Sprawozdanie z laboratorium 3. z algorytmów geometrycznych - Triangulacja wielokątów monotonicznych

Hubert Miklas
Grupa 5, Poniedziałek 15:00-16:30
Data wykonania ćwiczenia:
13-11-2024
Data oddania sprawozdania:
26-11-2024

1 Wstęp do ćwiczenia

1.1 Specyfikacja środowiska

System: Windows 10 Home
Procesor: Intel(R) Core(TM) i5-7400 CPU 3.00 GHz
Pamięć RAM: 24 GB
Karta Graficzna: NVIDIA GeForce GTX 1060
Środowisko: Jupyter Notebook, Python 3.12.7

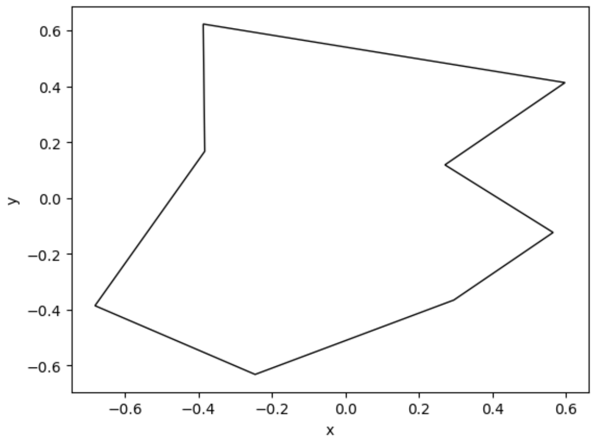
1.2 Opis i cel ćwiczenia

Celem ćwiczenia jest wygenerowanie różnych wielokątów monotonicznych na płaszczyźnie oraz wyznaczenie ich triangulacji.

2 Wstęp teoretyczny

2.1 Wielokąt y-monotoniczny

Wielokąt jest **y-monotoniczny**, jeżeli dla każdej linii poziomej przecina on tę linię w co najwyżej dwóch punktach. Na **Rysunku 1** przedstawiono przykładowy wielokąt y-monotoniczny, który jednocześnie nie jest x-monotoniczny.



Rysunek 1: Przykładowy wielokąt y-monotoniczny (nie jest x-monotoniczny)

2.2 Triangulacja

Triangulacja na płaszczyźnie to podział wielokąta na trójkąty (sympleksy) w taki sposób, że część wspólna dowolnych dwu różnych trójkątów jest ich wspólną krawędzią, wspólnym wierzchołkiem albo zbiorem pustym. Od sympleksów tworzących triangulację wymaga się ponadto, by dowolny obszar ograniczony przecinał tylko skończoną ich liczbę.

2.3 Sprawdzanie orientacji

Do wyznaczania, czy przekątna należy do wielokąta służy wyznacznik 2×2 z **Równania (1)**.

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

Oczywiście, dla lewej i prawej strony warunek nie jest taki sam, dlatego należy weryfikować to następująco:

$$side \times det(current, stack[-1], stack[-2]) > \epsilon \tag{2}$$

Gdzie *side* jest z góry narzucone przez kierunek, w którym wprowadzany jest wielokąt. W przypadku tego ćwiczenia jest to kierunek przeciwny do ruchu wskazówek zegara. W takim razie dla prawego łańcucha (względem najwyższego i najniższego punktu) $side = 1$ i dla lewego łańcucha $side = -1$. Stała $\epsilon = 10^{-18}$ jest przyjętą tolerancją dla zera, aby wykluczyć ewentualne punkty współliniowe.

3 Funkcje pomocnicze

3.1 Sprawdzanie, czy wielokąt jest y-monotoniczny

Funkcja `is_y_monotonic(polygon)` pozwala określić, czy podany wielokąt jest y-monotoniczny, co oznacza, że każdy poziomy przekrój wielokąta przecina go w co najwyżej dwóch punktach.

Działanie funkcji opiera się na następujących krokach:

1. Znalezienie wierzchołków o najmniejszej i największej współrzędnej y.
2. Weryfikacja porządku malejących współrzędnych y od wierzchołka maksymalnego do minimalnego.
3. Weryfikacja porządku rosnących współrzędnych y od wierzchołka minimalnego do maksymalnego.

