Kraków, 26.11.2020 r.

Paweł Kruczkiewicz,

Algorytmy geometryczne,

Grupa nr 8 (czwartek 16:15, tydzień B)

Sprawozdanie nr 3

Temat: Triangulacja wielokątów monotonicznych

Cel ćwiczenia

Zapoznanie się z zagadnieniami monotoniczności wielokątów, a w szczególności z algorytmem sprawdzania, czy dany wielokąt jest monotoniczny, algorytmem klasyfikacji wierzchołków w dowolnym wielokącie, algorytmem triangulacji wielokąta monotonicznego.

Wstęp teoretyczny

Triangulacja to jedno z ważniejszych pojęć w geometrii obliczeniowej. Intuicyjnie można rozumieć triangulację jako podział wnętrza wielokąta na trójkąty, takie, aby żadne na siebie nie nachodziły, a wspólnie tworzyły wnętrze owego wielokąta.

W tym sprawozdaniu omówimy algorytm podziału wierzchołków, przydatny przy triangulacji niemonotonicznych wielokątów, oraz liniowy algorytm triangulacji wielokątów ściśle y-monotonicznych[1].

Specyfikacja

Doświadczenie przeprowadzono na 64-bitowym systemie Windows 10 na 4-rdzeniowym procesorze firmy Intel o taktowaniu zegara 2.30 GHz. Kod oraz obliczenia wykonano w Jupyter Notebooku. Wersja pythona to 3.8. Precyzja obliczeń jest ograniczona przez zapis 64-bitowej liczby zmiennoprzecinkowej (odpowiednik typu double w językach C).

Jako wyznacznika użyto samodzielnie zaimplementowanego wyznacznika 3x3 o dokładności $\varepsilon = 10^{-13}$. Wyznacznik ten osiągnął najlepsze wyniki w testach opisanych w poprzednim sprawozdaniu.

Plan ćwiczenia

1. Dostosowano aplikację graficzną tak, by można było zadawać proste wielokąty przy użyciu myszki.

[1] Wielokąt nazywamy monotonicznym względem prostej *l*, 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 niniejszym sprawozdaniu bierzemy pod uwagę jedynie monotoniczność względem prostej OY. Można więc intuicyjnie rozumieć ścisłą y-monotoniczność tak, że gdy przechodzimy po kolejnych wierzchołkach w łańcuchu, to współrzędna *y* jest malejąca.

2. Zaimplementowano procedurę podziału wierzchołków na początkowe, końcowe, łączące, dzielące i prawidłowe oraz dostosowano program graficzny do pokazywania takich wielokątów.
3. Na bazie powyższego zaimplementowano algorytm sprawdzający y-monotoniczność wielokąta.
4. Zaimplementowano procedurę triangulacji wielokąta y-monotonicznego wraz z wizualizacją procesu.
5. Przetestowano program dla różnych zestawów danych.

Oznaczenia graficzne

Dla podziału wielokątów

Zielony – wierzchołek początkowy, **czerwony** – końcowy, **żółty** – łączący, **niebieski** – dzielący, **szary** – prawidłowy

Dla triangulacji

Niebieski – nieprzetworzony, **żółty** – na stosie, **czerwony** – przetwarzany, **zielony** – przetworzony.

Używane struktury przechowywania wierzchołków

W przeciwieństwie do algorytmu przedstawionemu m. in. w *Computational geometry* M. Berga[2], nie użyto w implementacji tzw. list DCEL (*Doubly Connect Edge List*), gdyż w przypadku języka Python poręczniej było używać jedynie dwuelementowych krotek reprezentujących punkty, uporządkowane w liście, która reprezentowała punkty wielokąta uporządkowane w ruchu przeciwnym do wskazówek zegara[3]. Poprzednika i następnika można wtedy łatwo znaleźć po indeksie przetwarzanego elementu.

