

Diskrétne Geometrické Štruktúry

5. Reprezentácie objektov

Martin Samuelčík

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

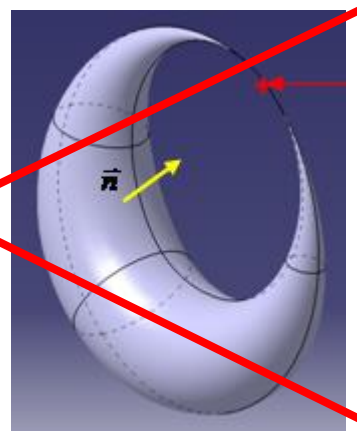
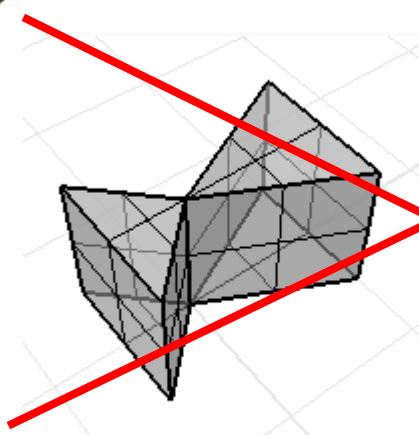
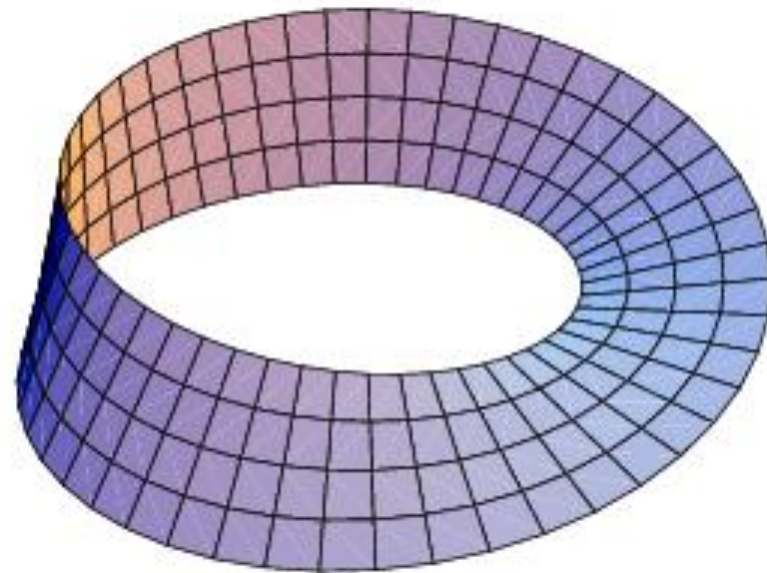
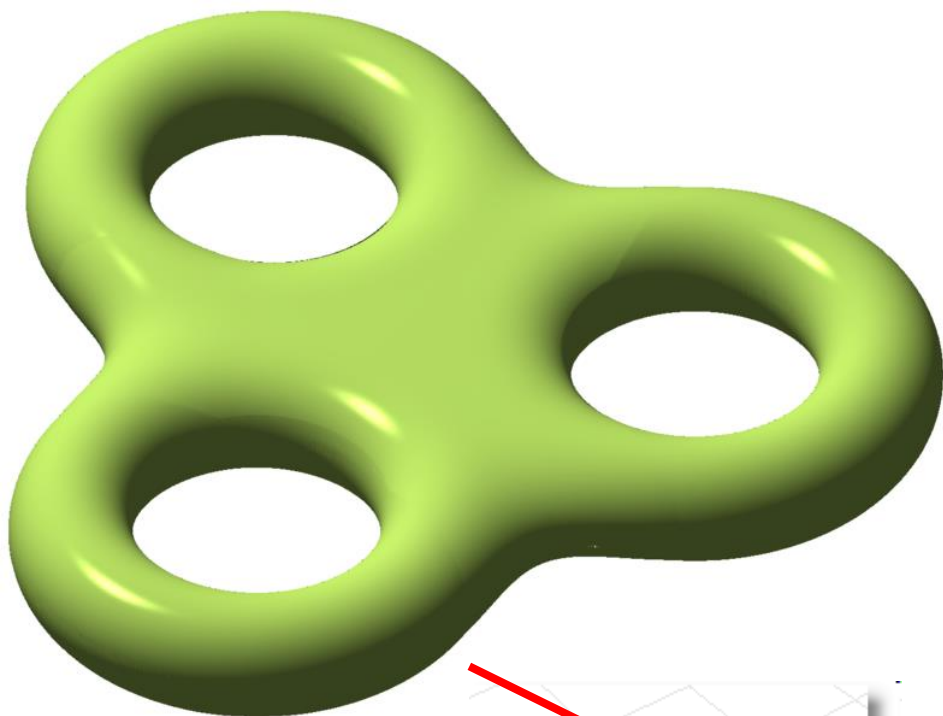
Variety

