Przetwarzanie i przechowywanie opisu siatki trójkątnej na płaszczyźnie

Mateusz Zając, Błażej Kapkowski

Opis problemu

- Najczęstszy sposób przechowywania siatki to użycie 2 list – listy wierzchołków (czyli współrzędnych) oraz listy trójkątów (czyli indeksów wierzchołków tworzących trójkąt)
- Taka struktura ułatwia odczyt i zapis siatek, ale utrudnia operacje na nich.
- Dlatego, do pewnych operacji, stosuje się strukturę Half Edge.

Testowane operacje

- Operacje, jakie wykonano na obu strukturach, a potem porównano, to:
- Wyznaczanie 1- i 2-warstwowego otoczenia wierzchołka, składającego się z sąsiednich wierzchołków.
- Wyznaczanie 1- i 2-warstwowego otoczenia trójkąta, składającego się z sąsiednich trójkątów.
- Przeszukiwanie sąsiednich trójkątów w poszukiwaniu wybranego wierzchołka, zaczynając od wybranego trójkąta.

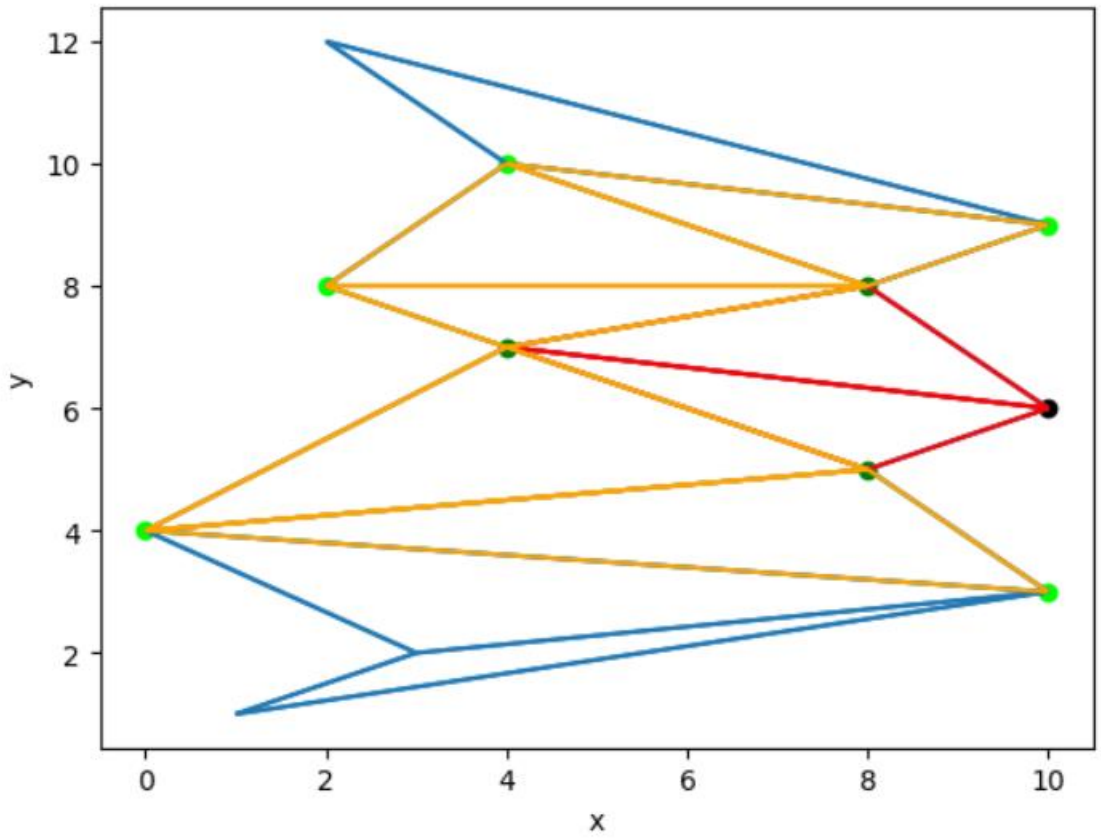
Przyjęte założenia

- Siatka to pewna triangulacja w 2D.
- Przy trzeciej operacji (poszukiwaniu wierzchołka) nie użyto żadnych heurystyk, przechodzi się po prostu przez kolejne warstwy trójkątów.
- Siatka nie może mieć dziur w środku, bo to znacznie utrudnia implementację Half Edge.

