

Diskrétne geometrické štruktúry

8.

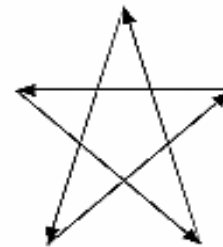
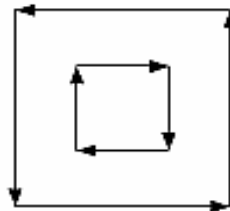
Martin Florek

florek@sccg.sk

www.sccg.sk/~florek

Zložité steny

- definované steny pomocou zložitých polygónov
- steny s dierami, so samopretínajúcimi sa hranami
- stena definovaná určitým počtom kontúr



Zložité steny 2

- reprezentované množinou kontúr – lomených čiar
- orientácia kontúr
- spracovanie → prepojenie dier, rozdelenie na jednoduché polygóny, triangulácia (GLU teselátor)

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

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

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

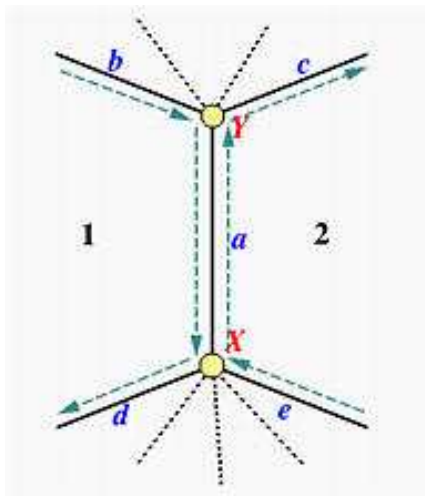
```
struct Face
{
    vector<Contour*> contours;
}
```

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

Winged Edge

- pozšírenie informácie pre hranu o susedné prvky
- v hrane informácie o susedných vrcholoch, hranách a stenách
- položky dané orientáciou



- a – aktuálna hrana
- X – začiatkový 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 2

- 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

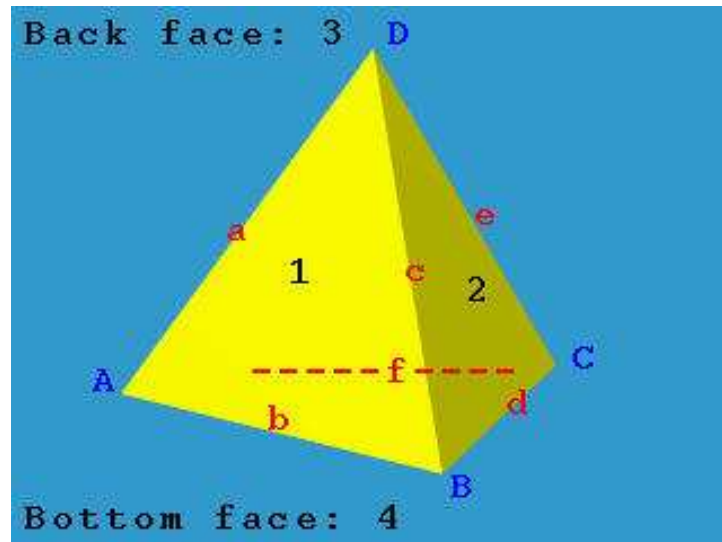
```
struct Vertex
{
    float x, y, z;
    WingedEdge* edge
}
```

```
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

- Double Connected Edge List
- 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
- 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

DCEL štruktúra

- ostatné prvky podobné ako vo Winged edge
- možnosť reprezentovať steny s dierami

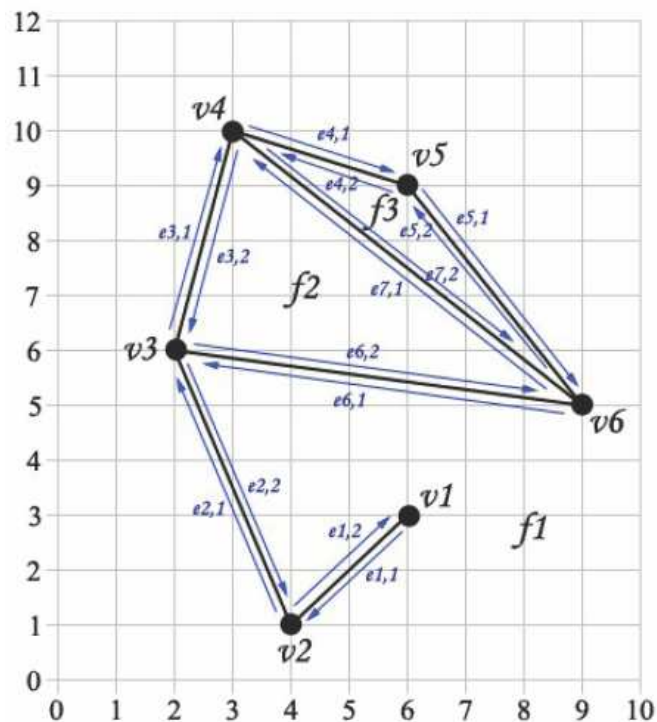
```
struct Vertex
{
    float x, y, z;
    HalfEdge* edge;
}
```

