Przemysław Rola, Juliusz Wasieleski
Informatyka II rok, grupa 1

Algorytmy geometryczne, projekt nr 5 - porównanie triangulacji wielokątów prostych - sprawozdanie

1. Triangulacja wielokąta prostego przy użyciu triangulacji wielokątów monotonicznych

1.1 Program

Program wymaga zaimportowania bibliotek:

- matplotlib - do rysowania wyników
- numpy – do obliczeń numerycznych
- sortercontainers – do struktury SortedList
- json

Struktura przechowywania Triangulacji:

Dwuelementowa krotka w której:

- pierwszy element to wejściowa lista punktów (punkty to dwuelementowe krotki współrzędnych)
- drugi element to lista trzyelementowych krotek których elementami są indeksy punktów z listy punktów

Dodatkowo korzystam z klasy Point:

- x – współrzędna x
- y – współrzędna y
- chain – łańcuch na którym znajduje się punkt
- index – indeks trzymany na stałe z wejściowej listy punktów
- index_tmp – pole na tymczasowy indeks podczas triangulacji wielokąta monotonicznego
- type – typ punktu

Algorytm znajduje się w funkcji triangulate która zwraca instancję klasy plot zadeklarowane w kodzie przykładowego narzędzia graficznego oraz strukturę do przechowywania triangulacji (opisaną wyżej).

Funkcja *triangulate* używa funkcji pomocniczych:

- ***read_from_file*** – funkcja czytająca wielokąt z pliku orzymuje na wejściu ścieżkę do pliku i zwraca listę krotek będących współrzędnymi punktów
- ***split_int_monotonic*** – funkcja dzieląca wielokąt na wielokąty monotoniczne, otrzymuje listę współrzędnych punktów, przerabia je na instacje klasy Punkt a następnie korzystając z algorytmu zamiatania wyznacza które krawędzie powinniśmy dodać aby uzyskać podział na wielokąty monotoniczne. Mając te krawędzie funkcja zwraca listę punktów (instacji klasy Point), listę linii (po współrzędnych, są potrzebne tylko do prezentacji graficznej) oraz listę list, których elementami są indeksy punktów z których zbudowane są wielokąty monotniczne
- ***triangulate_monotonic*** - funkcja wykonująca triangulację wielokąta który już jest monotoniczny, ustala jakie krawędzie powinniśmy dodać aby otrzymać triangulację i zwraca listę tych krawędzi po indeksach

- ***make_triangle* –** funkcja przerabiająca powyższe indeksy krawędzi na listę trzyelementowych krotek, których elementami są indeksy punktów tworzących dany trójkąt

1.2 Użytkownik

Aby otrzymać triangulację wielokąta, należy mieć przygotowany plik wejściowy z współrzędnymi punktów ułożonymi przeciwnie do wskazówek zegara. Następnie uruchamiamy funkcję *triangulate(read_from_file('ścieżka_do_pliku.txt')),* w której podajemy nazwę pliku.

1.3 Opis problemu

Problem triangulacji dla wielokątów można zawsze rozwiązać rekurencyjnie w czasie O(n^2), ale istnieją dużo lepsze algorytmy triangulujące wielokąty nawet w czasie O(n). Dużo łatwiejszy jest algorytm triangulujący wielokąty monotoniczne, ale żeby sprawić że nasz algorytm jest bardziej uniwersalny wystarczy wcześniej podzielić wielokąt na takie, które są monotoniczne.

To zmusza nas do zrobienia pewnych założeń na wielokąty montoniczne:

- Wielokąt prosty nazywamy ściśle monotonicznym względem prostej l (wyznaczającej kierunek monotoniczności), gdy jego brzeg można podzielić na dwa spójne łańcuchy takie, że dowolna prosta l' prostopadła do l przecina każdy z łańcuchów w co najwyżej jednym punkcie.
- W naszym wypadku, gdy zakładamy, że dzielimy na wielokąty y – monotoniczne zakładamy, że nie ma dwóch punktów o tej samej współrzędnej y-owej