Prosta struktura, 1. operacja

Funkcja rozpoczyna się od inicjalizacji dwóch zbiorów do przechowywania incydentnych wierzchołków na pierwszej warstwie i wierzchołków na drugiej warstwie. Następnie algorytm iteruje przez wszystkie trójkąty w triangulacji. Jeśli wierzchołek v należy do trójkąta, to funkcja dodaje wszystkie wierzchołki tego trójkąta do piewszego zbioru (z wyłączeniem v). Dla każdego wierzchołka z pierwszego zbioru, funkcja iteruje ponownie przez wszystkie trójkąty w triangulacji. Jeśli sąsiedni wierzchołek nie należy do trójkąta i v nie należy do trójkąta, to dodaje wierzchołki tego trójkąta do drugiego zbioru.

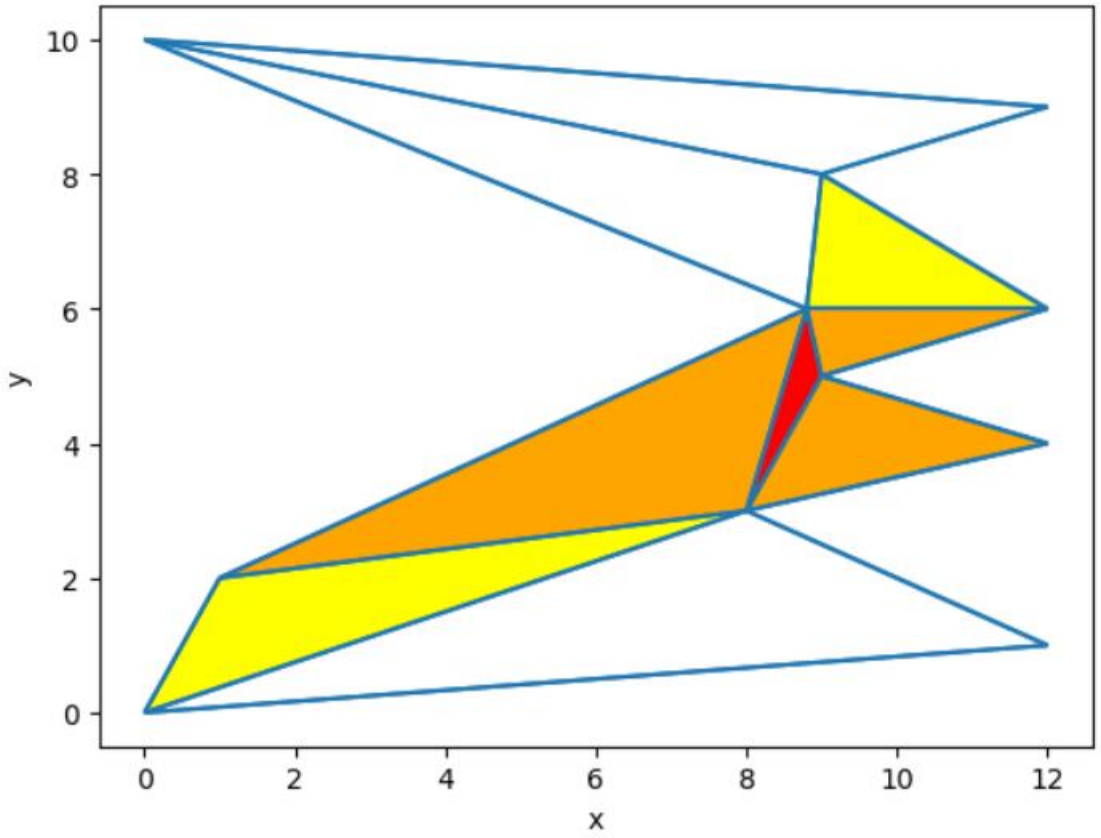
Prosta struktura, 1. operacja



Prosta struktura, 2. operacja

Funkcja służy do znajdowania sąsiednich trójkątów dla danego trójkąta w triangulacji. Algorytm inicjalizuje dwa zbiory: pierwszy dla jednowarstwowych sąsiadów i drugi dla dwuwarstwowych sąsiadów. Algorytm Iteruje przez wszystkie trójkąty w triangulacji. Jeśli trójkąt nie jest równy trójkątowi t i sąsiaduje z nim (co jest sprawdzane za pomocą osobnej funkcji), to dodaje go do pierwszego zbioru. Dla każdego trójkąta z pierwszego zbioru, iteruje ponownie przez wszystkie trójkąty w triangulacji. Jeśli trójkąt nie jest równy ani trójkątowi t, ani trójkątowi z pierwszego zbioru, i sąsiaduje z tym trójkątem, to dodaje go do drugiego zbioru.

