Algorytmy geometryczne – Triangulacja Delaunay'a chmury punktów 2D z wykorzystaniem algorytmu iteracyjnego – analiza etapów triangulacji

Paulina Jędrychowska i Oskar Simon

1. Dane techniczne urządzenia i wykorzystane narzędzia:

System Windows 10 x64

Procesor i5-9300H

Pamięć RAM 16GB

Środowisko Jupyter Notebook

Język programowania Python 3.0

2. Cel projektu:

Implementacja i porównanie dwóch metod iteracyjnych algorytmu triangulacji Delaunay'a.

3. Obsługa programu:

Do funkcjonowania programu należy uruchomić komórki opisane jako „Biblioteki", „Funkcje", „Klasy", „Triangulacja 1 metodą", „Triangulacja 2 metodą".

W celu narysowania własnego zbioru punktów należy uruchomić komórkę „Rysowanie punktów", następnie w otwartym oknie zaznaczyć myszką punkty. Następnie narysowane punkty można zapisać uruchamiając komórkę „Zapis narysowanych punktów do pliku". W komórce należy zmiennej „fileName" nadać nazwę pliku (docelowo ustawione 'tmp.json'). Nazwa powinna być w cudzysłowie i zawierać rozszerzenie json.

Program umożliwia również wygenerowanie losowego zbioru punktów z podanego przedziału. W komórce należy zmiennej „fileName" nadać nazwę pliku (docelowo ustawione 'random_points.json'). Nazwa powinna być w cudzysłowie i zawierać rozszerzenie json. Zmiennej „numerOfPoints" należy przypisać ilość punktów, które chcemy narysować (docelowo ustawione na 10), zmiennej „rangeOfPoints" należy przypisać zakres, z którego mają być punkty (docelowo ustawione na [-100, 100]).

W celu korzystania ze zbioru punktów zapisanego w pliku należy uruchomić komórkę „Odczyt z pliku". Dodatkowo odczytany z pliku zbiór zostanie wyświetlony. Dla odczytanego zbioru można wygenerować trzy wizualizację: „Wizualizacja pierwszej metody triangulacji", „Wizualizacja drugiej metody triangulacji", „Wizualizacja odnajdywania trójkąta". Dla dwóch pierwszych wyświetlana jest dodatkowa informacja ile odcinków zostało utworzonych w

wyniku triangulacji. Trzecia z nich obrazuje sposób w jaki algorytm odnajduje trójkąt, który zawiera dany punkt.

Uruchomienie ostatniej komórki „Porównanie czasu wykonywania algorytmów” wyświetli czasy wykonania obu algorytmów, dzięki czemu możemy ocenić, który z nich jest szybszy dla danego zestawu danych. Zmienna „numbersOfPoints” zawiera ilości punktów, dla których chcemy porównać algorytmy (docelowo [10, 100, 200, 500, 1000, 2000, 5000, 10000]). Można ustawić dowolne wartości liczbowe, dla większych liczb program będzie wykonywał się dłużej.

4. Utworzone klasy obiektów:

- Point – Reprezentuje punkt. Przechowuje wartości x i y punktów. Ma dodatkowe pola vx i vy, które trzymają wartości x i y używane przy wizualizacji. Są to zazwyczaj takie same wartości jak x i y, z wyjątkiem wierzchołków początkowego prostokąta pomocniczego. Jest to spowodowane tym, że wierzchołki prostokąta początkowego są znacznie bardziej oddalone od pozostałych punktów i w celu czytelnej wizualizacji są one rysowane bliżej.
- Line – Reprezentuje linię. Przechowuje obiekty typu Point, reprezentujące początek i koniec odcinka. Każda linia jeżeli tworzy bok jakiegoś trójkąta, to automatycznie zostaje mu przypisany obiekt typu Triangle, a po usunięciu trójkąta usuwany jest odnośnik do niego. Linia może mieć przypisany maksymalnie dwa trójkąty. Linie można porównywać po ich współrzędnych końcowych.
- Triangle – Reprezentuje trójkąt. W trzyelementowej liście przechowywane są trzy obiekty typu Line, reprezentujące odcinki tworzące ten trójkąt. Klasa ma metodę znajdującą środkowy punkt i promień okręgu opisanego na nim. Można również odnaleźć trójkąt sąsiadujący po krawędzi z nim. Obiekty można usuwać.