Jeżeli w obu przypadkach wartości y zmieniają się zgodnie z opisanym porządkiem, funkcja zwraca `True`. W przeciwnym razie zwraca `False`.

3.2 Klasyfikacja wierzchołków wielokąta

Funkcja `color_vertex(polygon)` klasyfikuje wierzchołki wielokąta na podstawie ich geometrii względem sąsiadujących punktów. Każdy wierzchołek otrzymuje jedną z poniższych kategorii:

- **0 - wierzchołek początkowy**: obaj jego sąsiedzi leżą poniżej i kąt wewnętrzny ma mniej niż 180 stopni. To wierzchołki, w których zaczyna się monotoniczny spadek. Oznaczono je kolorem zielonym.
- **1 - wierzchołek końcowy**: obaj jego sąsiedzi leżą powyżej i kąt wewnętrzny ma mniej niż 180 stopni. To wierzchołki, w których monotoniczność wielokąta się zmienia, czyli na przykład zaczyna się monotoniczny wzrost, jeśli wcześniej był spadek, lub na odwrót. Oznaczono je kolorem czerwonym.
- **2 - wierzchołek łączący**: obaj jego sąsiedzi leżą powyżej i kąt wewnęntrzny ma więcej niż 180 stopni. To wierzchołki, które są połączone liniami (przekątnymi) wewnątrz wielokąta, tworząc trójkąty. Oznaczono je kolorem niebieskim.
- **3 - wierzchołek dzielący**: obaj jego sąsiedzi leżą poniżej i kąt wewnęntrzny ma więcej niż 180 stopni. To wierzchołki, które wyznaczają przekątne (linie łączące), tworzące trójkąty podczas triangulacji. Oznaczono je kolorem jasno niebieskim.
- **4 - wierzchołek prawidłowy**: nie spełnia warunków żadnej z powyższych kategorii. Oznaczono je kolorem brązowym.

Opis działania algorytmu:

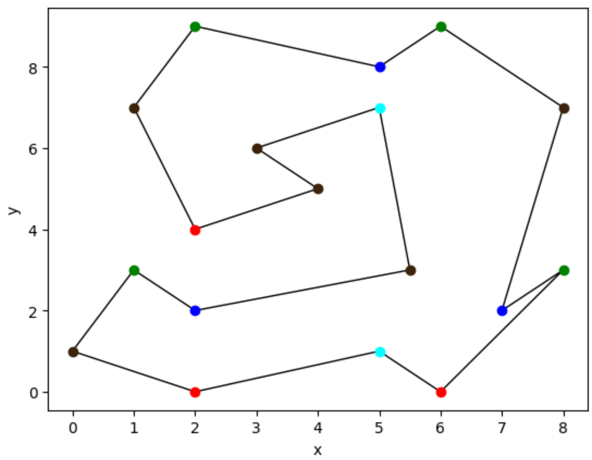
1. Iteracja po wierzchołkach:
   - Dla każdego wierzchołka `current_point` wyznacz poprzedni (`previous_point`) oraz następny (`next_point`) wierzchołek na podstawie indeksów cyklicznych.

2. Obliczanie wypukłości lub wklęsłości:
   - Używając funkcji `angle(a, b, c)`, oblicz wartość iloczynu wektorowego między wektorami ($a \to b$) oraz ($b \to c$), aby określić, czy kąt jest wypukły (`angle > 0`) czy wklęsły (`angle < 0`).

3. Klasyfikacja wierzchołka:
   - Jeśli kąt jest wypukły (`angle > E`): - Jeśli współrzędne y wierzchołka są mniejsze niż sąsiednich punktów, to wierzchołek jest końcowy (**1**). - Jeśli współrzędne y wierzchołka są większe niż sąsiednich punktów, to wierzchołek jest początkowy (**0**).
   - Jeśli kąt jest wklęsły (`angle < -E`): - Jeśli współrzędne y wierzchołka są mniejsze niż sąsiednich punktów, to wierzchołek jest łączący (**2**). - Jeśli współrzędne y wierzchołka są większe niż sąsiednich punktów, to wierzchołek jest dzielący (**3**). - W przeciwnym razie wierzchołek jest prawidłowy (**4**).

4. Zwracanie wyniku:
   - Algorytm zwraca tablicę `color`, gdzie każda pozycja odpowiada klasyfikacji danego wierzchołka.