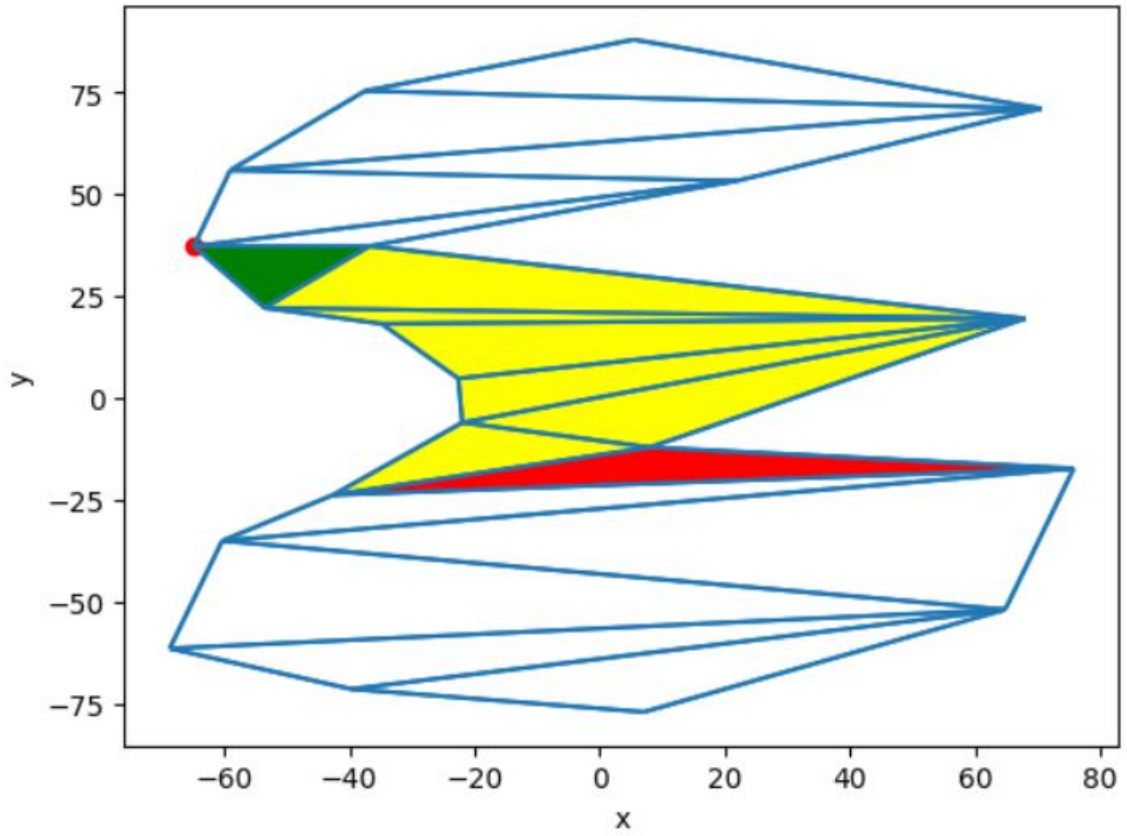
Prosta struktura, 2. operacja



Prosta struktura, 3. operacja

Algorytm zaczyna od trójkąta Start i zmierza do punktu docelowego Target. Inicjalizuje także zmienną pomocniczą dla trójkąta Current na wartość Start. Następnie wchodzi do pętli, która będzie powtarzana, dopóki Current nie zostanie znalezione, lub zostanie odwiedzony już wcześniej. Jeśli punkt Target znajduje się w trójkącie Current, algorytm zwraca ten trójkąt jako wynik. Dla trójkąta Current znajduje sąsiadujące trójkąty (takie, które mają przynajmniej 2 wspólne wierzchołki z Current). Spośród sąsiadujących trójkątów wybiera te, które nie zostały jeszcze odwiedzone, i ustawia Current na pierwszy z nich.

