Algorytmy geometryczne

Projekt nr 7

Triangulacja wielokąta prostego – porównanie metod

Dawid Zawiślak i Mateusz Kochelski

gr. Czw. 13:00 A

1. Triangulacja wielokąta prostego przy użyciu triangulacji wielokątów monotonicznych

1.1) Opis techniczny programu

Program korzysta z następujących modułów:

- **Visualizer** od KN BIT – wizualizacja wyników triangulacji oraz jej krokowego procesu
- **Matplotlib** – zadawanie nowych wielokątów za pomocą myszki
- **Sortedcontainers** – implementacja wzbogaconego drzewa przeszukiwań binarnych użytego do przechowywania stanu miotły

Dodatkowo, aby interaktywne widgety matplotlib działały poprawnie musimy mieć zainstalowany pakiet **ipympl**

Wynikowa triangulacja przechowywana jest w postaci krotki, gdzie pierwszy element to lista par współrzędnych punktów zadanego wielokąta w porządku przeciwnym do ruchu wskazówek zegara, a drugi element to lista trójek zawierających indeksy wierzchołków składających się na trójkąty wynikowej triangulacji.

Możliwe jest zapisanie danej triangulacji do pliku w formacie za pomocą odpowiedniej funkcji:

n – liczba wierzchołków

x_1 y_1

x_2 y_2

…

x_n y_n

$t_{1.1}$ $t_{1.2}$ $t_{1.3}$

$t_{2.1}$ $t_{2.2}$ $t_{2.3}$

…

$t_{m.1}$ $t_{m.2}$ $t_{m.3}$

Funkcja triangulująca zwraca również listę par indeksów przekątnych dodawanych podczas triangulacji.

Możliwe jest wczytywanie wielokątów zapisanych w plikach pod postacią:

x_1 y_1

x_2 y_2

...

x_n y_n

Lub zadawanie ich za pomocą myszki z możliwością ich zapisania do plików.

Triangulacja odbywa się w funkcji **triangulation_general**, jest również dostępny jej wariant z wizualizacją **triangulation_general_vis**.

Funkcja **triangulation_general** używa następujących funkcji pomocniczych:

- **split_into_monotone_fragments** – funcja dzieląca wielokąt na monotoniczne wielokąty składowe
- **get_point_side** – zwracająca tablice słownikową określającą w którym łańcuchu leży każdy punkt oraz indeks punktu końcowego
- **sort_points_by_y** – zwracająca indeksy punktów wielokąta w porządku punktów posortowanych malejąco po współrzędnej Y
- **is_neighbour** – sprawdzająca czy dana krawędź należy do pierwotnego wielokąta
- **inside_polygon** – sprawdzająca czy trójkąt utworzony przez dane 3 punkty leży wewnątrz wielokąta
- **is_y_monotonic** – sprawdza czy wielokąt jest monotoniczny i czy konieczne jest zastosowanie algorytmu dzielenia na składowe
- **color_vertex** – klasyfikująca wierzchołki
- **orient** – funkcja licząca wyznacznik 3x3

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

Dostępne są również funkcje użytkowe:

- **save_polygon_to_file** – zapisuje wielokąt do pliku
- **read_polygon_from_file** – wczytuje wielokąt z pliku
- **save_triangulation_to_file** – zapisuje triangulacje pod opisaną wyżej postacią

Za wartość tolerancji dla 0 zostało przyjęte 1e-12.

1.2) Użytkownik:

Schemat obsługi wygląda następująco:

```
polygon_to_triangulate = read_polygon_from_file("polygon.txt")

polygon, diags, triangles = triangulation_general(polygon_to_triangulate)

save_triangulation_to_file(polygon, triangles, "triangulation_result.txt")
```

Na początku wczytujemy wielokąt zapisany w pliku. Następnie przekazujemy go do funkcji triangulacji (mamy do wybory wariant standardowy oraz tez z wizualizacją). Otrzymujemy wynik triangulacji pod 2 postaciami. Możemy zapisać otrzymaną triangulacje do pliku pod opisaną wyżej postacią.

1.3) Sprawozdanie:

Opis problemu:

Naszym zadaniem jest odnalezienie triangulacji zadanego wielokąta. Dodatkowo zakładamy, że:

- Triangulacja wielokąta prostego to podział wielokąta na trójkąty
- Wierzchołki każdego trójkąta należą do pierwotnego wielokąta
- Trójkąty stykają się ze sobą tylko ścianami lub wierzchołkami, oraz zawierają się w pierwotnym wielokącie
- Dla każdego wielokąta istnieje triangulacja
- Może istnieć wiele triangulacji, ale będą one składać się z takiej samej ilości trójkątów
- Żadne 2 wierzchołki wielokąta wejściowego nie będą miały tej samej współrzędnej Y

Triangulacja wielokątów może być rozwiązana rekurencyjnie złożonością czasową O(n^2), jednak istnieją algorytmy triangulacji wielokątów, które działają nawet w czasie O(n). Algorytm triangulacji dla wielokątów monotonicznych jest znacznie prostszy i działa w czasie O(n), możemy użyć go również do triangulacji wielokątów niekoniecznie monotonicznych, wystarczy wcześniej podzielić wielokąt na fragmenty, które są monotoniczne.

Wielokąt monotoniczny względem osi y nazywamy y-monotonicznym, jeśli idąc z najwyższego wierzchołka do najniższego wzdłuż lewego (lub prawego) łańcucha zawsze poruszamy się w dół lub poziomo, nigdy w górę.

W naszym algorytmie zakładamy również, że wierzchołki wielokąta parami mają różne współrzędne y.

I. Podział wielokąta na fragmenty monotoniczne

Najpierw ustalamy kategorie wszystkich wierzchołków.

