Michał Szczurek
Informatyka II rok, grupa wtorek 12:50 B

Algorytmy geometryczne, ćwiczenie 3- Sprawozdanie

1. Wprowadzenie

Celem ćwiczenia była implementacja i przetestowanie trzech algorytmów: algorytmu sprawdzającego monotoniczność wielokąta względem osi OY układu współrzędnych, algorytmu klasyfikującego wierzchołki wielokąta i algorytmu wykonującego triangulację wielokąta y- monotonicznego.

2. Specyfikacja środowiska, w którym wykonano ćwiczenie

Ćwiczenie zostało wykonane przy pomocy jupyter notebook i Python 3 na komputerze z systemem Windows 10 (64 bitowym) i procesorem Intel Core i-5- 9300H (64 bitowy). Wykresy wygenerowano przy pomocy dostarczonego na laboratoria narzędzia graficznego i biblioteki matplotlib. Narzędzie graficzne zostało przystosowane do wprowadzania rysowania i wprowadzania wielokątów. Dzięki temu dane można wprowadzać zarówno poprzez narysowanie wielokąta jaki i poprzez podanie listy punktów. Należy jednak pamiętać, by punkty zadawać przeciwnie do ruchu wskazówek zegara.

3. Sposób wykonania ćwiczenia

Na wykonanie ćwiczenia składały się następujące czynności:

- Dostosowanie narzędzia graficznego.
- Zaimplementowanie algorytmu sprawdzającego y- monotoniczność wielokąta wraz funkcjami pomocniczymi.
- Zaimplementowanie algorytmu klasyfikującego wierzchołki wielokąta.
- Zaimplementowanie algorytmu wykonującego triangulację wielokąta wraz z funkcjami i strukturami pomocniczymi.
- Przetestowanie algorytmów dla wybranych danych.

4. Dostosowanie narzędzia graficznego

Dostarczone narzędzia graficzne zdawało się nie w pełni umożliwiać wprowadzanie wielokątów - według moich obserwacji wprowadzany wielokąt się nie domykał. Prawdopodobną przyczyną tego faktu jest sposób w jaki stwierdzane jest, czy użytkownik kliknął w ostatni punkt wielokąta. Narzędzie korzysta z tolerancji, która jest ustalana automatycznie w zależności od osi wykresu i sprawdza, czy punkt kliknięty przez użytkownika jest w jej zakresie. Sprawdzenie jest realizowane przez instrukcję:

```
if dist(self.rect_points[0], new_point) < (np.mean([self.ax.get_xlim(), self.ax.get_ylim()])*TOLERANCE):
```

Według mojej wiedzy taka realizacja w praktyce jest tożsama z przyjęciem tolerancji równej 0- instrukcja self.ax.get_xlim() zwraca granicę osi wykresu - domyślnie krotkę (-0.5,0.5). Podobnie self.ax.get_ylim(). Funkcja mean obliczająca średnią w przypadku otrzymania dwóch takich krotek oblicza średnią liczb z krotek, która jak można zauważyć wynosi w takim przypadku 0. Wobec tego zmieniłem powyższą instrukcję na:

```
if dist(self.rect_points[0], new_point) < (np.mean([self.ax.get_xlim()[1]-self.ax.get_xlim()[0],
self.ax.get_ylim()[1]- self.ax.get_ylim()[0]])*TOLERANCE):
```

Dzięki temu obliczana jest średnia z długości osi, a nie z jej punktów końcowych.

5. Sprawdzanie y-monotoniczności wielokąta

5.1 Lista kroków algorytmu:

1. Znalezieniu punktu o największej współrzędnej y. W przypadku remisów - położonego najbardziej na prawo.
2. Przeglądanie punktów przeciwnie do ruchu wskazówek zegara zaczynając od najwyższego, aż do znalezienia takiego punktu p, że następny ma większą współrzędną y od p.
3. Przeglądanie punktów zgodnie z ruchem wskazówek zegara zaczynając od najwyższego, aż do znalezienia takiego punktu p', że następny ma niemniejszą współrzędną y od p'.
4. Wielokąt jest y monotoniczny jeśli p jest równy p'.

