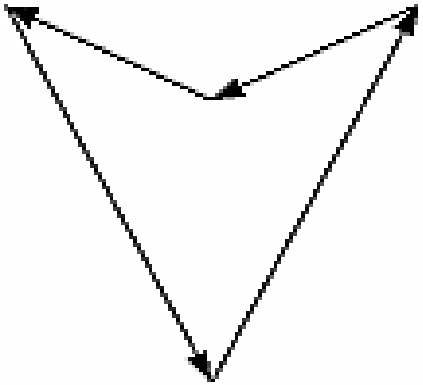
# Diskrétne Geometrické Štruktúry

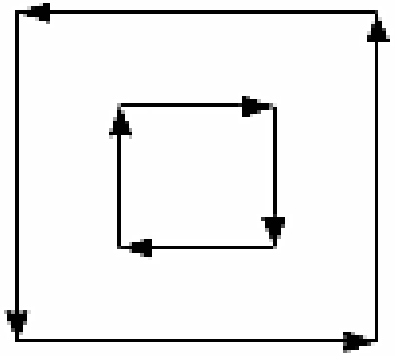
## 6. Reprezentácia objektov II

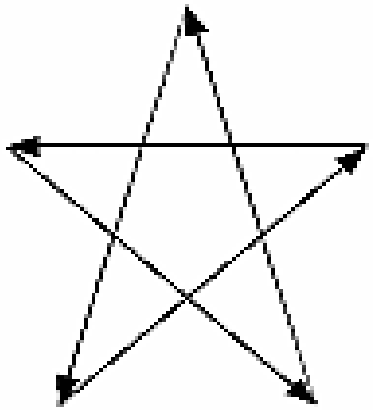
Martin Samuelčík
samuelcik@sccg.sk, www.sccg.sk/~samuelcik, I4

# Zložité steny

- Definované steny pomocou zložitých polygónov
- Steny s dierami, so samopretínajúcimi sa hranami
- Stena definovaná určitým počtom kontúr







# Zložité steny

- Reprezentované množinou kontúr – lomených čiar
- Orientácia kontúr
- Spracovanie → prepojenie dier, rozdelenie na jednoduché polygóny, triangulácia (GLU teselátor, CGAL, Visualization Library, ...)

```
struct Vertex
{
        float x, y, z;
}
```

```
struct Edge
{
        Vertex* v1;
        Vertex* v2;
}
```

```
struct Contour
{
        vector<Edge*> edges;
}
```

```
struct Face
{
        vector<Contour*> contours;
}
```

# Orientácia

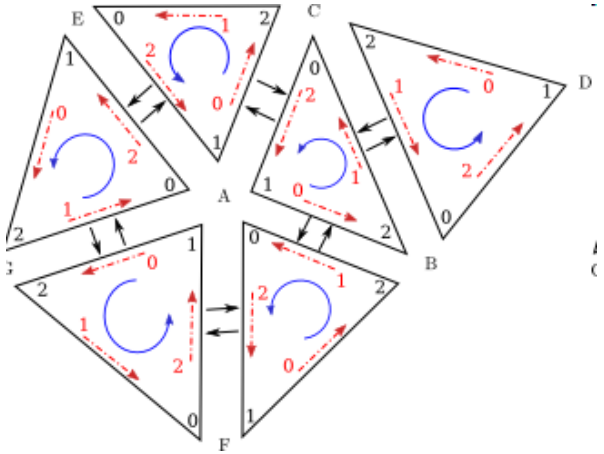
- Orientácia polygónu je daná postupnosťou vrcholov tvoriacou polygón, orientácia hrany je daná poradím jej 2 vrcholov
- Orientácia mnohostena (polygonálnej siete) je daná orientáciou stien tak, že na spoločnej hrane majú 2 steny opačnú orientáciu
- Obsah, objem

Jednoduchý polygón
s vrcholmi $[x_i, y_i]$

$$A = \frac{1}{2}\sum_{i=0}^{n-1}(x_i y_{i+1} - x_{i+1} y_i)$$

Konvexný mnohosten
$x_i$ je bod i-tej steny
$A_i$ je obsah i-tej steny
$n_i$ je normála i-tej steny

$$\text{volume} = \frac{1}{3}\sum_{\text{face } i} \vec{x}_i \cdot \hat{n}_i A_i$$