Drobną modyfikację wprowadzono jednak przy implementacji algorytmu triangulacji – wielokąt przechowywano jako kolejne wartości typu (*punkt, nazwa łańcucha*) (rys. 1). Informacja o tym, z którego wierzchołka pochodzi wierzchołek jest kluczowa dla poprawnego działania algorytmu.

```
((x1, x2), 'chain_name')  #wzór
((0,1), 'left')           #przykład
```

Rys. 1 – przykładowy zapis punktu w tabeli przechowującej wielokąt w algorytmie triangulacji

Algorytm podziału wierzchołków

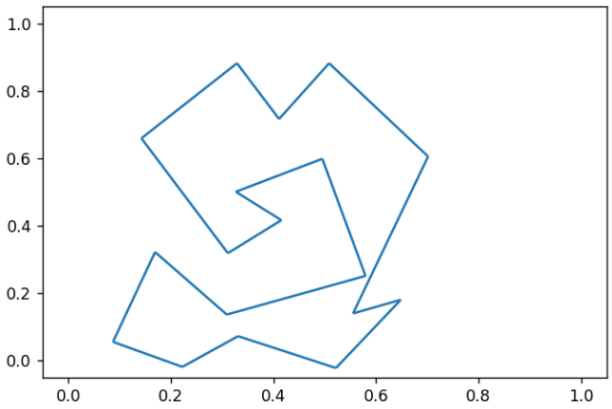
Algorytm podziału jest stosunkowo prosty. Dla każdego wierzchołka obliczono jego stosunkowe położenie względem sąsiadów oraz kąt wewnętrzny na bazie wyznacznika tego oraz sąsiadujących wierzchołków. Następnie przyporządkowano go do odpowiedniej grupy na bazie tych wyników.

Wizualizację dla wielokąta podobnego do przedstawionego na wykładzie, przedstawiono poniżej (rys. 2 i 3).

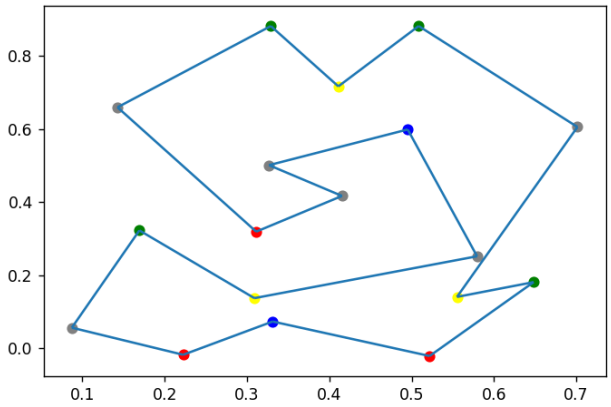
---

[2] Mark de Berg, Otfried Cheong, Marc van Kreveld, Mark Overmars *Computational Geometry Algorithms and Applications* Third Edition, wyd. *Springer*, ISBN 978-3-540-77973-5, str. 57

[3] Przeciwnie do ruchu wskazówek zegara – zgodnie z założeniami przyjętymi w poleceniu.



Rys. 2 – wejściowy wielokąt



Rys. 3 – wielokąt z podzielonymi wierzchołkami

Sprawdzanie y-monotoniczności

Do sprawdzenie Y-monotoniczności korzystamy z faktu, że Y-monotoniczny wielokąt nie posiada wierzchołków dzielących i łączących. Uruchamiamy wyżej przedstawiany algorytm i sprawdzamy, czy w tablicy wynikowej występują wspomniane wierzchołki. Jeżeli tak, zwracamy Fałsz, w przeciwnym wypadku Prawdę.

Dla przykładu powyższy wielokąt nie jest Y-monotoniczny, ponieważ posiada wierzchołki dzielące i łączące.

Triangulacja

Prosty algorytm triangulacji Y-monotonicznych wielokątów opiera się na użyciu stosu. Upraszczając – dajemy na niego te wierzchołki, które uznajemy za „potencjalne" punkty podziału. Jako wynik zwraca listę przekątnych wielokąta składających się na jego triangulację.

Algorytm przetwarza każdy wierzchołek idąc od najwyższego punktu (punkty są już posortowane względem współrzędnej y malejąco). Następnie na stos wrzucamy dwa pierwsze punkty. Przechodzimy po reszcie punktów. Dla każdego rozpatrzamy dwa przypadki:

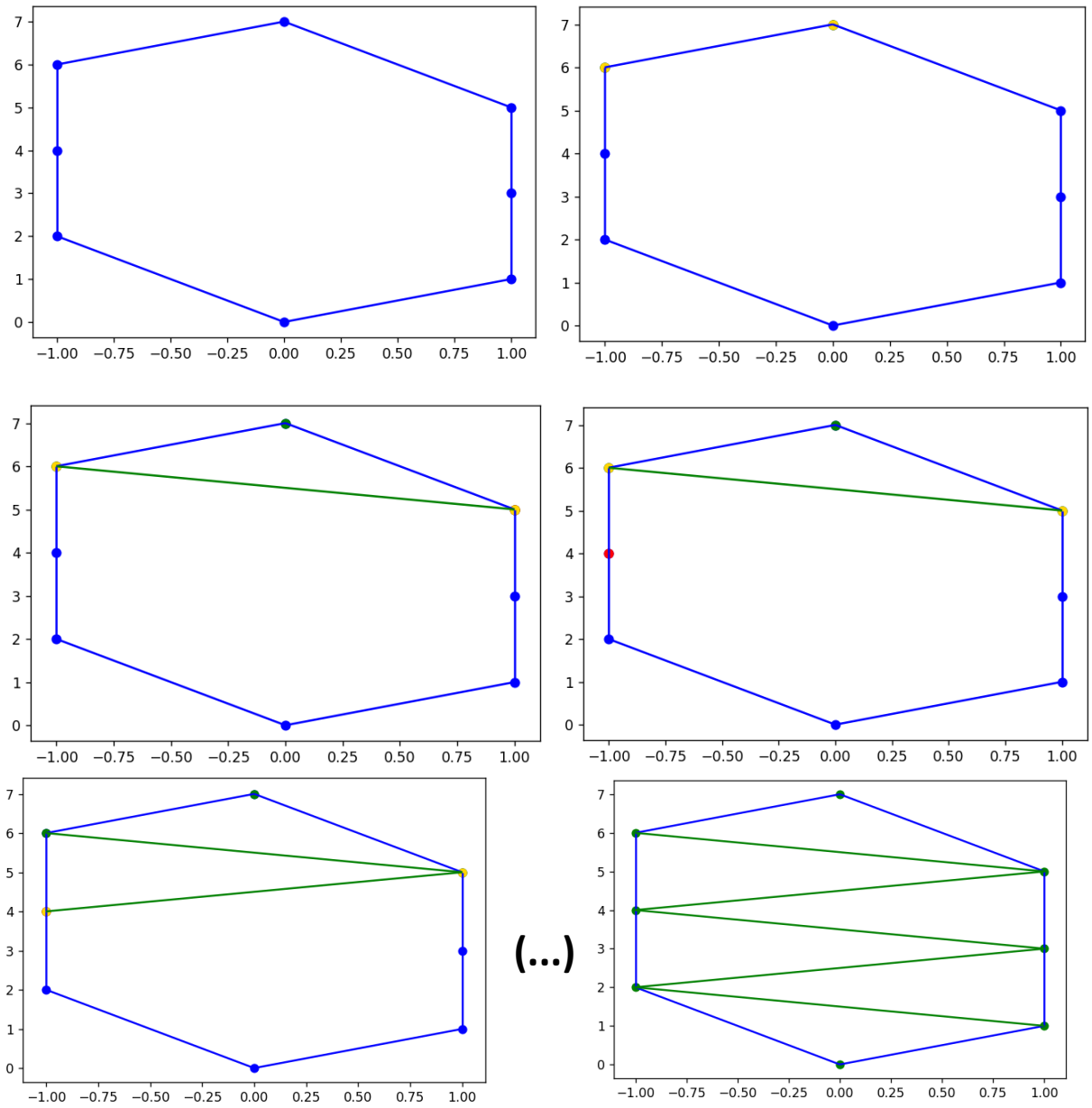
1. Przetwarzany wierzchołek znajduje się na innym łańcuchu niż wierzchołek ze szczytu stosu – wtedy dodajemy do tablicy wynikowej linie złożone z przetwarzanego wierzchołka oraz każdego z wierzchołków na stosie. Do opróżnionego stosu dodajemy poprzednik wierzchołka oraz przetwarzany wierzchołek.
2. Gdy znajduje sie na tym samym sprawdzamy, czy trójkąt stworzony z tego i dwóch wierzchołków na górze stosu znajduje się wewnątrz wielokąta. Jeśli tak, tworzymy linie od tego wierzchołka do ostatniego wyciągniętego wierzchołka ze stosu, dodajemy tę linię do tablicy wynikowej i wyciągamy kolejny wierzchołek, aż trójkąt nie będzie w wielokącie. Jeśli nie, wkładamy go na stos

Jest to algorytm liniowy, ponieważ scalanie dwóch łańcuchów w jeden (które już są posortowane względem współrzędnej *y*), odbywa się w czasie liniowym, a wszystkie opereracja dla przetwarzanych wierzchołków wykonywane są w czasie O(1). Zatem asymptotycznie program wykonuje się w czasie O(n)

Działanie algorytmu przedstawiono na poniższych wielokątach.