5.2 Krótkie omówienie poprawności algorytmu

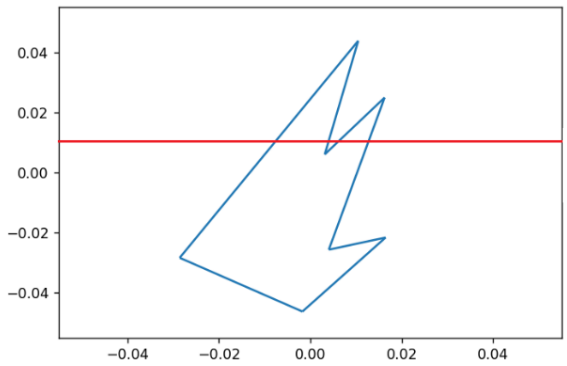
Algorytm polega na „spływaniu" w dół wielokąta, po obu łańcuchach aż dojedziemy do najniższego punktu. Jeśli najniższy punkt, do którego doszliśmy lewym łańcuchem jest tym samym do którego doszliśmy prawym łańcuchem - to w wielokącie nie ma wierzchołków dzielących i łączących, czyli jest on monotoniczny. Warto zwrócić uwagę, że zaimplementowany przeze mnie algorytm sprawdza słabą monotoniczność - gdyby istniało kilka najniższych punktów, to p będzie punktem najbardziej położonym na prawo, ponieważ remisy rozstrzygane na korzyść później odkrytego punktu. Punkt p' z kolei będzie pierwszym znalezionym punktem spośród najniższych czyli p = p'. Istotne jest również to, że punkt znaleziony w kroku pierwszym musi być skrajnym prawym punktem. W przeciwnym wypadku Przeglądanie punktów prawego łańcucha zakończy się już na etapie pierwszego punktu.

Zaimplementowano również algorytm sprawdzający ścisłą monotoniczność. Różni się on tym, że przeglądanie obu łańcuchów jest zakończone wtedy, gdy następny punkt jest położony nie niżej od obecnego.

5.3 Przykładowe działanie algorytmu na danych testowych.

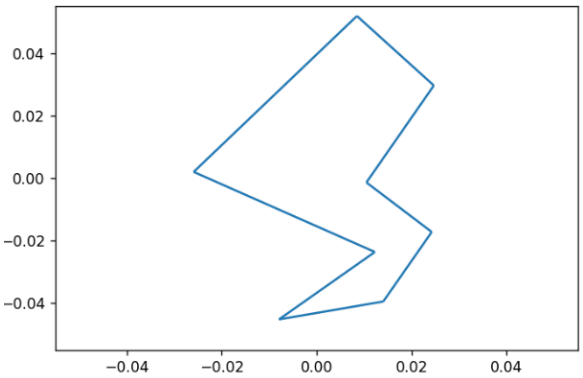
Zestawy danych, których ilustracje przedstawiono poniżej znajdują się na końcu notebooka. Zestawy dobrano tak, by przetestować algorytm dla wielokąta monotonicznego, niemonotonicznego i słabo monotonicznego.

Rys. 1. Wielokąt niemonotoniczny (wraz z przykładową prostą, która ukazuje na jego niemonotoniczność)



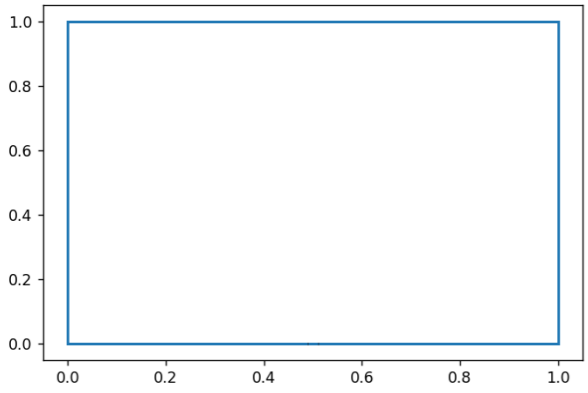
Algorytm poprawnie określił wielokąt jako niemonotoniczny.

Rys. 2. Wielokąt monotoniczny



Algorytm poprawnie określił wielokąt jako monotoniczny.

