Tarea 14 - Algoritmos para encontrar la envolvente convexa

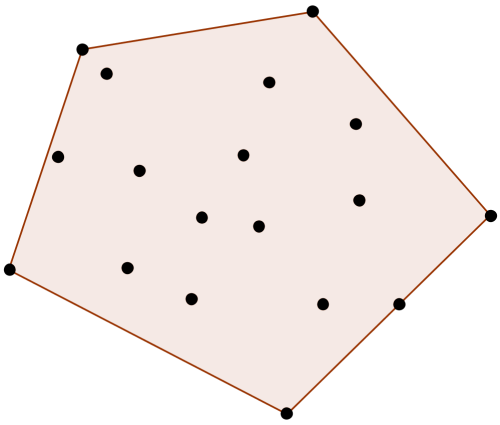
Oscar Esaú Peralta Rosales

Objetivo

- Implementar el algoritmo de Jarvis march para un conjunto de puntos. Presentar: Gráfica de puntos incluyendo su envolvente convexa; archivo conteniendo los puntos pertenecientes a la envolvente y mostrar por pantalla el tiempo de ejecución.
- Implementar el algoritmo de Graham, para envolver los objetos dentro de una imagen de entrada. Presentar: Archivo conteniendo los puntos pertenecientes a la envolvente, imagen de salida incluyendo su envolvente convexa y mostrar por pantalla el tiempo de ejecución
- Generar una discusión sobre las diferencias entre ambos algoritmos de CH.

Envolvente convexa

Sea un conjunto de puntos X en un espacio, aquel subconjunto de X con cardinalidad mínima que contenga a todos los puntos de X en su representación espacial se le conoce como envolvente convexa.



Algoritmos

Jarvis's March

Se basa en el cálculo de los ángulos formados por un punto pivote de X el cual debe pertenecer al borde contra todos los demás. Se encuentra el primer punto pivote; aquel con las coordenadas en x o en y más pequeña o grande (en $2D$). Luego se procede a calcular los ángulos formados con los demás puntos y aquel punto que forme el ángulo más chico es tomado como nuestro nuevo pivote. El procedimiento se repite hasta llegar encontrar que el nuevo pivote es el pivote inicial.

Para una implementación mas sencilla, se usa el producto cruz formado por una triada de puntos, si se conserva el orden en el que se hacen las operaciones podemos usar los signos de dicho producto para saber si un punto está dentro del polígono o forma parte de la envoltura.

```
points[] <- Vector de puntos iniciales
hull[] <- Vector de que contendra los puntos de la envolvente convexa
start_point <- Encontrar algún punto más alejado en X o Y

hull.add(start_point) // Agregar el primer punto de la envolvente

current_point = start_point

while True:
    target_point = points[0]
    for point in points[1:]:
        if current_point == point: // El mismo punto
            continue
        // Producto punto de los vectores formados por puntos
        // (current_point, target_point) y (current_point, point)
        cross_v = cross(current_point, target_point, point)
        if cross_v == 0: // Collinear
            continue // Se puede manejar el caso para aceptar colineares
        if cross_v > 0: // El punto está a la derecha del actual
            next_point = point

        if next_point == start_point: // Se ha dado la vuelta
            break

        hull.add(next_point)
        current = next_point

return hull
```

Graham's Scan

A diferencia del algoritmo anterior, este algoritmo ordena los puntos con respecto al ángulo formado con el primer pivote elegido. Lo que permite descartar puntos que ya se sabe que están dentro de la envolvente y así evitar realizar operaciones nuevamente con estos.

```
points[] <- Vector de puntos iniciales
hull[] <- Vector de que contendra los puntos de la envolvente convexa
start_point <- Encontrar algún punto más alejado en X o Y

hull.add(start_point) // Agregar el primer punto de la envolvente

// Ordenar los puntos con respecto al ángulo formado con el pivote, orden
ascendete
// excepto el primero
sort(points+1, points + points.size())

// Inicializamos los primeros tres puntos del poligono convexo
hull.push(points[points.size-1]);
hull.push(points[0]);
hull.push(points[1]);

index = 2
```

