

# Diskrétne geometrické štruktúry

7.

Martin Florek

[florek@sccg.sk](mailto:florek@sccg.sk)

[www.sccg.sk/~florek](http://www.sccg.sk/~florek)

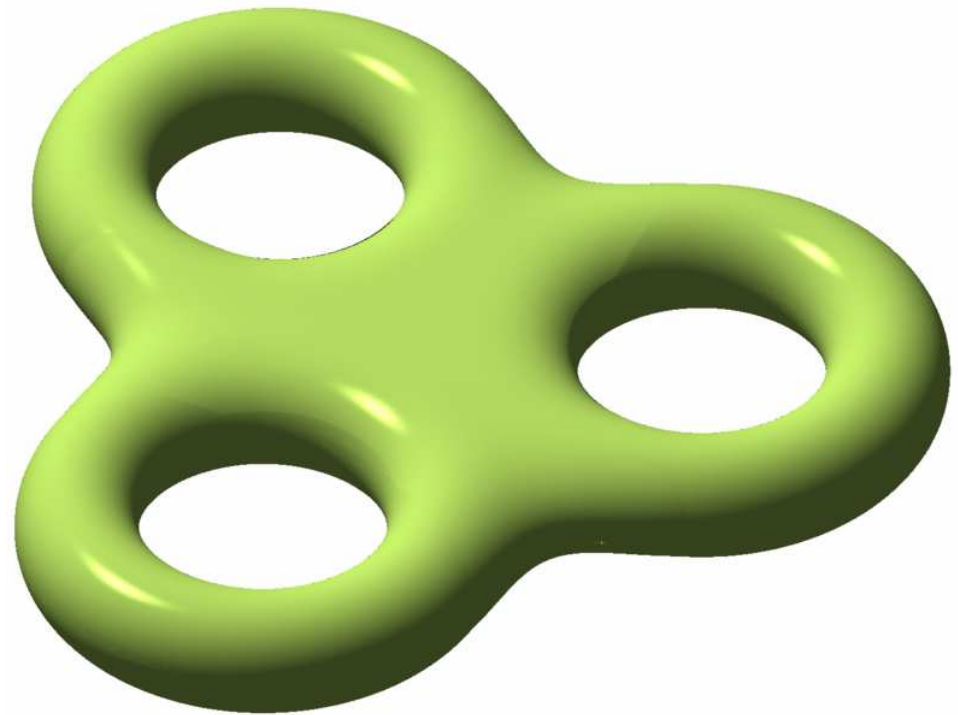
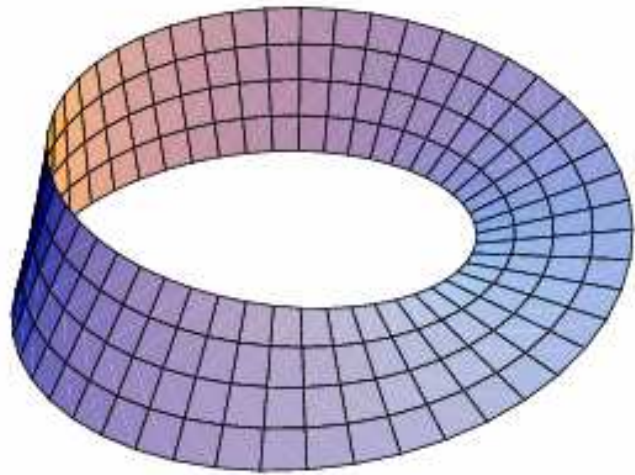
# Manifold

- n-manifold – lokálne homeomorfný s Euklidovským priestorom
- pre každý bod manifoldu 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





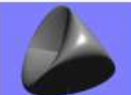
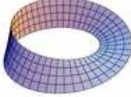
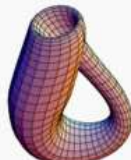
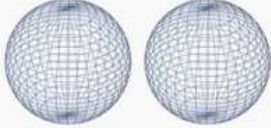
# Manifoldy 2