- n -varieta – množina bodov lokálne homeomorfná s n rozmerným Euklidovským priestorom
- Pre každý bod variety existuje jeho okolie homeomorfné s otvorenou n -rozmernou sférou

$$\mathbf{B}^n = \{(x_1, x_2, \dots, x_n) \in \mathbb{R}^n \mid x_1^2 + x_2^2 + \dots + x_n^2 < 1\}.$$

- Homeomorfizmus – spojitá bijekcia
- Predstavuje topologickú ekvivalenciu
- n -manifold


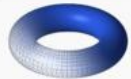


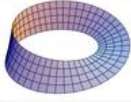
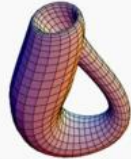
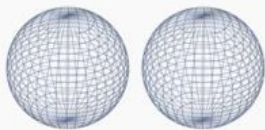
Variety








Vlastnosti variet

- Orientovateľnosť
- Rod (genus) – počet „dier“
- Eulerova charakteristika
 - konvexné polygóny, mnohosteny
 - $V-E+F = 2$
- Orientovateľné a uzavreté 2-variety
 - $V-E+F=2-2g$
 - $g = \text{genus}$
- Mapy, atlasy – zobrazenia častí variety na n -rozmernú sféru

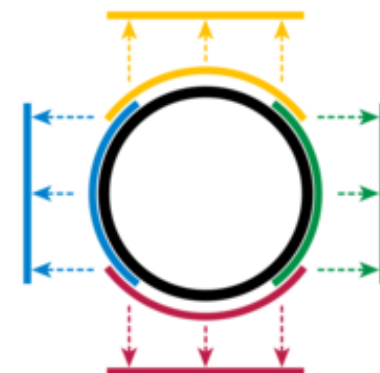
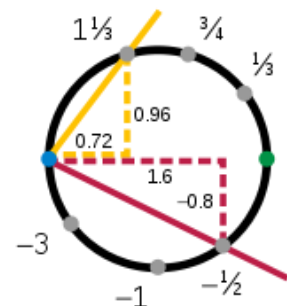
Eulerova charakteristika

Name	Image	Euler characteristic
Sphere		2
Torus		0
Double torus		-2
Triple torus		-4
Real projective plane		1
Möbius strip		0
Klein bottle		0
Two spheres (not connected)		$2 + 2 = 4$

Name	Image	Vertices V	Edges E	Faces F	Euler characteristic: $V - E + F$
Tetrahedron		4	6	4	2
Hexahedron or cube		8	12	6	2
Octahedron		6	12	8	2
Dodecahedron		20	30	12	2
Icosahedron		12	30	20	2

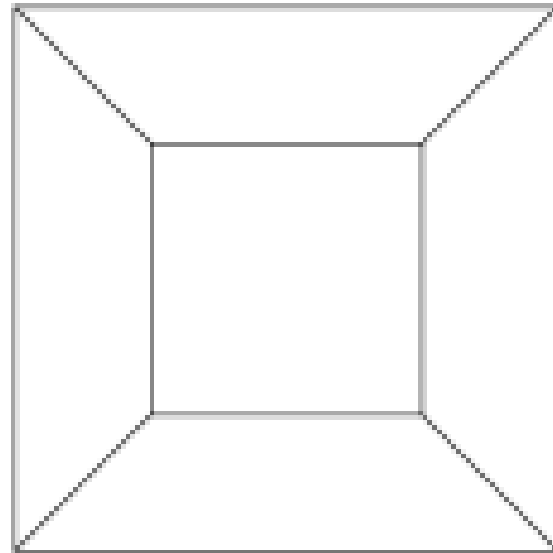
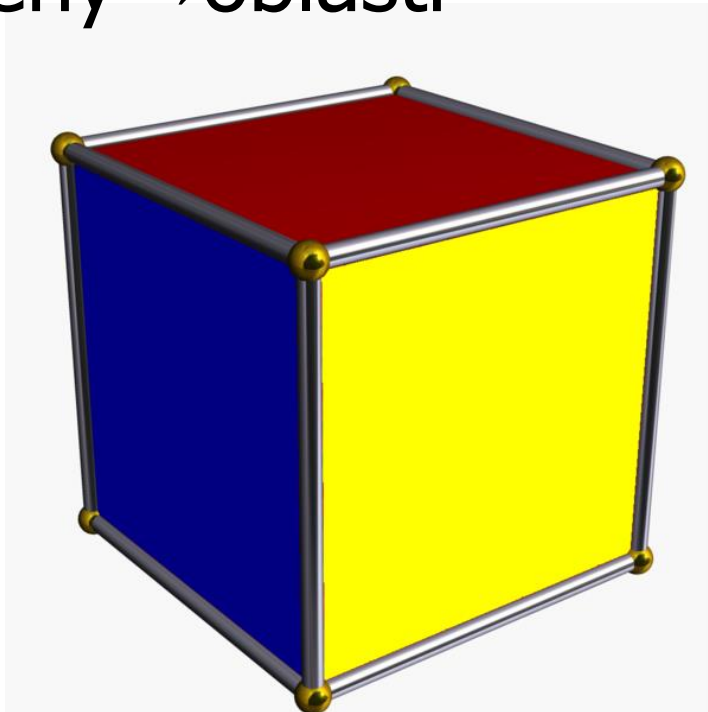
Mapy a Atlas