PODZIAŁ NA WIELOKĄTY MONOTONICZNE- Algorytm podziału na wielokąty monotoniczne opiera się na algorytmie zamiatania i w zależności od typu punktu (początkowy, końcowy, dzialący, łączący, prawidłowy) wykonuje odmienne działania. Tak jak to w algorytmach zamiatania mamy strukturę stanu oraz strukturę zdarzeń:

- **Struktura zdarzeń** – lista L uporządkowanych malejąco względem y-ów wierzchołków P.
- **Struktura stanu** – drzewo poszukiwań binarnych T przechowujące w liściach ciąg aktualnie przecinanych przez miotłę krawędzi ograniczających wielokąt P z lewej strony wraz z dowiązaniami do ich pomocników.

W naszej implementacji zamiast drzewa poszukiwań binarnych została użyta struktura SortedList

TRIANGULACJA WIELOKĄTA MONOTONICZNEGO – Polega na wykonywaniu następujących kroków:

• Określamy lewy i prawy łańcuch wielokąta względem kierunku monotoniczności
• Porządkujemy wierzchołki wzdłuż kierunku monotoniczności
• Wkładamy dwa pierwsze wierzchołki na stos.
• Jeśli kolejny wierzchołek należy do innego łańcucha niż wierzchołek stanowiący szczyt stosu, to możemy go połączyć ze wszystkimi wierzchołkami na stosie. Na stosie zostają dwa wierzchołki, które były „zamiatane" ostatnie.
• Jeśli kolejny wierzchołek należy do tego samego łańcucha co wierzchołek ze szczytu stosu, to analizujemy kolejne trójkąty, jakie tworzy dany wierzchołek z wierzchołkami zdejmowanymi ze stosu:
    • Jeśli trójkąt należy do wielokąta, to usuwamy wierzchołek ze szczytu stosu
    • w przeciwnym przypadku umieszczamy badane wierzchołki na stosie

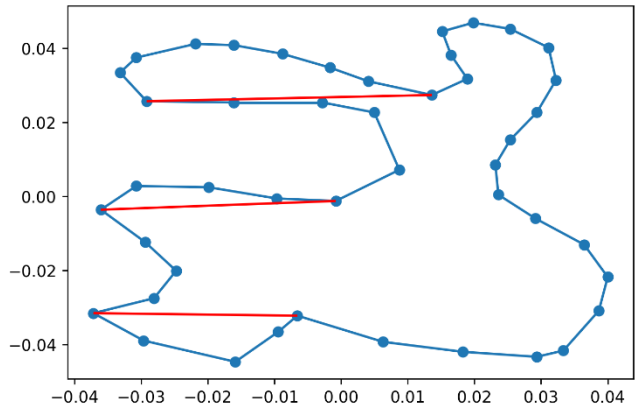
Podstawową pomocą matematyczną są dwie funkcje:

orient(a,b,c) - pozwala stwierdzić, czy punkt c znajduje się ponad czy poniżej prostej tworzonej przez punkty a, b,
zwracając odpowiednio wartość większą, mniejszą od zera, lub 0 w przypadku jeśli leży na jej linii.

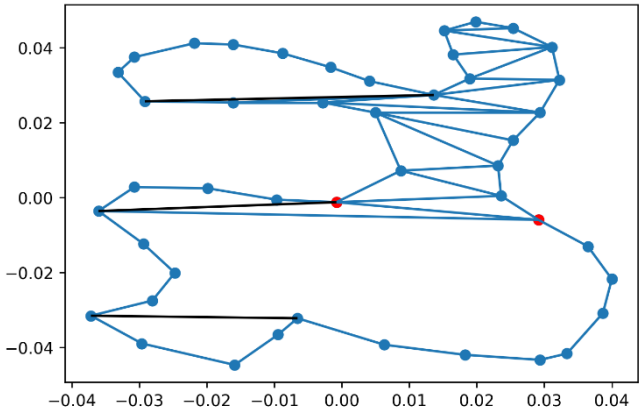
$$orient(a,b,c) = \begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{vmatrix}$$