Rys. 3. Wielokąt słabo monotoniczny



Algorytm poprawnie określił wielokąt jako monotoniczny w przypadku sprawdzania nieścisłej monotoniczności i jako niemonotoniczny w ścisłym sensie

6. Klasyfikacja wierzchołków

6.1. Opis algorytmu

Algorytm przegląda wierzchołki wielokąta i na podstawie wysokości sąsiednich wierzchołków i kąta jaki tworzy z sąsiednimi wierzchołkami przypisuje je do jednej z grup:

- Wierzchołki początkowe- obaj sąsiedzi leżą poniżej i kąt wewnętrzny tworzony przez wierzchołek i sąsiadów < π.
- Wierzchołki końcowe- gdy obaj sąsiedzi leżą powyżej i kąt wewnętrzny tworzony przez wierzchołek i sąsiadów < π.
- Wierzchołki łączące - gdy obaj sąsiedzi leżą powyżej i kąt wewnętrzny tworzony przez wierzchołek i sąsiadów > π.
- Wierzchołki dzielące- gdy obaj sąsiedzi leżą poniżej i kąt wewnętrzny tworzony przez wierzchołek i sąsiadów > π.
- Wierzchołki prawidłowe- w pozostałych przypadkach.

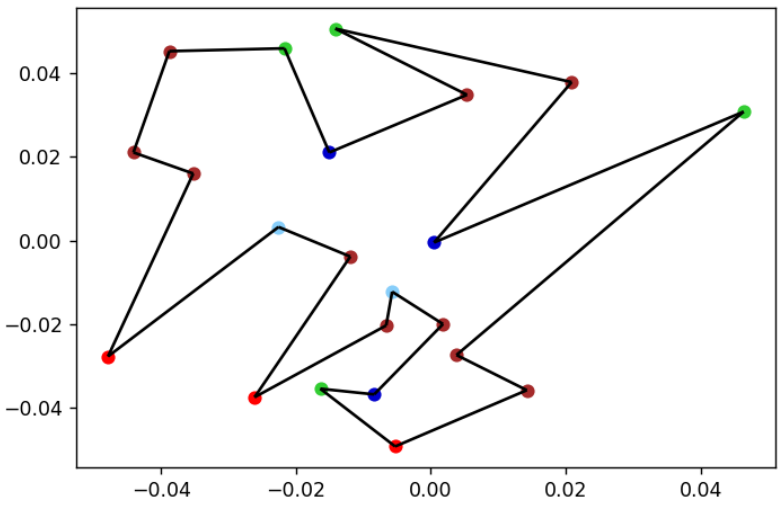
6.2. Przykład działania algorytmu

Na poniższym wykresie wierzchołki oznaczono odpowiednimi kolorami:

- Zielonym - wierzchołki początkowe
- Czerwonym - wierzchołki końcowe
- Granatowym - wierzchołki łączące
- Jasnoniebieskim - wierzchołki dzielące
- Brązowym - wierzchołki prawidłowe

Wielokąt dobrano tak, by pokazać wszystkie typy wierzchołków

Rys. 4. Klasyfikacja wierzchołków



Przykład dobrano tak, by przedstawić wszystkie rodzaje wierzchołków.

7. Triangulacja wielokątów ściśle monotonicznych

7.1. Lista kroków algorytmu:

0. Sprawdzenie, czy wielokąt jest ściśle y -monotoniczny.

1. Znalezienie wierzchołka o największej współrzędnej y w wielokącie - h i wierzchołka o najmniejszej współrzędnej y - v.

2. Przypisanie wierzchołkom innym niż v i h łańcucha do którego należą (prawy lub lewy) przechodząc po nich od najwyższego do najniższego i dodanie ich do kolejki. Kolejka jest posortowaną listą punktów, powstałą podobnie jak ma o miejsce w algorytmie mergesort.

3. Włożenie wierzchołka h i pierwszego wierzchołka z posortowanej listy na stos.

4. Przeglądanie reszty wierzchołków z posortowanej listy:

- Jeśli kolejny wierzchołek należy do innego łańcucha niż wierzchołek stanowiący szczyt stosu, to należy go połączyć ze wszystkimi wierzchołkami ze stosu, z którymi nie tworzy on boku wielokąta. Na stosie pozostaje wierzchołek ze szczytu stosu i obecnie przeglądany punkt

- W przeciwnym wypadku należy analizować trójkąty utworzone przez przeglądany wierzchołek z wierzchołkami zdejmowanymi ze stosu- tak długo jak trójkąty należą do wielokąta, należy połączyć ich wierzchołki (tam gdzie połączenie już nie istnieje). Na stosie zostają wierzchołki, dla których nie udało się utworzyć trójkąta.