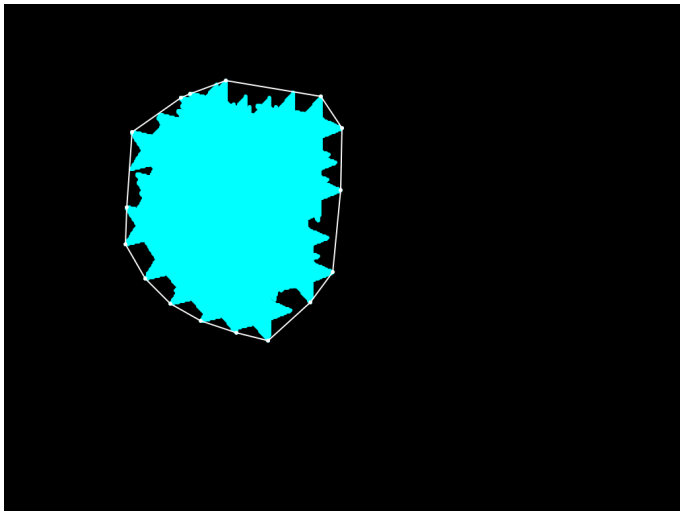
```
while(index < points.size):
    last = hull.size -1
    // Producto cruz
    cross_v = cross(hull[last-1], hull[last], points[index])
    if cross_v < 0: // Forma parte de la envolvente
        hull.push(points[index])
    else:
        hull.pop()

return hull
```

Implementación y Resultados

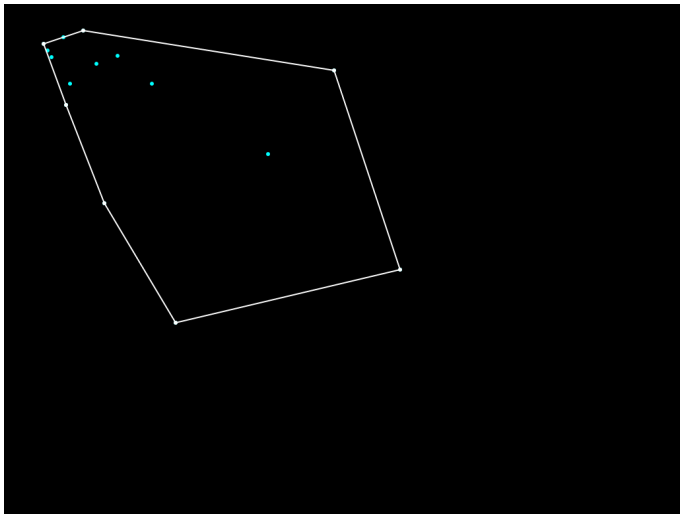
Se implementaron los algoritmos antes descritos para funcionar con:

1. Un conjunto de puntos predefinido
2. Generación aleatoria de n puntos
3. Una imagen binaria formato pgm

La siguiente imagen muestra el resultado obtenido de una imagen binaria:



Resultado obtenido para una serie de puntos



Conclusiones

La idea principal de ambos algoritmos es muy similar, se usan los ángulos formados desde la perspectiva de algún punto fijo para saber si el punto forma parte o no de la envoltura convexa.

La principal deferencia del Algoritmo de Jarvis's March a el algoritmo de Graham's Scan es que este segundo ordena los puntos de menor a mayor con respecto al ángulo formado con el punto pivote elegido. Lo que le permite ser más eficiente al poder descartar y no volver a procesar puntos que ya han sido detectados como dentro de la envoltura convexa. Así, el algoritmo Jarvi's March tienen una complejidad de $O(n^2)$ con n como el número de puntos, y el algoritmo de Graham's Scan tiene una complejidad de $o(nlog(n))$.