1.4 Testy

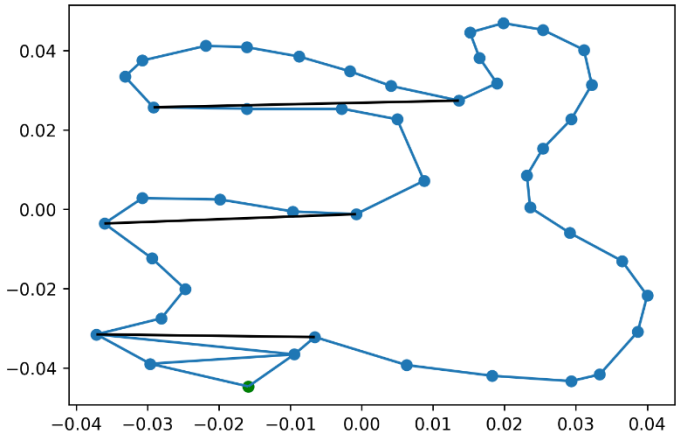
Podział na wielokąty monotoniczne



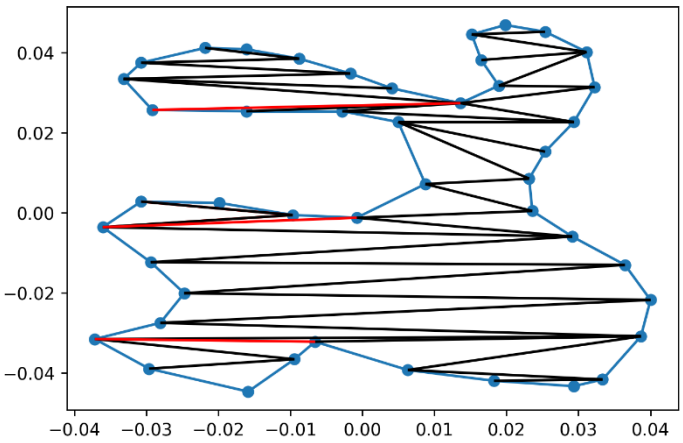
Przykładowy moment triangulacji jednego z wielokątów



Triangulacja kolejnego



Ostateczny efekt podziału i triangulacji



2. Triangulacja Delunaya wielokąta prostego

2.1 Program

Program wymaga zaimportowania bibliotek:

- matplotlib - do rysowania wyników
- deque - jako struktura kolejki

Struktura przechowywania Triangulacji: Half Edge
Składa się ona z 3 struktur:

Point:

- x,y - współrzędne x-owe i y-owe
- edges - zbiór półkrawędzi w których punkt jest początkiem

Edge:

- origin - wierzchołek początkowy
- twin - półkrawędź skierowana przeciwnie pomiędzy tymi samymi punktami
- next - następna krawędź w ścianie
- prev - poprzednia krawędź w ścianie
- face - ściana przy której jest krawędź

Face:

- edges - krawędzie składające się na ścianę
- visited - pole potrzebne przy wyszukiwaniu punktów, zaznaczający czy byliśmy już w tej krawędzi

Dla każdej z powyższych klas została zdefiniowana funkcja hashująca i równości, aby działanie zbiorów przechodziło pomyślnie.

Algorytm znajduje się w klasie Delunay(), przyjmującą przy utworzeniu listę.
W niej zdefiniowana jest funkcja start() która uruchamia triangulację i pokazuje wynik.

Podstawowe wykorzystywane funkcje w klasie Delunay(L):

Przy uruchomieniu klasy zostaje utworzona baza triangulacji. Zostają znalezione zewnętrzne punkty, a następnie utworzony zostaje prostokąt o punktach odsuniętych od maksimów.

.addPoints() - dodaje wszystkie punkty do triangulacji.
Dla każdego punktu P:

- znajduje ścianę w której się znajduje,
- sprawdza wszystkie ściany, których dodanie P spowoduje zmianę krawędzi
- łączy ze wszystkimi wierzchołkami z powyższych ścian tworząc nowe, zawierające dany punkt

.regainEdges() - przywraca do triangulacji krawędzie wielokąta które w trakcie programu mogły zostać zatracone.