5. Odnajdowanie trójkąta do którego należy punkt

W obu algorytmach potrzebujemy wiedzieć, w środku którego trójkąta znajduje się rozpatrywany punkt. W tym celu trzymany jest trójkąt startowy, z którego przechodząc po krawędziach docieramy do odpowiedniego trójkąta. Dla kolejnych trójkątów sprawdzamy czy punkt znajduje się w nich. Przechodzimy po trzech bokach aktualnego trójkąta. Porównujemy położenie względem boku szukanego punktu i punktu leżącego naprzeciwko danego boku w trójkącie. Jeżeli punkty znajdują się po przeciwnych stronach, to znaczy że szukany punkt znajduje się za krawędzią. Następnym rozpatrywanym trójkątem staje się sąsiad po tej krawędzi. Jeżeli dla jakiegoś trójkąta dla wszystkich z jego boków punkty były po tej samej stronie, to jest on naszym szukanym trójkątem.

Trójkątem startowym jest na początku jeden z trójkątów utworzonych z pomocniczego prostokąta początkowego. Następnie trójkątem startowym staje się każdy nowo utworzonych trójkąt, jest to potrzebne ponieważ wcześniejszy trójkąt startowy mógł zostać usunięty przy powstawaniu nowego trójkąta.

Do porównania położenia punktów względem prostej wykorzystujemy metodę korzystającą z wyznacznika macierzy. Jeżeli oba punkty mają taki sam znak wyznacznika to znaczy że są po tej samej stronie, jeżeli nie to po przeciwnej. Punkt znajdujący się na linii nie ma znaczenia do którego z trókątów zostanie zakwalifikowany.

Metoda służąca do określenia położenia punktów względem prostej korzysta z wyznacznika 3x3 własnej implementacji na podstawie wzoru 1. Gdzie (a_x, a_y), (b_x, b_y) są końcami prostej a (c_x, c_y) rozpatrywanym punktem.

Wzór 1. Wyznacznik 3x3

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

Funkcja orient() zwraca znak wyznacznika, przy określonej tolerancji. Jako tolerancję dla zera przyjęto 10^{-13} .

6. Algorytmy triangulacji Delaunay'a:

W pierwszym kroku algorytmu tworzony jest prostokąt , w środku którego znajdują się wszystkie punkty. Prostokąt dzielony jest na dwa trójkąty, które są naszą początkową triangulacją. Aby uniknąć naruszenia triangulacji naszej chmury punktów przez pomocnicze trójkąty tworzone są one znacznie większe. W głównej pętli algorytmu dodawane są kolejne punkty. Dalsza część algorytmu zaimplementowana jest na dwa różne sposoby:

- Sposób pierwszy:

  Dla dodanego punktu znajdujemy trójkąt, który zawiera ten punkt. Następnie znaleziony trójkąt dzielimy na trzy trójkąty. Jeżeli znaleziony punkt leży na linii to tworzymy cztery trójkąty. Usuwamy trójkąt, w którym znajduje się aktualnie rozpatrywany punkt (dla punktu leżącego na linii usuwamy też jego sąsiada. Dla nowo utworzonych trójkątów znajdujemy ich sąsiadów względem starej linii (która wcześniej należała do usuniętego trójkąta). Dla każdego z trójkąta tworzymy okrąg, w który jest wpisany. Jeżeli sąsiedni trójkąt zawiera się w tym okręgu to zmieniana jest przekątna czworoboku utworzonego z tych trójkątów.

