Algorytmy geometryczne

Sprawozdanie z ćwiczenia 3.

Krzysztof Kwiecień
gr. Wt_14.40_B

Dane techniczne urządzenia na którym wykonano ćwiczenie:

Komputer z systemem Windows 10 x64
Procesor: Intel Core I5 2500k @4.5Ghz
Pamięć RAM: 8GB
Środowisko: Jupyter notebook

Ćwiczenie zrealizowano w języku Python 3, z wykorzystaniem bibliotek *numpy* (umożliwiających wiele operacji numerycznych), *random* (do generacji losowych punktów) oraz *matplotlib* (do rysowania wykresów).
Jako tolerancje dla zera dla wyznacznika przyjęto 10^{-12}.

Opis realizacji ćwiczenia:

Ćwiczenie polegało na implementacji algorytmów:

- Sprawdzania y-monotoniczności zadanego wielokąta
- Klasyfikacji wierzchołków w dowolnym wielokącie
- Triangulacji wielokąta y-monotonicznego wraz z wizualizacją

1. Tworzenie zbiorów danych (figur)

Aplikacja pozwala na tworzenie figur za pomocą dostarczonego narzędzia graficznego. Za pomocą myszki należy wprowadzić kolejne punkty należące do figury. Po narysowaniu figura jest zapisywana do pliku json w postaci listy krawędzi, w kolejności ich dodania.

Dodatkowo przygotowane zostały funkcje pomocnicze zamieniające figury na listę linii oraz listę lini na listę kolejnych punktów.

Figury można wczytywać z wcześniej utworzonych plików i pobierać z nich potrzebne dane oraz wyświetlać.

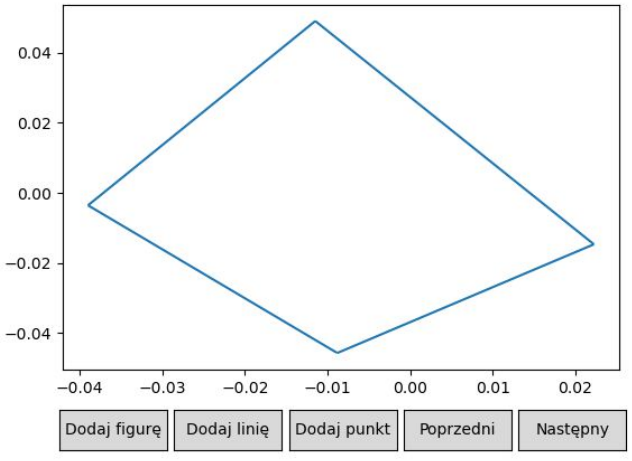
W celu przetestowania algorytmów zostały ręcznie utworzone następujące figury:

- Czworokąt (“czworokąt.json”)
- Okrąg (“okrag.json”)
- Figura niemonotoniczna przedstawiona na wykładzie (“wyklad.json”)
- Figura zaproponowana na ćwiczeniach (“cw.json”)
- Lotka tenisowa (“strzalka.json”)
- Wąska lotka zgięta przy dziobie (“rzutka.json”)
- Ptak (“ptak.json”)
- Choinka (“choinka.json”)

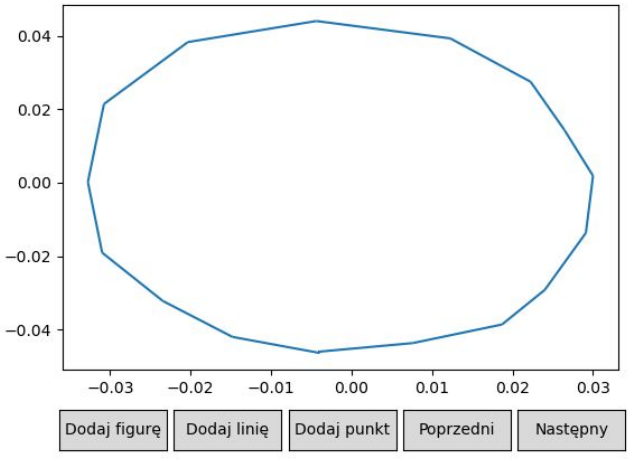
Figury ‘Czworokąt’ i ‘Okrąg’ testują podstawowe przypadki (figury wypukłe, każdy punkt “widoczny” z każdego innego). ‘Figura niemonotoniczna przedstawiona na wykładzie’ służy sprawdzeniu procedury określania y-monotoniczności. Pozostałe figury nie są wypukłe i testują “odcinanie” krawędzi z wierzchołków, niezależność działania algorytmu triangulacji od orientacji figury, zrównoważenia wielkości obu gałęzi oraz położenia punktów “ekstremalnych”, czyli takich z którymi połączonych będzie wiele innych punktów.

Wszystkie zaimplementowane w ramach tego ćwiczenia algorytmy korzystają z reprezentacji wielokątów w postaci listy wierzchołków w kolejności ich dodawania. Pozwala to na szybkie znajdowanie sąsiadów wierzchołków oraz lewej i prawej gałęzi figury.

