Algorytmy geometryczne

Dokumentacja projektu

grupa nr.4 Czw_12.20_B

Paweł Surdyka

Paweł Derbisz

Dane techniczne urządzenia na którym wykonano ćwiczenie:

Komputer z systemem Windows 10 x64
Procesor: Intel Core I5 9300HF CPU @2.4Ghz
Pamięć RAM: 8GB
Środowisko: Jupyter notebook
Język: Python 3

Opis ćwiczenia

Ćwiczenie polegało na wczytaniu zawartości przykładowych triangulacji 2D, zapisanych w postaci tekstowej jako lista punktów oraz lista elementów, zilustrowaniu tej siatki oraz stworzeniu odpowiednich struktur aby otrzymać reprezentację „Half Edge Data Structure” a następnie wykonaniu 4 operacji z wykorzystaniem obydwu reprezentacji i porównaniu ich pod kątem czasowym i pamięciowym.

1.Tworzenie struktur Half-Edge

Aplikacja pozwala na stworzenie reprezentacji siatki przy pomocy struktur half-edge z wczytanej reprezentacji w postaci tablicy krawędzi oraz tablicy trójkątów.

Reprezentacja Half-Edge zawiera podane poniżej struktury danych:

- Vertices zawierającej współrzędne wierzchołka oraz informacje która ,,pół-krawędź” z niego wychodzi

```
class Vertex:
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.edge = None
```

-Half_edges zawierającej informacje z jakiego wierzchołka wychodzi, do jakiej ściany przylega oraz jakiego ma poprzednika (prev), następnika (next) oraz przeciwieństwa (twin)

```
class HalfEdge:
    def __init__(self):
        self.vertex = None
        self.twin = None
        self.prev = None
        self.next = None
        self.face = None
```

-Face zawierającej jedną ,,pół krawędź’’ (dzięki funkcjom prev i next wystarczy tylko jedna ,,pół krawędź’’.

```
class Face:
    def __init__(self):
        self.edge = None
```

Aby uzyskać reprezentacje Half-edge z reprezentacje wejściowej należy:

1. Dla każdego trójkąta utworzyć trzy half-edge'y, które będą reprezentować jego krawędzie.

```
e1 = HalfEdge()
e2 = HalfEdge()
e3 = HalfEdge()
```

2. Dla każdego half-edge'a określić punkt z którego wychodzi oraz dla każdego punktu określić który half-edge z niego wychodzi (jeżeli cykl a->b->c->a idzie przeciwnie do ruchu wskazówek zegara to tak też przypisujemy ,,pół krawędź’’ do wierzchołków, a jeżeli cykl idzie zgodnie z ruchem wskazówek zegara to tworzymy cykl c->b->a->c i tak też przypisujemy kolejne ,,pół krawędź’ie”).

3. Dla każdego half-edge'a określić, która krawędź jest jego poprzednikiem a która następnikiem (określamy to przy użyciu informacji z punktu 2.) oraz jaka krawędź jest jego przeciwieństwem.

```
det = det2x2([(vertex1.x-vertex3.x), (vertex1.y-vertex3.y)], [(vertex2.x-vertex3.x), (vertex2.y-vertex3.y)])
if(det > 0):
    vertex1.edge = e1
    vertex2.edge = e2
    vertex3.edge = e3
    e1.vertex = vertex1
    e2.vertex = vertex2
    e3.vertex = vertex3
    e1.next = e2
    e1.prev = e3
    e2.next = e3
    e2.prev = e1
    e3.next = e1
    e3.prev = e2
else:
    vertex1.edge = e3
    vertex2.edge = e1
    vertex3.edge = e2
    e1.vertex = vertex2
    e2.vertex = vertex3
    e3.vertex = vertex1
    e1.next = e3
    e1.prev = e2
    e2.next = e1
    e2.prev = e3
    e3.next = e2
    e3.prev = e1
```