5. Połączenie punktów ze stosu z wierzchołkiem v (o ile wierzchołki nie są już połączone).

7.2. Szczegóły implementacyjne algorytmu

Algorytm wykorzystuje dwie samodzielnie zaimplementowane struktury danych. Pierwszą z nich jest Polygon czyli wielokąt. Struktura składa się w zasadzie tylko z listy punktów wielokąta w postaci krotek. Jest ona używana we wszystkich algorytmach jako opakowanie tablicy krotek, co biorąc pod uwagę, że punkty są zadane przeciwnie do ruchu wskazówek zegara pozwala na łatwe odtworzenie figury.

Drugą strukturą jest Tri_point. Jest to reprezentacja punktów zawierająca współrzędne punktu, jego etykietę, która pozwala na identyfikacje łańcucha, do jakiego należy punkt oraz krotki reprezentujące sąsiadów punktu w wielokącie. Taka reprezentacja umożliwia wygodny dostęp do informacji o punkcie potrzebnych w procesie triangulacji, jaki i umożliwia sprawdzenie czy dwa punktu są sąsiadami w wielokącie w czasie stałym. Funkcjonalność ta jest wykorzystywana podczas dodawania nowej przekątnej- przed jej dodaniem należy sprawdzić, czy nie jest krawędzią grafu, czyli czy punkty wyznaczające ją nie są sąsiadami w wielokącie.

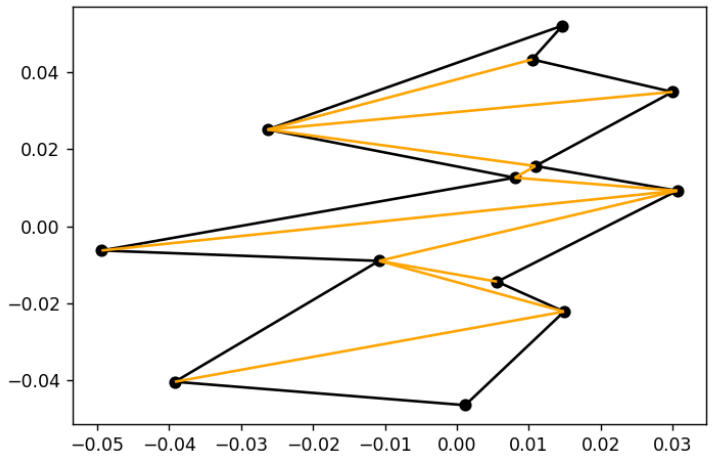
Triangulacja jest przechowywana w formie listy dodanych przekątnych (przekątna ma postać krotki zawierającej punkty będące krotkami współrzędnych). Taka realizacja pozwala na proste odtworzenie przekątnych i ich narysowanie.

Sprawdzanie, czy trójkąt należy do wielokąta polega na obliczeniu wartości wyznacznika 3x3 dla punktów należących do trójkąta i sprawdzenia jego znaku skorygowanego o informację o łańcuchu do którego należą punkty (wynik jest odwrotny dla lewego i prawego łańcucha).

Program zwraca dwa elementy- lines- dodane przekątne i scenes- sceny, które umożliwiają stworzenie wykresu odtwarzającego działanie algorytmu. Dodane przekątne mają na wykresie kolor pomarańczowy, wierzchołki znajdujące się na stosie - czerwony, a punkty, dla których sprawdzane jest, czy trójkąt, który tworzą należy do wielokąta – na niebiesko.

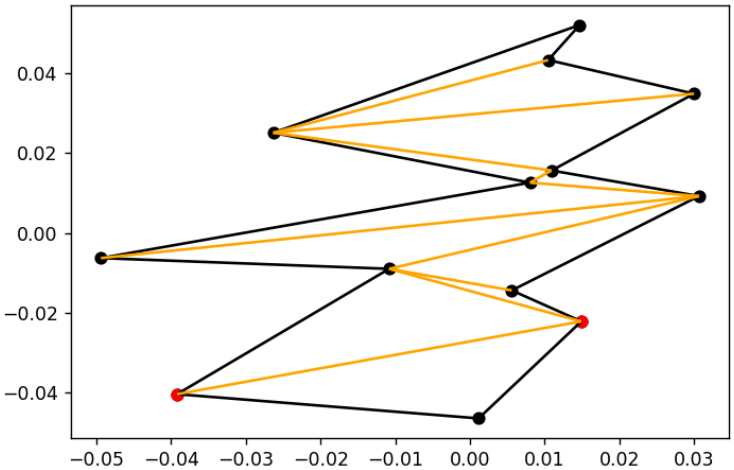
7.3. Przykładowe działanie algorytmu na danych testowych

Rys. 5. Przykładowa triangulacja 1.