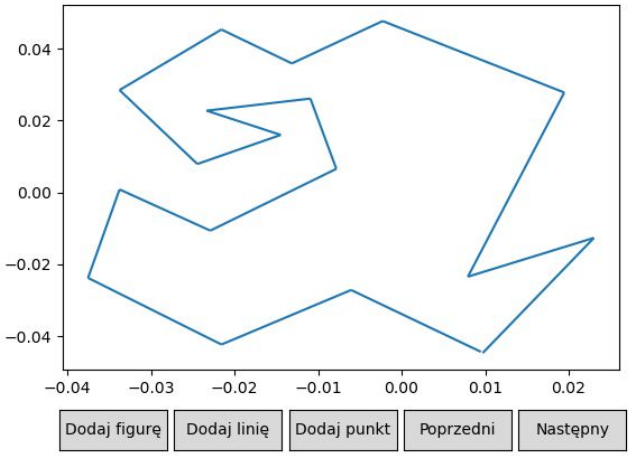
Rysunek 1.1 Figura ‘Czworokąt’



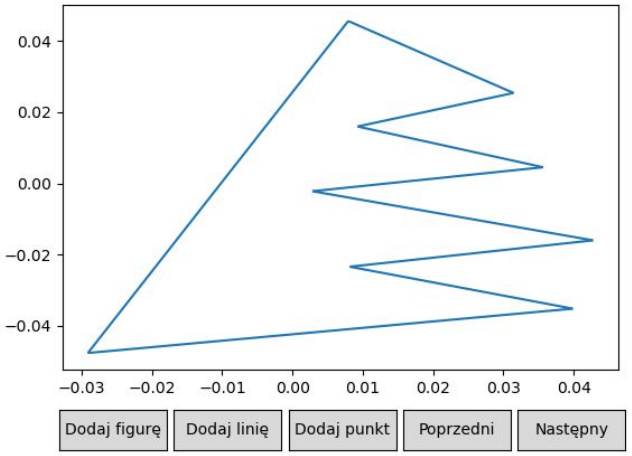
Rysunek 1.2 Figura ‘Okrąg’



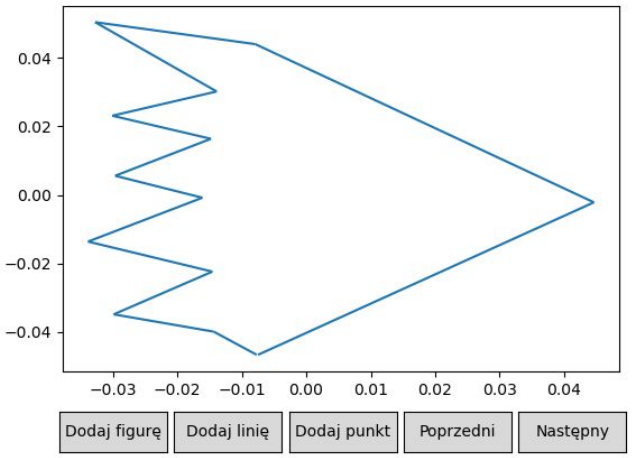
Rysunek 1.3 Figura ‘Figura niemonotoniczna przedstawiona na wykładzie’



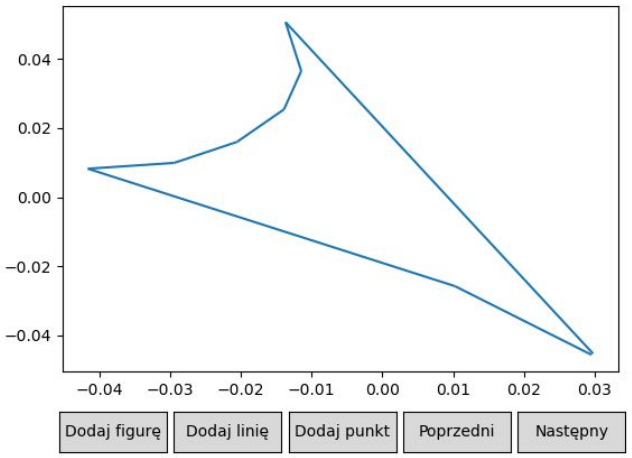
Rysunek 1.4 Figura ‘Figura zaproponowana na ćwiczeniach’



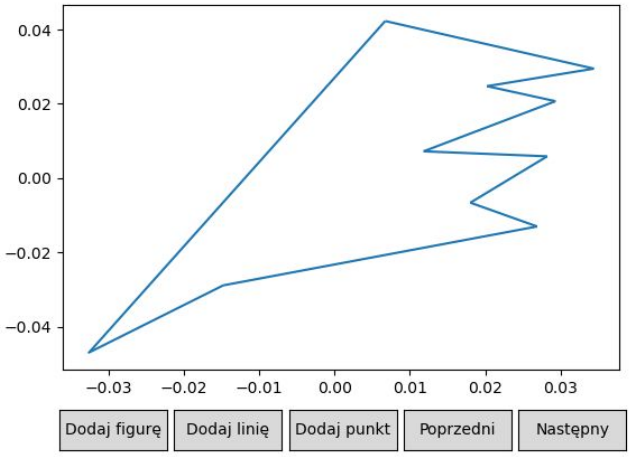
Rysunek 1.5 Figura ‘Lotka tenisowa’