- Mapa predstavuje jedno zobrazenie časti n -variety do n -rozmerného priestoru
- Mapy tvoria atlas variety, snažíme sa o čo najmenší počet máp v atlase
- Väčšina variet potrebuje viac ako jednu mapu
- Kružnica
 - 4 mapy pre celú kružnicu
 - 1 mapa pre kružnicu bez jedného bodu
 - neexistuje atlas pre kružnicu s jednou mapou



Rovinný graf

- Orientovateľný polytop s rodom 0 sa dá pretransformovať na rovinný graf, vrcholy \rightarrow vrcholy, hrany \rightarrow hrany, steny \rightarrow oblasti

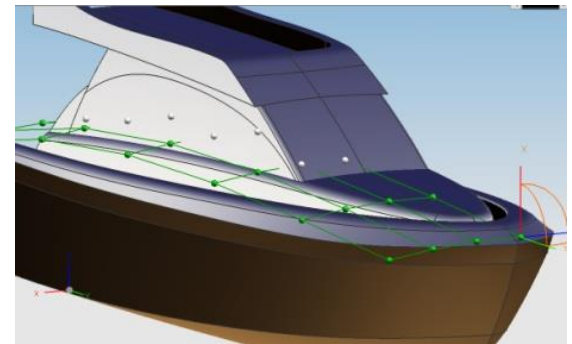
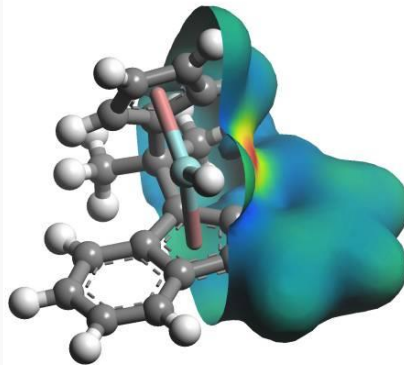
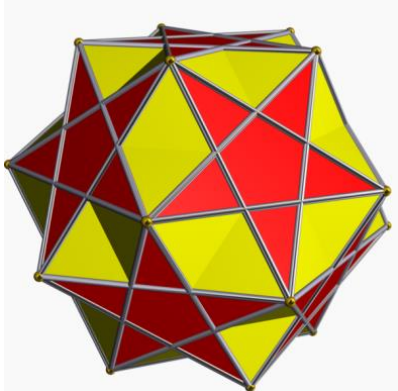
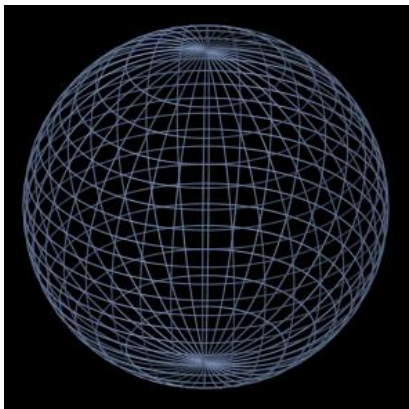


Kritériá pre reprezentácie

- Pamäťová náročnosť
- Reprezentácia neuzavretých objektov
- Konverzia medzi reprezentáciami
- Generovanie na základe vstupných dát
- Reprezentácia vnútra
- Geometrické algoritmy (prieniky, transformácie, ...)
- Topologické algoritmy
- Vizualizácia

