Algorytmy geometryczne

Przetwarzanie i przechowywanie opisu siatki trójkątnej na płaszczyźnie

Dominik Jastrząb

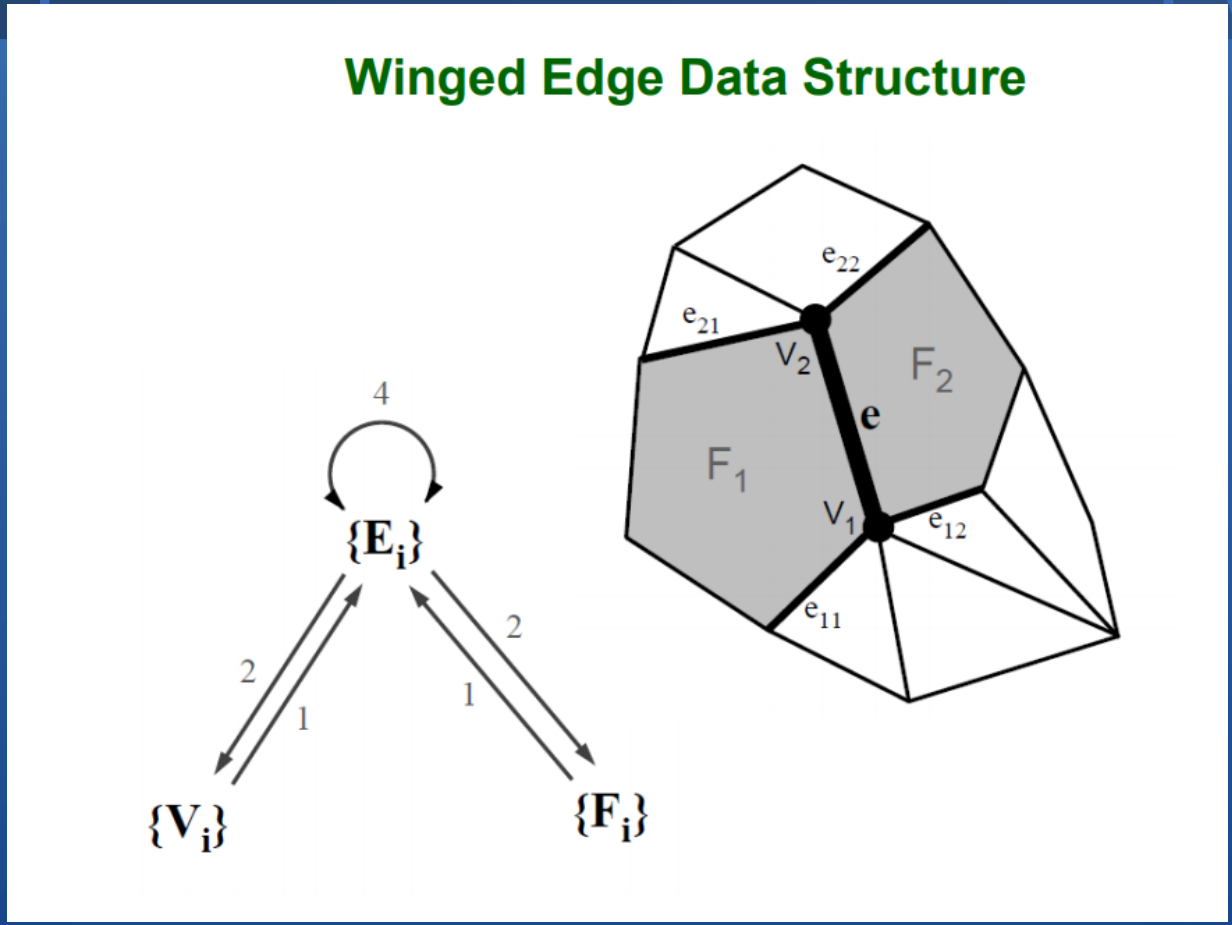
Przechowywanie i przetwarzanie opisu siatki

- Główną trudnością w przechowywaniu opisu siatki jest problematyczne, ze względu na konieczność przechowywania danych topologicznych i geometrycznych.
- Rozdzielenie danych geometrycznych od topologicznych pozwala osiągnąć mniejszą złożoność pamięciową
- Przechowywanie wszystkich powiązań topologicznych pozwala na wykonanie większości operacji na siatce w czasie liniowym. Jest niestety znacznie obciążające pamięciowo

Przechowywanie i przetwarzanie opisu siatki

- Aby doprowadzić do kompromisu między złożonością pamięciową, a obliczeniową stosuje się określone struktury danych. Pozwalają one zoptymalizować oba aspekty, unikając jednocześnie redundancyjnych informacji