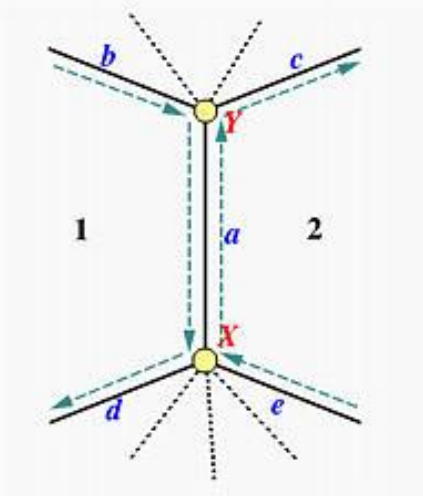


# Rozšírenie zoznamov

- Podľa aktuálneho použitia objektov
- Pre topologické požiadavky na povrchu objektu
  - Pridanie topologickej informácie pre časti štruktúry
  - Pridanie informácií pre každú časť, najčastejšie a najviac sa pridávajú informácie pre hrany
  - Vrcholy a steny obsahujú odkaz na jednu incidentnú hranu

# Winged Edge

- Rozšírenie informácie pre hranu o susedné prvky
- V hrane informácie o susedných vrcholoch, hranách a stenách
- Položky dané orientáciou




*a* – aktuálna hrana
*X* – začiatočný bod hrany
*Y* – koncový bod hrany
*b* – predchádzajúca hrana pri prechode ľavou stenou
*d* – nasledujúca hrana pri prechode ľavou stenou
*c* – nasledujúca hrana pri prechode pravou stenou
*e* – predchádzajúca hrana pri prechode pravou stenou
1 – ľavá stena
2 – pravá stena

# Winged Edge

- V niektorých prípadoch sa ukladajú len dve susedné hrany, pre každú stenu jedna - nasledujúca hrana
- Pre každú stenu a vrchol sa uloží jedna incidentná hrana, v prípade zložitých stien hrana pre každú kontúru

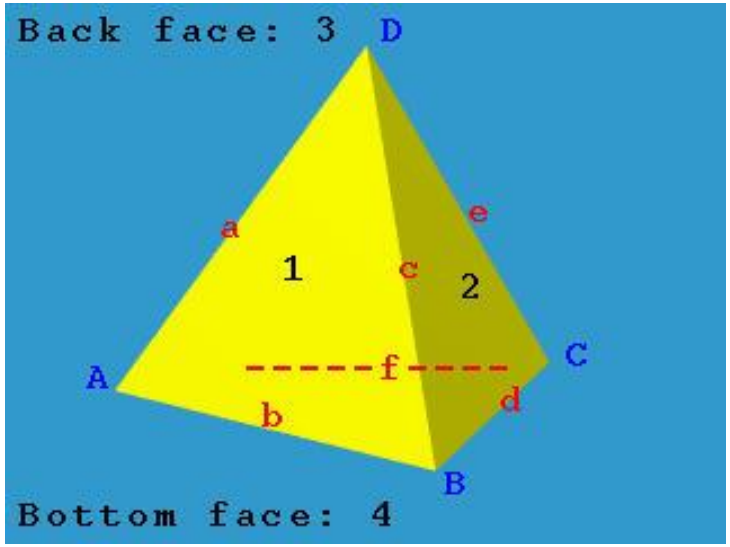
```
struct Vertex
{
        float x, y, z;
        WingedEdge* edge
}
```

```
struct Face
{
        WingedEdge* outer_edge;
        //vector< WingedEdge*> inner_edges;
}
```

```
struct WingedEdge
{
        Vertex* X;
        Vertex* Y;
        WingedEdge* b;
        WingedEdge* c;
        WingedEdge* d;
        WingedEdge* e;
        Face* 1;
        Face* 2;
}
```

```
struct Mesh
{
        vector<Vertex*> vertices;
        vector<WingedEdge*> edges;
        vector<Face*> faces;
}
```

