

Wyznaczanie otoczki wypukłej

Sprawozdanie z laboratorium 13 – Piotr Sarna LK1

Cel ćwiczenia

Podczas zajęć zapoznaliśmy się ze sposobami rozwiązywania problemów geometrycznych. Jednym z nich jest Algorytm Jarvisa.

Wstęp teoretyczny

Algorytm Jarvisa służy do wyznaczenia punktów, które połączone ze sobą stanowią wielokąt okalający wszystkie pozostałe punkty w zbiorze (jest „otoczką wypukłą” zbioru).

Punkty posiadają współrzędne (X, Y) reprezentujące ich położenie na płaszczyźnie.

W algorytmie możemy wykorzystać iloczyn wektorowy, do oszacowania wzajemnego położenia punktów. Dla trzech punktów a, b, c , które tworzą wektory ab oraz ac obliczamy ich iloczyn wektorowy i patrzymy na znak iloczynu:

- > 0 – skręt w lewo, punkt c po lewej stronie
- < 0 – skręt w prawo, punkt c po prawej stronie
- $= 0$ – punkty leżą na jednej linii

Opis algorytmu

Algorytm rozpoczynamy od znalezienia najniżej położonego punktu z całego zbioru, oznaczamy go jako P0:

```
Point P0 = pointVector[0];

for (int i = 1; i < pointCount; i++) {
    if (pointVector[i].Y < P0.Y) {
        P0 = pointVector[i];
    }
}
```

Następnie zaczynając z punktu P0, iteracyjnie wybieramy kolejny punkt, który tworzy największy kąt w prawo względem poprzedniego wektora. Aby porównać, który punkt daje największy skręt, używamy iloczynu wektorowego trzech punktów:

- Poprzedniego punktu r
- Obecnego kandydata pointVector[tempPointIndex]
- Nowego kandydata

```
do {
    convexHull.push_back(r);
    int tempPointIndex = -1;

    for (int i = 0; i < pointCount; i++) {
        const Point& t = pointVector[i];
        if (arePointsEqual(r, t)) continue;

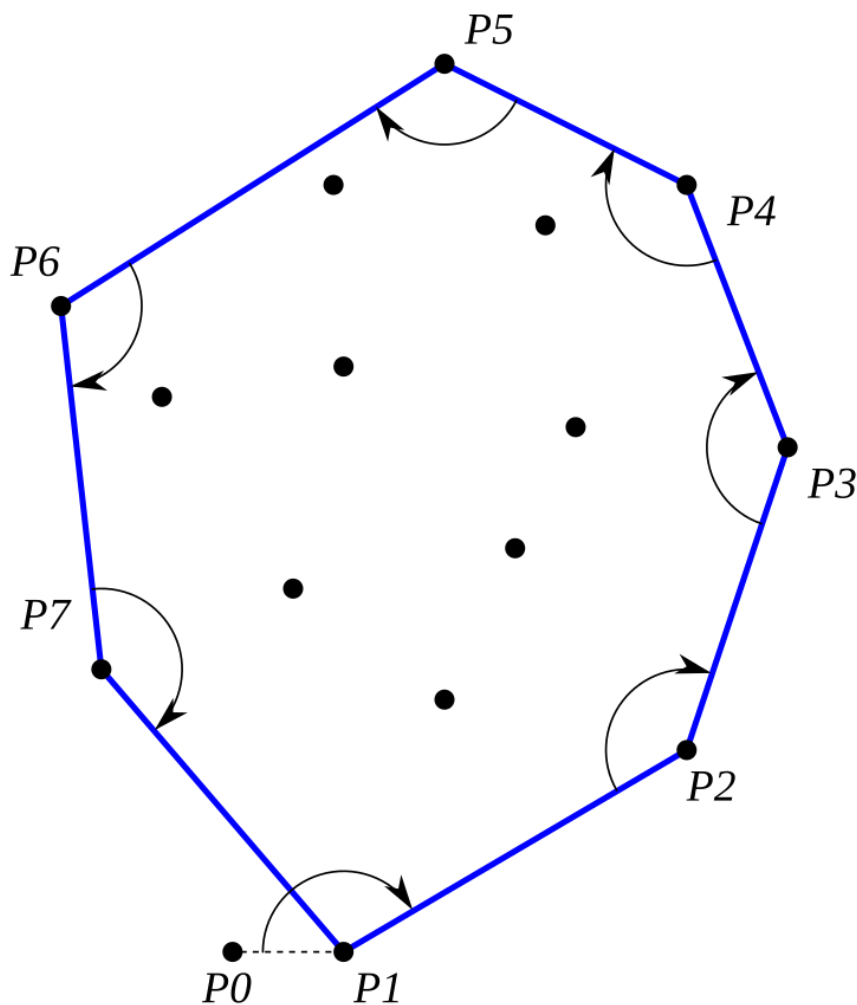
        if (tempPointIndex == -1 || vectorProduct(r, pointVector[tempPointIndex], t) < 0) {
            tempPointIndex = i;
        }
    }

    r = pointVector[tempPointIndex];
} while (!arePointsEqual(r, P0));
```

Jeżeli iloczyn wektorowy jest mniejszy od zera, oznacza to że nowo badany punkt jest jeszcze mocniej wysunięty, co za tym idzie tworzy jeszcze większy kąt. Aktualizujemy tempPointIndex.