4. Dla każdego half-edge'a określić do jakiej ściany należy.

```
face = Face()
face.edge = e1

e1.face = face
e2.face = face
e3.face = face
```

2. OP1: Wyznaczanie otoczenia dla wybranego wierzchołka

Znalezienie otoczenia składającego się z jednaj warstwy wierzchołków dla pierwszej reprezentacji polegało na przeglądaniu wszystkich trójkątów w celu znalezienia takich trójkątów w których jest nasz szukany punkt.

```
def get_surr(triangles,point_id,surr):
    for triangle in triangles:
        if point_id in triangle:
            for i in triangle:
                surr.append(i)
            if point_id in surr:
                surr.remove(point_id)
    surr = list(dict.fromkeys(surr))
    return surr
```

Dla znalezienia drugiej warstwy należało powtórzyć powyższy algorytm dla wszystkich wierzchołków z pierwszego otoczenia.

```
def surrounding(triangles,point_id):
    first_surrounding = []
    first_surrounding = get_surr(triangles,point_id,[])
    second_surrounding = []
    for i in first_surrounding:
        second_surrounding = get_surr(triangles,i,second_surrounding)
    for i in first_surrounding:
        if i in second_surrounding:
            second_surrounding.remove(i)
    if point_id in second_surrounding:
        second_surrounding.remove(point_id)
    return first_surrounding,second_surrounding
```

W drugiej reprezentacji przechodzimy po wszystkich ,,pół krawędziach'' wychodzących z danego wierzchołka za pomocą operacji h = h.prev.twin i dodajemy je do tablicy. Żeby następnie pobrać informacje o wierzchołku z otocznia za pomocą operacji h.twn.vertex.

```
def all_half_edges_leaving(v):
    h = v.edge
    t = h
    while h.twin != None:
        if h.twin.next != t:
            h = h.twin.next
        else:
            break
    first_edge = h
    edges = []
    edges.append(first_edge)
    h = h.prev
    if h.twin != None:
        w = h.twin
        t = w
        while w != h:
            edges.append(w)
            if w.prev.twin != None:
                if w.prev.twin != t:
                    w = w.prev.twin
                else:
                    break
            else:
                break

        if w.prev.twin == None:
            w = w.prev
            edges.append(w)
    else:
        edges.append(h)

    return edges
```

3. OP2: Wyznaczanie otoczenia dla wybranego trójkąta

Dla pierwszej reprezentacji przechodzimy po wszystkich trójkątach i dodajemy trójkąty o wspólnych krawędziach z naszym wybranym trójkątem do tablicy incydent_triangles.

```
def traingle_surrounding(points,triangles,trian):
    v1 = trian[0]
    v2 = trian[1]
    v3 = trian[2]
    incident_triangles = []
    for triangle in triangles:
        count = 0
        if v1 in triangle:
            count += 1
        if v2 in triangle:
            count += 1
        if v3 in triangle:
            count += 1
        if count == 2:
            incident_triangles.append(triangle)
    return incident_triangles
```

W reprezentacji Half-edge wystarczy sprawdzić czy dana krawędź składająca się na trójkąt na krawędź ,,twin'' a jeśli tak to dodajemy do tablicy wynikowej ,,face'' krawędzi ,,twin''

```
def first_sur(face):
    e1 = face.edge
    e2 = e1.next
    e3 = e2.next
    first_s = []
    if e1.twin != None:
        first_s.append(e1.twin.face)
    if e2.twin != None:
        first_s.append(e2.twin.face)
    if e3.twin != None:
        first_s.append(e3.twin.face)
    return first_s

def traingle_surrounding_for_half_edges(face):
    first_s = first_sur(face)
    second_s = []
    for f in first_s:
        temp = first_sur(f)
        for fa in temp:
            second_s.append(fa)
    return first_s,second_s
```