Na **Rysunku 2** przedstawiono kolorowanie przykładowego wielokąta:



Rysunek 2: Wynik podziału wierzchołków wielokąta na typy opisane powyżej

Figura na **Rysunku 2** przedstawia również wielokąt niemonotoniczny, w szczególności nie-y-monotoniczny.

3.3 Funkcja sortująca wierzchołki

Funkcja `build_events(polygon)` tworzy posortowaną listę wierzchołków według współrzędnej y (nierosnąco). Korzystając z faktu, że zadana figura jest y-monotoniczna (jest nierosnąca zaczynając od punktu z maksymalnym y) możemy wykonać scalenie dwóch list - lewej i prawej względem najwyższego punktu - analogicznie do algorytmu sortowania przez scalanie. W przypadku remisów wierzchołki są sortowane według współrzędnej x.

3.4 Funkcja dzieląca wielokąt

Funkcja `divide(polygon)`, wyznacza prawy i lewy łańcuch (przypisuje każdemu wierzchołkowi wartość *side* z **Nierówności (2)**) przechodząc kolejno od punktu z maksymalnym y do punktu z minimalnym y wyznaczając lewy (-1) łańcuch. Następnie przechodzi od minimalnego punktu do maksymalnego wyznaczając prawy (1) łańcuch.

4 Opis algorytmu i uzasadnienie wyboru struktur

Podane figury były wprowadzane przeciwnie do ruchu wskazówek zegara. Taki sposób wprowadzania wymusza odpowiednie dostosowanie klasyfikacji, czy przekątna należy do wielokąta (Jak opisano powyżej). Jeśli spróbujemy narysować wielokąt w odwrotny sposób, algorytm w przypadku, gdy punkty należą do tego samego łańcucha, zwróci wierzchołki spoza figury, jednocześnie wykluczając te, które powinny się w niej znaleźć. Wynikowa triangulacja jest przechowywana w tablicy krotek, gdzie każda krotka reprezentuje indeksy wierzchołków z tablicy reprezentującej wielokąt, połączonych przekątną. Wybrałem taką reprezentację przede wszystkim ze względu na wygodę weryfikacji poprawności wyniku i oszczędzanie pamięci, w przypadku triangulacji dużego wielokąta. Podział punktów na lewy bądź prawy przechowuję w tablicy z wartościami 1 lub -1 (specjalne przypadki dla najwyższego i najniższego punktu, dla których wartość nie ma znaczenia, jest 0).

1. **Weryfikacja, czy wielokąt jest monotoniczny**: Funkcja `is_y_monotonic(polygon)` sprawdza, czy podany wielokąt jest y-monotoniczny. Jeżeli nie jest, zwraca pustą tablicę i wyświetla komunikat o błędzie.

2. **Podział na łańcuchy**: Funkcja `divide(polygon)` dzieli wierzchołki wielokąta na lewy (-1) i prawy (1) łańcuch.

3. **Budowa zdarzeń**: Funkcja `build_events(polygon)` tworzy posortowaną listę wierzchołków według współrzędnej y (nierosnąco).

4. **Inicjalizacja stosu**: Dwa wierzchołki o najwyższych współrzędnych y są dodawane na stos. Stos przechowuje aktywne wierzchołki służące do budowy trójkątów.

5. **Iteracja po zdarzeniach**: Dla każdego wierzchołka z listy zdarzeń wykonywane są następujące kroki:

   (a) **Przejście między łańcuchami**: Jeśli bieżący wierzchołek należy do innego łańcucha niż ostatni wierzchołek na stosie, wszystkie elementy stosu są usuwane, a dla każdego z nich dodawana jest przekątna do wynikowej triangulacji. Następnie bieżący wierzchołek i ostatni element stosu przed wykonaniem powyższych działań są dodawane na nowo.

   (b) **Dodawanie przekątnych w tym samym łańcuchu**: Jeśli bieżący wierzchołek należy do tego samego łańcucha co ostatni na stosie, analizowana wzajemne położenie między bieżącym wierzchołkiem, ostatnim i przedostatnim na stosie. Jeśli nowo dodana przekątna będzie leżeć wewnątrz wielokąta, o czym decyduje warunek z równania 2 (gdzie $\epsilon = 10^{-18}$ to przyjęta tolerancja dla zera) jest ona dodawana do tablicy wynikowej. Ostatni element stosu jest usuwany. Następnie bieżący wierzchołek jest dodawany na stos.