Rozkład wielokąta na wielokąty monotoniczne jest realizowany poprzez zastosowanie algorytmu zamiatającego. Proces ten polega na przechodzeniu przez kolejne wierzchołki wielokąta i dokonywaniu podziału na podstawie ich kategorii (wierzchołki początkowe, końcowe, łączące, dzielące i prawidłowe). Miotła porusza się od góry do dołu, przechodząc przez poszczególne wierzchołki i zatrzymując się przy każdym z nich. Dodatkowo, w ramach tego algorytmu, wykorzystywana jest struktura stanu (**SortedSet**), która przechowuje wszystkie krawędzie aktualnie przecinane przez miotłę posortowane względem współrzędnej X.

Definicje:

- v_i – i-ty wierzchołek z ciągu wierzchołków ułożonych przeciwnie do wskazówek zegara
- e_i – krawędź łącząca wierzchołki v_i v_{i+1}
- helper(e) - najniższy wierzchołek powyżej miotły taki, że odcinek łączący ten wierzchołek z e leży wewnątrz wielokąta (zainicjalizowany jako górny wierzchołek każdej krawędzi)
- e_j – krawędź łeżąca bezpośrednio na lewo od wierzchołka v_i
- T – struktura przechowująca stan miotły (aktualnie przecinane krawędzie posortowane po współrzędnej X od prawej do lewej)

Algorytm zatrzymując się w kolejnych punktach obsługuje je w odmienny sposób ze względu na kategorie tych wierzchołków:

- Wierzchołek początkowy:
  - Dodaj e_i do T
  - helper(e_i) = v_i

- Wierzchołek końcowy:
  - Jeśli helper(e_{i-1}) jest wierzchołkiem łączącym połącz v_i z helper(e_{i-1})
  - Usuń e_{i-1} z T

- Wierzchołek dzielący:
  - Jeśli helper(e_{i-1}) jest wierzchołkiem łączącym połącz v_i z helper(e_{i-1})
  - Usuń e_{i-1} z T
  - Znajdź w T krawędź e_j
    (leżąca bezpośrednio po lewo od v_i)
  - Jeśli helper(e_j) jest wierzchołkiem łączącym połącz v_i z helper(e_j)
  - helper(e_j) = v_i

- Wierzchołek łączący:
  - Znajdź w T krawędź e_j
    (leżąca bezpośrednio po lewo od v_i)
  - Połącz v_i z helper(e_j)
  - helper(e_j) = v_i
  - Dodaj e_i do T
  - helper(e_i) = v_i

- Wierzchołek prawidłowy:
  Jeśli wnętrze wielokąta leży na lewo od v_i :
  - Znajdź w T krawędź e_j
    (leżąca bezpośrednio po lewo od v_i)
  - Jeśli helper(e_j) jest wierzchołkiem łączącym połącz v_i z helper(e_j)
  - helper(e_j) = v_i

  W przeciwnym wypadku :
  - Jeśli helper(e_{i-1}) jest wierzchołkiem łączącym połącz v_i z helper(e_{i-1})
  - Usuń e_{i-1} z T oraz dodaj e_i do T
  - helper(e_i) = v_i

II. Triangulacja każdej składowej monotonicznej

Posortowanie punktów po współrzędnej Y

Określenie dla każdego punktu w jakim łańcuch się znajduje

Wstawienie 2 pierwszych wierzchołków na stos

Przechodzimy pętlą po pozostałych posortowanych wierzchołkach:

a. Jeśli analizowany wierzchołek znajduje się w innym łańcuchu niż ten odłożony na górę stosu. Wtedy możemy dodać przekątne pomiędzy analizowanym wierzchołkiem i każdym innym wierzchołkiem znajdującym się na stosie. Po zakończeniu procedury na stos odkładane są wierzchołek który poprzednio był na górze stosu oraz analizowany przez nas wierzchołek.

b. W przeciwnym wypadku ściągamy 2 wierzchołki ze stosu i sprawdzamy, czy trójkąt utworzony z nich i analizowanego wierzchołka leży wewnątrz wielokąta. Jeśli tak dodajemy przekątną pomiędzy aktualnie analizowanym wierzchołkiem i wierzchołkiem ściągniętym ze stosu jako drugi. Proces powtarzamy dopóki można już dodać kolejnej przekątnej spełniając te warunki. W wypadku gdy utworzony trójkąt nie leży wewnątrz wielokąta odkładamy z powrotem 2 ściągnięte wierzchołki na stos i również dodajemy na stos aktualnie analizowany wierzchołek. Jeśli na stosie nie zostanie żaden wierzchołek odkładamy na niego ostatnio ściągnięty wierzchołek oraz aktualnie analizowany.

III. Wynikiem triangulacji jest suma triangulacji poszczególnych składowych

Wizualizacja zwraca końcowy efekt triangulacji oraz proces krokowy do plików (triangulation_result.png, triangulation_process.gif), a w notebooku widzimy dodatkowo wejściowy wielokąt, kategorie wierzchołków, podział na składowe monotoniczne i końcową triangulacje na wykresach.

Legenda dla gifu krokowej wizualizacji:

- zielone punkty – punkty na stosie

- niebieski punkt – aktualnie analizowany punkt

- niebieskie linie – podział na składowe monotoniczne

- przerywane linie czerwone/żółte – kolejne trójkąty triangulacji

2. Triangulacja Delaunaya z odzyskiwanie krawędzi i usuwaniem nadmiernych trójkątów.

Opis problemu :

Triangulacja Delaunaya to algorytm, który efektywnie dzieli przestrzeń płaszczyzny na trójkąty, gwarantując, że żadne z punktów nie znajduje się wewnątrz okręgu opisanego na trójkącie. W rezultacie uzyskujemy strukturę trójkątów, idealną do analizy i efektywnego przetwarzania danych geometrycznych.