# Príklad




| Edge | Vertices | | Faces | | Left Traverse | | Right Traverse | |
|---|---|---|---|---|---|---|---|---|
| Name | Start | End | Left | Right | Pred | Succ | Pred | Succ |
| a | A | D | 3 | 1 | e | f | b | c |
| b | A | B | 1 | 4 | c | a | f | d |
| c | B | D | 1 | 2 | a | b | d | e |
| d | B | C | 2 | 4 | e | c | b | f |
| e | C | D | 2 | 3 | c | d | f | a |
| f | A | C | 4 | 3 | d | b | a | e |

| Vertex Name | Incident Edge |
|---|---|
| A | a |
| B | b |
| C | d |
| D | e |

| Face Name | Incident Edge |
|---|---|
| 1 | a |
| 2 | c |
| 3 | a |
| 4 | b |

# Topologické prehľadávanie

```
WingedEdgeFF(Face* face)
{
	WingedEdge* start_edge = face->outer_edge;
	WingedEdge* current_edge;
	if (start_edge->1 == face)
	{
		result.Add(start_edge->2);
		current_edge = start_edge->d;
	}
	else if (start_edge->2 == face)
	{
		result.Add(start_edge->2);
		current_edge = start_edge->c;
	}
	else return;
	while (current_edge != start_edge)
	{
		if (current_edge->1 == face)
		{
			result.Add(current_edge->2);
			current_edge = current_edge->d;
		}
		else if (current_edge->2 == face)
		{
			result.Add(current_edge->1);
			current_edge = current_edge->c;
		}
	}
	return result;
}
```

```
WingedEdgeVE(Vertex* vertex)
{
	WingedEdge* start_edge = vertex->edge;
	WingedEdge* current_edge;
	WingedEdge* prev_edge = start_edge;
	if (vertex == start_edge->X)
		current_edge = start_edge->d;
	else
		current_edge = start_edge->c;
	result.Add(start_edge);
	while (current_edge != start_edge)
	{
		result.Add(current_edge);
		if (vertex == current_edge->X)
		{
			if (prev_edge == current_edge->e)
				current_edge = current_edge->d;
			else
				current_edge = current_edge->e;
		}
		else
		{
			if (prev_edge == current_edge->c)
				current_edge = current_edge->b;
			else
				current_edge = current_edge->c;
		}
		prev_edge = result.Last();
	}
	return result;
}
```

# DCEL

- Winged edge
  - Problémy s orientáciou
  - Problém so stenami s dierami
- Riešenie – rozbitie hrany na dve polhrany, polhrany sú prepojené s danou stenou, určujú orientáciu danej steny
- Double Connected Edge List
- Half edge

# Half-edge

- Polhrana obsahuje identifikátor na opačnú polhranu, spolu generujú hranu
- Obsahuje identifikátor steny, ku ktorej patrí
- Obsahuje jeden identifikátor vrcholu, kde polhrana začína alebo končí
- Uložené sú aj nasledujúca resp. predchádzajúca polhrana v rámci steny
- V prípade opačnej polhrany sa môže odkazovať na NULL polhranu

# DCEL štruktúra

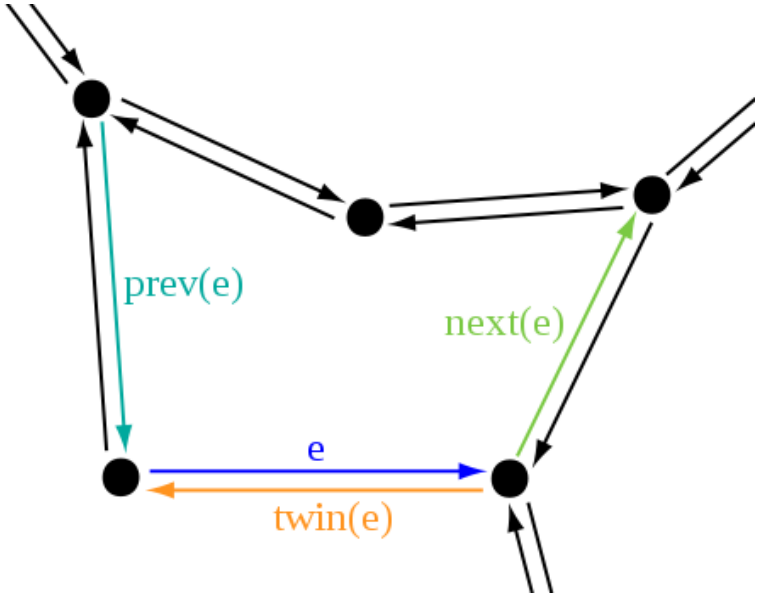
- Použitie polhrany
- Ostatné prvky podobné ako vo Winged edge
- Možnosť reprezentovať steny s dierami

```
struct Vertex
{
        float x, y, z;
        HalfEdge* edge;
}
```