   (c) **Odrzucanie krawędzi należących do wielokąta**: Przed każdym dodaniem krawędzi sprawdzane jest, czy różnica indeksów punktów jest większa niż 1 lub czy nie jest równa n-1. Jeśli tak jest, oznacza to, że próbujemy dodać krawędź, która należy do wielokąta i nie należy jej dodawać.

6. **Rysowanie wynikowej triangulacji**: Funkcja `draw_polygon_tri` wizualizuje wielokąt wraz z dodanymi przekątnymi, umożliwiając weryfikację poprawności obliczeń.

7. **Dodanie przekątnych obrysowych**: Funkcja `add_polygon(polygon)` dodaje krawędzie łączące sąsiednie wierzchołki wielokąta, aby stworzyć pełną triangulację.

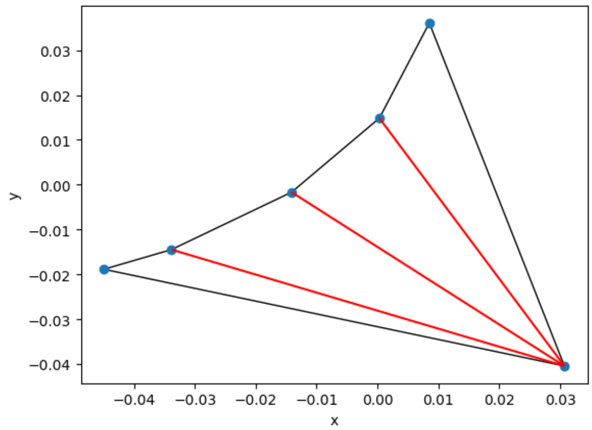
8. **Zwrócenie wyniku**: Funkcja `full_triangulation(polygon)` zwraca listę przekątnych wraz z oryginalnymi krawędziami wielokąta jako wynik końcowy triangulacji.

5 Testy

Skorzystałem z testów zaproponowanych przez KN BIT.

5.1 Parabola z jednym punktem

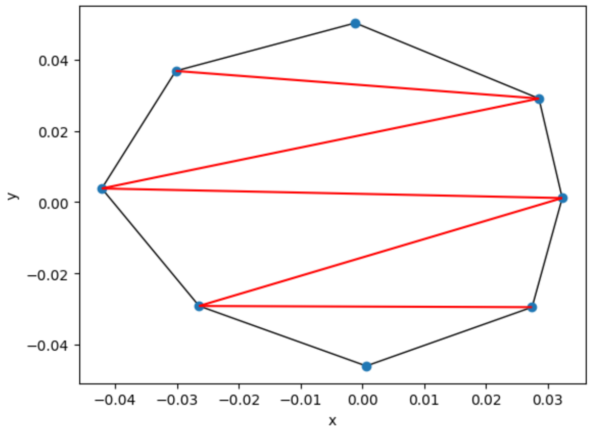
Ten test miał na celu sprawdzić, czy logika przyłączania punktu z jednego łańcucha do wszystkich wierzchołków innego łańcucha jest poprawna (punkt 5a opisanego wyżej algorytmu). Na **Rysunku 3** widać, że podział na trójkąty jest poprawny.



Rysunek 3: Wynik triangulacji dla pseudo-paraboli z jednym punktem

5.2 Wielokąt wypukły

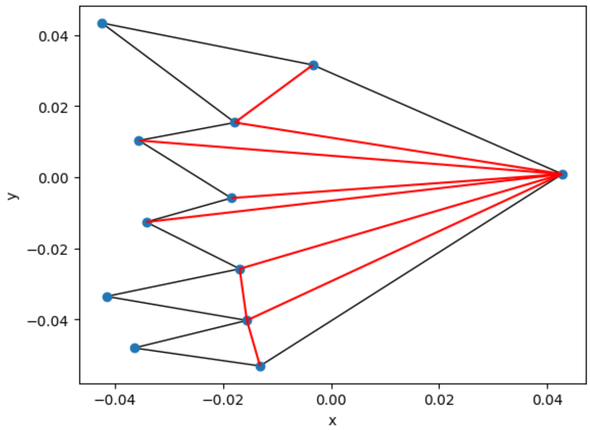
Ten test miał na celu sprawdzić, naprzemienna przyłączanie rozważanego punktu ze stosu do poprzednich poprawna (punkt 5a opisanego wyżej algorytmu, stosowany naprzemiennie). Na **Rysunku 4** wynik algorytmu.