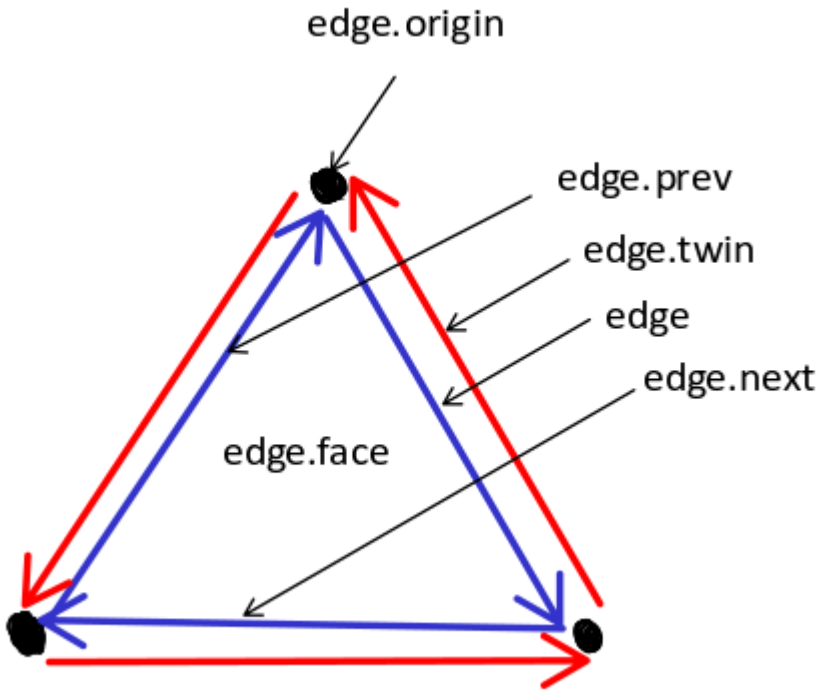
Prosta struktura, 3. operacja



Struktura Half Edge

- Struktura Half Edge (lub też Double Connected Edge List) dzieli każdą nieskierowaną krawędź na dwie skierowane.
- Każda taka krawędź ma wskaźnik do drugiej krawędzi z pary.
- Poza tym, każda krawędź wskaźnik na następną oraz poprzednią krawędź należącą do trójkąta – trzy kolejne krawędzie tworzą cykl.
- Krawędź zawiera też indeks wierzchołka, z którego wychodzi, a także indeks trójkąta, z którym sąsiaduje (lub None, jeśli z żadnym)