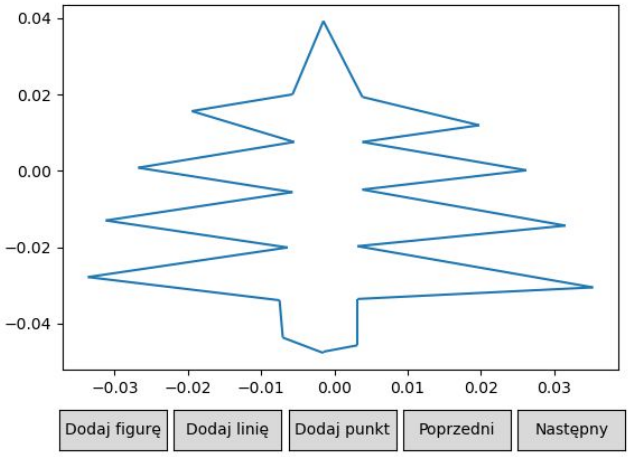
Rysunek 1.6 Figura ‘Wąska lotka zgięta przy dziobie’



Rysunek 1.7 Figura ‘Ptak’



Rysunek 1.8 Figura ‘Choinka’



2. Sprawdzanie monotoniczności

W celu sprawdzenia monotoniczności zadanego wielokąta wykorzystano kolejność punków go tworzących. Wielokąt został zamieniony na listę kolejnych punktów z których się składa, następnie znaleziono indeks punktu o największej i najmniejszej współrzędnej y. Kolejnym krokiem było sprawdzenie, czy każdy kolejny punkt idąc od najwyższego do najniższego leży poniżej wcześniejszego punktu. Warunek ten należało sprawdzić dla obydwóch gałęzi (po obu stronach wierzchołka najwyższego). W przypadku choć jednego odstępstwa od tej zasady można stwierdzić, że figura nie jest monotoniczna.

Procedura ta została przetestowana dla wszystkich zaproponowanych figur, zgodnie z oczekiwaniami jedynie figura ‘Figura niemonotoniczna przedstawiona na wykładzie’ została oznaczona jako niemonotoniczna.

3. Klasyfikacja wierzchołków

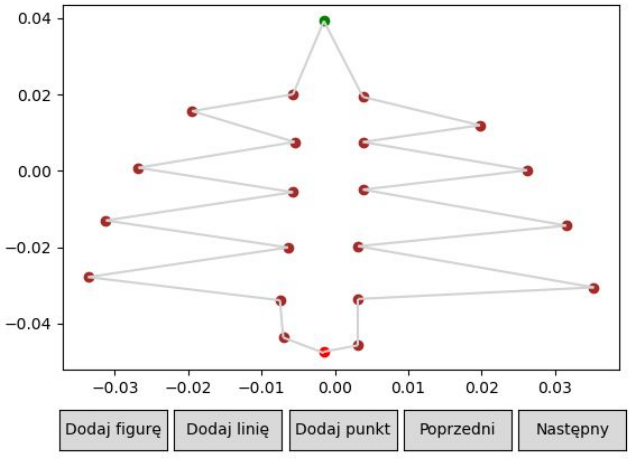
Algorytm klasyfikacji wierzchołków wielokąta na początkowe, końcowe, łączące, dzielące i prawidłowe polega na jednokrotnym przejściu po liście wierzchołków i ich klasyfikacji na podstawie ich położenia w relacji do sąsiadów (wierzchołka poprzedniego i następnego). Należało również uwzględnić to, że pierwszy i ostatni wierzchołek na liście również ze sobą sąsiadują.
W celu określenia kąta wewnętrznego jaki tworzą rozpatrywane wierzchołki użyto wyznacznika.
Punkty były klasyfikowane na podstawie następujących warunków, z następującymi oznaczeniami:

- początkowy, gdy obaj jego sąsiedzi leżą poniżej i kąt wewnętrzny < π (zielony)
- końcowy, gdy obaj jego sąsiedzi leżą powyżej i kąt wewnętrzny < π (czerwony)
- łączący, gdy obaj jego sąsiedzi leżą powyżej i kąt wewnętrzny > π (ciemny niebieski)
- dzielący, gdy obaj jego sąsiedzi leżą poniżej i kąt wewnętrzny > π (jasnoniebieski)
- prawidłowy, w pozostałych przypadkach (ma jednego sąsiada powyżej, drugiego – poniżej). (brązowy)

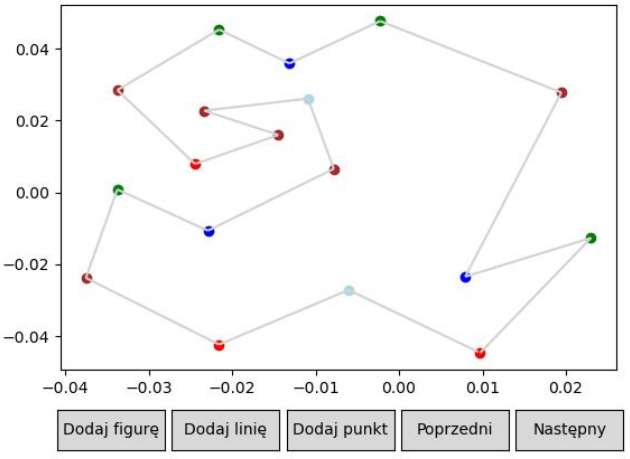
Dla wszystkich zestawów wierzchołki zostały sklasyfikowane poprawnie. Zgodnie z oczekiwaniami wielokąty monotoniczne zawierały po jednym wierzchołku początkowym (najwyższy), końcowym (najniższym), a pozostałe wierzchołki były prawidłowe.
Figura ‘Figura niemonotoniczna przedstawiona na wykładzie’ zawierała wszystkie rodzaje wierzchołków.