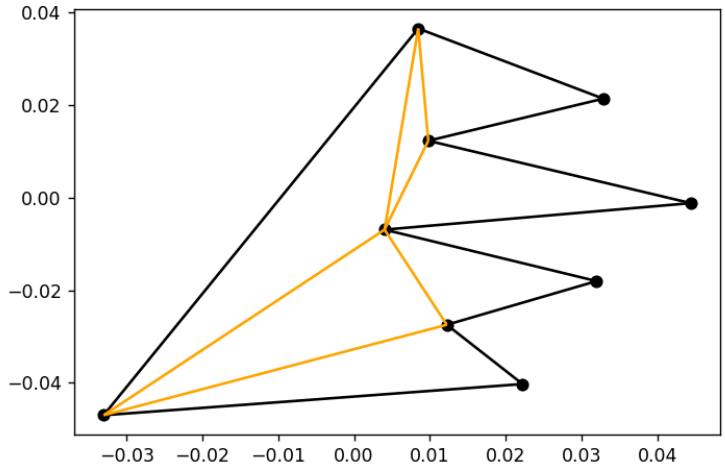
Powyższy wielokąt dobrze testuje, czy algorytm nie dodaje linii pokrywających linie wielokąta podczas łączenia najniższego wierzchołka z innymi- na koniec zarówno lewy jak i prawy sąsiad najniższego wierzchołka jest na stosie.

Rys. 6. Przedostatni krok triangulacji 1.



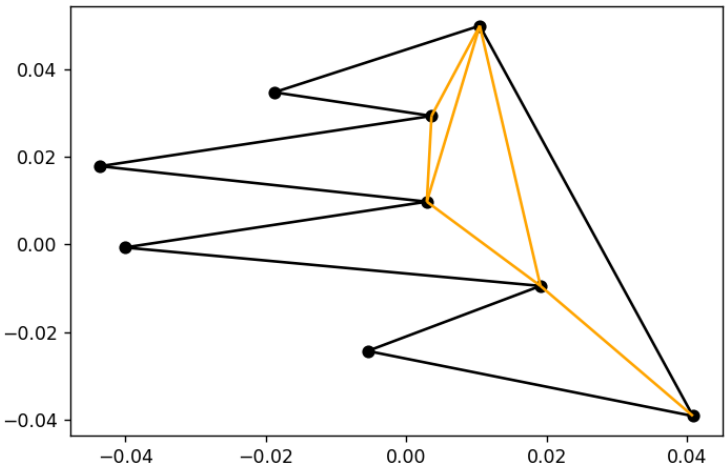
Powyższy rysunek obrazuje sytuację, przed rozpoczęciem próby łączenia najniższego wierzchołka z wierzchołkami ze stosu (na czerwono).

Rys. 7. Przykładowa triangulacja 2.



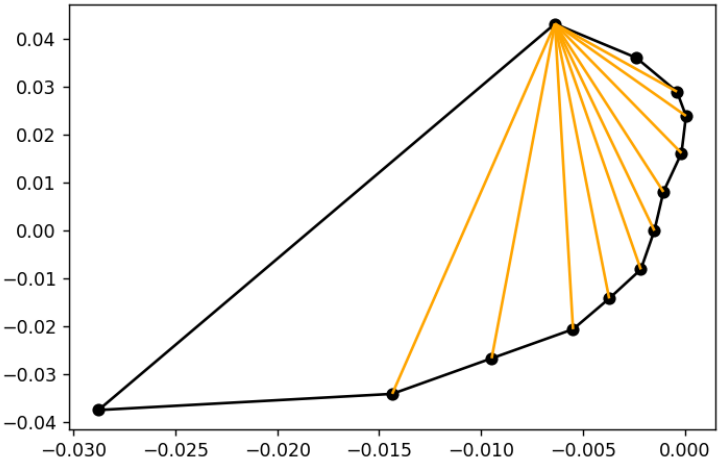
Celem tego zestawu danych było sprawdzenie, czy łączenie punktów leżących na tym samym łańcuchu (zakładając, że punkt najwyższy należy do obu) jest poprawne oraz czy nie algorytm nie generuje krawędzi już istniejących w wielokącie.

Rys. 8. Przykładowa triangulacja 3.