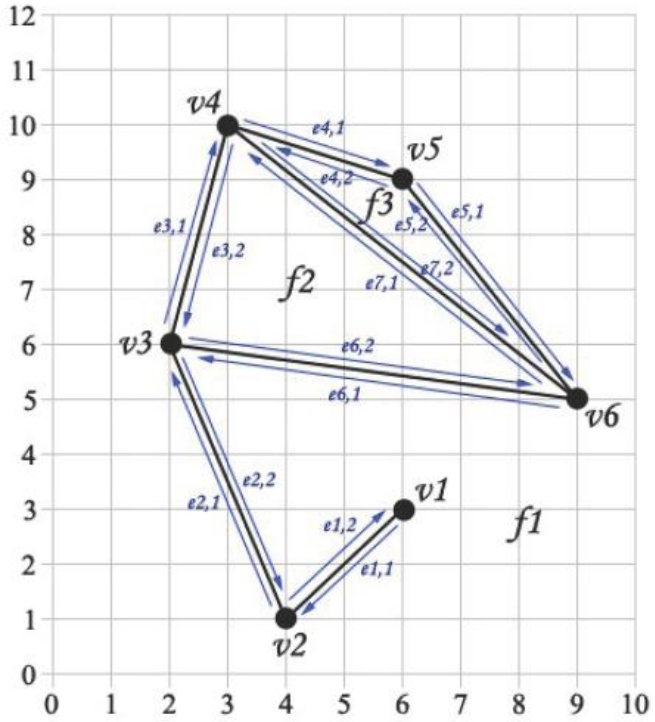
```
struct Face
{
        HalfEdge* outer_edge;
        //vector< HalfEdge*> inner_edges;
}
```

```
struct HalfEdge
{
        Vertex* origin;
        HalfEdge* opp;
        HalfEdge* next;
        //HalfEdge* prev;
        Face* face;
}
```

```
struct DCEL
{
        vector<Vertex*> vertices;
        vector<HalfEdge*> edges;
        vector<Face*> faces;
}
```



# Príklad




| Half-Edge | Origin | Twin | Incident Face | Next | Prev |
|---|---|---|---|---|---|
| e1,1 | v1 | e1,2 | f1 | e2,1 | e1,2 |
| e1,2 | v2 | e1,1 | f1 | e1,1 | e2,2 |
| e2,1 | v2 | e2,2 | f1 | e3,1 | e1,1 |
| e2,2 | v3 | e2,1 | f1 | e1,2 | e6,1 |
| e3,1 | v3 | e3,2 | f1 | e4,1 | e2,1 |
| e3,2 | v4 | e3,1 | f2 | e6,2 | e7,1 |
| e4,1 | v4 | e4,2 | f1 | e5,1 | e3,1 |
| e4,2 | v5 | e4,1 | f3 | e7,2 | e5,2 |
| e5,1 | v5 | e5,2 | f1 | e6,1 | e4,1 |
| e5,2 | v6 | e5,1 | f3 | e4,2 | e7,2 |
| e6,1 | v6 | e6,2 | f1 | e2,2 | e5,1 |
| e6,2 | v3 | e6,1 | f2 | e7,1 | e3,2 |
| e7,1 | v6 | e7,2 | f2 | e3,2 | e6,2 |
| e7,2 | v4 | e7,1 | f3 | e5,2 | e4,2 |

| Face | Outer Comp. | Inner Comp. |
|---|---|---|
| f1 | nil | f2, f3 (e3,1) (e4,1) |
| f2 | f1, f3 (e3,2) | nil |
| f3 | f1, f2 (e4,2) | nil |

| Vertex | Coordinates | IncidentEdge |
|---|---|---|
| v1 | (6, 3) | e1,1 |
| v2 | (4, 1) | e2,1 |
| v3 | (2, 6) | e3,1 |
| v4 | (3, 10) | e4,1 |
| v5 | (6, 9) | e5,1 |
| v6 | (9, 5) | e6,1 |

# Topologické prehľadávanie

```
HalfEdgeFF(Face* face)
{
        HalfEdge* start_edge = face->outer_edge;
        if (start_edge->opp)
                result.Add(start_edge->opp->face);
        HalfEdge* current_edge = start_edge->next;
        while (current_edge  && current_edge != start_edge)
        {
                result.Add(current_edge->opp->face);
                current_edge = current_edge->next;
        }
        return result;
}
```

Časová náročnosť je konštantná resp. závislá len od počtu výstupných elementov.