Dodatkowe ograniczenia dotyczące triangulacji Delaunaya

- Triangulacja Delaunaya może być dostosowana do specyficznych wymagań poprzez dodanie dodatkowych ograniczeń.
- Pierwszym ograniczeniem jest wymaganie by krawędzie pierwotnego wielokąta znajdowały się w triangulacji
- Drugim ograniczeniem jest brak trójkątów które znajdują się poza pierwotnym wielokątem

Opis Algorytmu:

Triangulacja Delaunaya z ograniczeniami składa się z 3 kroków, które później zostaną dokładnie omówione.

1. Dokonywanie Triangulacji bazowej np. Metodą odwróceń Lawsona
2. Odzyskiwanie krawędzi pierwotnego wielokąta
3. Usunięcie zewnętrznych trójkątów

Opis Metody odwróceń Lawsona

Potrzebne struktury:

- Point przechowująca współrzędne x i y punktu
- Edge przechowująca indeksy punktów które są połączone krawędzią oraz referencje do trójkątów po obu stronach tej krawędzi oraz informacje czy jest to krawędź należąca do trójkąta zewnętrznego.
- Triangle przechowująca krawędzi i punkty z których składa się trójkąt, pole Visited potrzebne do wykonywania algorytmów przeszukiwań po trójkątach oraz Punkt będący środkiem tego trójkąta
- Słownik Tpoints mapujący indeks punktu na punkt (Point)

- Słownik Edges mapujący 2 indeksy punktów na krawędź (Edge) łączącą te punkty

Definicja krawędzi „Nielegalnej”

Oprócz szczególnych w których rozważamy krawędzi połączone z trójkątem zewnętrznym krawędź PiPj jest nielegalna wtedy i tylko wtedy kiedy punkt Pl znajduję wewnątrz okręgu utworzonego na punktach Pi,Pj,Pk gdzie Pk to zawsze punkt dopiero co wstawiany do triangulacji

Pierwszym krokiem w algorytmie jest stworzenie Zewnętrznego trójkąta o współrzędnych (-3M,-3M), (3M,0), (0,3M), gdzie M to maksymalna wartość bezwzględna pojedynczej współrzędnej w naszym zbiorze punktów.

Najpierw szukamy w którym z istniejących już trójkątów znajduję się punkt który chcemy dodać, następnie sprawdzamy czy krawędzie przeciwległe do punktu który dodaliśmy w trójkącie są legalne za pomocą warunku koła. Jeżeli nie są legalne to zamieniamy z trójkątem który leży po drugiej stronie tej krawędzi. Jeżeli zamienimy krawędź to musimy potem rekurencyjne sprawdzić legalność powstałych nowych dwóch trójkątów.

Szukanie trójkąta do którego mamy wstawić punkt

Aby znaleźć trójkąt w którym znajduje się nasz punkt, zaczynamy poszukiwanie od ostatniego utworzonego trójkąta. Następnie do kolejki priorytetowej wrzucamy kolejnych jego sąsiadów oraz dystans który go dzieli od trójkąta.

Cała triangulacja

Po wstawieniu do naszej Triangulacji po kolei wszystkich punktów w ten sposób otrzymujemy triangulacje która spełnia warunki bycia Triangulacją Delaunaya.

Odzyskiwanie krawędzi pierwotnego wielokąta

Jeżeli którejś z pierwotnych krawędzi wielokąta brakuje w końcowym słowniku Edges to przechodzimy rekurencyjnie po trójkątach których krawędzie przecinają i potem wracając zamieniamy krawędzie między trójkątami sąsiadów krawędzi która jest przecinana. Wiemy od którego trójkąta zacząć bo szukamy trójkąta który zawiera początek brakującej krawędzi lekko wzdłuż niej przesunięty. Obok na obrazku widać jaką dostaliśmy triangulację a jaka jest pożądana.