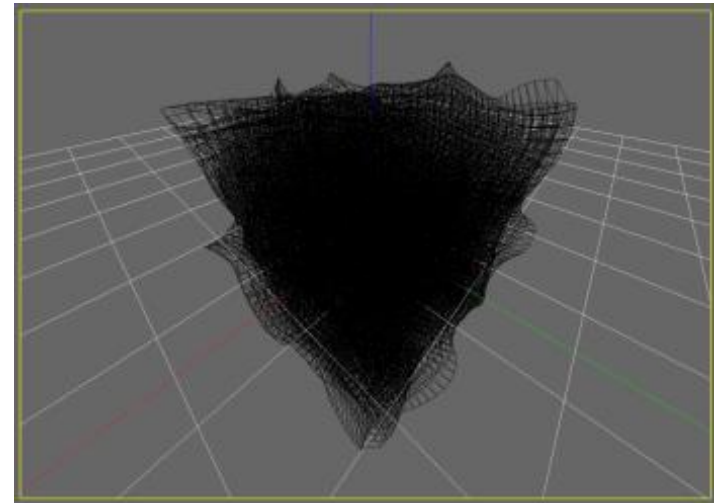
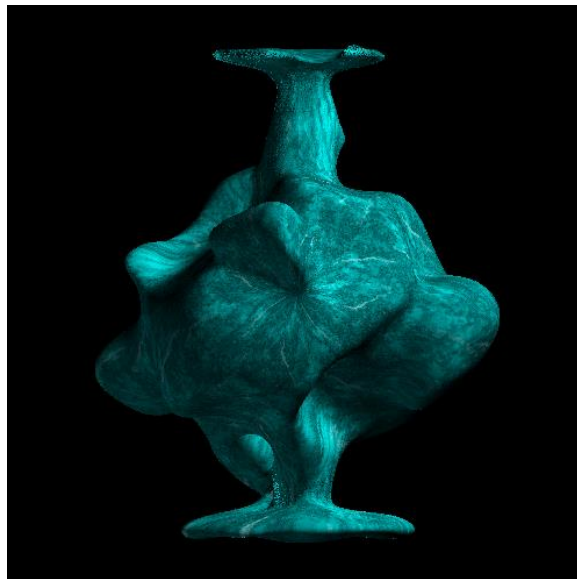
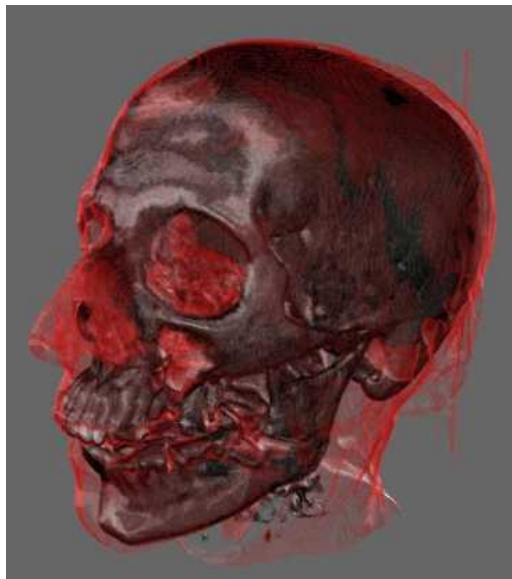
Reprezentácie 2-variet

- Povrchová reprezentácia (boundary representation, b-rep)
 - Drôtený model (wireframe)
 - Množina polygónov (meshes)
 - Implicitné povrchy (blobby)
 - Parametrické povrchy (Bézier, NURBS)



Reprezentácie 3-variet

- Diskrétna volumetrická reprezentácia (binárna, vzdialenostné polia)
- Mračná bodov
- Funkcionálna reprezentácia, f-rep
- Parametrické telesá



Množina polygónov

- Polytopy (mnohosteny), meše, planárne grafy
- Pamäťová náročnosť
- Definovanie častí (vrcholy, hrany, steny)
- Indexy alebo smerníky
- Zložitosť vytvorenia štruktúry
- Topologické algoritmy (susednosť)
- Geometrické algoritmy (prieniky)
- Vizualizačné algoritmy