- dla każdej krawędzi z początkowego wielokąta sprawdza czy istnieje taka w triangulacji.

Jeśli powyższy warunek nie jest spełniony, to wykonywane są kroki:

- dla brakującej krawędzi znajdywane są krawędzie kolizyjne, i zostają usuwane poprzez przełączanie przekątnych w dwóch trójkątach

.eraseOuterLayer() - funkcja usuwa „otoczkę" zbędnych trójkątów powstałą wokół wielokąta.

- znajdowane są ściany zawierające się wewnątrz wielokąta
- pozostałe zostają usunięte, a razem z nimi wierzchołki pomocnicze

.start() - uruchamia w podanej kolejności wszystkie powyższe funkcje, tworząc pełną triangulację.

2.2 Użytkownik

Aby otrzymać triangulację wielokąta, należy mieć przygotowany plik. Następnie uruchamiamy funkcję *Triangulate(steps),* w której podajemy nazwę pliku.

Istnieje opcja włączenia kroków programu, odpowiadając twierdząco na zapytanie programu.

2.3 Opis problemu

Triangulacja Delunaya wielokąta prostego to przypadek triangulacji Delunaya z ograniczeniami.

Triangulacja Delunaya to metoda triangulacji która gwarantuje nam, że każde trójkąty będą w optymalny sposób rozmieszczone w siatce, tzn. w okrąg opisany na trójkącie nie wchodzą żadne inne wierzchołki niż te które są w danym trójkącie.

Oznacza to założenia:

- Nie możemy dodawać ani usunąć z triangulacji żadnego wierzchołka który znajdował się w wielokącie początkowym.
- Każda krawędź z wejściowego wielokąta musi być reprezentowana w końcowej triangulacji.
- Poprzez ograniczenia czasami może nie zostać spełniony warunek nie zachodzenia na siebie obcych punktów do okręgu trójkąta, jednak ze względu na drugie ograniczenie nie dodawania nowych punktów jest to niezbywalny kompromis.

WSTĘPNA TRIANGULACJA - Algorytm tworzenia wstępnej triangulacji działa na bazie rekurencyjnej:

- Mając triangulację T_n n-elementową, dokładamy punkt P tworząc T_{n+1} poprzez znalezienie wszystkich ścian w których punkt P wchodzi w okrąg opisany. Następnie każda z tych ścian zostaje zamieniona na taką z wierzchołkiem w punkcie P
- Aby algorytm za każdym razem miał pełną triangulację na początku, stworzone są tymczasowe wierzchołki zewnętrzne, które okrywają prostokątnym polem cały wielokąt. Na końcu zostają one usunięte.

ODZYKIWANIE - Następnie przystępuje do odzyskania krawędzi, poprzez zamienianie przekątnych w parach trójkątów, których obecne przekątne zaburzają krawędź pierwotną.

USUWANIE- Na koniec poprzez znalezienie wszystkich wewnętrznych ścian zostaje usunięty zewnętrzna część powstała na skutek zewnętrznych wierzchołków pomocniczych oraz połączeń poza geometrią wielokąta.

Podstawową pomocą matematyczną są dwie funkcje:

orient(Edge,P) - pozwala stwierdzić, czy punkt znajduje się ponad czy poniżej krawędzi Edge,
zwracając odpowiednio wartość większą, mniejszą od zera, lub 0 w przypadku jeśli leży na jej linii.

$$orient(Edge, P) = \begin{vmatrix} Edge.origin_x & Edge.origin_y & 1 \\ Edge.twin.origin_x & Edge.twin.origin_y & 1 \\ P_x & P_y & 1 \end{vmatrix}$$

ta funkcja jest wykorzystywana do określenia ściany na której znajduje się punkt, czy też w późniejszych etapach czy krawędź przecina brakującą pomiędzy punktami.