Wielokąt 0

Wielokąt zero jest zbiorem trywialnym – przedstawia działanie algorytmu w najprostszym przypadku – kiedy każdy kolejny wierzchołek znajduje się na przeciwległym łańcuchu. Tutaj wybrano ośmiokąt jak na rysunku niżej



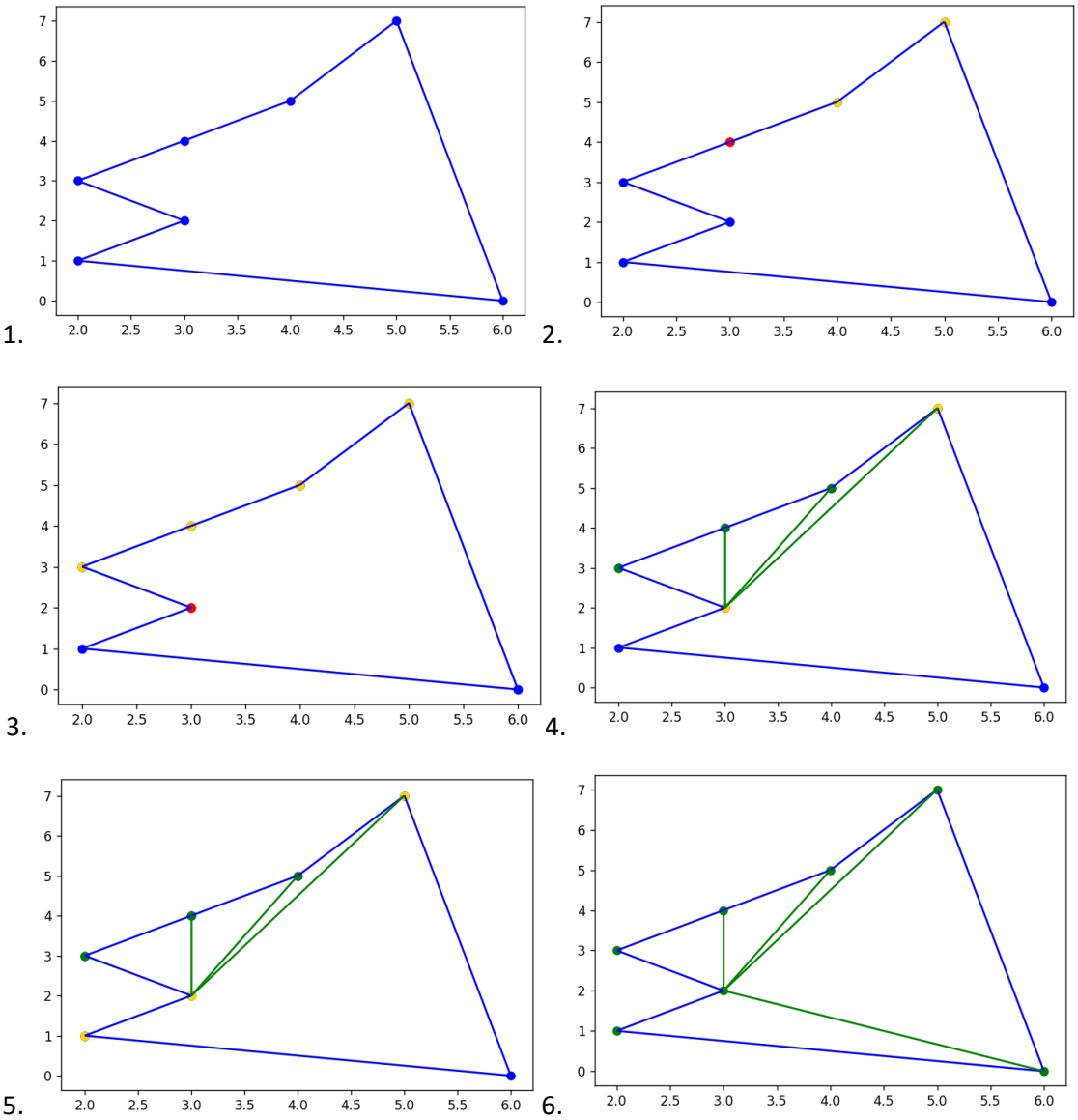
Rys. 4 – przedstawienie działania algorytmu triangulacji dla wielokąta 0

Jak widać, najpierw algorytm dodaje dwa pierwsze wierzchołki do stosu, następnie sprawdza na którym łańcuchu względem wierzchołka najwyżej na stosie znajduje się przetwarzany punkt. Następnie go dodaje. Sytuacja powtarza się aż do przetworzenia wszystkich wierzchołków aż do ostatniego.