Struktura Half Edge



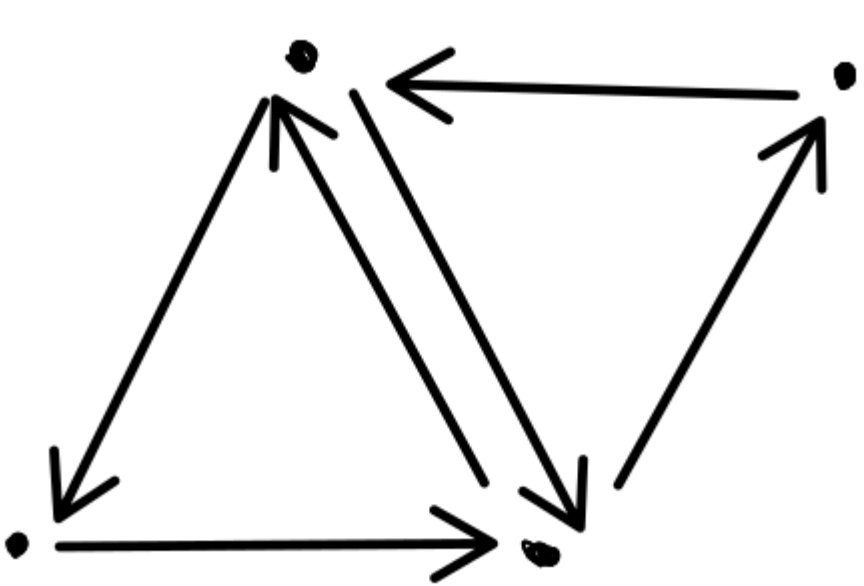
Struktura Half Edge

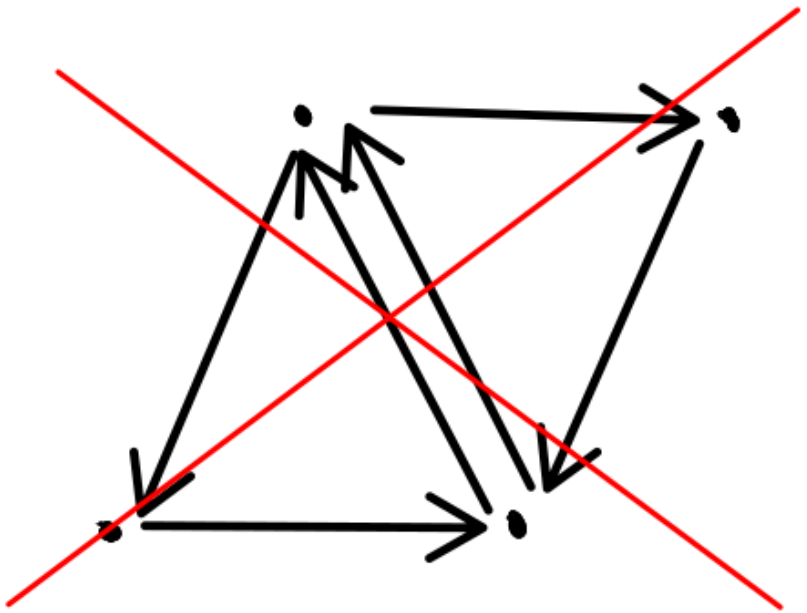
Ponadto, do każdego wierzchołka przypisano jedną, dowolną krawędź, która z niego wychodzi.
Do każdego trójkąta zaś przypisano jedną, dowolną krawędź, która do niego należy.
Dzięki temu, podczas wyznaczania sąsiednich wierzchołków lub trójkątów, mając indeks tego, z którego się zaczyna, można od razu przejść do związanej z obiektem krawędzi.

Half Edge - konstrukcja

- Do konstrukcji struktury wykorzystano słownik. Kluczami były krotki, w których na 1. miejscu jest indeks tego wierzchołka, z którego krawędź wychodzi, a na 2. indeks tego, do którego wchodzi. Wpierw iteruje się po wszystkich trójkątach i tworzy się na ich podstawie krawędzie. Jeśli w słowniku umieszczona była już jedna z krawędzi trójkąta, wszystkie jego krawędzi się odwraca. Tworząc krawędzi trójkąta, przypisuje się jedną z nich do tego trójkąta, a także po jednej do każdego jego wierzchołka, jeśli jeszcze jej nie przypisano. Każde trzy krawędzie z trójkąta łączy się w cykl: do każdej przypisuje się poprzednią.

