. Нахождение экстремальной точки:**

```python
def find_min_point(x_coords: list, y_coords: list):  2 usages
    y_min, x_min, index = 1000, 1000, 0
    for i in range(len(y_coords)):
        if y_coords[i] < y_min:
            y_min = y_coords[i]
            x_min = x_coords[i]
            index = i
        elif y_coords[i] == y_min:
            if x_coords[i] < x_min:
                y_min = y_coords[i]
                x_min = x_coords[i]
                index = i
    return x_min, y_min, index
```

1.2. Нахождение максимальной точки:

```python
def find_max_point(x_coords: list, y_coords: list):  1 us
    y_max, x_max, index = 0, 0, 0
    for i in range(len(y_coords)):
        if y_coords[i] > y_max:
            y_max = y_coords[i]
            x_max = x_coords[i]
            index = i
        elif y_coords[i] == y_max:
            if x_coords[i] > x_max:
                y_max = y_coords[i]
                x_max = x_coords[i]
                index = i
    return x_max, y_max, index
```

1.3. Вычисление значений полярного угла:

```python
def calculate_polar_angle(point_x, point_y, x_coord: list, y_coord: list) -> list:
    polar_angles = []
    for i in range(len(y_coord)):
        temp_list = []
        if x_coord[i] - point_x != 0:
            arctan = math.atan((y_coord[i]-point_y)/(x_coord[i]-point_x))
            if y_coord[i]-point_y < 0 or x_coord[i]-point_x < 0:
                arctan += math.pi
        elif x_coord[i] - point_x == 0 and y_coord[i] - point_y >= 0:
            arctan = math.pi / 2
        elif x_coord[i] - point_x == 0 and y_coord[i] - point_y < 0:
            arctan = 3 * math.pi / 2
        temp_list.append(arctan)
        temp_list.append(i)
        polar_angles.append(temp_list)
    polar_angles.sort(key=lambda x: x[0])
    return polar_angles
```

1.4. Проверка, лежит ли точка на границе полигона:

```python
def is_point_on_segment(p, q, r):  1 usage
    if (q[0] <= max(p[0], r[0]) and q[0] >= min(p[0], r[0]) and
        q[1] <= max(p[1], r[1]) and q[1] >= min(p[1], r[1])):
        if abs((r[0]-p[0])*(q[1]-p[1]) - (q[0]-p[0])*(r[1]-p[1])) < 1e-9:
            return True
    return False
```

1.5. Проверка принадлежности точки полигону:

```python
def point_in_polygon(point, polygon):  2 usages
    x, y = point
    inside = False
    n = len(polygon)
    poly = polygon if polygon[0] == polygon[-1] else polygon + [polygon[0]]
    for i in range(len(poly) - 1):
        if is_point_on_segment(poly[i], point, poly[i + 1]):
            return "on boundary"
    inside = False
    p1x, p1y = poly[0]
    for i in range(1, len(poly)):
        p2x, p2y = poly[i]
        if y > min(p1y, p2y):
            if y <= max(p1y, p2y):
                if p1y != p2y:
                    xinters = (y - p1y) * (p2x - p1x) / (p2y - p1y) + p1x
                else:
                    xinters = p1x
                if p1x == p2x or x <= xinters:
                    inside = not inside
        p1x, p1y = p2x, p2y
    return inside
```

2. Основные функции:

2.1. Построение выпуклой оболочки алгоритмом Грэхема:

```
def graham_algorithm(self):  1 usage
    x_coord, p0, p0_index = find_min_point(self.x_coords, self.y_coords)
    polygon = []
    remaining_x = self.x_coords[:]
    remaining_y = self.y_coords[:]
    remaining_x.pop(p0_index)
    remaining_y.pop(p0_index)
    polar_angles = calculate_polar_angle(x_coord, p0, remaining_x, remaining
    sorted_points = []
    for angle, idx in polar_angles:
        sorted_points.append([remaining_x[idx], remaining_y[idx]])
    polygon.append([x_coord, p0])
    if sorted_points:
        polygon.append(sorted_points[0])
        sorted_points.pop(0)
    i = 0
    while i < len(sorted_points):
        e = vector_multiplic(polygon[-2], polygon[-1], sorted_points[i])
        if e > 0:
            polygon.append(sorted_points[i])
            i += 1
        else:
            polygon.pop(-1)
            if len(polygon) < 2:
                polygon.append(sorted_points[i])
                i += 1
    return polygon
```