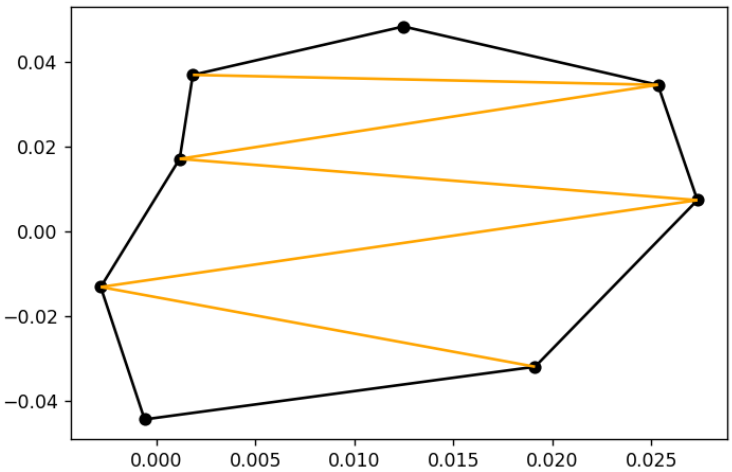
Ten zestaw miał pełnić podobną funkcję jak 2. Z tą różnicą, że teraz większość punktów należy do lewego łańcucha. Miało to na celu wykryć, czy w kodzie istnieją anomalie powodujące inne zachowanie w zależności od tego, który łańcuch jest „dominujący”.

Rys. 9. Przykładowa triangulacja 4.



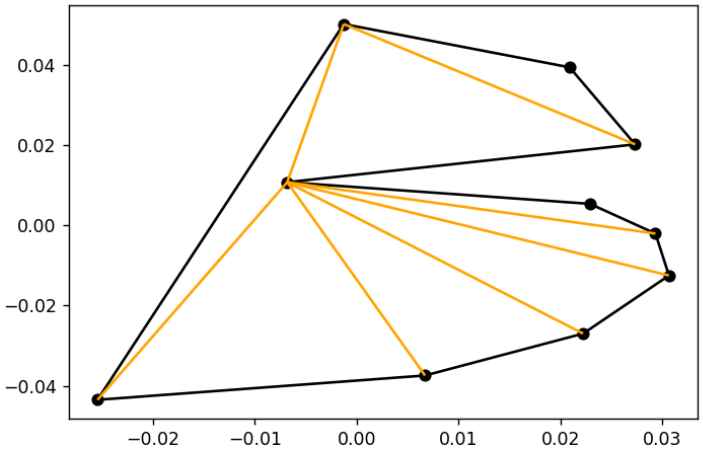
Wielokąt przedstawiony na rysunku powyżej został przetestowany, by sprawdzić czy program działa poprawnie, gdy wszystkie przekątne tworzone są z jednym punktem i z punktami z jednego łańcucha.

Rys. 10. Przykładowa triangulacja 5.



Ten zestaw danych miał na celu sprawdzenie, czy algorytm działa poprawnie, gdy przekątne powinny być tworzone tylko pomiędzy punktami należącymi do różnych łańcuchów.

Rys 11. Przykładowa triangulacja 6.



Powyższy przykład miał za zadanie sprawdzenie, czy program poprawnie połączy punkty należące do jednego łańcucha niebędące punktami najwyższymi lun najniższymi.

8. Funkcje pomocnicze

Poniżej znajduje się krótki opis funkcji użytych w programie, które nie są bezpośrednio realizacją opisanych algorytmów:

- points_from_plot - tworzy listę punktów na podstawie wykresu.

- polygon_from_plot - tworzy wielokąt na podstawie wykresu (implementacja wielokąta jest opisana w punkcie 7.2)

- det – oblicza wyznacznik 3x3

- to_lines – tworzy listę linii z listy punktów

- draw_classification – tworzy wykres klasyfikacji wierzchołków na podstawie wielokąta i list poszczególnych typów punktów.

- triangle_in_polygon – w oparciu o wyznacznik, punkty, wielokąt i łańcuch, do którego należą punkty stwierdza, czy punkt zawiera się w wielokącie.

- stack_to_points – przekształca stos ze strukturami Tri_point na listę punktów jako krotek.

- add_scenes – dodaje jeden z typów scen. Używane w funkcji triangulation.

9. Wniosek

Testowane dane nie wykazały niepoprawności w zaimplementowanych algorytmach.