Na rysunku spośród wszystkich figur monotonicznych przedstawiony tylko jeden przykład, ponieważ w reszcie przypadków rezultat był bardzo podobny i opisany powyżej.

Rysunek 2.1 Klasyfikacja punktów w figurze ‘Choinka’



Rysunek 2.2 Klasyfikacja punktów w figurze ‘Figura niemonotoniczna przedstawiona na wykładzie’



4. Triangulacja

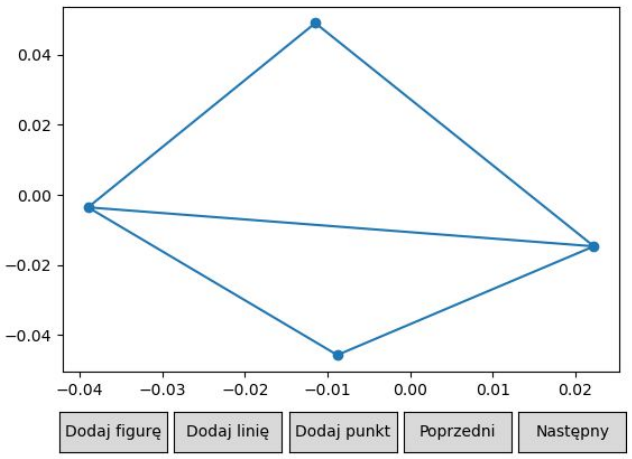
Algorytm triangulacji na wejściu przyjmuje listę punktów, będących wierzchołkami tworzącymi dany wielokąt w kolejności ich wprowadzenia, zgodnie z ruchem przeciwnym do ruchu wskazówek zegara.
Algorytm generuje listę krawędzi, która zawiera zarówno krawędzie należące do wielokąta oraz przekątne wewnętrzne, będące rezultatem triangulacji. Użycie listy krawędzi ma następujące zalety:
Krawędzie są najprostszym typem obiektów które definiują rezultat triangulacji - zapis w postaci listy trójek punktów reprezentujących trójkąty byłby trudniejszy w przeglądaniu, rysując go musielibyśmy uważać na duplikaty krawędzi, które w tym rozwiązaniu nie mają miejsca; W prosty sposób możemy znaleźć krawędzie (oraz ich ilość), które wychodzą z zadanego wierzchołka. Ponadto taka reprezentacja nie wymaga deklarowania dodatkowych struktur, jest przejrzysta oraz kompatybilna z zadanym narzędziem graficznych bez żadnych konwersji.

Poza listą krawędzi dodatkowo zwracana jest lista scen zawierających kolejne kroki procesu triangulacji (kolejno dodawane krawędzie) w celach wizualizacji.

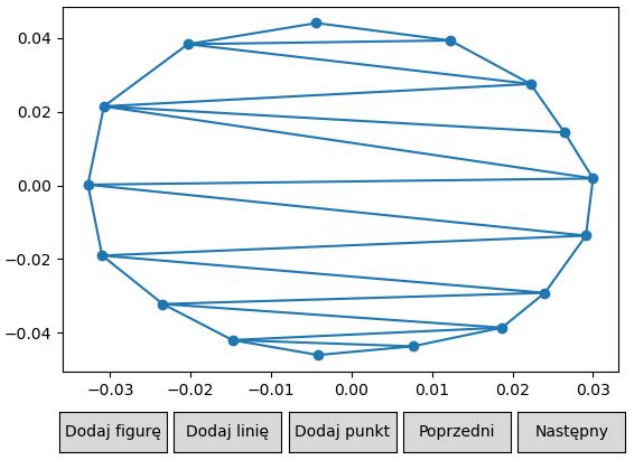
Na początku algorytm dokonuje podziału punktów na dwie gałęzie - lewą i prawą. Jest to dokonane poprzez znalezienie indeksów punktów najniższego i najwyższego. Dzięki uporządkowaniu punktów w liście w kolejności ich wstawiania punkty znajdujące się w przedziale od indeksu punktu najniższego do najwyższego stanowią jedną gałąź (w tym przypadku prawą), natomiast pozostałe drugą (lewą). Punkt najniższy znajduje się w prawej gałęzi, natomiast najwyższy w lewej. Dla każdego punktu przechowujemy informacje do której gałęzi przynależy. Następnie punkty są sortowane od najwyższego do najniższego (wg. współrzędnej y).

