Algorytmy Geometryczne

Przetwarzanie i przechowywanie opisu siatki trójkątnej na płaszczyźnie

Dokumentacja

Michał Ramut, Hubert Tułacz

Styczeń 2024

1. **Opis projektu**

Porównanie Halfedge Data structure ze strukturą składającą się ze zbioru punktów i trójkątów na podstawie trzech operacji.

OP1: wyznaczanie otoczenia dla wybranego wierzchołka (kolejne warstwy incydentnych wierzchołków – należy rozpatrzyć otoczenia składające się z jednej warstwy oraz dwóch warstw),

OP2: wyznaczanie otoczenia dla wybranego trójkąta (kolejne warstwy incydentnych trójkątów – należy rozpatrzyć otoczenia składające się z jednej warstwy oraz dwóch warstw),

OP3: przeglądanie incydentnych trójkątów od wybranego trójkąta w kierunku wybranego punktu (dla odszukania trójkąta zawierającego dany punkt).

Do realizacji projektu została zaimplementowana struktura HalfEdge, w celu łatwego przechodzenia po wierzchołkach/trójkątach.

1. **Podstawowe informacje techniczne**

• Język Python w wersji 3.10,

• Precyzja operacji zmiennoprzecinkowych — domyślna w python - zazwyczaj odpowiada double w języku C

• Narzędzie do wizualizacji - bibiloteka mathplotlib

• Reprezentacja punktu — krotka współrzędnych punku (liczb zmiennoprzecinkowych)

1. **Przykłady użycia**

from halfedge import \*

from visualizer import \*

p = [(1.0, 1.0),

(2.0, 6.5),

(9.0, 1.0),

(17.5, 5.0),

(19.0, 13.5),

(12.0, 16.0),

(11.5, 12.0),

(6.0, 12.0)]

t = [(0, 2, 1),

(1, 2, 7),

(6, 7, 2),

(2, 3, 6),

(6, 3, 4),

(6, 4, 5)]

mesh = triangles\_to\_segments(p,t)

he = create\_half\_edge\_structure(p, t)

draw\_halfedge\_structure(he,mesh)

1. **Przykładowa wizualizacja**

Obraz zawierający linia, diagram, origami

Opis wygenerowany automatycznie

Rysunek 1 Przykładowa wizualizacja

1. **Opis klas i funkcji z pliku halfedge.py**

class halfedge.Face(points: list[tuple[float, float]], index: int)

Klasy bazowe: object

Reprezentacja trójkąta.

class halfedge.HalfEdge(v: Vertex, face: Face)

Klasy bazowe: object

Reprezentacja półkrawędzi.

class halfedge.Vertex(cords: tuple[float, float], index: int)

Klasy bazowe: object

Reprezentacja wierzchołka.

halfedge.ccw(a: tuple[float, float], b: tuple[float, float], c: tuple[float, float]) → float

Pole równoległoboku rozpiętego na wektorach (ab) i (ac). Będzie ujemne, gdy podane punkty będą ułożone zgodnie z ruchem wskazówek zegara.

halfedge.sort\_points(points: list[tuple[tuple[float, float], int]]) → list[tuple[tuple[float, float], int]]

Funkcja sortująca punkty.

halfedge.create\_half\_edge\_structure(points: list[tuple[float, float]], triangles: list[int]) → list[HalfEdge]

Funkcja tworzy strukturę HalfEdge.

1. **Opis funkcji z pliku halfedge.ipynb**

halfedge.point\_in(triangle: tuple[int, int, int], point: tuple[float, float]) → bool

Funkcja sprawdza czy punkt zawiera się w podanym trójkącie.

halfedge.triangles\_to\_connections(triangles: list[tuple[int, int, int]]) → list[tuple[int, int]]

Funkcja tworzy i zwraca listę odcinków łączących punkty w postaci listy par indeksów.

halfedge.triangles\_to\_segments(points: list[tuple[float, float]], triangles: list[tuple[int, int, int]]) →

list[tuple[tuple[float, float], tuple[float, float]]]

Funkcja tworzy i zwraca listę odcinków łączących punkty.

halfedge.triangulation\_reader(number: int) → tuple[list[tuple[float, float]], list[tuple[int, int, int]]]

Funkcja czyta triangulację o zadanym numerze z katalogu tests/.