# Vlastnosti

- orientovateľnosť
- rod (genus) – počet „dier“
- Eulerova charakteristika  $V-E+F$ 
  - polytopy
    - $V-E+F = 2$
  - orientovateľné a uzavreté 2-manifolds
    - $V-E+F = 2-2g$
    - $g$  = genus
- mapy, atlasy – zobrazenia častí manifoldu na  $n$ -rozmernú sféru

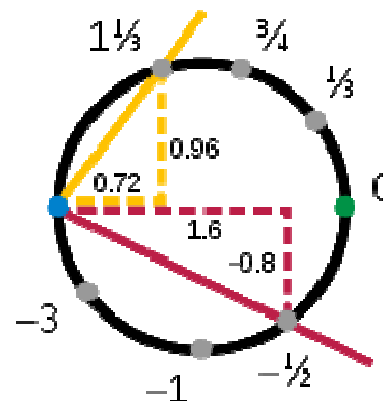
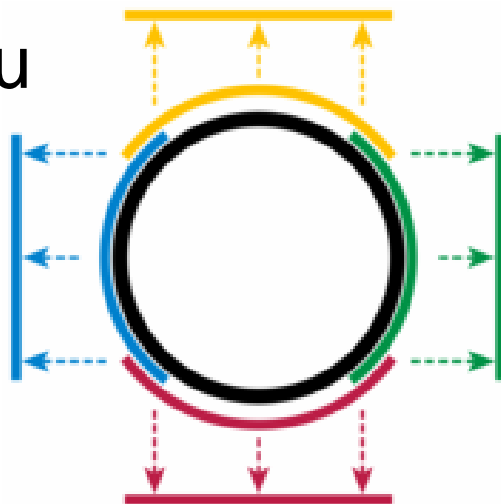
# Eulerova charakteristika

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

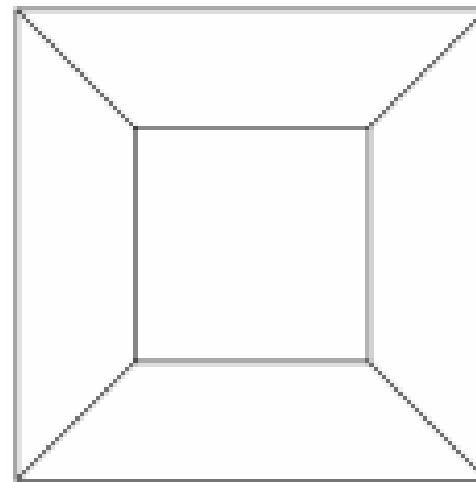
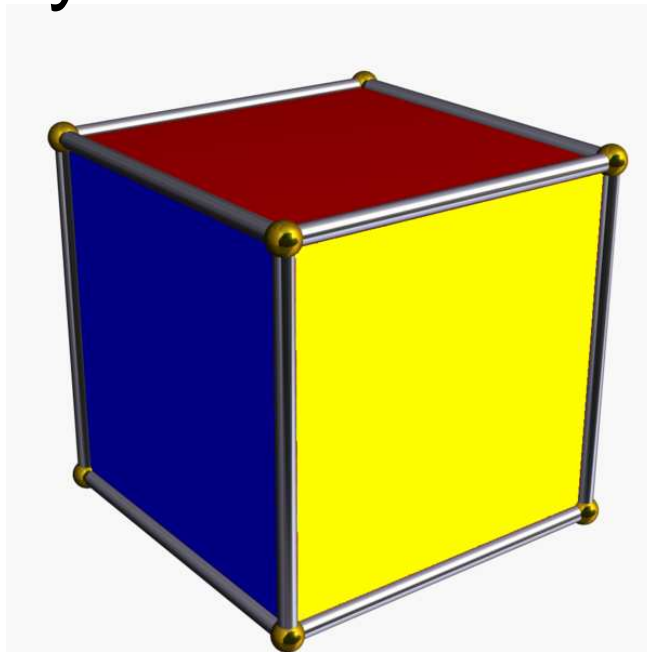
# Mapy a Atlas