Przed rozpoczęciem głównej pętli na stos dodajemy dwa najwyższe wierzchołki.
Następnie wykonywana jest główna pętla, w której rozważamy pojedynczo każdy kolejny wierzchołek.
Jeśli rozważany wierzchołek znajduje się na innej gałęzi niż ostatni ze stosu, to łączymy go ze wszystkimi wierzchołkami na stosie (podczas łączenia są one zdejmowane ze stosu), po wykonaniu tej operacji na stosie umieszczane są dwa ostatnio rozważane wierzchołki.
W przypadku gdy rozważany wierzchołek znajduje się na tej samej gałęzi co wierzchołek ze szczytu stosu dodajemy krawędź między tymi dwoma wierzchołkami (jest to krawędź zewnętrzna wielokąta), a następnie podobnie jak w poprzednim przypadku łączymy rozważany punkt ze wszystkimi znajdującymi się na stosie, ale tylko pod warunkiem, że krawędź je łącząca będzie zawierała się we wnętrzu wielokąta (co sprawdzamy przy pomocy wyznacznika). Po wykonaniu tej operacji na stosie umieszczany jest rozważany wierzchołek oraz ostatni wierzchołek zdjęty ze stosu.
Po wykonaniu się głównej pętli otrzymujemy rezultat w postaci listy krawędzi wielokąta oraz utworzonych przekątnych. Poniżej przedstawiono wyniki działania algorytmu dla testowanych figur.

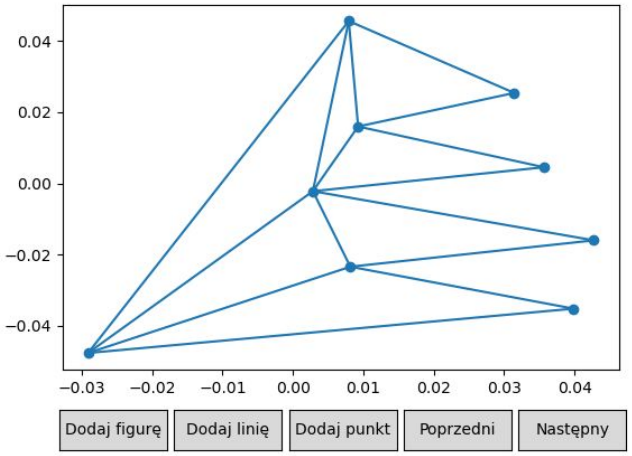
Rysunek 3.1 Wynik algorytmu triangulacji dla figury ‘Czworokąt’



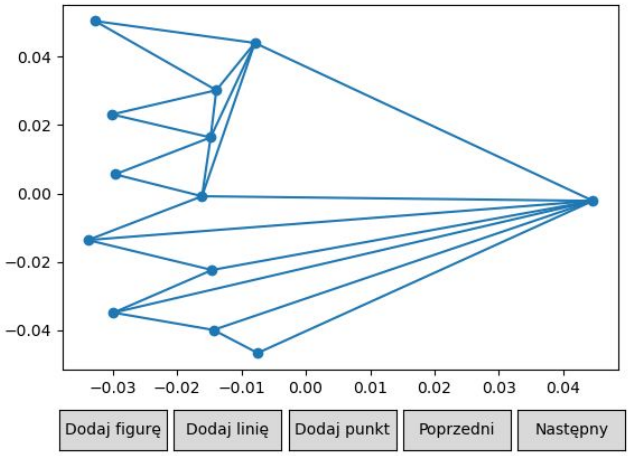
Rysunek 3.2 Wynik algorytmu triangulacji dla figury ‘Okrąg’



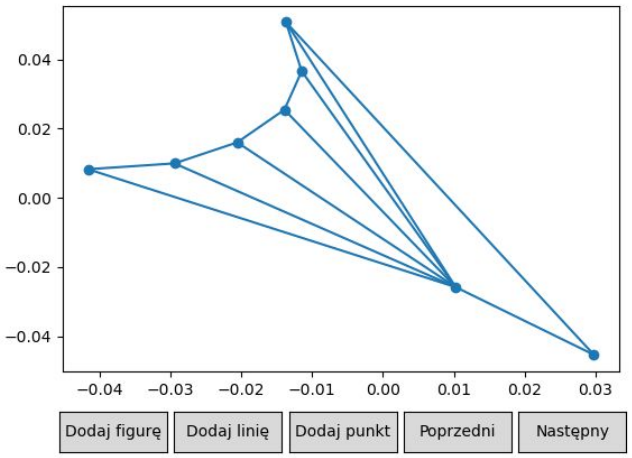
Rysunek 3.3 Wynik algorytmu triangulacji dla figury ‘Figura zaproponowana na ćwiczeniach’



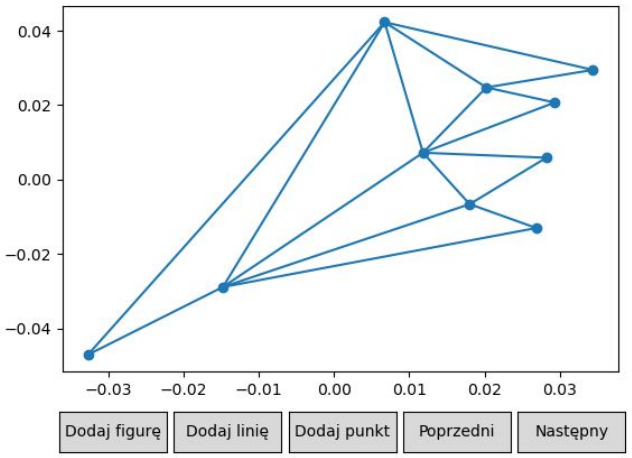
Rysunek 3.4 Wynik algorytmu triangulacji dla figury ‘Lotka tenisowa’