- Sposób drugi:

  Znajdujemy trójkąt, w którym znajduje się aktualnie rozpatrywany punkt. Ten trójkąt zostaje dodany do listy trójkątów do usunięcia oraz do pustego stosu trójkątów. Następnie, dopóki stos nie jest pusty, wykonywane są następujące kroki: ze stosu ściągany jest trójkąt i każdy jego topologiczny sąsiad, którego opisane na nim koło

zawiera aktualnie rozpatrywany punkt, zostaje dodany do listy trójkątów do usunięcia, oraz wyżej wspomnianego stosu. Jeżeli dany sąsiad nie spełnia wspomnianego warunku, to krawędź, którą graniczy on z aktualnie rozpatrywanym trójkątem dodawana jest do listy krawędzi potrzebnej do dalszej części algorytmu. Tak samo dzieje się z krawędziami rozpatrywanych trójkątów, które stanowią brzeg figury. Po opróżnieniu stosu, usunięte zostają wszystkie trójkąty na liście trójkątów do usunięcia. Następnie zastępowane są one przez trójkąty zbudowane z aktualnie rozpatrywanego punktu i krawędzi zapisanych na wcześniej wspomnianej liście.

Po rozpatrzeniu wszystkich punktów ze zbioru usuwane są wszystkie krawędzie należące do pomocniczego prostokąta.

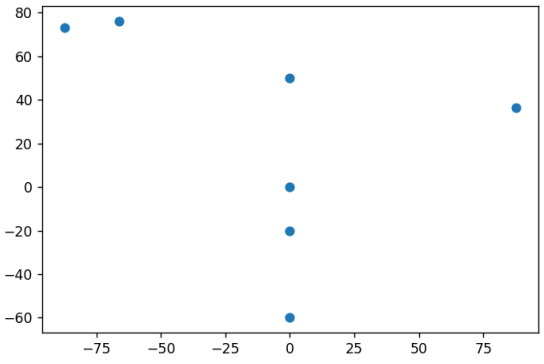
W rezultacie algorytmu uzyskujemy: listę wszystkich uzyskanych odcinków , wykres kolejnych kroków algorytmu dla danego zbioru.

7. Utworzone zbiorów punktów w celu przetestowania algorytmu:

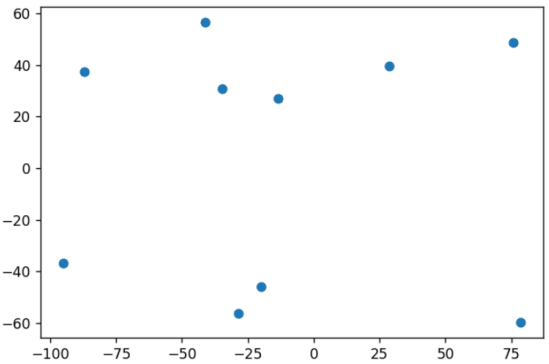
- Tarcza („shield”)
- 10 punktów („10_random_points”)
- 100 punktów („100_random_points”)

Wszystkie zbiory zostały zwizualizowane przez dostarczone narzędzie graficzne oparte o bibliotekę matplotlib.