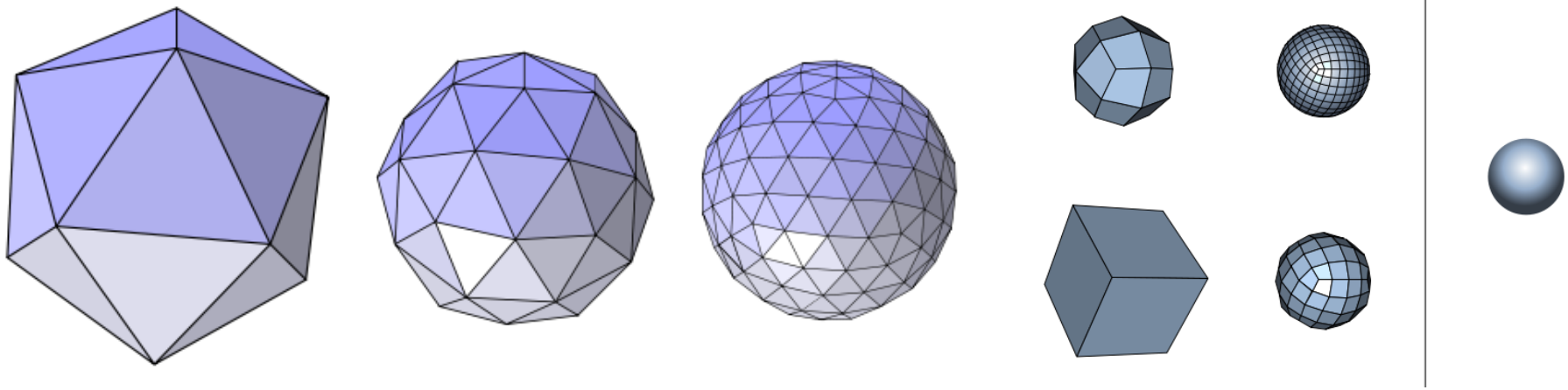
```
HalfEdgeVE(Vertex* vertex)
{
        HalfEdge* start_edge = vertex->edge;
        result.Add(start_edge);
        HalfEdge* current_edge = start_edge->opp->next;
        while (current_edge  && current_edge != start_edge)
        {
                result.Add(current_edge);
                current_edge = current_edge->opp->next;
        }
        return result;
}
```

# Generovanie DCEL

- Najčastejší prípad – generovanie zo zoznamu stien:
  - naplň zoznam vrcholov a stien v DCEL podľa daného zoznamu
  - pre každú stenu prejdi hrany tej steny, vytvor half-edge, naplň origin, next, prev, face
  - pre naplnenie opp (opačnej polhrany) v každej half-edge treba zistiť susednosti jednotlivých stien
  - do vrcholov a stien sa pridá jedna ľubovoľná incidentná polhrana

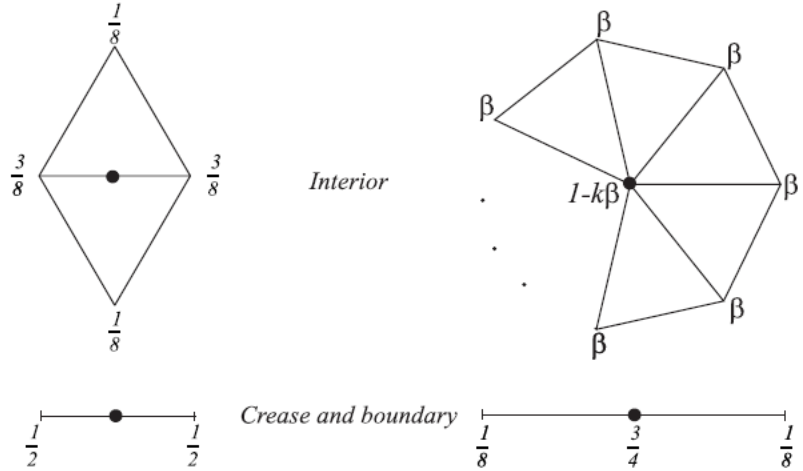
# Prerozdeľovacie plochy

- Algoritmus vytvorí z množiny polygónov A hustejšiu množinu polygónov B tak aby sa splnili vlastnosti
  - Hladkosť, spojitosť, aproximácia, interpolácia vrcholov A pomocou B
- Algoritmy pre trojuholníkové, polygonálne siete