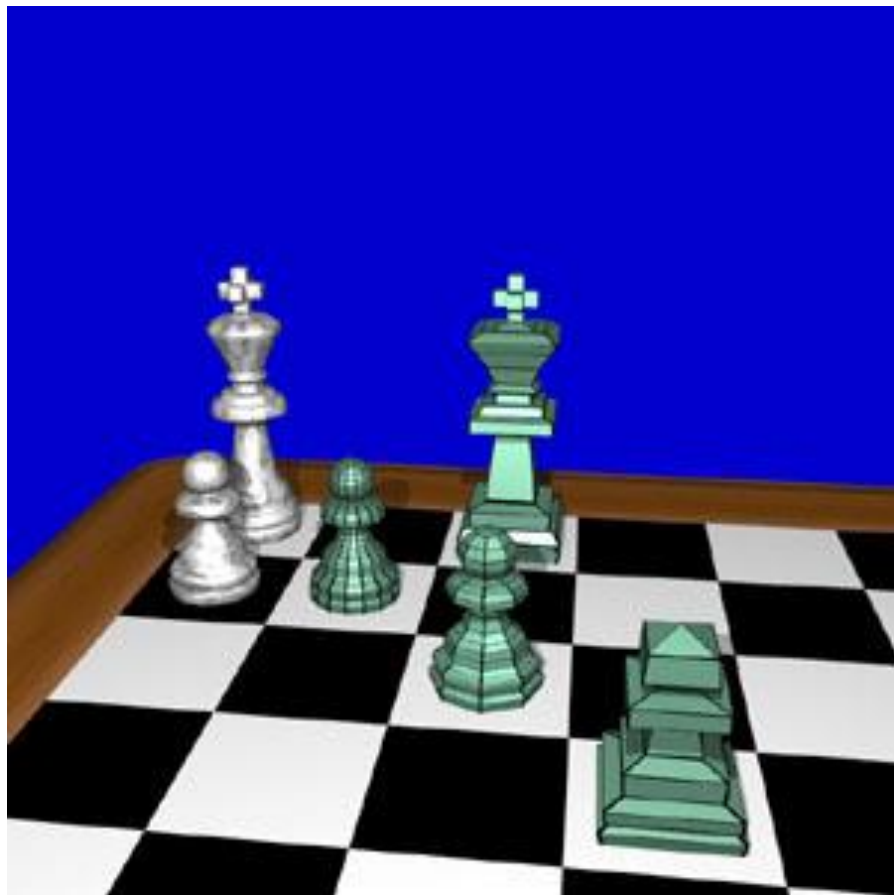


Topologické algoritmy

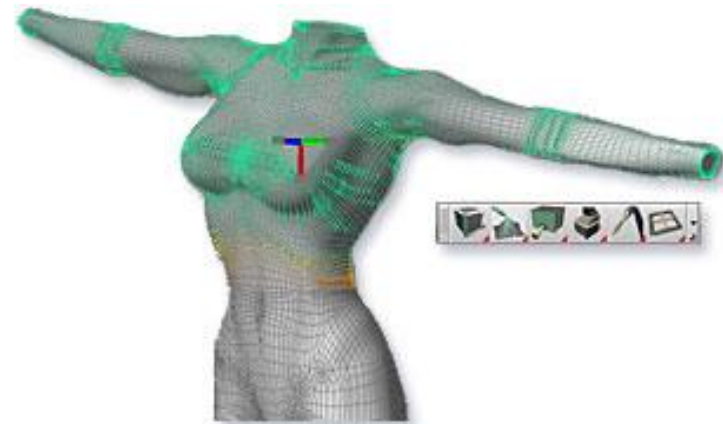
- Pre daný prvok, nájsť všetky susedné prvky
- Hľadanie susednosti na viacero krokov
- Hľadanie spojitosti dvoch prvkov na povrchu

	Vrchol	Hrana	Stena
Vrchol	VV	VE	VF
Hrana	EV	EE	EF
Stena	FV	FE	FF

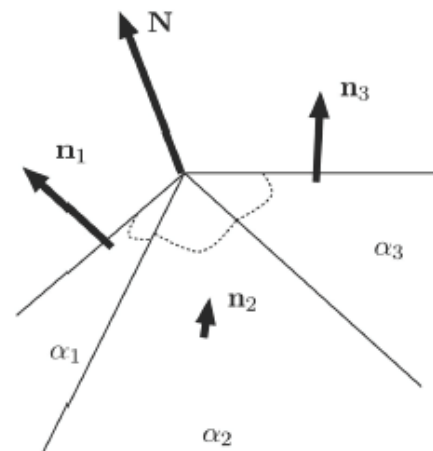
Použitie topologických algoritmov



prerozdeľovacie plochy



modelovanie povrchu



výpočty na povrchu

Matica susednosti

- Množina vrcholov $\{v_1, v_2, \dots, v_n\}$
- Množina hrán $\{e_1, e_2, \dots, e_m\}$
- Matica susednosti vrcholov $B = (b_{i,j})$ rozmerov $n \times n$
 - $b_{i,j} = 1$, ak \exists hrana (v_i, v_j)
 - $b_{i,j} = 0$, inak
- Matica susednosti hrán $D = (d_{i,j})$ rozmerov $m \times m$
 - $d_{i,j} = 1$, ak hrany e_i, e_j majú spoločný vrchol
 - $d_{i,j} = 0$, inak