Half Edge – tworzenie krawędzi

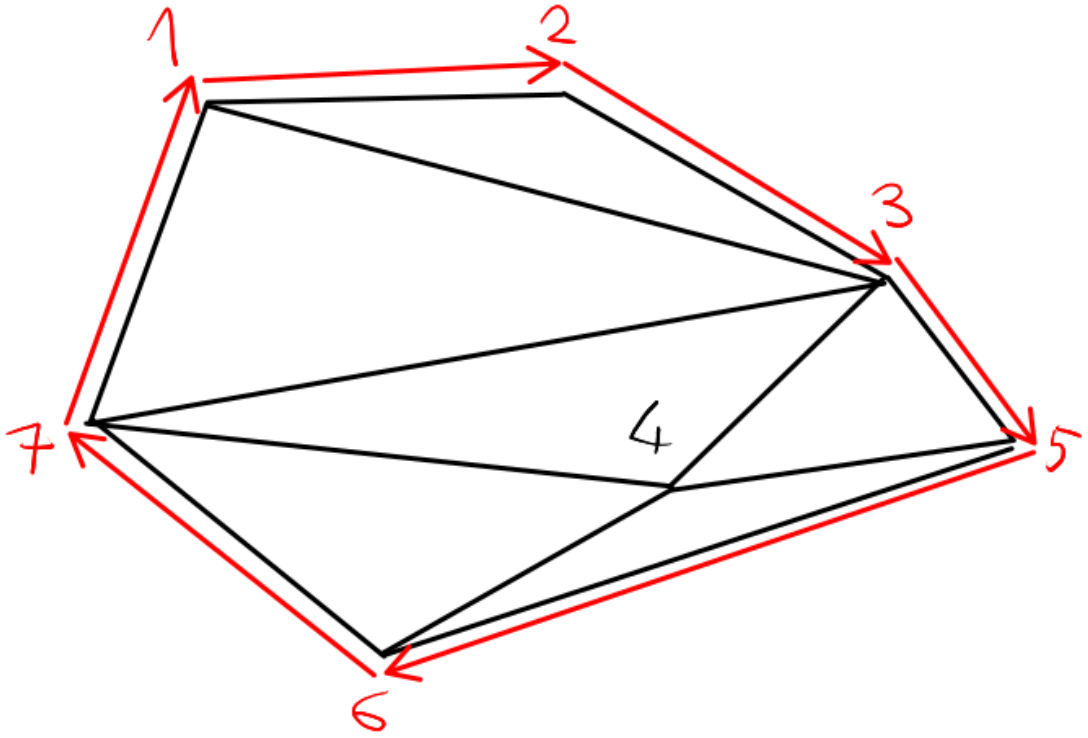




Half Edge – konstrukcja cd.

- Następnie iteruje się po kluczach słownika. Dla każdej krotki postaci (start, end), sprawdza się, czy krotka (end, start) występuje w słowniku. Jeśli nie, tworzy się krawędź wychodzącą z end, ale zamiast trójkąta, przypisuje się do niej wartość None. Wszystkie takie krawędzi zbiera się do osobnej listy.
- Tworzy się słownik, w którym kluczami są wierzchołki początkowe krawędzi bez żadnego trójkąta, a wartościami ich wierzchołki końcowe. Czyta się kolejne początkowe wierzchołki: current = empty_edges[current]. Iterując w ten sposób po słowniku, można uporządkować krawędzie w kolejności, w jakiej występują w cyklu, a następnie połączyć, tak jak poprzednio krawędzi trójkątów.

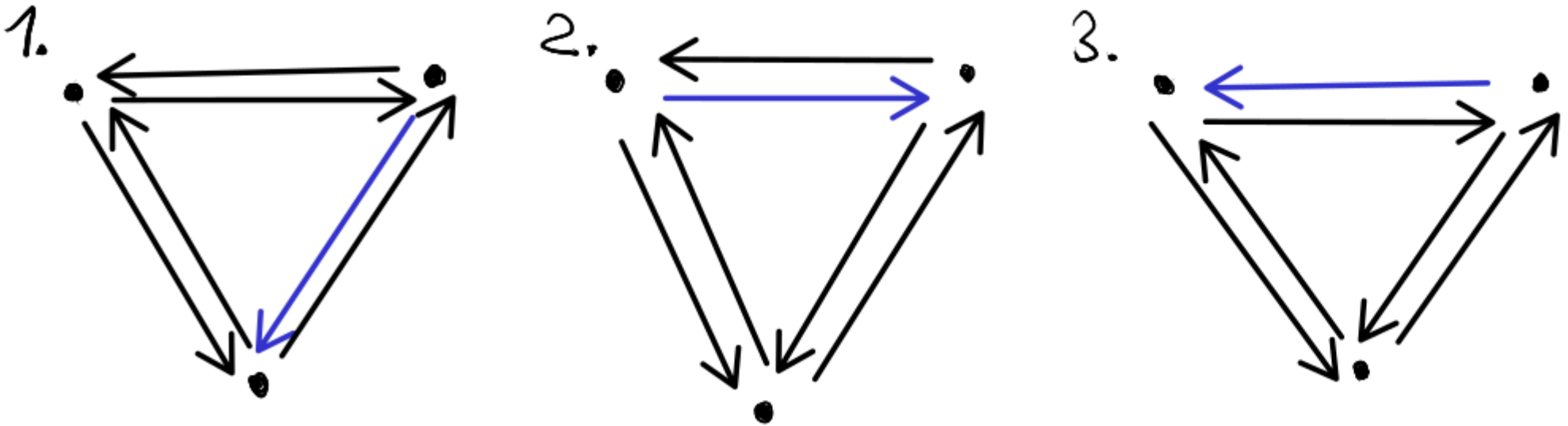
Half Edge – krawędzi zewnętrzne



Half Edge, 1. operacja

- Podstawą algorytmu jest sposób przechodzenia po sąsiadujących wierzchołkach. Zaczyna się od krawędzi v.edge wychodzącej z wierzchołka v. Następnie ustawia się zmienną current = v.edge.
- Dopóki current != v.edge, po kolejnych krawędziach iteruje się za pomocą operacji:
- current = current.prev.twin
- W ten sposób znajduje się wszystkie krawędzi wychodzące z wierzchołka v.