* 1. **OP1: wyznaczanie otoczenia dla wybranego wierzchołka**

halfedge.apex\_surroundings\_basic(point: int, points: list[tuple[float, float]], connections: list[tuple[int, int]]) → list[int]

Funkcja wyznacza otoczenie punktu (2 warstwy incydentnych wierzchołków), mając do dyspozycji triangulację w postaci listy punktów i listy odcinków.

* Iterujemy po wszystkich połączeniach i sprawdzamy czy jest to połączenie z interesującego nas punktu, jeżeli tak to ponownie iterujemy po wszystkich połączeniach aby znaleźć sąsiadów sąsiadów.

halfedge.apex\_surroundings\_halfedges(p: int, halfedges: list[HalfEdge]) → list[int]

Funkcja wyznacza otoczenie punktu (2 warstwy incydentnych wierzchołków), mając do dyspozycji triangulację w postaci stuktury HalfEdge.

* Szukamy sąsiadów wierzchołka startowego, następnie szukamy sąsiadów jego sąsiadów.
* Aby znaleźć sąsiadów wierzchołka najpierw znajdujemy wszystkie pół krawędzie wychodzące z tego wierzchołka a następnie dla każdej takiej pół krawędzi przechodzimy po jej ścianie i zapisujemy znalezione punkty.
  1. **OP2: wyznaczanie otoczenia dla wybranego trójkąta**

halfedge.triangle\_surroundings\_basic(tri\_id: int, triangles: list[tuple[int, int, int]]) → tuple[list[int],

list[int]]

Funkcja wyznacza otoczenie trójkąta (2 warstwy incydentnych trójkątów), mając do dyspozycji listę trójkątów.

* Iterujemy po liście połączeń(przedstawionej jako trójkąty) i szukamy trójkątów sąsiadujących z trójkątem startowym
* Następnie dla każdego sąsiada szukamy jego sąsiadów w ten sam sposób

halfedge.triangle\_surroundings\_halfedge(tri\_id: int, halfedges: list[HalfEdge]) → list[int]

Funkcja wyznacza otoczenie trójkąta (2 warstwy incydentnych trójkątów), mając do dyspozycji triangulację w postaci stuktury HalfEdge.

* Szukamy sąsiednich ścian ściany początkowej, następnie powtarzamy to dla znalezionych sąsiadów
* Aby znaleźć sąsiadów szukamy pół krawędzi należącej do badanej ściany, a następnie dodaje (jeżeli istnieją) ściany do których należą bliźniacze pół krawędzie powiązane z cyklem wewnątrz początkowej ściany
  1. **OP3: przeglądanie incydentnych trójkątów od wybranego trójkąta w kierunku wybranego punktu**

halfedge.find\_triangle\_basic(triangles: list[tuple[int, int, int]], points: list[tuple[float, float]], point: tuple[float, float], start\_triangle: tuple[int, int, int]) → int

Funkcja szuka trójkąta, w którym znajduje się dany punkt.

* Tworzymy stos przechowywujący trójkąty które należy sprawdzić
* Dopóki stos nie jest pusty lub nie znaleźliśmy już szukanego trójkąta, dodajemy sąsiednie trójkąty

halfedge.find\_triangle\_halfedge(halfedges: list[HalfEdge], point: tuple[float, float], start\_triangle: int) → int

Funkcja szuka trójkąta, w którym znajduje się dany punkt, mając do dyspozycji triangulację w postaci stuktury HalfEdge.

* Tworzymy stos który przechowuje trójkąty, które należy sprawdzić
* Dopóki stos nie jest pusty lub nie znaleźliśmy już szukanego trójkąta, dodajemy na stos sąsiadów aktualnego trójkąta

1. **Porównanie czasów algorytmów**

**Obraz zawierający tekst, zrzut ekranu, diagram, linia

Opis wygenerowany automatycznie**

Wykres 2 Porównanie czasów algorytmów dla OP1

**Obraz zawierający tekst, zrzut ekranu, diagram, linia

Opis wygenerowany automatycznie**

Wykres 2 Porównanie czasów algorytmów dla OP2

**Obraz zawierający tekst, zrzut ekranu, linia, diagram

Opis wygenerowany automatycznie**

Wykres 3 Porównanie czasów algorytmów dla OP3