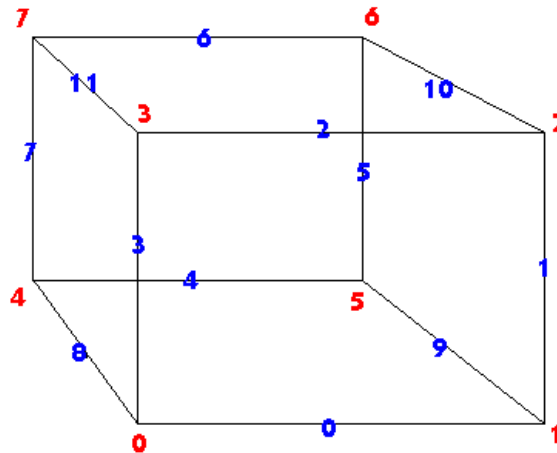
Matica incidencie

- Množina vrcholov $V = \{v_1, v_2, \dots, v_n\}$
- Množina hrán $E = \{e_1, e_2, \dots, e_m\}$
- Matica $A = (a_{i,j})$ rozmerov $n \times m$
- $a_{i,j} = 1$, ak hrana e_j začína vo vrchole v_i ,
- $a_{i,j} = -1$, ak hrana e_j končí vo vrchole v_i ,
- $a_{i,j} = 0$, inak
- $O(n \cdot m)$ pamäte, $2 \cdot m$ informácií

Matice susednosti a incidencie

	0	1	2	3	4	5	6	7
0	0	1	0	1	1	0	0	0
1	1	0	1	0	0	1	0	0
2	0	1	0	1	0	0	1	0
3	1	0	1	0	0	0	0	1
4	1	0	0	0	0	1	0	1
5	0	1	0	0	1	0	1	0
6	0	0	1	0	0	1	0	1
7	0	0	0	1	1	0	1	0

	0	1	2	3	4	5	6	7	8	9	10	11
0	1	0	0	1	0	0	0	0	1	0	0	0
1	1	1	0	0	0	0	0	0	0	1	0	0
2	0	1	1	0	0	0	0	0	0	0	1	0
3	0	0	1	1	0	0	0	0	0	0	0	1
4	0	0	0	0	1	0	0	1	1	0	0	0
5	0	0	0	0	1	1	0	0	0	1	0	0
6	0	0	0	0	0	1	1	0	0	0	1	0
7	0	0	0	0	0	0	1	1	0	0	0	1



Zoznam hrán

- Viacero spôsobov zadávania koncových vrcholov hrán
- Potrebné vhodné oindexovanie vrcholov
- Chýbajú niektoré údaje o susednostiach, pomalé $O(m)$ riešenie VV, VE, EE

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

```
struct Edge1
{
    float x1, y1, z1;
    float x2, y2, z2;
}
```

```
struct Edge2
{
    int i1;
    int i2;
}
```

```
struct Edge3
{
    Vertex* v1;
    Vertex* v2;
}
```

```
struct Mesh
{
    vector<Vertex> vertices;
    vector<Edge> edges;
}
```

Zoznam stien

- V štruktúre sú vrcholy a steny
- Možnosť pridania hrán → viacero spôsobov reprezentácie
- Postupnosť vrcholov hrán v stene → orientácia

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

```
struct Edge2
{
    Vertex* v1, v2;
}
```

```
struct Face1
{
    vector<int> vertices;
}
```

```
struct Face2
{
    vector<Edge2*> edges;
}
```

```
struct Mesh1
{
    vector<Vertex> vertices;
    vector<Face1> faces;
}
```

```
struct Mesh2
{
    vector<Vertex*> vertices;
    vector<Edge2*> edges;
    vector<Face2*> faces;
}
```

Zoznam stien

- Minimálna štruktúra na reprezentáciu vrcholov, hrán aj stien
- Chýba zložitejšia topologická informácia
- Rýchle niektoré topologické algoritmy: EV, FV, FE
- Možnosť rozšírenia o ďalšie topologické informácie

Načítanie zo súborov

- V súborových formátoch sú objekty najčastejšie uložené v zoznamoch vrcholov a v zoznamoch stien s pomocou indexov
- VRML, Collada

```
<mesh>
  <source id="box-lib-positions" name="position">
    <float_array id="box-lib-positions-array" count="24">-1 1 1 1 1 1 -1 -1 1 1 -1 1 -1 1 -1 1 -1 -1 -1 1 -1 -1 </float_array>
    <technique_common>
      <accessor count="8" source="#box-lib-positions-array" stride="3">
        <param name="X" type="float"/>
        <param name="Y" type="float"/>
        <param name="Z" type="float"/>
      </accessor>
    </technique_common>
  </source>
  <vertices id="box-lib-vertices">
    <input semantic="POSITION" source="#box-lib-positions"/>
  </vertices>
  <polylist count="6" material="BlueSG">
    <input offset="0" semantic="VERTEX" source="#box-lib-vertices"/>
    <vcount>4 4 4 4 4 4 </vcount>
    <p>0 2 3 1 0 1 5 4 6 7 3 2 0 4 6 2 3 7 5 1 5 7 6 4 </p>
  </polylist>
</mesh>
```