- kružnica
  - 4 mapy pre celú kružnicu
  - 1 mapa pre kružnicu bez jedného bodu
  - neexistuje atlas pre kružnicu s jednou mapou
- mapy tvoria atlas manifoldu
- väčšina manifoldov potrebuje viac ako jednu mapu



# Rovinný graf

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



# Reprezentácia

- počítačová reprezentácia 2-manifoldov a 3-manifoldov
- reprezentácia množín polygónov (mnohosteny, meše)
- reprezentácia rovinných grafov

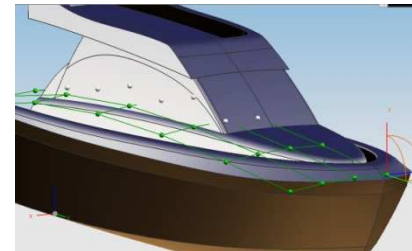
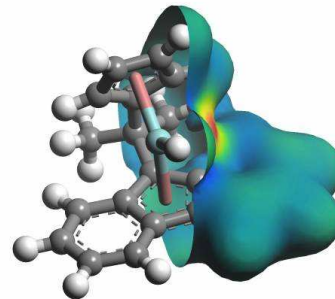
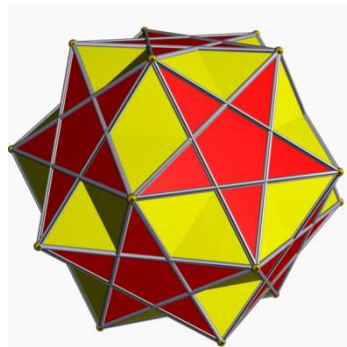
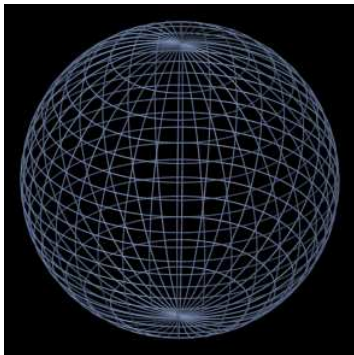


# Výber reprezentácie

- pamäťová náročnosť
- konverzia medzi reprezentáciami
- generovanie na základe vstupných dát
- reprezentácia vnútra
- geometrické algoritmy (prieniky, transformácie, ...)
- vizualizácia

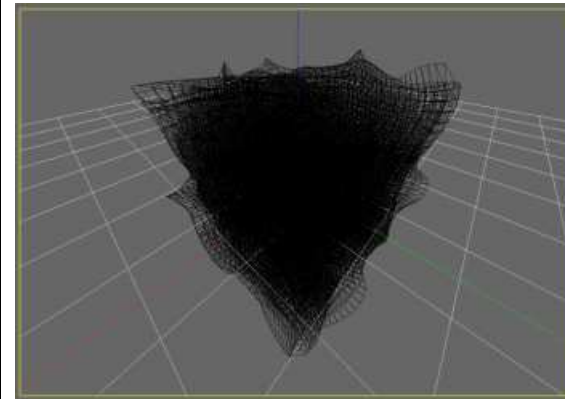
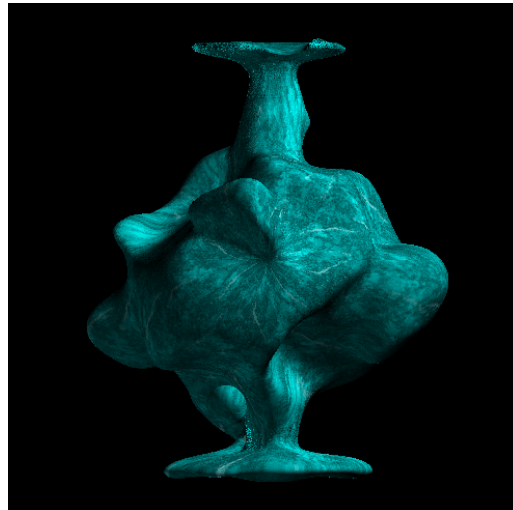
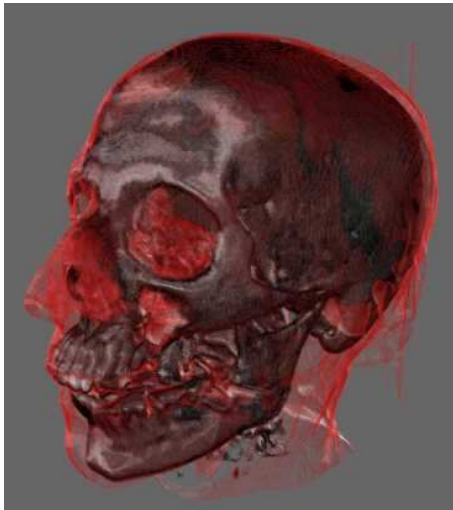
# Reprezentácie 2-manifoldov