Winged Edge Data Structure



Winged Edge Data Structure

- Struktura krawędzi jest głównym nośnikiem danych
- Tylko wierzchołki zawierają dane geometryczne
- Wierzchołki i ściany zawierają tylko należność do danej krawędzi (z danych topologicznych)
- Pozwala na realizację zapytań o sąsiedztwo (krawędzi, wierzchołków i ścian) w czasie liniowym

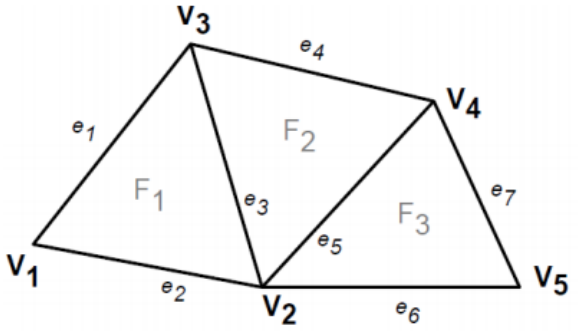
Winged Edge Data Structure

Winged Edge Data Structure

EDGE TABLE					11	12	21	22
e_1	V_1	V_3		F_1	e_2	e_2	e_4	e_3
e_2	V_1	V_2	F_1		e_1	e_1	e_3	e_6
e_3	V_2	V_3	F_1	F_2	e_2	e_5	e_1	e_4
e_4	V3	V_4		F_2	e_1	e_3	e_7	e_5
e_5	V_2	V_4	F_2	F_3	e_3	e_6	e_4	e_7
e_6	V_2	V_5	F_3		e_5	e_2	e_7	e_7
e_7	V_4	V_5		F_3	e_4	e_5	e_6	e_6

FACE TABLE	
F_1	e_1
F_2	e_3
F_3	e_5

VERTEX TABLE				
V_1	X_1	Y_1	Z_1	e_1
V_2	X_2	Y_2	Z_2	e_6
V_3	X_3	Y_3	Z_3	e_3
V_4	X_4	Y_4	Z_4	e_5
V_5	X_5	Y_5	Z_5	e_6



Winged Edge Data Structure

```
class EdgeW:
    def __init__(self,vOrg,vDest,fl=None,fr=None,elcw=None,elcww=None,ercw=None,ercww=None):
        self.vertexOrigin=vOrg
        self.vertexDestination=vDest
        self.faceLeft=fl
        self.faceRight=fr
        self.edgeLeftCw=elcw
        self.edgeLeftCww=elcww
        self.edgeRighCw=ercw
        self.edgeRightCww=ercww
class VertexW:
    def __init__(self,x,y,edge):
        self.x=x
        self.y=y
        self.edge=edge
class FaceW:
    def __init__(self,edge):
        self.edge=edge
```

Winged Edge Data Structure

- Tworzenie Struktury z danych podanych jako wierzchołki, oraz trójkąty
- 1.Stworzenie wierzchołków i przypisanie do nich współrzędnych (O(V))
- 2. Stworzenie krawędzi i ścian, przypisanie do krawędzi nich wierzchołków końcowych, początkowych i nowostoworzonych ścian. (O(T))
- 3.Przypisanie krawędzi do wierzchołków (O(V*T))
- 4.Przypisanie krawędziom następników, poprzedników i lewych oraz prawych ścian (O(T^2*E))
- Ostatecznie: (O(T^2*E))

Winged Edge Data Structure

Sąsiedztwo 1 stopnia – złożoność jednostkowa O(1) – przeglądanie sąsiadów krawędzi startowej

```
def simpleFindAdjacentVertexesCCW(vertex,edges):
    firstEdge=vertex.edge
    edge=vertex.edge
    adjacentVertexes=[]
    if(edge.vertexDestination==vertex):
        adjacentVertexes.append(edge.vertexOrigin)
        edge=edge.edgeRightCw
    else:
        adjacentVertexes.append(edge.vertexDestination)
        edge=edge.edgeLeftCww

    while(edge != firstEdge):
        if(edge.vertexDestination==vertex):
            adjacentVertexes.append(edge.vertexOrigin)
            edge=edge.edgeRightCw
        else:
            adjacentVertexes.append(edge.vertexDestination)
            edge=edge.edgeLeftCww
    return adjacentVertexes
```

Winged Edge Data Structure

Sąsiednie ściany, złożoność jednostkowa (O(1))

```
def findAdjacentFaces(face,edges,visualization,scenes=None,redlines=None):
    startEdge=face.edge
    edge=face.edge
    adjacentFaces=[]
    firstFaceEdges=findFaceEdges(face,edges)
    while(True):
        if(face==edge.faceLeft):
            adjacentFaces.append(edge.faceRight)
            edge=edge.edgeLeftCw
        else:
            adjacentFaces.append(edge.faceLeft)
            edge=edge.edgeRightCww

        if(edge==startEdge):
            break

    return adjacentFaces,scenes
```