Konverzia

- Z oindexovaného zoznamu vrcholov a stien ku smerníkovému zoznamu vrcholov, hrán a stien
- Predpokladá sa nízka valencia vrcholov = počet vychádzajúcich hrán z vrchola

```
struct Vertex
{
    float x, y, z;
    vector<Edge2*> edges;
}
```

```
ConvertMesh1ToMesh2(Mesh1* mesh)
{
    Mesh2* result = new Mesh2;
    for (int i = 0; i < mesh->vertices.size(); i++)
    {
        Vertex* vert = new Vertex;
        vert->x = mesh->vertices[i].x; vert->y = mesh->vertices[i].y; vert->z = mesh->vertices[i].z;
        result->vertices.add(vert);
    }
    for (int i = 0; i < mesh->faces.size(); i++)
    {
        Face2* face = new Face2;
        for (int j = 0; j < mesh->faces[i]->vertices.size(); j++)
        {
            int index = mesh->faces[i]->vertices[j];
            int next_index = mesh->faces[i]->vertices[(j+1) % mesh->faces[i]->vertices.size()];
            Vertex* new_vertex = result->vertices[index];
            Vertex* new_next_vertex = result->vertices[next_index];
            bool already_connected = false;
            for (int k = 0; k < new_vertex->edges.size(); k++)
                if (new_vertex->edges[k]->v1 == new_next_vertex || new_vertex->edges[k]->v2 == new_next_vertex )
                {
                    already_connected = true;
                    face->edges.add(new_vertex->edges[k]);
                }
            if (already_connected) continue;
            Edge2* edge = new Edge2;
            edge->v1 = new_vertex; edge->v2 = new_next_vertex;
            new_vertex->edges.add(edge); new_next_vertex->edges.add(edge);
            result->edges.add(edge);
            face->edges.add(edge);
        }
        result->faces.add(face);
    }
    return result;
}
```

Topologické algoritmy

	Vrchol	Hrana	Stena
Vrchol	$O(m)$	$O(m)$	$O(l+k)$
Hrana	$O(1)$	$O(m)$	$O(l+k)$
Stena	$O(k)$	$O(k)$	$O(m+k)$

- n – počet vrcholov
- m – počet hrán
- l – počet stien
- k – maximálny počet vrcholov (hrán) v stene

FF pre zoznam

```
struct Edge2
{
    Vertex* v1, v2;
    Face2* f1;
    Face2* f2;
}
```

```
MeshFF(Face2* face, Mesh2* mesh)
{
    vector<Face2*> result;
    for (int i = 0; i < mesh->faces.size(); i++)
        mesh->edges[i]->f1 = mesh->edges[i]->f2 = NULL;
    for (int i = 0; i < mesh->faces.size(); i++)
        for (int j = 0; j < mesh->faces[i]->edges.size(); j++)
            {
                if (mesh->faces[i]->edges[j]->f1 == NULL)
                    mesh->faces[i]->edges[j]->f1 = mesh->faces[i];
                else if (mesh->faces[i]->edges[j]->f2 == NULL)
                    mesh->faces[i]->edges[j]->f2 = mesh->faces[i];
            }
    for (int i = 0; i < face->edges.size(); i++)
        {
            if (face == face->edges[i]->f1 && face->edges[i]->f2 != NULL)
                result.add(face->edges[i]->f2);
            if (face == face->edges[i]->f2 && face->edges[i]->f1 != NULL)
                result.add(face->edges[i]->f1);
        }
    return result;
}
```

Vizualizácia

- Moderné grafické karty → zoznam vrcholov a indexov trojuholníkov (polygónov)
- Možnosť prenášať veľa dát naraz
- Využitie vyrovnávacej pamäte

```
struct Vertex
{
    float x, y, z;
    // uv coordinates, normals
}
```