```
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

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

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

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;
}
```

```
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;
}
```

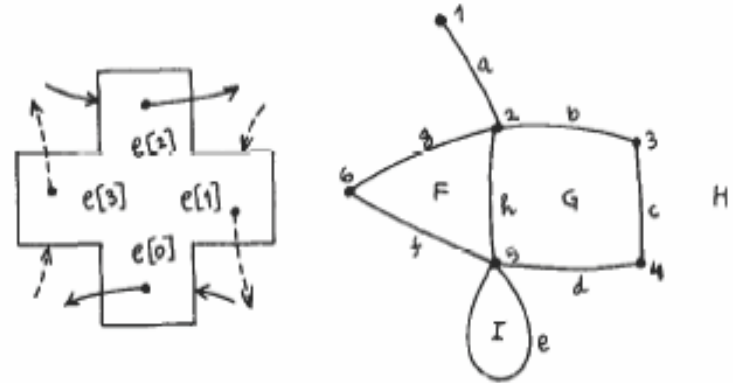
Generovanie DCEL

- najčastejší prípad – generovanie zo zoznamu stien:
 - naplň zoznam vrcholov a stien v DCEL podľa daného zoznamu
 - pre každú stenu prejdí 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

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 2

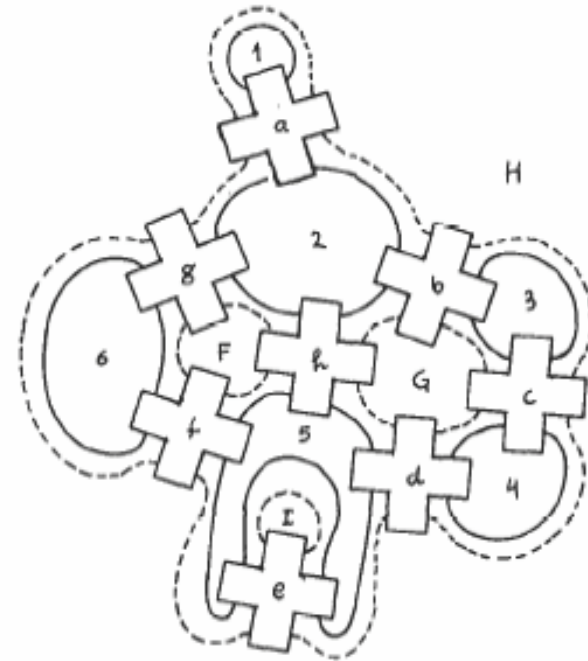


```
struct Vertex
{
    float x, y, z;
    Edge* edge;
}
```

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

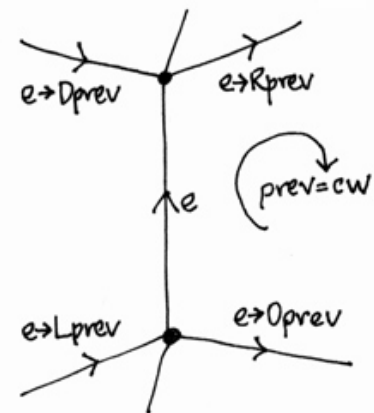
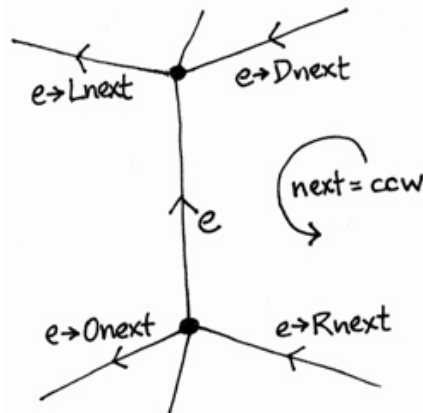
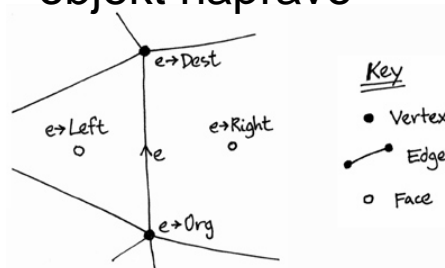
```
struct Edge
{
    Edge* next; // Onext
    void* data; // vertex, face info
    QuadEdge* parent;
}
```

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



Algebra na hranách

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



Algebra na hranách 2

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

• na všetko stačí Rot a Onext

Topologické prehľadávanie

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

```
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;
}
```

Rozšírenie pre nemanifolds

- nemanifolds – 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
- urýchlujú topologické prehľadávanie až na konštantnú časovú zložitosť
- zvýšená pamäťová náročnosť
- pre vizualizáciu sa musia naspäť vytvárať jednoduché zoznamy
- pomerne náročná príprava a aktualizácia štruktúr

koniec (-:

`florek@sccg.sk`