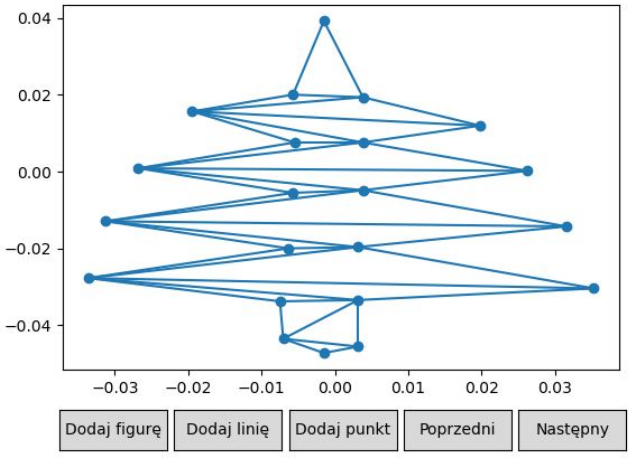
Rysunek 3.5 Wynik algorytmu triangulacji dla figury ‘Wąska lotka zgięta przy dziobie’



Rysunek 3.6 Wynik algorytmu triangulacji dla figury ‘Ptak’



Rysunek 3.7 Wynik algorytmu triangulacji dla figury ‘Choinka’

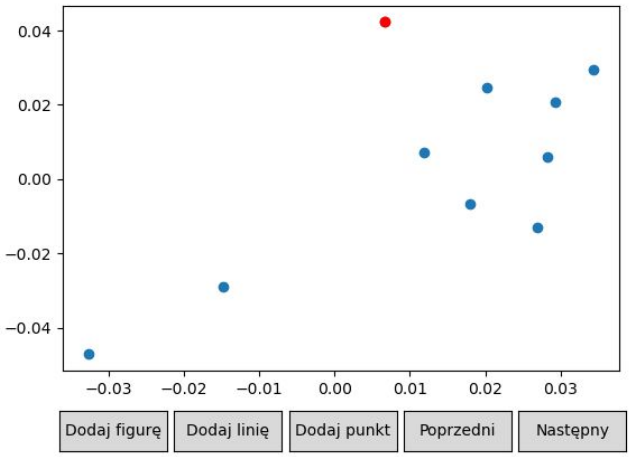


5. Wizualizacja procesu triangulacji

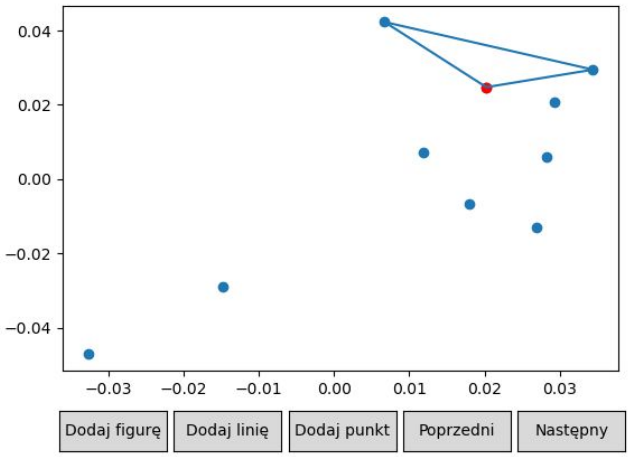
Procedura triangulacji zapisuje proces przebiegu triangulacji z dokładnością do każdej dodanej linii. W wyniku jej działania zwracana jest lista scen, które można interaktywnie przewijać z wykorzystaniem dostarczonego narzędzia graficznego. W wizualizacji punkty niebieskie to punkty należące do wielokąta, punktem czerwonym oznaczono punkt aktualnie rozpatrywany (w głównej pętli), natomiast niebieskie krawędzie to krawędzie dodane w wyniku działania algorytmu.

Poniżej przedstawiono przykład takiej wizualizacji, każdy kolejny rysunek różni się od poprzedniego dodaniem nowego trójkąta (a więc o kilka scen).

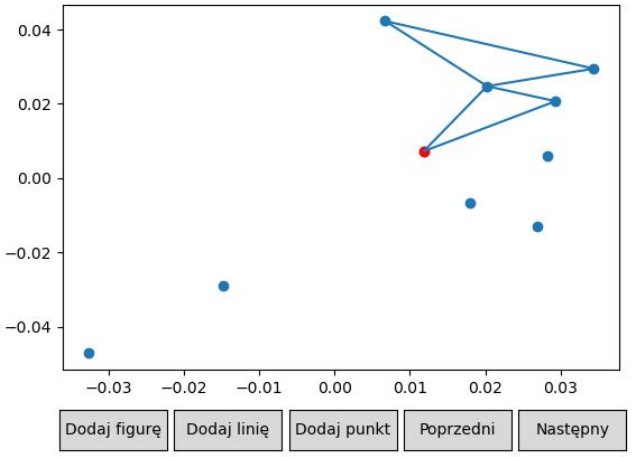
Rysunek 4.1 Punkty tworzące figurę ‘Ptak’