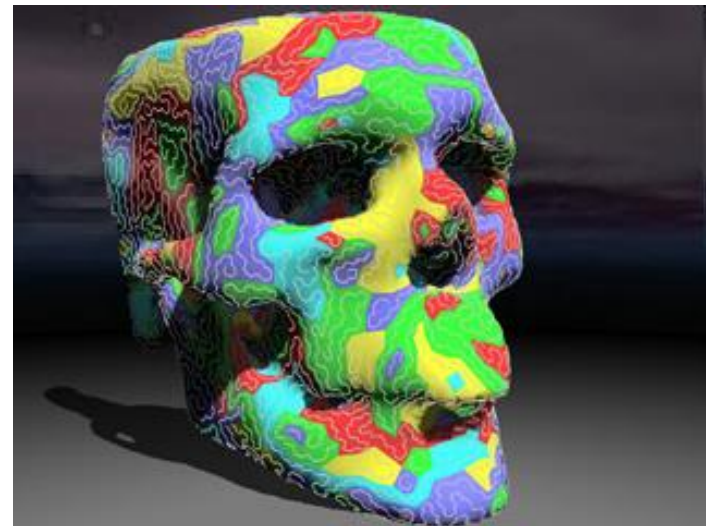
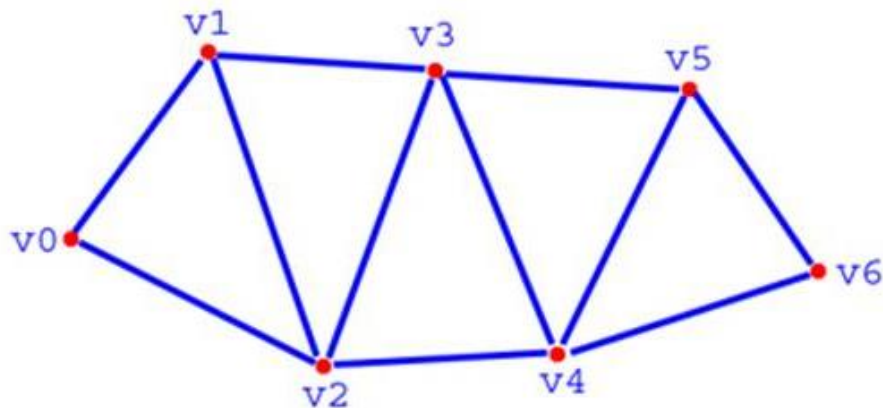
```
struct Triangle
{
    int i, j, k;
}
```

```
struct Mesh
{
    int num_vertices;
    Vertex* vertices;
    int num_triangles;
    Triangle* triangles;
}
```



Vytvorenie stripov

- Zadanie trojuholníkov, štvoruholníkov menším počtom vrcholov
- Prvý trojuholník je daný tromi vrcholmi
- Každý ďalší trojuholník je daný jedným vrcholom (+ dva predchádzajúce vrcholy)



Vytvorenie stripov

- NP-úplný problém
- SGI algoritmus
 - 1. vyber prvý trojuholník stripu, ktorý ešte nebol spracovaný, ak taký neexistuje, tak skonči
 - Prvý trojuholník s malým počtom spracovaných susedov
 - 2. vyber smer v ktorom má z aktuálneho trojuholníka strip pokračovať
 - 3. rozšír strip o trojuholník v danom smere, ak taký smer ešte existuje, choď na 2. Ak sa smer nenájde, môže sa celý aktuálny strip otočiť a pokúsiť sa o pokračovanie (nájdanie smeru) v prvom trojuholníku
 - 4. choď na 1.
- Rozšírenie v podobe vytvárania viac stripov a výberu toho najlepšieho
- Spájanie stripov pomocou degenerovaných trojuholníkov (swaps)

Zoznam okolí vrcholov

- Pre každý vrchol zoznam vrcholov, ktoré s ním susedia
- Chýba informácia o stenách (oblastiach), ani hrany nie sú implicitne dané
- Vhodná pamäťová náročnosť
- Vyhľadávanie iba susedov
- Použiteľné pre alg. na grafoch (toky, ...)

```
struct Vertex3
{
    float x, y, z;
    vector<int> vertices;
}
```

```
struct Vertex3
{
    float x, y, z;
    vector<Vertex*> vertices;
}
```

```
struct Mesh3
{
    vector<Vertex3*> vertices;
}
```

Vytvorenie zoznamu

```
struct Vertex2
{
    float x, y, z;
    Vertex3* new_vertex;
}
```

```
CreateMesh3FromMesh2(Mesh2* mesh)
{
    Mesh3* result = new Mesh3;
    for (int i = 0; i < mesh->vertices.size(); i++)
    {
        Vertex3* vert = new Vertex3;
        vert->x = mesh->vertices[i].x; vert->y = mesh->vertices[i].y; vert->z = mesh->vertices[i].z;
        mesh->vertices[i]->new_vertex = vert;
        result->vertices.add(vert);
    }
    for (int l = 0; l < mesh->edges.size(); l++)
    {
        Vertex3* v1 = mesh->edges[l]->v1->new_vertex;
        Vertex3* v2 = mesh->edges[l]->v2->new_vertex;
        v1->vertices.add(v2);
        v2->vertices.add(v1);
    }
    return result;
}
```



Otázky?