

# Algorytmy geometryczne

Przetwarzanie i przechowywanie opisu siatki  
trójkątnej na płaszczyźnie

Dominik Jastrząb

# Przechowywanie i przetwarzanie opisu siatki

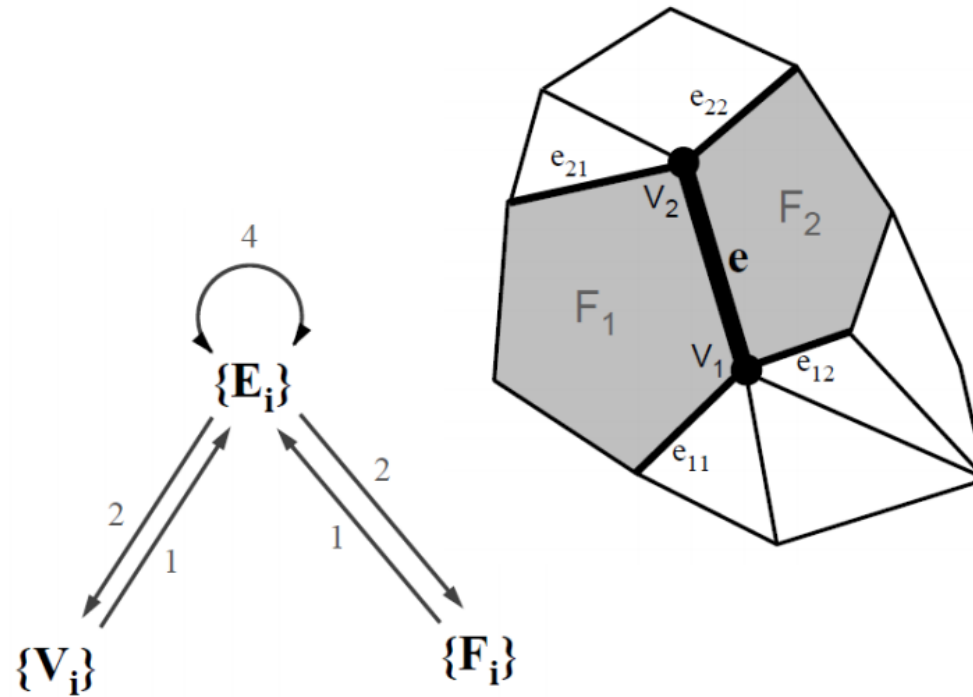
- Główną trudnością w przechowywaniu opisu siatki jest problematyczne, ze względu na konieczność przechowywania danych topologicznych i geometrycznych.
- Rozdzielenie danych geometrycznych od topologicznych pozwala osiągnąć mniejszą złożoność pamięciową
- Przechowywanie wszystkich powiązań topologicznych 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