Wielokąt 1

Wielokąt podobny do przedstawionego na wykładzie posiada wszystkie punkty na lewym łańcuchu . Jest dobrą ilustracją działania algorytmu, gdyż zawiera wszystkie możliwe przypadki, jakie mogą się

zdarzyć w trakcie przetwarzania punktu o tym samym łańcuchu. Fragment działania algorytmu przedstawiono na rysunku 5.



Rys. 5 – przedstawienie działania algorytmu triangulacji na wielokącie 1

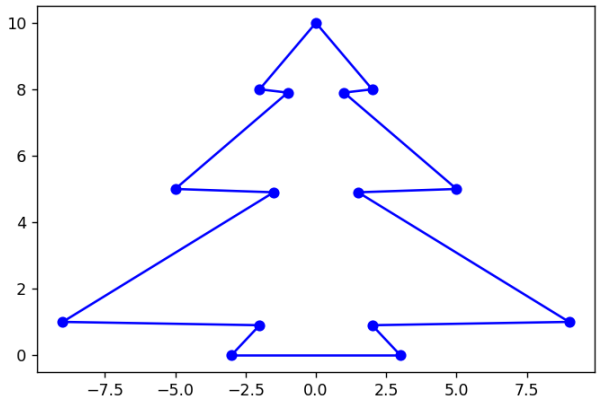
Algorytm do daje dwa punkty wielokąta (przedstawionego na rys 5.1) dwa punkty na stos i przetwarza 3. punkt (5.2). Nie mieści się on w wielokącie, więc zostaje dodany na stos. Podobnie dodany zostaje kolejny punkt. Punkt 5. (Czerwony na 5.3) posiada trójkąt należący do trójkąta, więc linia między nim a 3. Punktem zostaje utworzona podobnie jak do wszystkich poprzedzających go punktów (gdyż spełniają one warunek o zawieraniu się w wielokącie) (rys. 5.4). Warto zauważyć, że do stosu dodano również ostatni wierzchołek, do którego dorysowano linię.

Kolejny punkt znajduje nie spełnia warunku zawierania się w wielokącie, więc zostaje dodany do stosu. Na koniec łączymy ostatni wierzchołek z wszystkimi punktami na stosie oprócz pierwszego i ostatniego.

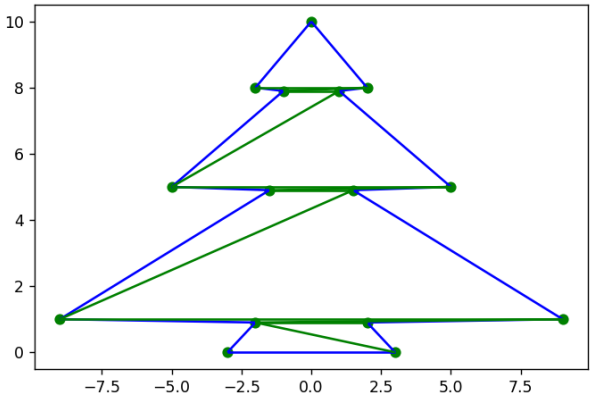
Wielokąt 2

Na koniec sprawdzono algorytm triangulacji dla zbioru, który wygląda jak choinka. Wybrano go z dwóch powodów – pierwszym jest symetryczność względem osi OY, a co za tym idzie wielokąt posiada po 2 wierzchołki o tych samych współrzędnych *y*. Znajdują się jednak na przeciwległych łańcuchach, więc wielokąt jest ciągle ściśle monotoniczny. Drugim powodem jest czas – nim prześlę to sprawozdanie do oceny będzie już grudzień. „Choinka" stanowi zatem delikatne przypomnienie, że czas świąteczny coraz bliżej i pora przestać zajmować się algorytmami, a pomyśleć o prezentach dla najbliższych.