4. OP3: przeglądanie incydentnych trójkątów od wybranego trójkąta w kierunku wybranego punktu

W pierwszej reprezentacji przeglądamy trójkąty za pomocą zmodyfikowanej funkcji BFS aby znaleźć najkrótszą ścieżkę od wybranego trójkąta do najbliższego trójkąta zawierającego wybrany punkt.

```
def bfs_for_triangles(triangles,t):
    n = len(triangles)
    visited = [False for i in range(n)]
    parent = [None for i in range(n)]
    childs = [[] for i in range(n)]
    d = [0 for i in range(n)]
    queue = []
    queue.append(t)
    visited[t] = True
    while queue:
        # t indeks aktualnie przeglądanego trójkąta
        t = queue.pop(0)
        v1 = triangles[t][0]
        v2 = triangles[t][1]
        v3 = triangles[t][2]
        for i in range(n):
            if visited[i] == False:
                count = 0
                if v1 in triangles[i]:
                    count += 1
                if v2 in triangles[i]:
                    count += 1
                if v3 in triangles[i]:
                    count += 1
                if count == 2:
                    queue.append(i)
                    visited[i] = True
                    parent[i] = t
                    childs[t].append(i)
                    d[i] = d[t] +1
    return visited,parent,d,childs
```

W drugiej reprezentacji najpierw szukamy w jakich trójkątach znajduje się nasz punkt, a następnie podobnie jak w pierwszej reprezentacji puszczamy zmodyfikowaną funkcje BFS. Przy czym wymagało to lekkiego zmienienia struktury face aby mogła przechowywać informacje o tym czy jest już odwiedzona, jaka jest jej odległość od wskazanego trójkąta oraz informacja jakie ,,twarze'' są jej dziećmi a jakie rodzicem.

```
def bfs_for_faces(faces,face):
    n = len(faces)
    d = [[] for i in range(n)]
    d[0].append(face)
    queue = []
    queue.append(face)
    face.visited = True
    while queue:
        t = queue.pop(0)
        e1 = t.edge
        e2 = e1.next
        e3 = e2.next
        if e1.twin != None:
            if e1.twin.face.visited == False:
                f1 = e1.twin.face
                f1.visited = True
                f1.distance = t.distance+1
                t.childs.append(f1)
                f1.parent.append(t)
                d[f1.distance].append(f1)
                queue.append(f1)
        if e2.twin != None:
            if e2.twin.face.visited == False:
                f2 = e2.twin.face
                f2.visited = True
                f2.distance = t.distance+1
                t.childs.append(f2)
                f2.parent.append(t)
                d[f2.distance].append(f2)
                queue.append(f2)
        if e3.twin != None:
            if e3.twin.face.visited == False:
```

5. OP4: zamiana krawędzi dla wskazanej pary incydentnych trójkątów

Dla pierwszej reprezentacji polegało to na wykorzystaniu funkcji z podpunktu OP2 (tj. wyznaczenie jednowarstwowego otoczenia) i zamianie punktów składających się na dane trójkąty.

```
triangle = triangles[6]
incydent_triangles = traingle_surrounding(points,triangles,triangle)
# 0 , 1 lub 2
choosen_triangle = incydent_triangles[0]
```

```
incydent_triangles = traingle_surrounding(points,triangles,triangle)
temp_tirangle = triangle
triangle = choosen_triangle
choosen_triangle = temp_tirangle
```

Dla reprezentacji drugiej polegało to na zamianie ,,pół krawędzi'' definiujących dane ,,twarze'', zamianie ,,twarzy'' przypisanej danej ,,pół krawędzi'' oraz zamianie wszystkich wierzchołków wychodzących dla danych ,, pół krawędzi''

```
trian4 = f[2]
incident_faces = first_sur(trian4)
choosen_face = incident_faces[0]
```

```
temp_face = trian4.edge
trian4.edge = choosen_face.edge
choosen_face.edge = temp_face
```

```
e1.face = choosen_face
e1.next.face = choosen_face
e1.prev.face = choosen_face
e2.face = trian4
e2.next.face = trian4
e2.prev.face = trian4
```