# 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

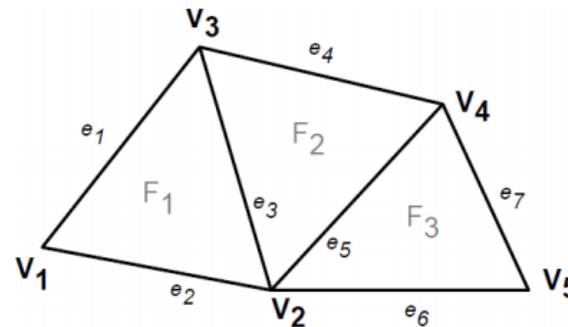
# Winged Edge Data Structure

## Winged Edge Data Structure

EDGE TABLE							
				11	12	21	22
e <sub>1</sub>	V <sub>1</sub> V <sub>3</sub>		F <sub>1</sub>	e <sub>2</sub>	e <sub>2</sub>	e <sub>4</sub>	e <sub>3</sub>
e <sub>2</sub>	V <sub>1</sub> V <sub>2</sub>	F <sub>1</sub>		e <sub>1</sub>	e <sub>1</sub>	e <sub>3</sub>	e <sub>6</sub>
e <sub>3</sub>	V <sub>2</sub> V <sub>3</sub>	F <sub>1</sub> F <sub>2</sub>		e <sub>2</sub>	e <sub>5</sub>	e <sub>1</sub>	e <sub>4</sub>
e <sub>4</sub>	V <sub>3</sub> V <sub>4</sub>		F <sub>2</sub>	e <sub>1</sub>	e <sub>3</sub>	e <sub>7</sub>	e <sub>5</sub>
e <sub>5</sub>	V <sub>2</sub> V <sub>4</sub>	F <sub>2</sub> F <sub>3</sub>		e <sub>3</sub>	e <sub>6</sub>	e <sub>4</sub>	e <sub>7</sub>
e <sub>6</sub>	V <sub>2</sub> V <sub>5</sub>	F <sub>3</sub>		e <sub>5</sub>	e <sub>2</sub>	e <sub>7</sub>	e <sub>7</sub>
e <sub>7</sub>	V <sub>4</sub> V <sub>5</sub>		F <sub>3</sub>	e <sub>4</sub>	e <sub>5</sub>	e <sub>6</sub>	e <sub>6</sub>

FACE TABLE	
F <sub>1</sub>	e <sub>1</sub>
F <sub>2</sub>	e <sub>3</sub>
F <sub>3</sub>	e <sub>5</sub>

VERTEX TABLE				
V <sub>1</sub>	X <sub>1</sub>	Y <sub>1</sub>	Z <sub>1</sub>	e <sub>1</sub>
V <sub>2</sub>	X <sub>2</sub>	Y <sub>2</sub>	Z <sub>2</sub>	e <sub>6</sub>
V <sub>3</sub>	X <sub>3</sub>	Y <sub>3</sub>	Z <sub>3</sub>	e <sub>3</sub>
V <sub>4</sub>	X <sub>4</sub>	Y <sub>4</sub>	Z <sub>4</sub>	e <sub>5</sub>
V <sub>5</sub>	X <sub>5</sub>	Y <sub>5</sub>	Z <sub>5</sub>	e <sub>6</sub>



# 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.edgeRightCw=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 nowostworzonych ścian. ( $O(T)$ )
- 3. Przypisanie krawędzi do wierzchołków ( $O(V \cdot T)$ )
- 4. Przypisanie krawędziom następników, poprzedników i lewych oraz prawych ścian ( $O(T^2 \cdot E)$ )
- Ostatecznie: ( $O(T^2 \cdot E)$ )



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

# Winged Edge Data Structure

Sąsiedztwo 2 stopnia – złożoność liniowa  $O(st.(V))$  – znalezienie sąsiadów pierwszego stopnia, sąsiadów dla sąsiadów pierwszego stopnia, oraz eliminacja niewłaściwych wierzchołków i krawędzi wierzchołków

```
def findSecondAdjacentVertexes(vertex, edges):
    firstEdge=vertex.edge
    edge=vertex.edge
    adjacentVertexes=findAdjacentVertexes2(vertex, edges)
    adjacentedges=[]
    secondAdjacentVertexes=[]

    for av in adjacentVertexes:
        adjacentEdges=findAdjacentEdges2(av, edges)
        for edge in adjacentEdges:
            if(edge.vertexDestination in adjacentVertexes and edge.vertexOrigin in adjacentVertexes):
                adjacentEdges.remove(edge)
        for edge in adjacentEdges:
            if(edge.vertexDestination in adjacentVertexes):
                if(edge.vertexOrigin not in secondAdjacentVertexes ):
                    secondAdjacentVertexes.append(edge.vertexOrigin)
            if(edge.vertexOrigin in adjacentVertexes):
                if(edge.vertexDestination not in secondAdjacentVertexes ):
                    secondAdjacentVertexes.append(edge.vertexDestination)
    secondAdjacentVertexes.remove(vertex)
    return secondAdjacentVertexes
```