Half Edge, 1. operacja – sposób iteracji



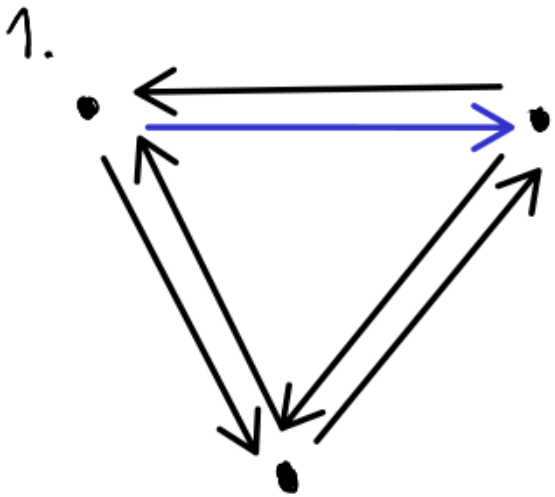
Half Edge, 1. operacja

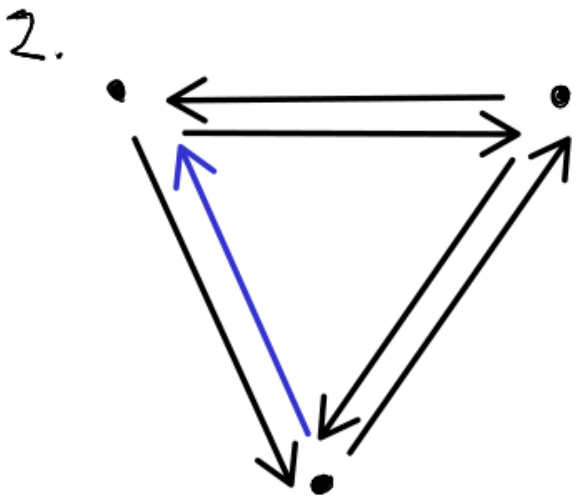
- Następnie tworzy się listę zbiorów, gdzie każdy zbiór odpowiada kolejnej warstwie, począwszy od zerowej, zawierającej pierwszy wierzchołek. Iteruje się po sąsiadach każdego wierzchołka należącego do zbioru na pozycji x w liście. Jeśli wierzchołek nie został umieszczony w żadnym poprzednim zbiorze, dodaje się go do zbioru na pozycji x+1.
- Zwracana przez funkcję lista zbiorów to lista kolejnych warstw otoczenia wierzchołka.

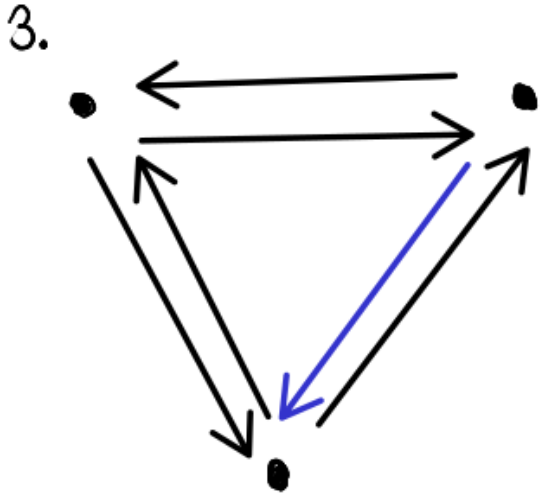
Half Edge, 2. operacja

- Żeby móc iterować po wszystkich sąsiadach trójkąta, najpierw trzeba umieć iterować po jego krawędziach. Zaczyna się od krawędzi t.edge należącej do trójkąta t. Tworzy się zmienną current = t.edge. Następnie, iteruje się po krawędziach za pomocą: current = current.prev.
-