# Loop algoritmus

- Charles Loop, 1987
- Trojuholníkové siete
- Vrcholy siete A sa posunú na nové pozície
- Na hranách siete A sa vytvoria nové body
- Pre každú stenu siete A sa vytvoria 4 nové trojuholníky siete B s vrcholmi v nových bodoch




a. Masks for odd vertices

b. Masks for even vertices

Original Loop

$$\beta = \frac{1}{n}\left(\frac{5}{8} - \left(\frac{3}{8} + \frac{1}{4}\cos\frac{2\pi}{n}\right)^2\right)$$

Warren

$$\beta = \begin{cases} \frac{3}{8n} & n > 3 \\ \frac{3}{16} & n = 3 \end{cases}$$

# Loop pre half-edge

- Pre každý vrchol z A sa vytvorí nový posunutý “even” vrchol pre sieť B, pričom v starom vrchole sa zachová odkaz na nový vrchol
- Pre každý trojuholník v A sa vztvoria 3 nové “odd” vrcholy (ak už neboli vytvorené v susednom trojuholníku) a priradia sa príslušnej starej polhrane
- Pre každý trojuholník v A sa vytvorí 12 nových polhrán a naplní sa príslušnými vrcholmi, každá stará polhrana si zapamätá 2 príslušné nové polhrany + sa prepoja protiľahlé nové polhrany tam kde to je známe

# Quad-edge

- Štruktúra používaná hlavne pre reprezentáciu duálnych grafov
- Vrcholy a steny majú podobné postavenie v štruktúre
- Hrany spájajúce vrcholy a „hrany“ spájajúce steny sú zoskupené do jednej štruktúry
- Používanie orientovaných polhrán – 4 polhrany tvoria jeden celok

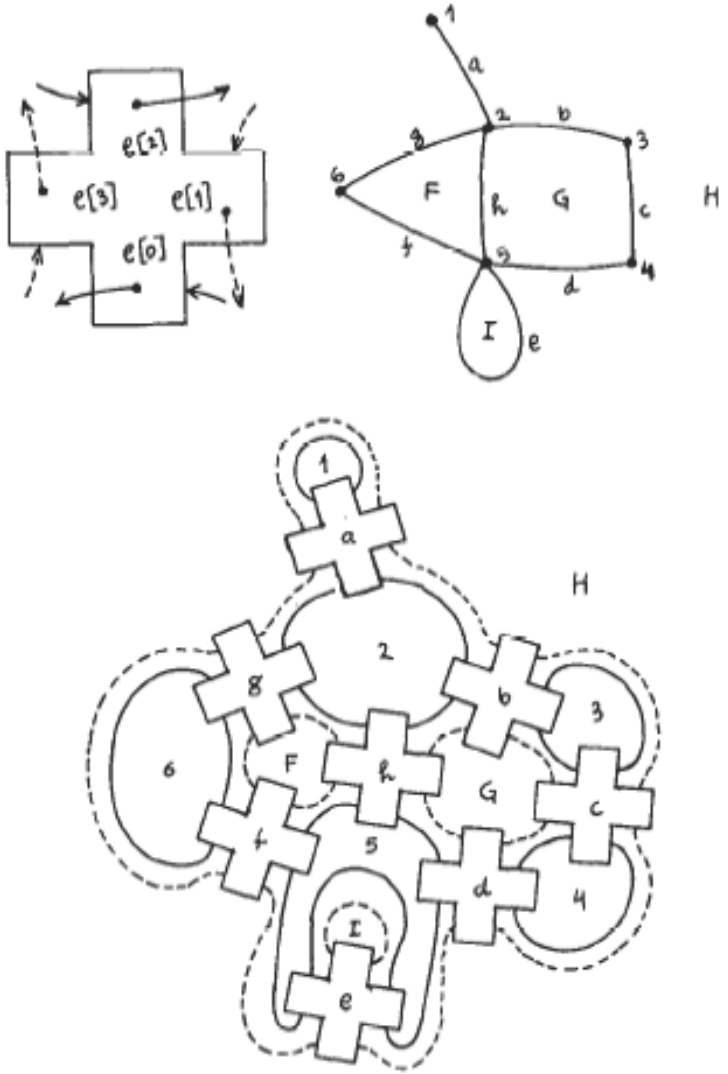
# Quad-edge

```
struct Vertex
{
        float x, y, z;
        Edge* edge;
}
```

```
struct Face
{
        Edge* edge;
}
```