pointInCircle(face, p) - funkcja pozwalająca na określenie, czy punkt wpada w okrąg opisany na trójkącie *face*

$$\text{pointInCircle}(face, P) = \begin{vmatrix} A_x - D_x & A_y - D_y & \left(A_x{}^2 - D_x{}^2\right) + \left(A_y{}^2 - D_y{}^2\right) \\ B_x - D_x & B_y - D_y & \left(B_x{}^2 - D_x{}^2\right) + \left(B_y{}^2 - D_y{}^2\right) \\ C_x - D_x & C_y - D_y & \left(C_x{}^2 - D_x{}^2\right) + \left(C_y{}^2 - D_y{}^2\right) \end{vmatrix}$$

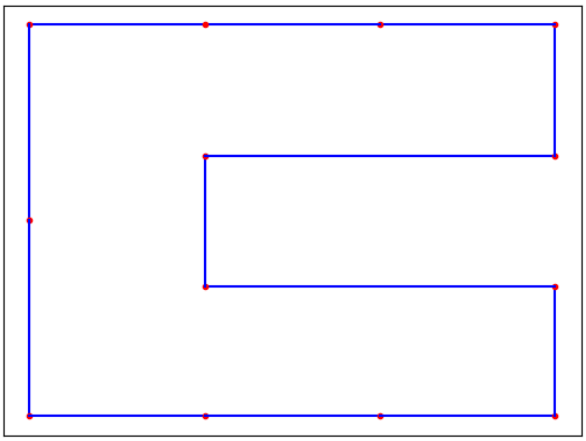
gdzie A,B i C to punkty ściany *face.*

Wynik pozytywny oznacza należenie do okręgu, ujemny o położeniu poza, a 0 o położeniu na brzegu.

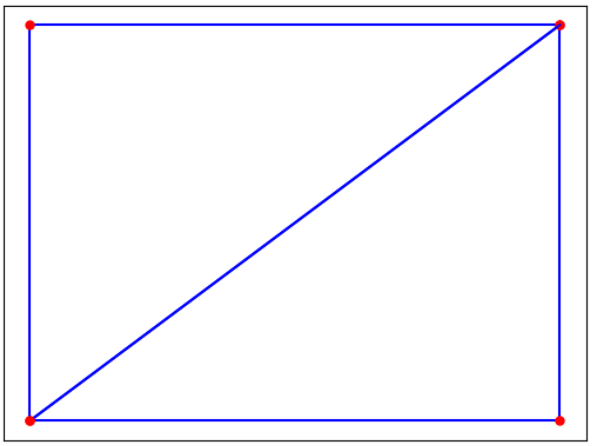
2.4 Testy

Pierwszym testem był test poddający próbie wszystkie kroki algorytmu. Zawiera on wklęsły fragment, powodujący iż w trakcie wstępnej triangulacji zostanie zatracona krawędź pomiędzy punktami.