Half Edge, 2. operacja – sposób iteracji



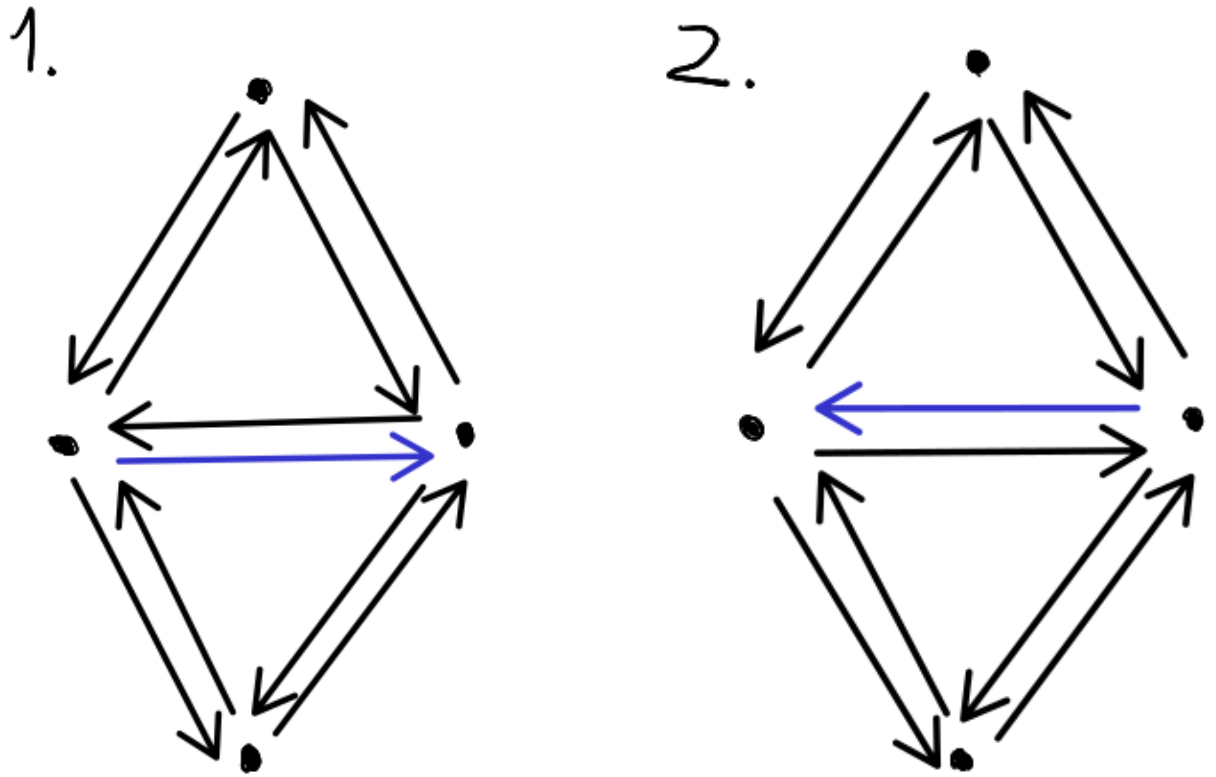




Half Edge, 2. operacja

- Żeby otrzymać sąsiedni trójkąt, należy odnieść się do sąsiedniej krawędzi, a potem do przypisanego do niej trójkąta: current.twin.triangle.
- Sam sposób wyznaczania kolejnych warstw jest taki sam jak przy pierwszej operacji. Używa się listy zbiorów, gdzie każdy zbiór przechowuje indeksy trójkątów z danej warstwy otoczenia.

Half Edge, 2. operacja – odwiedzenie trójkąta



Half Edge, 3. operacja

- Trzecia operacja działa na bazie poprzedniej. Wyznacza się kolejne warstwy otoczenia trójkąta tak długo, jak któryś z trójkątów należących do otoczenia będzie zawierał poszukiwany wierzchołek. Tak więc, przed wpisaniem wierzchołka do jednego ze zbiorów, sprawdza się czy jest tym poszukiwanym przez funkcję.

Bibliografia

- https://jerryyin.info/geometry-processing-algorithms/half-edge/
- https://cs184.eecs.berkeley.edu/sp19/article/15/the-half-edge-data-structure
- https://stackoverflow.com/questions/15365471/initializing-half-edge-data-structure-from-vertices
- https://kaba.hilvi.org/homepage/blog/halfedge/halfedge.htm
- Mark de Berg – „Computational Geometry – Algorithms and Applications”

Koniec

- Dziękujemy za uwagę!