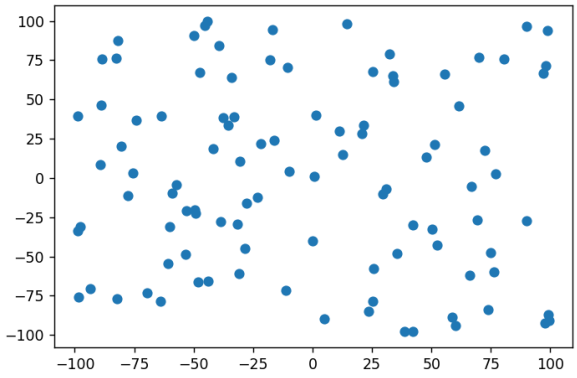
Wykres 1.1. Tarcza



Wykres 1.2. 10 punktów



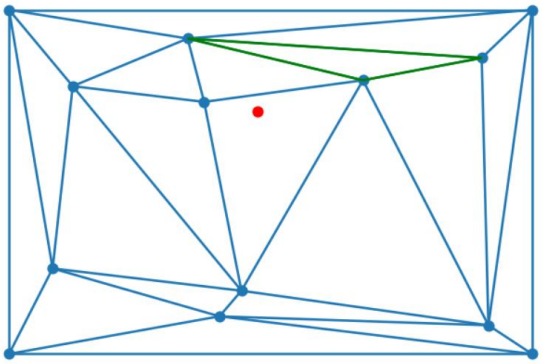
Wykres 1.3. 100 punktów



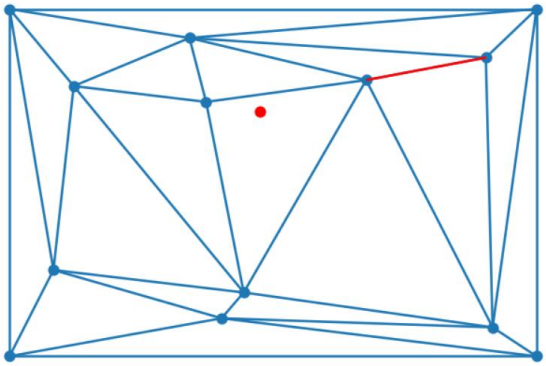
8. Wizualizacja przebiegu algorytmu znajdującego trójkąt zawierający punkt:

Wykres 2. Wizualizacja przebiegu algorytmu znajdującego trójkąt dla zbioru 10 punktów

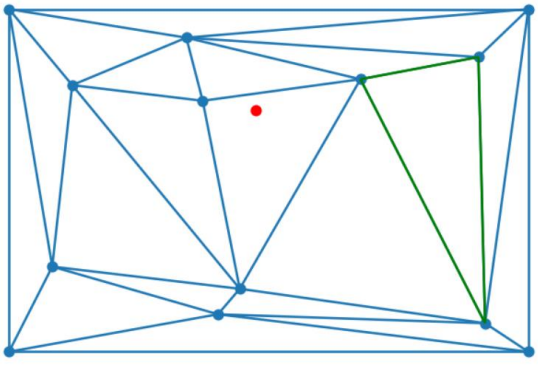
Wykres 2.1. Znajdowany jest trójkąt startowy



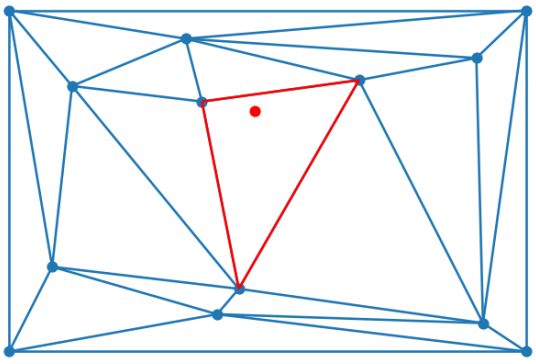
Wykres 2.2. Znajdowana jest krawędź, względem której szukany punkt jest po przeciwnej stronie



Wykres 2.3. Przechodzimy do trójkąta sąsiadującego, powtarzamy kroki algorytmu



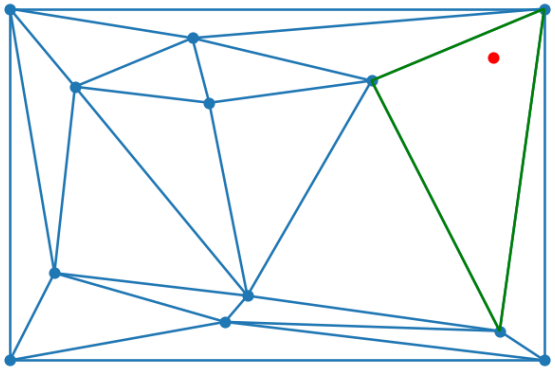
Wykres 2.4. Znaleziony trójkąt



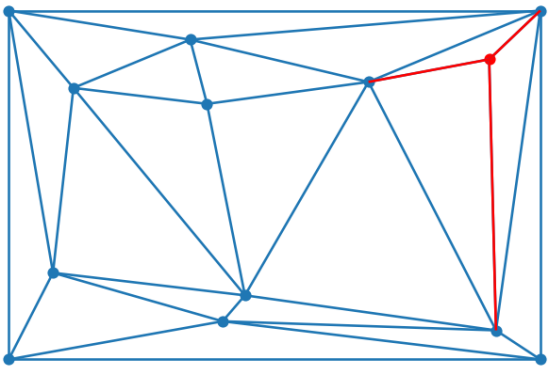
9. Wizualizacja przebiegu pierwszgo algorytmu triangulacji Delaunay'a:

Wykres 3. Wizualizacja przebiegu pierwszego algorytmu triangulacji Delaunay'a dla zbioru 10 punktów

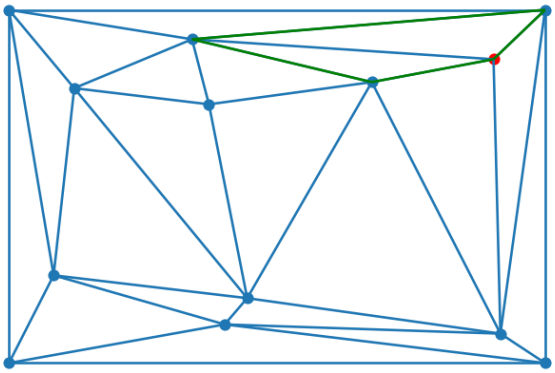
Wykres 3.1. Znajdowany jest punkt



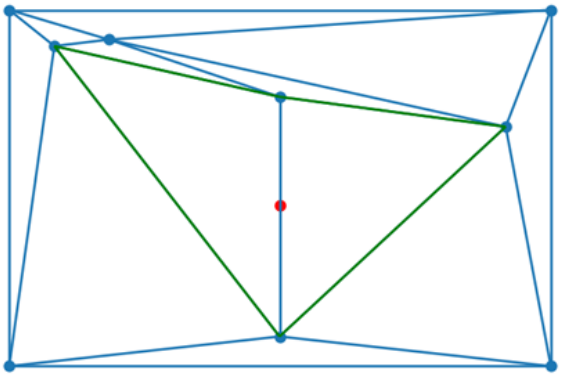
Wykres 3.2. Punkt łączony jest liniami z wierzchołkami trójkąta, do którego należy

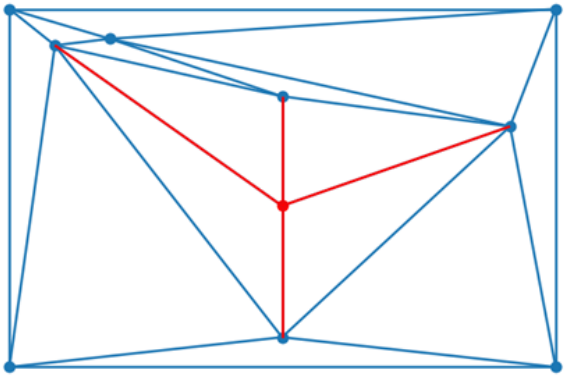


Wykres 3.3. W czworokątach zmieniane są przekątne, powtarzane są kroki



Wykres 4. Przypadek specjalny, występowanie punktu na linii dla zbioru punktów Tarcza

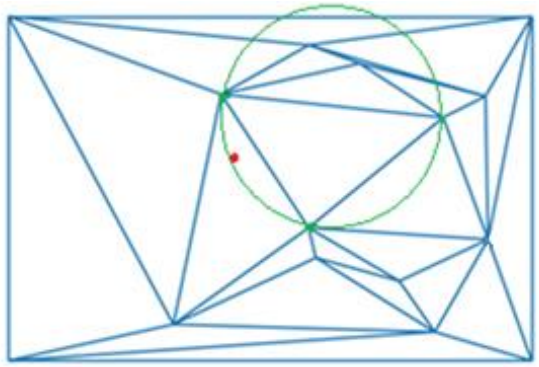




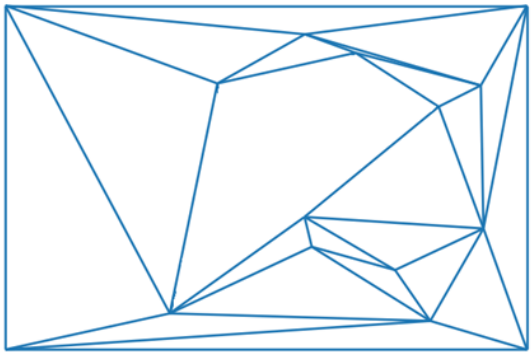
10. Wizualizacja przebiegu drugiego algorytmu triangulacji Delaunay'a:

Wykres 5. Wizualizacja przebiegu pierwszego algorytmu triangulacji Delaunay'a

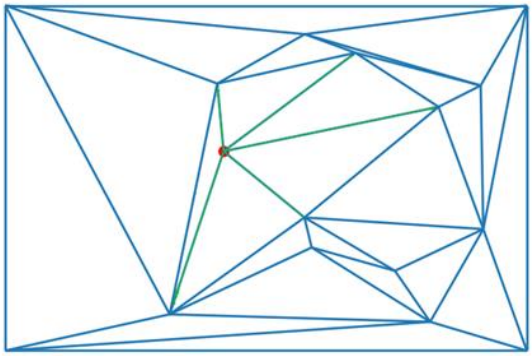
Wykres 5.1. Znajdowane są trójkąty, których koła na nich opisane zawierają ten punkt



Wykres 5.2. Usuwane są odpowiednie trójkąty



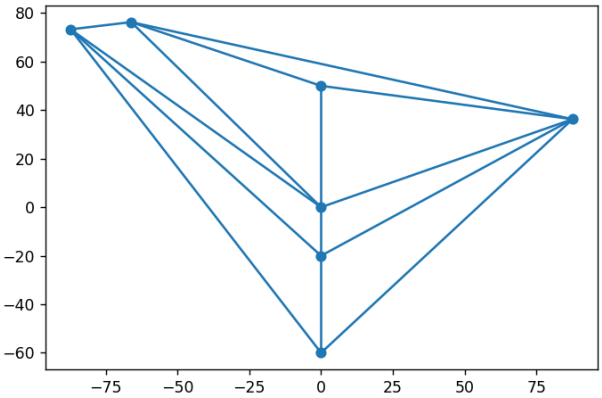
Wykres 5.3. Dziura zostaje wypełniona

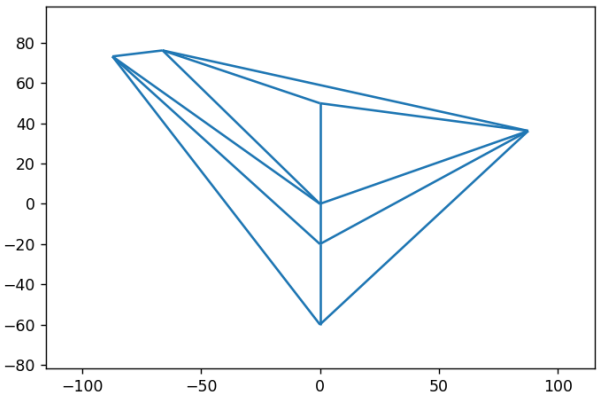


11. Wyniki triangulacji:

Wykres 6.1. Wynik triangulacji Delaunay'a

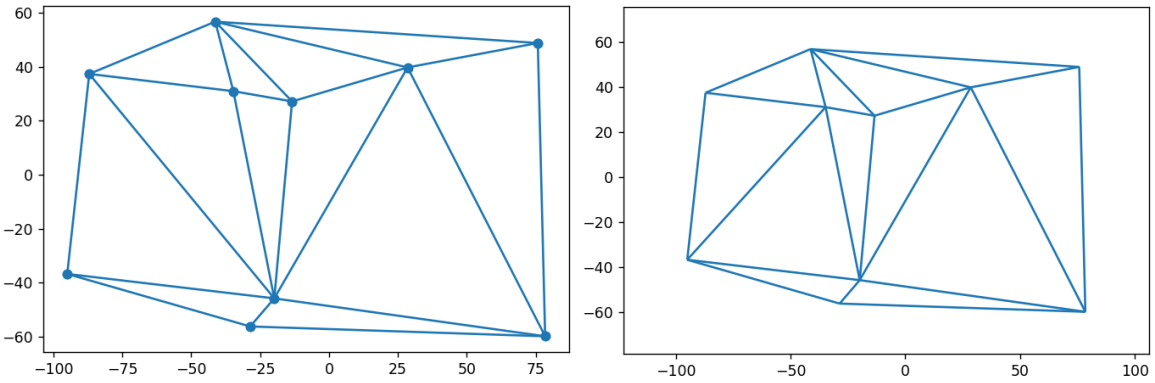
dla zbioru Tarcza sposobem pierwszym i drugim