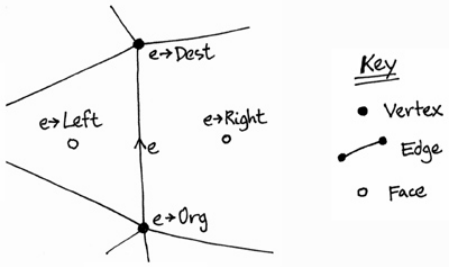
```
struct Edge
{
        Edge* next;  // Onext
        void* data;    // vertex, face info
        QuadEdge* parent;
}
```

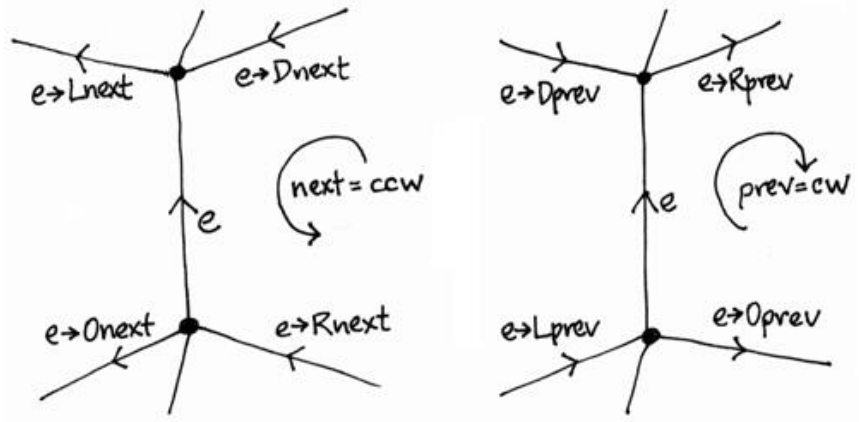
```
struct QuadEdge
{
        Edge* e[4];
}
```



# Algebra na hranách

- Prehľadávanie hrán v rámci štruktúry pre danú polhranu:
  - Rot – otočenie polhrany o 90° proti smeru hod. ručičiek
  - Sym – symetrická polhrana k danej polhrane
  - Next – nasledújúca polhrana; môže byť okolo začiatku, konca, ľavého alebo pravého objektu (Onext, Dnext, Lnext, Rnext)
  - Prev – predchádzajúca polhrana
  - Org – začiatočný objekt
  - Dest – koncový objekt
  - Left – objekt naľavo
  - Right – objekt napravo



# Algebra na hranách

- Rot(e) = e->parent->e[(r+1) mod 4];
- Sym(e) = Rot(Rot(e));
- Org(e) = e->data;
- Dest(e) = Sym(e)->data;
- $Rot^{-1}(e)$ = e->parent->e[(r+3) mod 4] = Rot(Rot(Rot(e)));
- Right(e) = $Rot^{-1}(e)$->data;
- Left(e) = Rot(e)->data;
- Onext(e) = e->next;
- Oprev(e) = Rot(Onext(Rot(e)));
- Dnext(e) = Sym(Onext(Sym(e)));
- Dprev(e) = $Rot^{-1}$(Onext($Rot^{-1}$(e)));
- Lnext(e) = Rot(Onext($Rot^{-1}$(e)));
- Lprev(e) = Sym(Onext(e));
- Rnext(e) = $Rot^{-1}$(Onext(Rot(e)));
- Rprev(e) = Onext(Sym(e));