Usuwanie zewnętrznych trójkątów

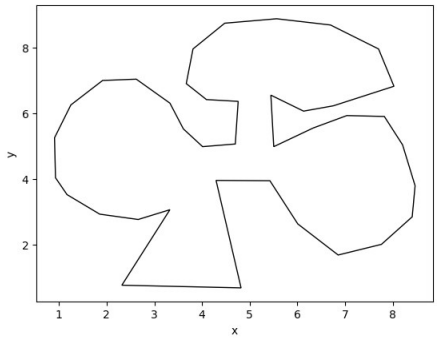
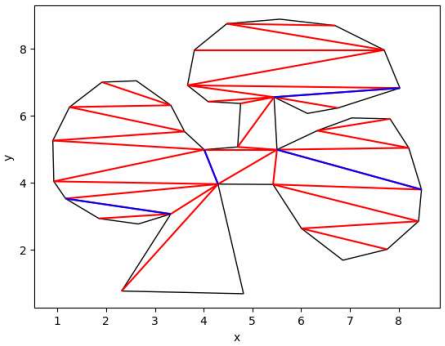
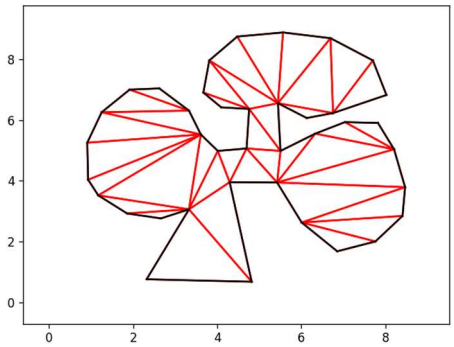
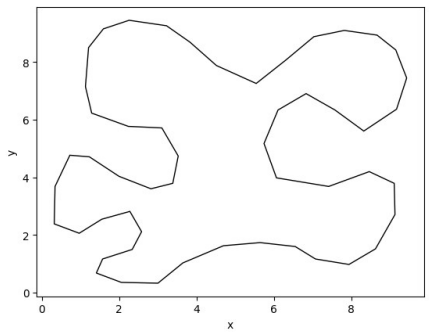
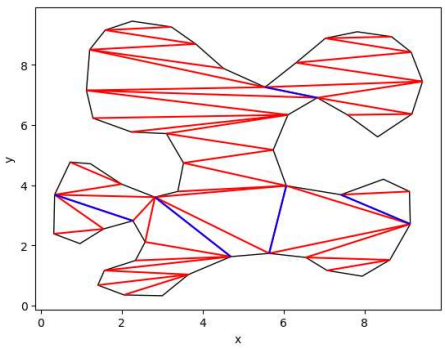
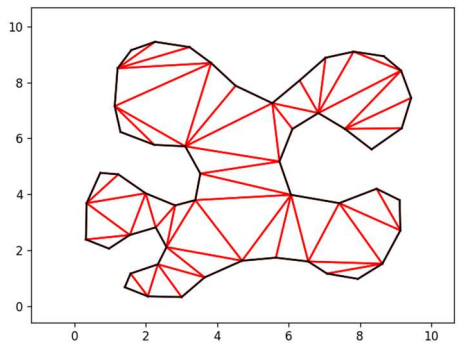
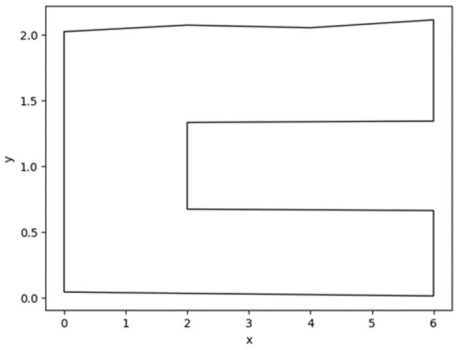
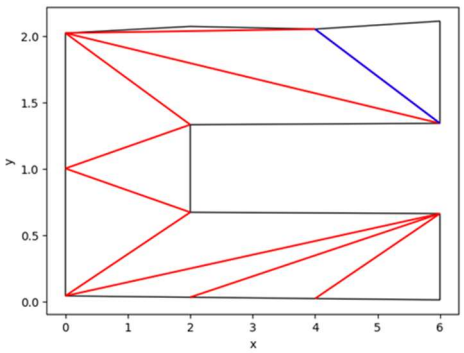
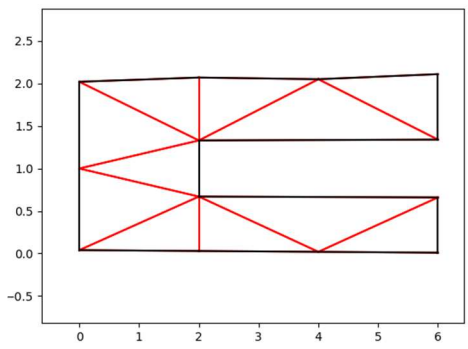
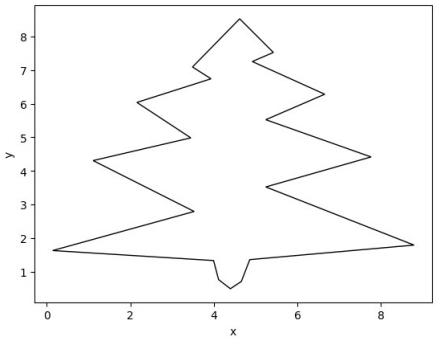
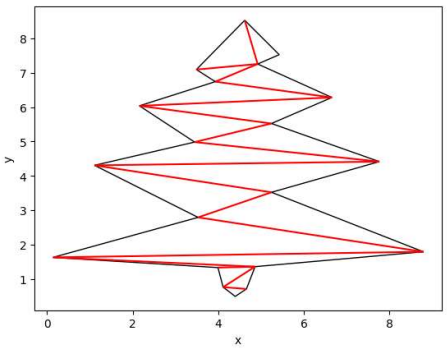
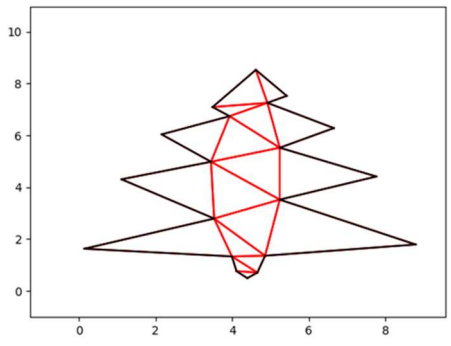
Z trójkąta który znajduje się w środku wielokąta puszczamy algorytm DFS który nie może przejść przez krawędź która należy do pierwotnego wielokąta zaznaczamy wszystkie trójkąty które odwiedzimy i to jest wynik naszej triangulacji.