Rysunek 4: Wynik triangulacji dla pseudo-paraboli z jednym punktem

5.3 Wielokąt zębaty skierowany w lewo

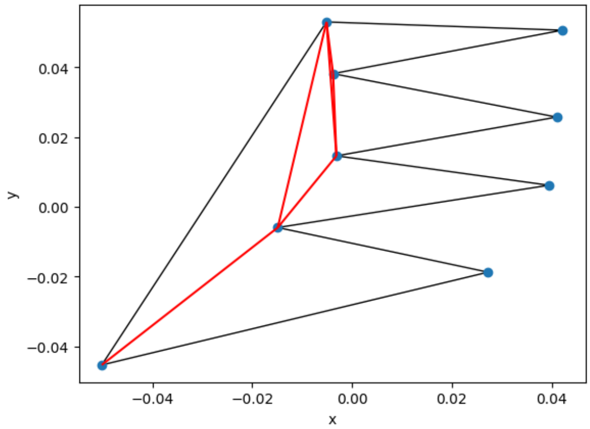
Test weryfikował, czy możliwość, dla której rozważany punkt należy do tego samego łańcucha, jest poprawnie obsługiwana - to znaczy czy dodawane są przekątne z lewej strony należące do wielokąta. Wynik został przedstawiony na **Rysunku 5**.



Rysunek 5: Wynik triangulacji dla wielokąta zębatego skierowanego w lewo

5.4 Wielokąt zębaty skierowany w prawo

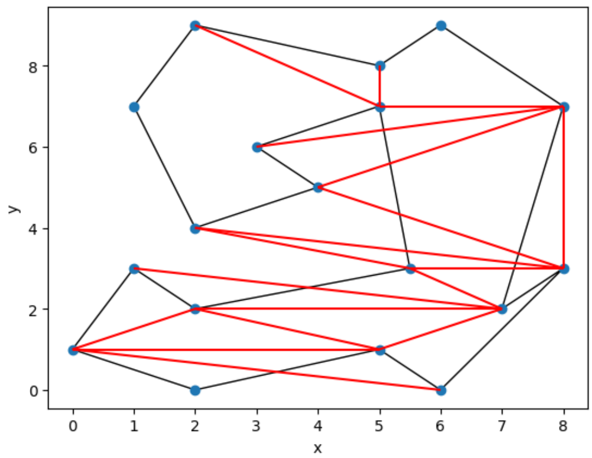
Ten sam test co wyżej, jednak w tym przypadku mamy mamy wielokąt zębaty skierowany w prawo, aby sprawdzić, czy warunek **Nierówności (2)** jest poprawnie zastosowany dla prawej (względem najwyższego punktu) strony wielokąta. Wynik został przedstawiony na **Rysunku 6**.



Rysunek 6: Wynik triangulacji dla wielokąta zębatego skierowanego w prawo

5.5 Dodatkowy test

Pomijając warunek y-monotoniczności sprawdziłem, jak wygląda triangulacja dla figury, która nie jest y-monotoniczna. Wynik przedstawiony jest na **Rysunku 7**.



Rysunek 7: Nieudana triangulacja dla wielokąta nie-y-monotonicznego

6 Wnioski

Na podstawie ćwiczenia można wyciągnąć następujące wnioski:

- Algorytm został zaimplementowany poprawnie. Dla wszystkich testów algorytm zwraca oczekiwany rezultat.
- Program opierał się na algorytmie omawianym na wykładach, który nie obsługuje wielokątów niemonotonicznych. Wynik próby triangulacji figury nie-y-monotonicznej widać na **Rysunku 7**.
- Jesteśmy w stanie podzielić zadaną y-monotoniczną figurę na trójkąty w czasie $O(n)$. Jest to optymalne rozwiązanie dla tego problemu, albowiem trzeba rozważyć każdy punkt należący do wielokąta. To wynika z faktu, że dla każdego n-kąta ilość trójkątów poprawnej triangulacji to n-2.