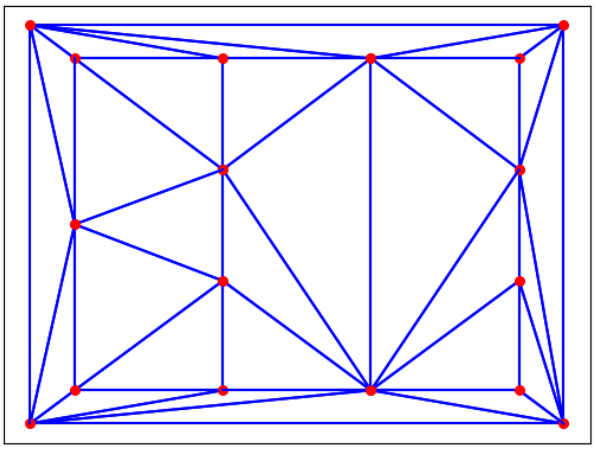
Wielokąt wejściowy



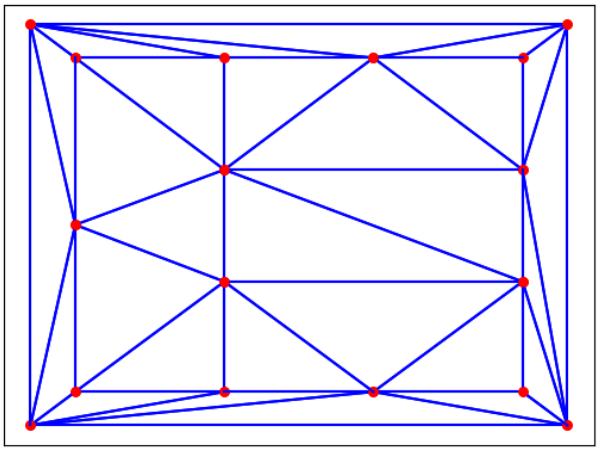
Pole Startowe



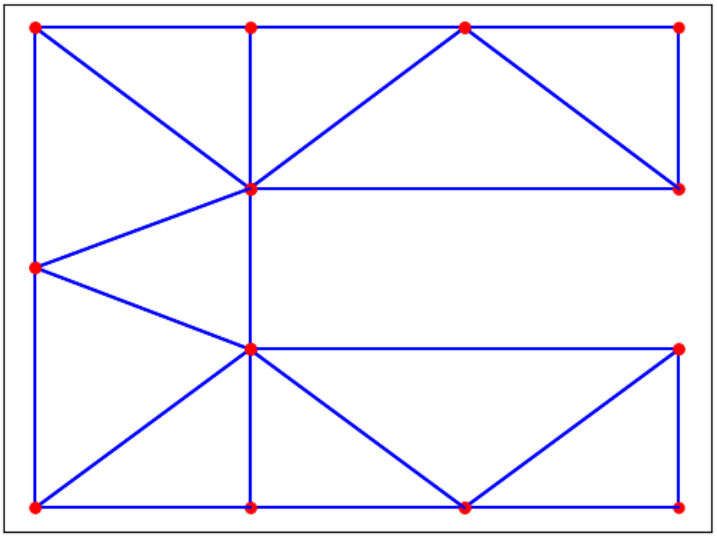
Wstępna Triangulacja



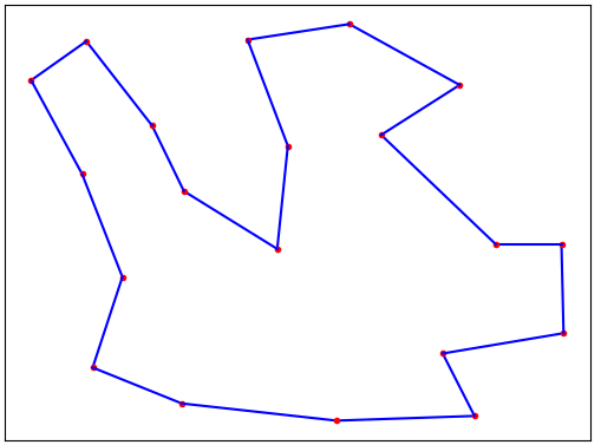
Odzyskiwanie krawędzi

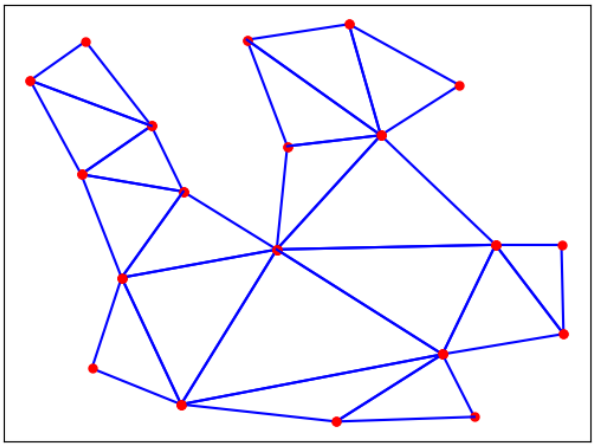


Końcowy wynik



Kolejnym testem było sprawdzenie czy faktycznie trójkąty powstałe są jak najlepiej umiejscowione.
Aby sprawdzić to napisany został „test_!”, w którym jest wiele schodzących krawędzi z góry na dół, gdzie zwykła monotoniczna triangulacja zrobiłaby spłaszczone trójkąty.





Źródła Triangulacji Delunaya:
Wykład „Algorytmy Geometryczne” AGH dr. Barbary Głut
Wikipedia