2.2. Построение выпуклой оболочки алгоритмом Джарвиса:

```
def jarvis_algorithm(self):  3 usages
    x_coord, p0, p0_index = find_min_point(self.x_coords, self.y_coords)
    x_max_coord, pk, pk_index = find_max_point(self.x_coords, self.y_coords)
    inverted_axis = False
    polygon = []
    polygon.append([x_coord, p0])
    i = 0
    while i < len(polygon):
        if not inverted_axis:
            polar_angles = calculate_polar_angle(polygon[i][0], polygon[i][1], self.x_coords, self.y_coords)
            if self.x_coords[polar_angles[0][1]] == polygon[-1][0] and self.y_coords[polar_angles[0][1]] == polygon[-1][1]:
                polygon.append([self.x_coords[polar_angles[1][1]], self.y_coords[polar_angles[1][1]]])
                if polygon[-1][0] == x_max_coord and polygon[-1][1] == pk:
                    inverted_axis = True
                i += 1
                continue
            polygon.append([self.x_coords[polar_angles[0][1]], self.y_coords[polar_angles[0][1]]])
            if polygon[-1][0] == x_max_coord and polygon[-1][1] == pk:
                inverted_axis = True
            i += 1
        i += 1
    elif inverted_axis:
        polar_angles = calculate_polar_angle(polygon[i][0], polygon[i][1], self.x_coords, self.y_coords)
        if self.x_coords[polar_angles[-1][1]] == polygon[-1][0] and self.y_coords[polar_angles[-1][1]] == polygon[-1][1]:
            if self.x_coords[polar_angles[-2][1]] == x_coord and self.y_coords[polar_angles[-2][1]] == p0:
                break
            polygon.append([self.x_coords[polar_angles[-2][1]], self.y_coords[polar_angles[-2][1]]])
            i += 1
            continue
        if self.x_coords[polar_angles[-1][1]] == x_coord and self.y_coords[polar_angles[-1][1]] == p0:
            break
        polygon.append([self.x_coords[polar_angles[-1][1]], self.y_coords[polar_angles[-1][1]]])
        i += 1
    return polygon
```

2.3. Проверка полигона на выпуклость:

```
def check_polygon(self, polygon: list):
    polygon.append(polygon[0])
    vectors, results = [], []
    for i in range(len(polygon) - 1):
        vector = [polygon[i+1][0]-polygon[i][0], polygon[i+1][1]-polygon[i][1]]
        vectors.append(vector)
    vectors.append(vectors[0])
    for i in range(len(vectors) - 1):
        results.append(vectors[i][0] * vectors[i+1][1] - vectors[i][1] * vectors[i+1][0])
    positives = sum(1 for x in results if x > 0)
    negatives = sum(1 for x in results if x < 0)
    if positives and negatives:
        return "Полингон вогнутый."
    elif positives and not negatives:
        return "Полигон выпуклый. Внутренние нормали ориентированы влево от его контура."
    elif negatives and not positives:
        return "Полигон выпуклый. Внутренние нормали ориентированы вправо от его контура."
    elif not positives and not negatives:
        return "Полигон вырождается в отрезок."
```

2.4. Построение внутренних нормалей к сторонам полигона:

```
def find_internal_normals(self, polygon: list):
    polygon.append(polygon[0])
    vectors, hordes, multiplic, results = [], [], [], []
    for i in range(len(polygon) - 1):
        n = [(-1)*(polygon[i+1][1]-polygon[i][1]), (polygon[i+1][0]-polygon[i][0])]
        vectors.append(n)
    polygon.pop(-1)
    for i in range(len(polygon)):
        vi, vj = polygon[i], polygon[(i+2) % len(polygon)]
        hord = [vj[0] - vi[0], vj[1] - vi[1]]
        hordes.append(hord)
    for i in range(len(hordes)):
        multipl = vectors[i][0] * hordes[i][0] + vectors[i][1] * hordes[i][1]
        multiplic.append(multipl)
    for i in range(len(multiplic)):
        if multiplic[i] > 0:
            results.append(vectors[i])
        elif multiplic[i] < 0:
            results.append([vectors[i][0] * (-1), vectors[i][1] * (-1)])
    return results
```

Вывод к лабораторной работе: в результате выполнения лабораторной работы были изучены и реализованы алгоритмы построения выпуклых оболочек полигонов, такие как метод обхода Грэхема и метод Джарвиса, проверки полигона на выпуклость и нахождения его внутренних нормалей. Результаты данной лабораторной работы были объединены с результатами лабораторной работы №1 с целью добавления в программу возможности нахождения точек пересечения полигонов с прямыми первого порядка.