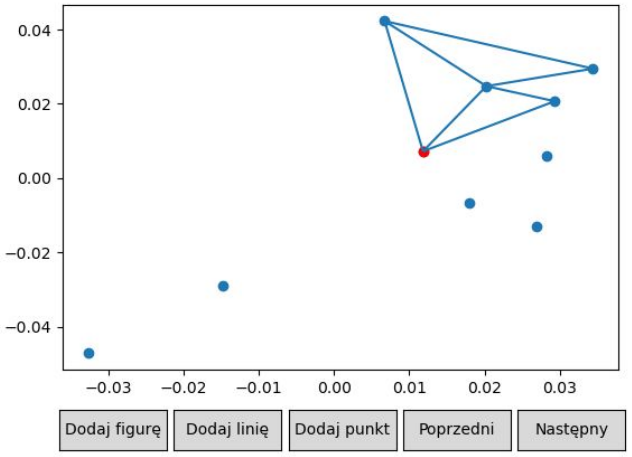
Rysunek 4.2 Stan algorytmu triangulacji po dodaniu pierwszego trójkąta dla figury ‘Ptak’



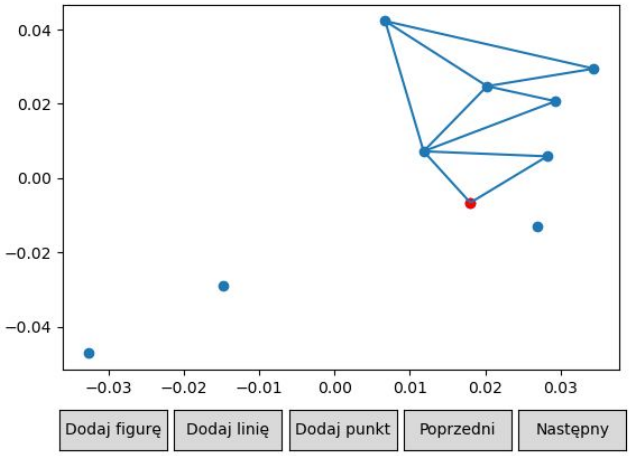
Rysunek 4.3 Stan algorytmu triangulacji po dodaniu pierwszego trójkąta dla figury ‘Ptak’



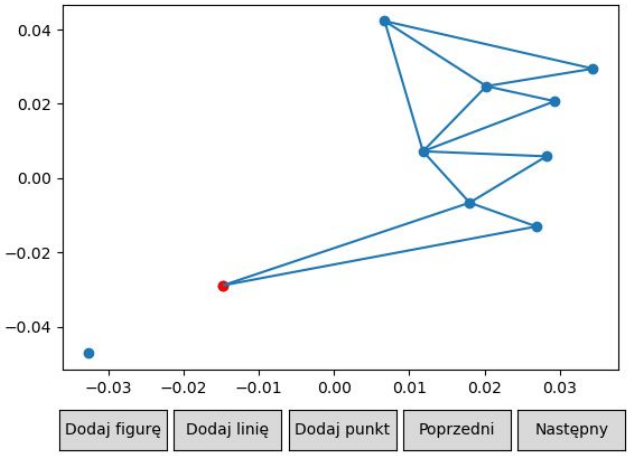
Rysunek 4.4 Stan algorytmu triangulacji po dodaniu pierwszego trójkąta dla figury ‘Ptak’



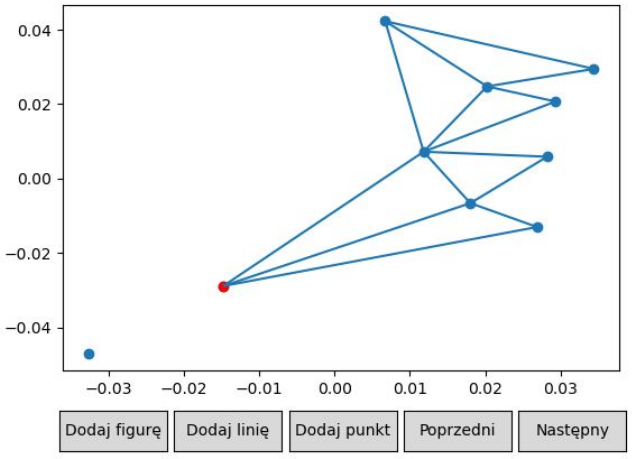
Rysunek 4.5 Stan algorytmu triangulacji po dodaniu pierwszego trójkąta dla figury ‘Ptak’