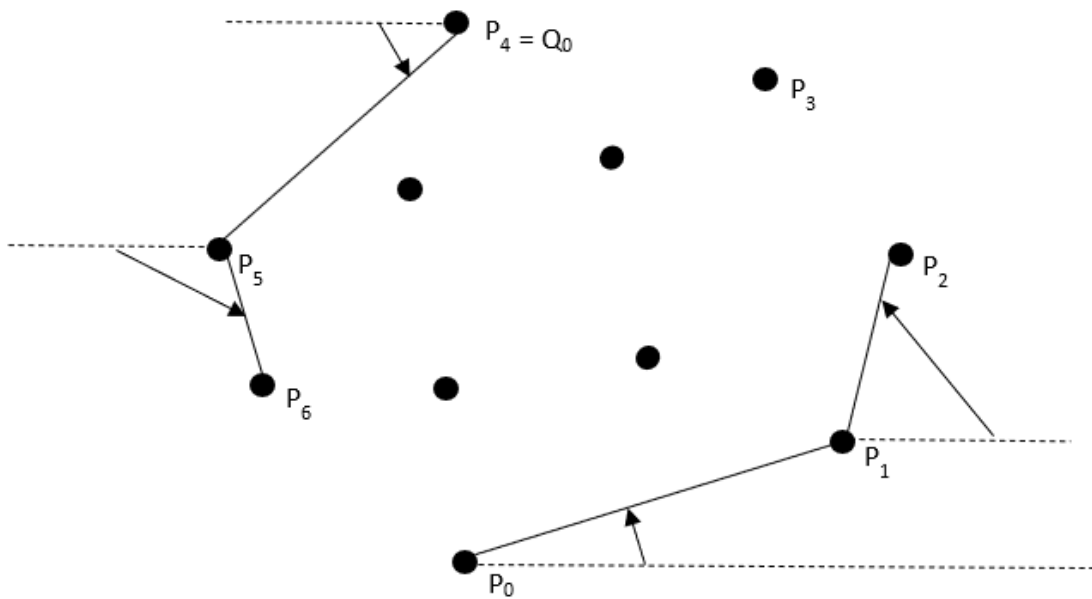
Powtarzamy iterację aż do powrotu do początkowego punktu P_0 – oznacza to, że wyznaczyliśmy pełną otoczkę.

Gotową wyznaczoną otoczkę poprzez nasz algorytm można zwizualizować w następujący sposób:



Źródło: https://pl.wikipedia.org/wiki/Plik:Jarvis_algorithm2.svg

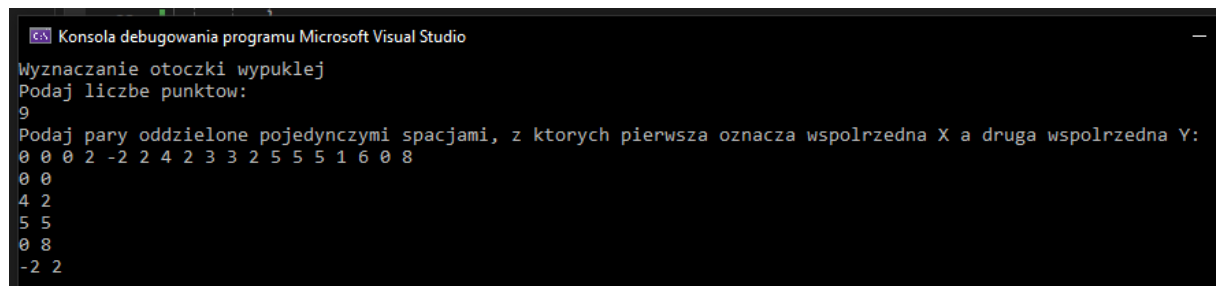
Alternatywną wersją algorytmu jest wyznaczenie najniżej i najwyżej położonych punktów na płaszczyźnie. Następnie porównując kąty tworzone odpowiednio z dodatnią oraz ujemną półosią OX wyznaczamy największe kąty. Przesuwamy się najpierw z najniższego wierzchołka P_0 do najwyższego Q_0 a następnie analogicznie w drugą stronę.



To podejście niesie za sobą jednak następujące wady:

- Funkcja $\text{atan2}(y, x)$ zwraca wynik w przedziale $[-\pi, \pi]$. Taka komplikacja nie występuje przy podejściu z iloczynem wektorowym.
- Należy osobno rozpatrywać przypadki z punktami współliniowymi – funkcja $\text{atan2}()$ zwróci tę samą wartość.

Prezentacja działania implementacji w C++



```
Konsola debugowania programu Microsoft Visual Studio
Wyznaczanie otoczki wypukłej
Podaj liczbę punktów:
9
Podaj pary oddzielone pojedynczymi spacjami, z których pierwsza oznacza współrzędna X a druga współrzędna Y:
0 0 0 2 -2 2 4 2 3 3 2 5 5 5 1 6 0 8
0 0
4 2
5 5
0 8
-2 2
```

Wnioski

Algorytm cechuje złożoność obliczeniowa $O(n * h)$, gdzie n oznacza liczbę punktów zbioru przechowanych w `pointVector` a h oznacza liczbę punktów otoczki.

Bibliografia

https://pl.wikipedia.org/wiki/Algorytm_Jarvisa