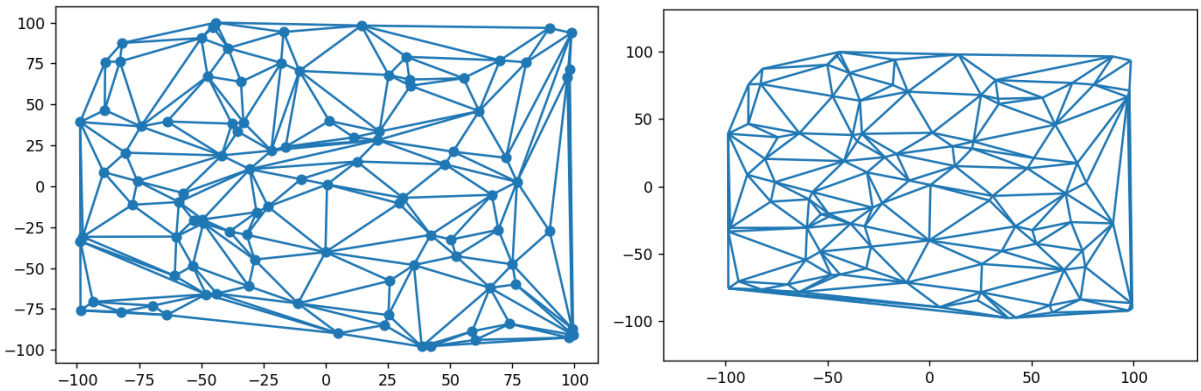
Wykres 6.2. Wynik triangulacji Delaunay'a

dla zbioru 10 punktów sposobem pierwszym i drugim



Wykres 6.3. Wynik triangulacji Delaunay'a

dla zbioru 100 punktów sposobem pierwszym i drugim



12. Porównanie czasu wykonywania algorytmów:

Tabela 1. Porównanie czasów działania obu algorytmów

względem liczby punktów

Algorytm	100 pkt	500 pkt	1000 pkt	10 000 pkt	50 000 pkt	100 000 pkt	200 000 pkt
Pierwszy	0.04 s	0.14 s	0.32 s	8.38 s	95.52 s	266.59 s	714.18 s
Drugi	0.04 s	0.26 s	0.40 s	8.41 s	95.52 s	262.51 s	700.64 s

Czasy działania algorytmów były do siebie zbliżone. Dla mniejszych zbiorów punktów szybszy okazywał się być pierwszy algorytm, a od zbioru zawierającego 100 000 punktów szybszy okazywał się być drugi algorytm. Natomiast różnice czasowe były bardzo małe nawet przy dużych zbiorach.

13. Wnioski:

Algorytmy w większości przypadków wyznaczają triangulację tak samo. Natomiast dla niektórych przypadków pierwszy algorytm wyznacza linie błędnie. Jest to spowodowane błędem w kodzie, który jest ciężki do wykrycia. Porównując ze sobą dwa wyniki triangulacji można zauważyć, że niektóre trójkąty w drugim algorytmie wyglądają inaczej niż w pierwszym i spełniają warunki triangulacji Delaunay'a. Próbując prześledzić przebieg pierwszego algorytmu nie jesteśmy w stanie zauważyć gdzie algorytm popełnia błąd.

Wyniki czasowe obu algorytmów po naprawie pierwszego kodu nie powinny ulec znacząco zmianie, gdyż algorytm wykonuje tylko drobny błąd.

Klasy użyte do implementacji były dobrze przemyślane i znacznie ułatwiły implementacje obu algorytmów.

Jako, że czasy wykonania algorytmów nie różnią się od siebie znacznie lepszym pomysłem wydaje się używanie drugiego algorytmu, ponieważ jest on łatwiejszy w implementacji.

14. Bibliografia:

https://pl.wikipedia.org/wiki/Triangulacja_(grafika_komputerowa)

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

http://web.mit.edu/alexmv/Public/6.850-lectures/lecture09.pdf?fbclid=IwAR2lgWJPYhq0qIEDLJiMnxf1EvYlwkXrWiXT51sf_xAV6kMDA8VQvTkZjqI