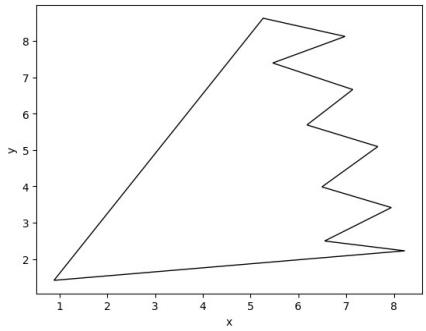
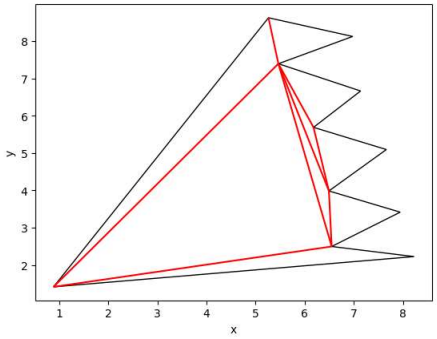
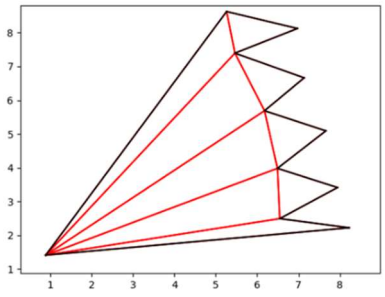
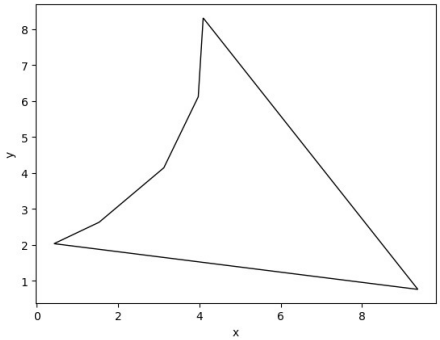
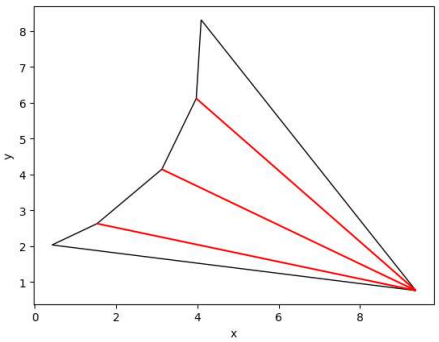
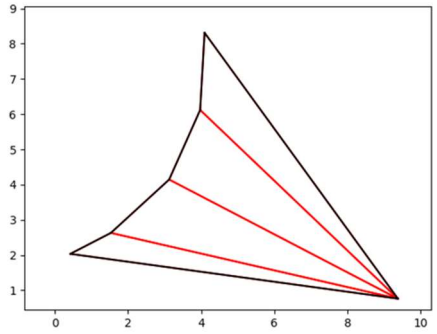
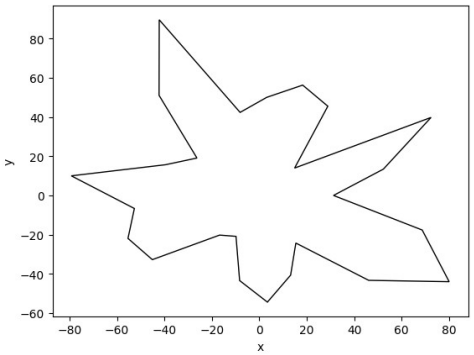
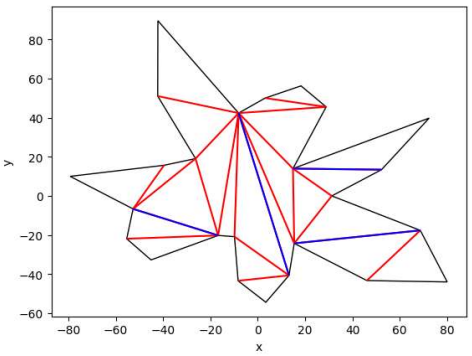
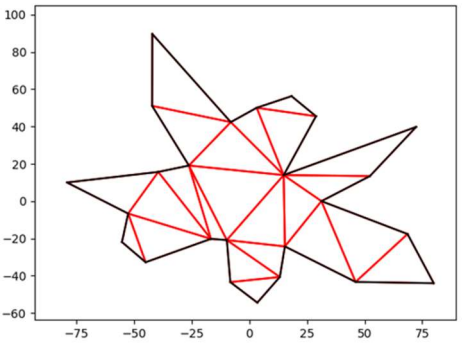
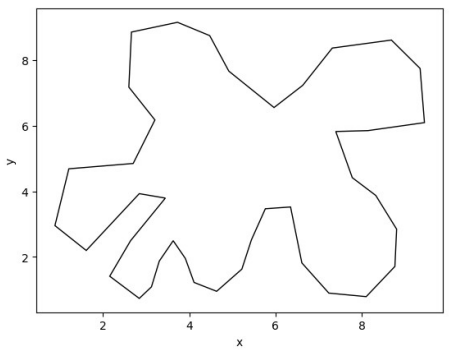
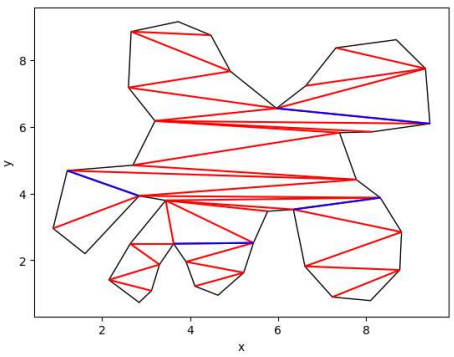
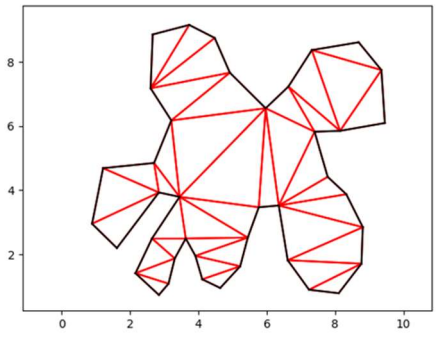
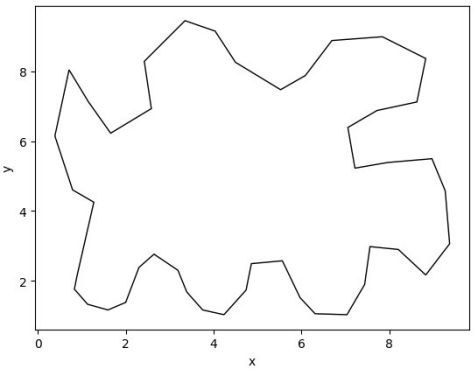
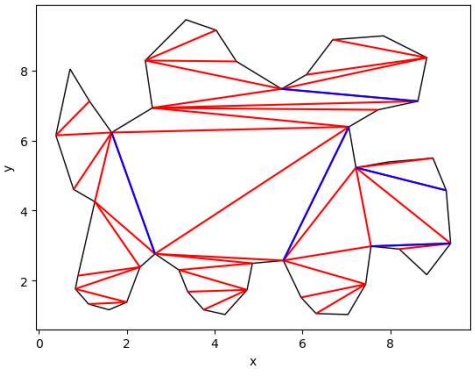
2.1) Wyniki działania obu algorytmów (testy dostępne są w paczce test_polygons.zip), poniżej widoczne kolejno test_polygon1-10 oraz test_perf3, test_perf4, test_perf2, test_perf1:

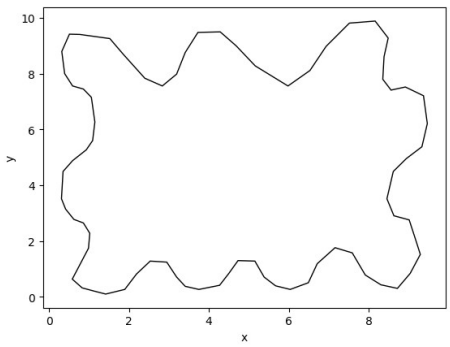
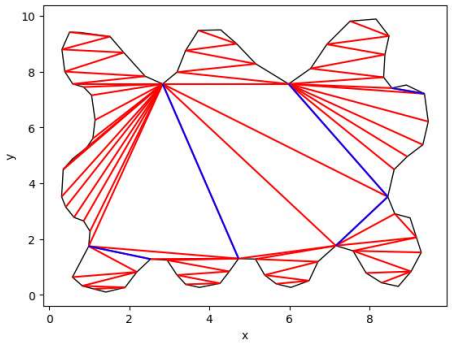
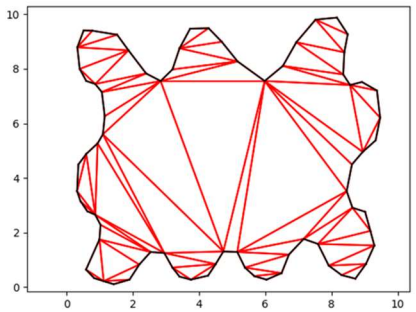
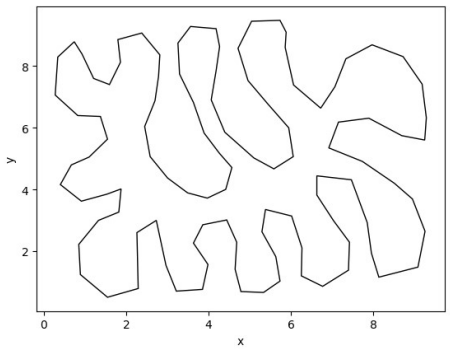
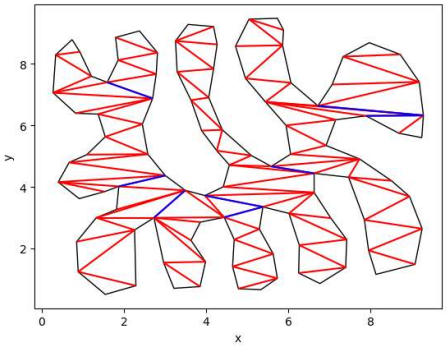
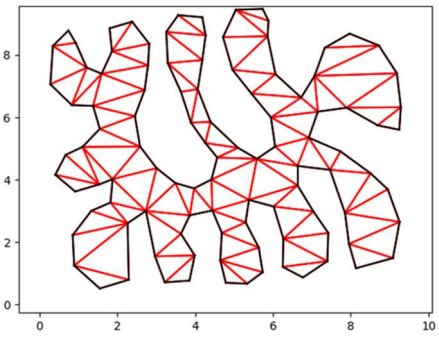
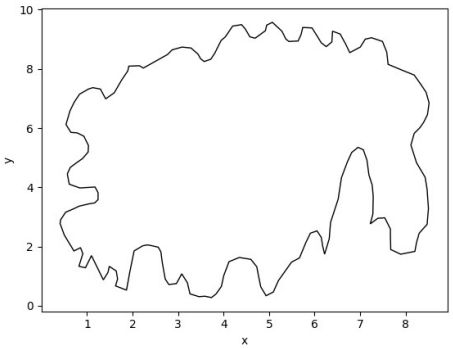
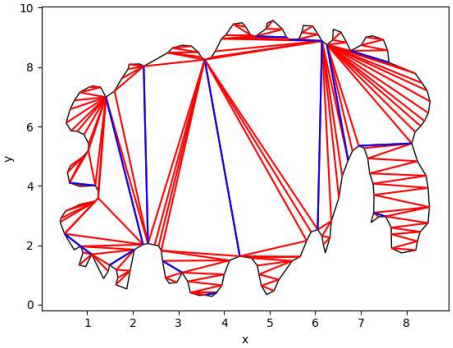
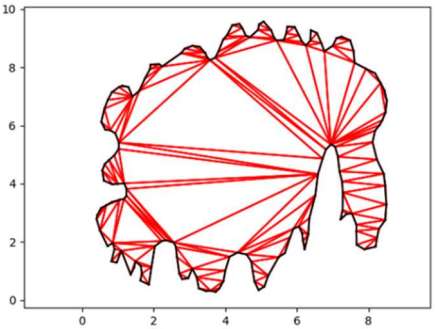
Wielokąty wejściowe