Działanie algorytmu przedstawiono na poniższym rysunku (rys 6).

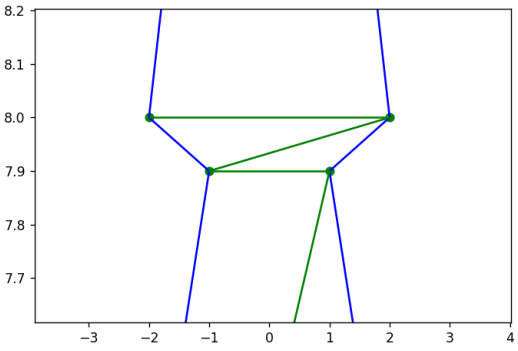


Rys. 6.1 – choinka „naga"



Rys. 6.2 – choinka „udekorowana"

Zauważmy, że „ząbki" choinki nie mogą być na tym samym poziomie co „wcięcia" – wtedy algorytm wskazałby niewłaściwą triangulację. Ponieważ jednak „wcięcia" są niżej niż ząbki, algorytm działa poprawnie. Chociaż może się wydawać, że przekątne w pobliżu wcięć nachodzą na siebie, jest to mylne wrażenie, co potwierdza rys. 7.

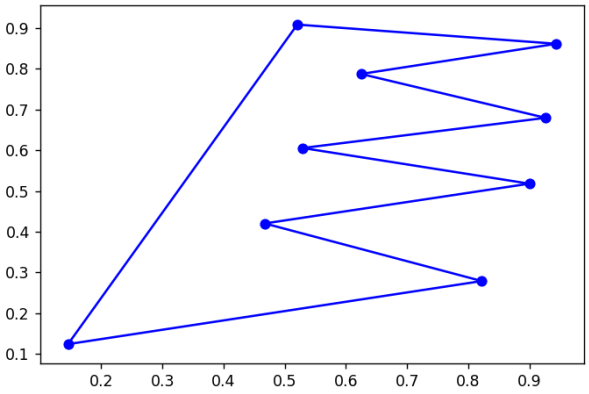


Rys 7 – przybliżenie na pierwsze „wcięcie" choinki. Linie nie przecinają się.

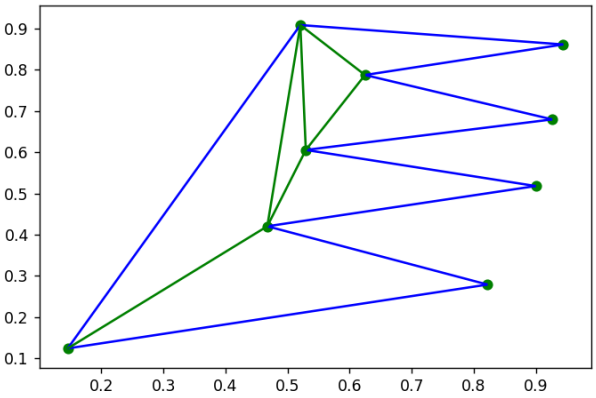
Kolejnym wartym uwagi faktem jest to, że mamy tutaj remis pod względem ostatniego wierzchołka. W implementacji nie jest rozważane, który w takim wypadku wierzchołek wybrać. Nie ma to jednak większego znaczenia dla poprawności algorytmu, gdyż takich wierzchołków może być co najwyżej dwa i algorytm nigdy nie wybierze dla nich wspólnej linii.

Wielokąt 3

Dla sprawdzenia poprawności algorytmu, w szczególności czy triangulacja prawego łańcucha działa poprawnie, striangulizowano wielokąt o „poszarpanym” prawym łańcuchem. Przedstawiono to na rysunku 8.



Rys. 8.1 – wejściowy wielokąt



Rys. 8.2 – triangulacja owego wielokąta

Jak widać – algorytm poprawnie wyznaczył triangulację wielokąta również dla prawego łańcucha.

Wnioski

Powyższe algorytmy to proste w implementacji oraz szybkie algorytmy liniowe, dzięki którym można łatwo rozwiązać trudniejsze zagadnienia jak np. problem monitorowania galerii. Minusem jednak algorytmu triangulacji jest jego ograniczoność – stosuje się go jedynie do monotonicznych wielokątów, co przy praktycznym użyciu (np. w grafice komputerowej) może nie zawsze być wystarczające.