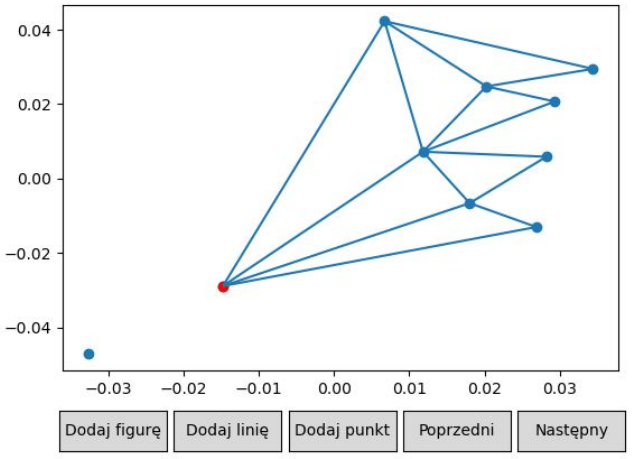
Rysunek 4.6 Stan algorytmu triangulacji po dodaniu pierwszego trójkąta dla figury ‘Ptak’



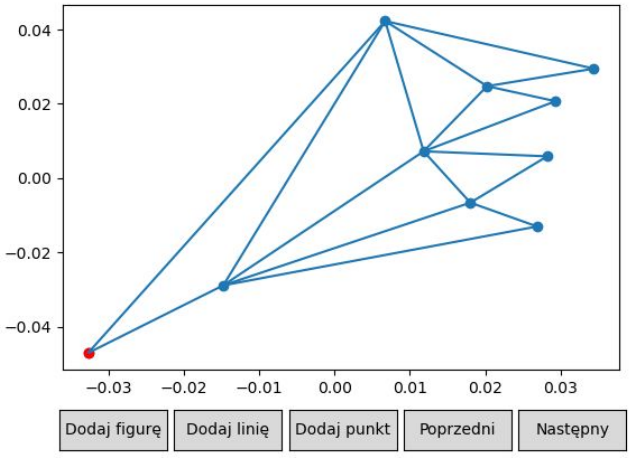
Rysunek 4.7 Stan algorytmu triangulacji po dodaniu pierwszego trójkąta dla figury ‘Ptak’



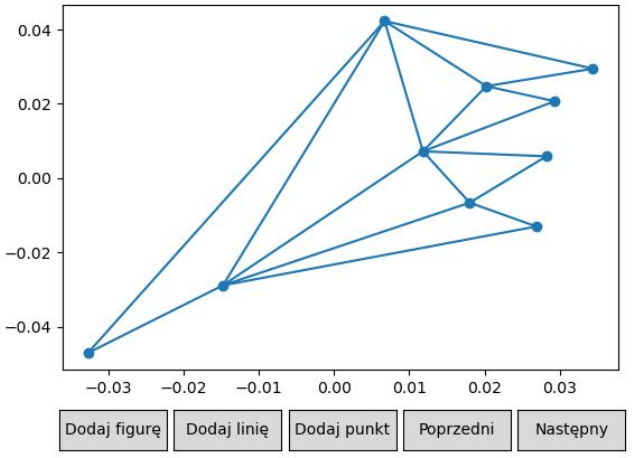
Rysunek 4.8 Stan algorytmu triangulacji po dodaniu pierwszego trójkąta dla figury ‘Ptak’



Rysunek 4.9 Stan algorytmu triangulacji po dodaniu pierwszego trójkąta dla figury ‘Ptak’



Rysunek 3.6 (powt.) Wynik algorytmu triangulacji dla figury ‘Ptak’



Wnioski i spostrzeżenia

Wszystkie zaimplementowane procedury i algorytmy zadziałały poprawnie dla testowanych wielokątów. Jedną ze zmian, jaką należało wprowadzić w algorytmie triangulacji względem pierwotnej wersji było łączenie wszystkich wierzchołków znajdujących się na stosie w przypadku gdy rozpatrywany wierzchołek należał do tej samej gałęzi co zdjęty ze stosu na początku głównej pętli. W pierwotnej wersji łączony był tylko jeden wierzchołek, co działało dla wielu testowanych wcześniej figur, lecz nie zadziałało dla figury ‘Wąska lotka zgięta przy dziobie’. Konieczność takiej zmiany jest ewidentna po przeanalizowaniu kroków działania algorytmu dla tej figury, ponieważ idąc cały czas wzdłuż jednej gałęzi rysujemy krawędzie zewnętrzne wielokąta; W przypadku, gdy gałąź ta będzie wklęsła do jego wnętrza nie uda nam się utworzyć żadnych trójkątów do momentu, aż dojdziemy do wierzchołka leżącego naprzeciw “wgięcia” tej gałęzi. W momencie dojścia do takiego wierzchołka oczywista jest konieczność połączenia go ze wszystkimi wierzchołkami na stosie, aby utworzyć brakujące trójkąty. Poprawność triangulacji można było łatwo zweryfikować korzystając z własności wielokątów prostych, mówiącej że dla wielokąta o n wierzchołkach efektem triangulacji będzie n-2 trójkątów. Ponadto algorytm poprawnie nie generuje duplikatów krawędzi. Użyte struktury danych doskonale pasowały do zadanych problemów - pozwalały na szybki dostęp do potrzebnych elementów, nie wymagały dodatkowych konwersji (poza pierwotną - z narysowanej figury) oraz były bezpośrednio kompatybilne z narzędziem do wizualizacji. Możliwość zapisu i odczytu figur z pliku w postaci listy krawędzi znacząco ułatwia korzystanie i testowanie aplikacji dla różnych przypadków.