Triangulacja monotonicznych wielokątów

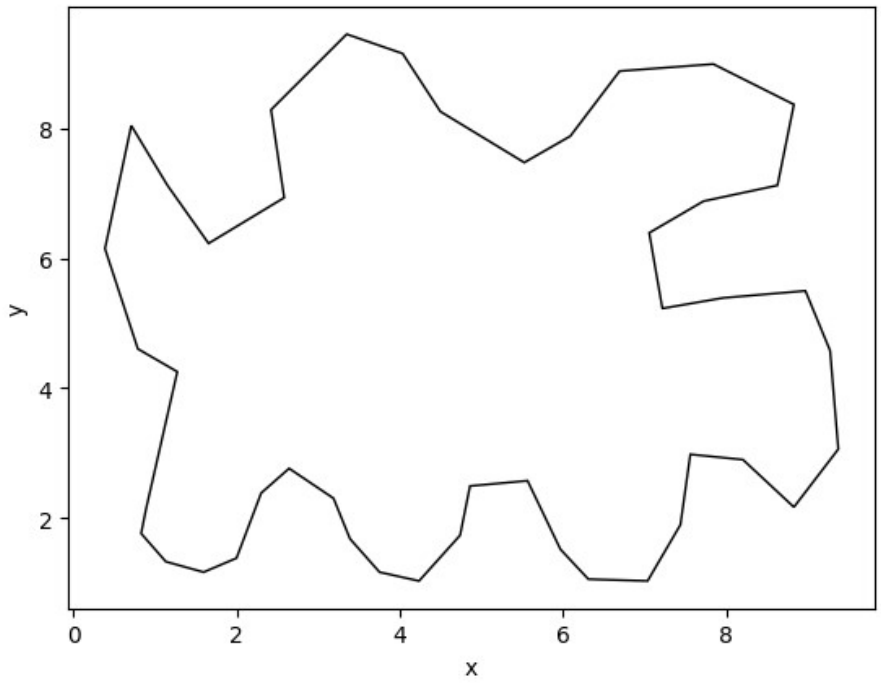
Triangulacja Delunaya

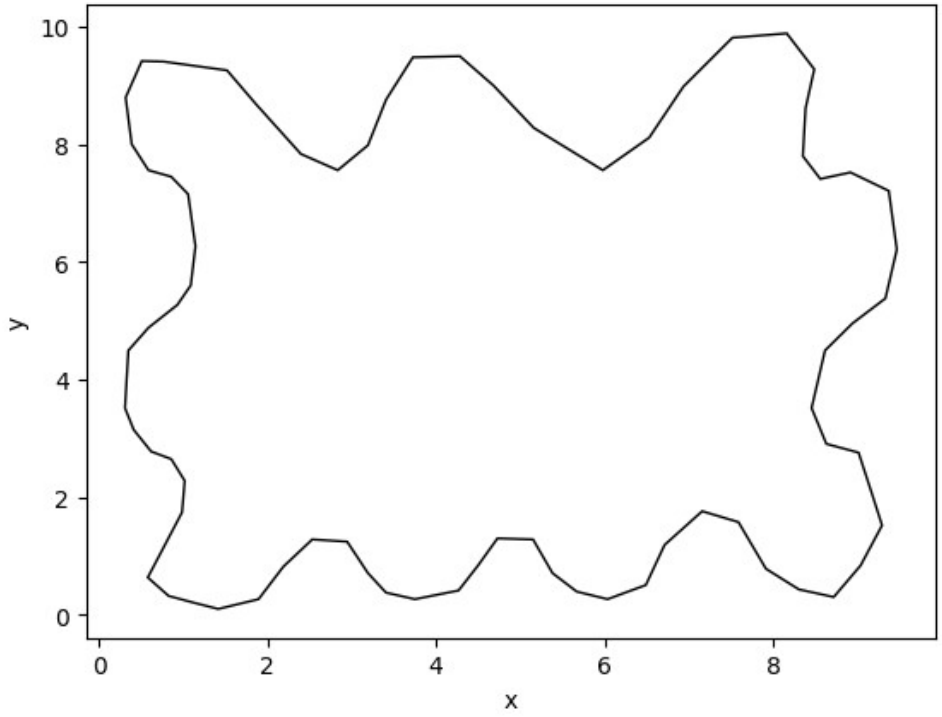















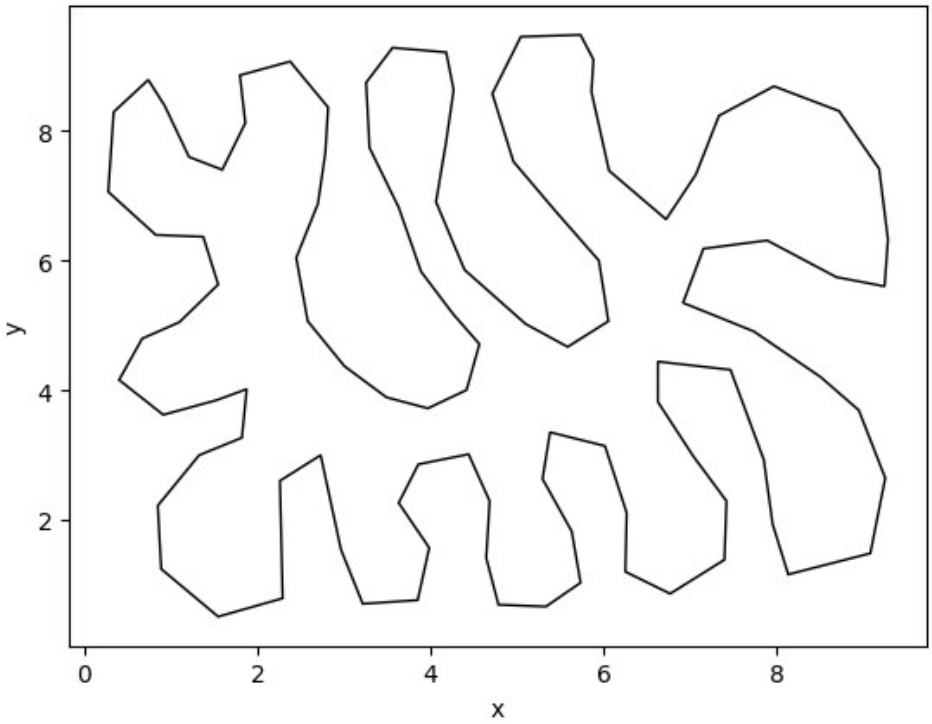
2.2) Testy wydajnościowe porównujący oba algorytmy:



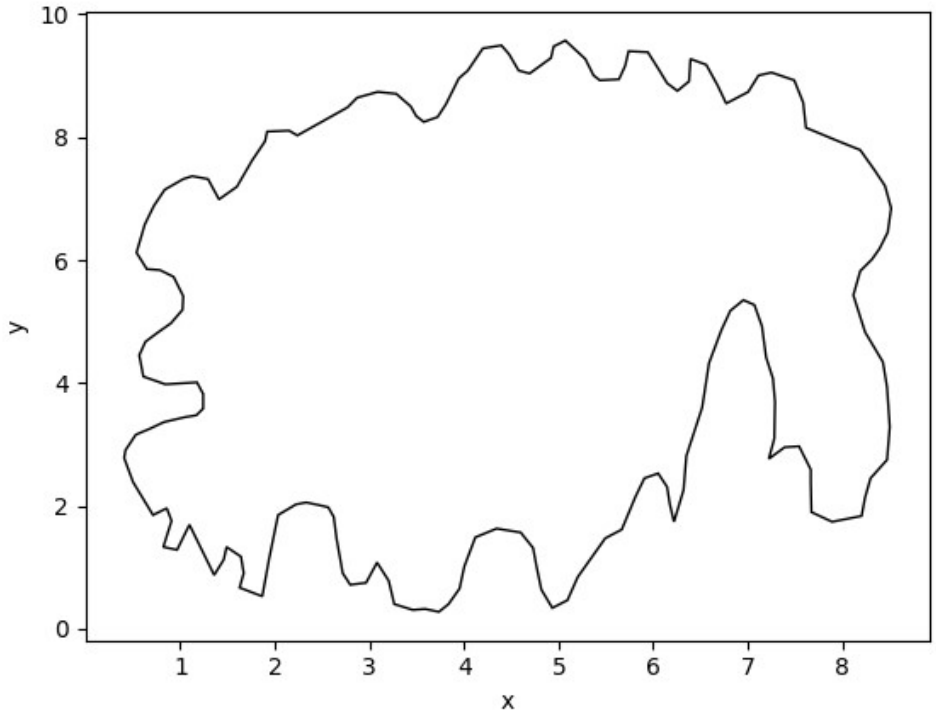
Rysunek 1 Test 1 (45 wierzchołków)



Rysunek 2 Test 2 (74 wierzchołków)

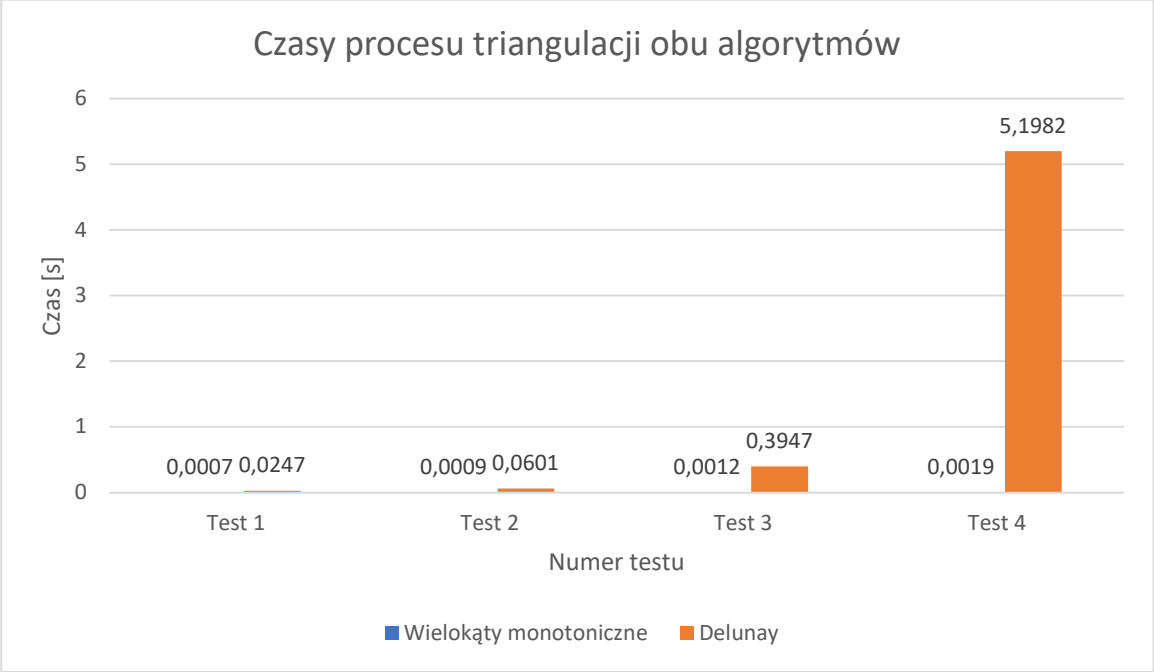


Rysunek 4 Test 3 (104 wierzchołków)



Rysunek 3 Test 4 (165 wierzchołków)

Porównanie czasów obu algorytmów:



Możemy zauważyć, że w triangulacji Delaunay trójkąty są bardziej zbliżone do kształtu równoramiennego i unikają tworzenia kątów rozwartych. Dlatego też triangulacja Delaunay jest preferowana np. do reprezentowania powierzchni. Natomiast, jeśli spojrzymy na porównanie czasów obliczeń (jak na poprzednim slajdzie), zauważymy, że mimo „gorszego" wizualnego efektu, triangulacja wielokątów monotonicznych jest szybsza.