•na všetko stačí Rot a Onext

# Topologické prehľadávanie

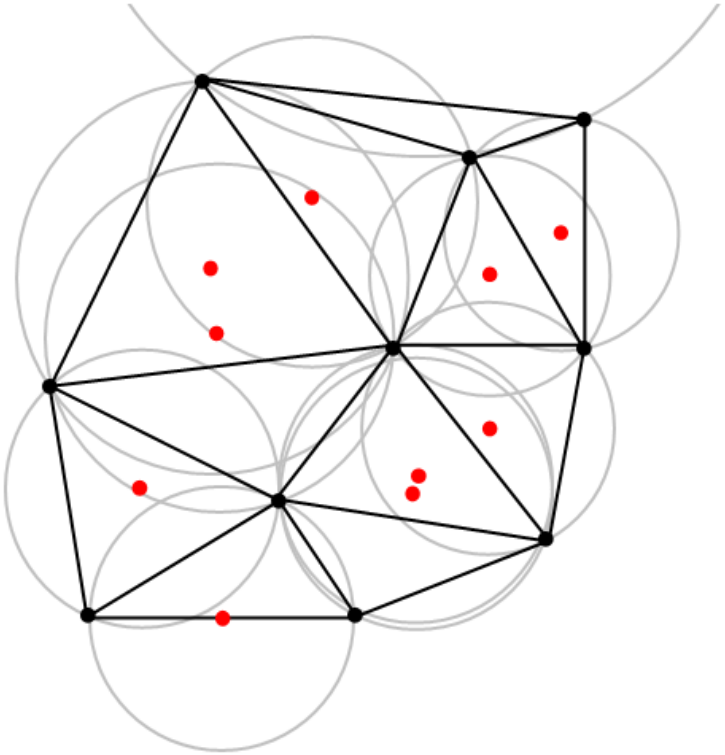
```
QuadEdgeFF(Face* face)
{
        Edge* start_edge = face->edge;
        result.Add(Sym(start_edge)->data);
        Edge* current_edge = Onext(start_edge);
        while (current_edge  && current_edge !=
        start_edge)
        {
                result.Add(Sym(current_edge)->data);
                current_edge = Onext(current_edge);
        }
        return result;
}
```

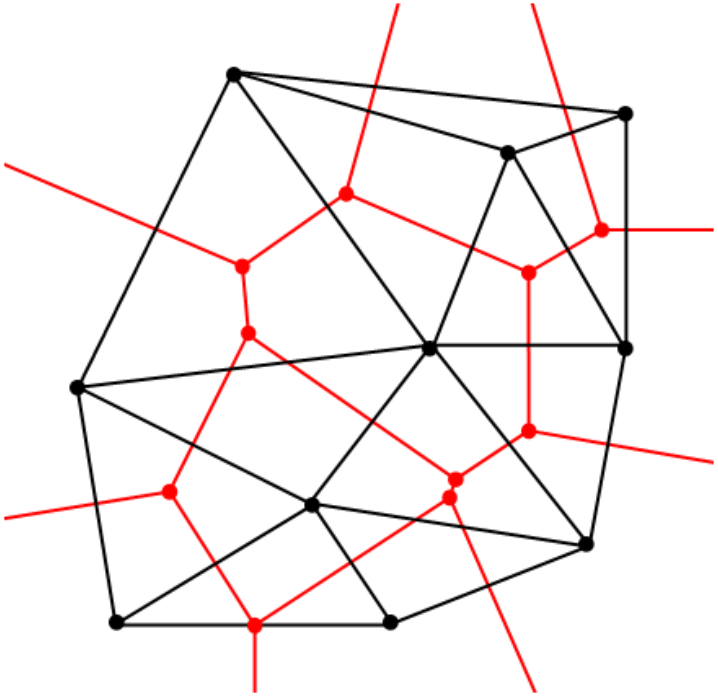
Časová náročnosť je konštantná resp. závislá len od počtu výstupných elementov.

```
QuadEdgeVE(Vertex* vertex)
{
        Edge* start_edge = vertex->edge;
        result.Add(start_edge);
        Edge* current_edge = Onext(start_edge);
        while (current_edge  && current_edge != start_edge)
        {
                result.Add(current_edge);
                current_edge = Onext(current_edge);
        }
        return result;
}
```

# Delaunay-Voronoi

- Delaunayova triangulácia
- Voronoiov diagram
- Duálne grafy





# Rozšírenie pre nevariety

- Nevariety – viac ako jeden prstenec v okolí vrcholu, viac ako 2 steny incidentné s jednou hranou
- Riešenie – zoznam stien pre hranu, zoznam hrán z vrcholu, pre každý prstenec jedna hrana

```
struct Vertex
{
        float x, y, z;
        vector<HalfEdge*> ring_edge;
}
```

```
struct HalfEdge
{
        Vertex* origin;
        vector<HalfEdge*> opp_edges;
        HalfEdge* next;
        HalfEdge* prev;
        Face* face;
}
```

# Výhody a nevýhody

- Kompaktné topologické štruktúry
- Urýchľujú topologické prehľadávanie až na konštantnú časovú zložitosť
- Zvýšená pamäťová náročnosť
- Pre vizualizáciu sa musia naspäť vytvárať jednoduché zoznamy
- Pomerne náročná príprava a aktualizácia štruktúr


# Otázky?