- povrchová reprezentácia (boundary representation, b-rep)
  - drôtený model (wireframe)
  - množina polygónov
- implicitné povrchy (blobby)
- parametrické povrchy (NURBS)



# Reprezentácie 3-manifoldov

- diskretná volumetrická reprezentácia (binárna, vzdialenostné polia)
- funkcionálna reprezentácia, f-rep
- parametrické telesá



# Reprezentácie množiny polygónov

- pamäťová náročnosť
- definovanie častí (vrcholy, hrany, steny)
- indexy, smerníky
- zložitosť vytvorenia štruktúry
- topologické algoritmy (susednosť)
- geometrické algoritmy (prieniky)
- vizualizačné algoritmy

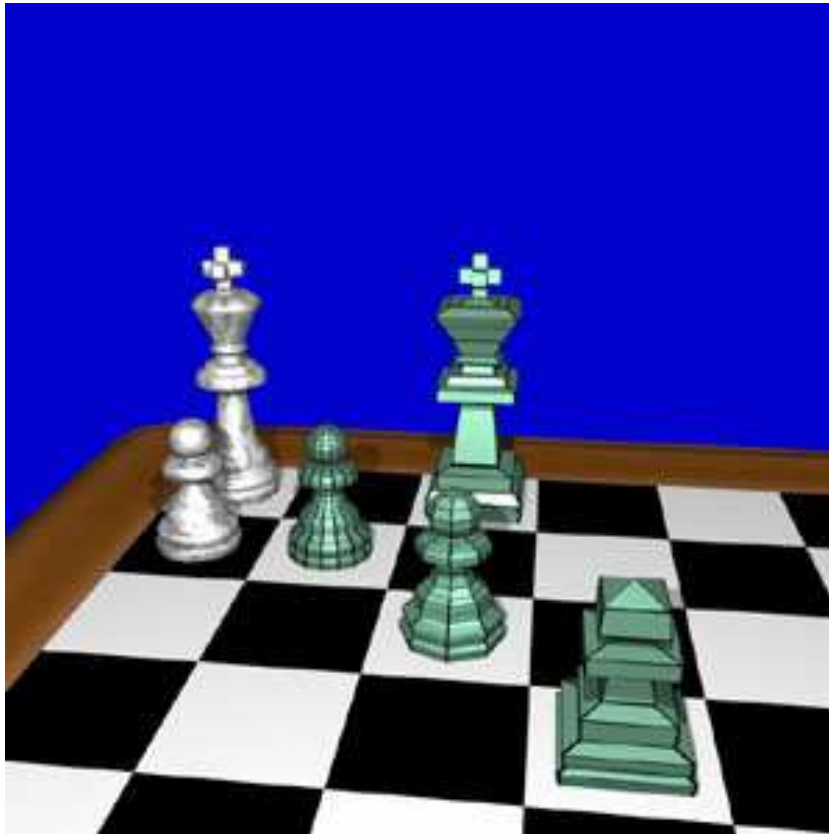
# Topologické algoritmy

- pre daný prvok, nájdí všetky susedné prvky

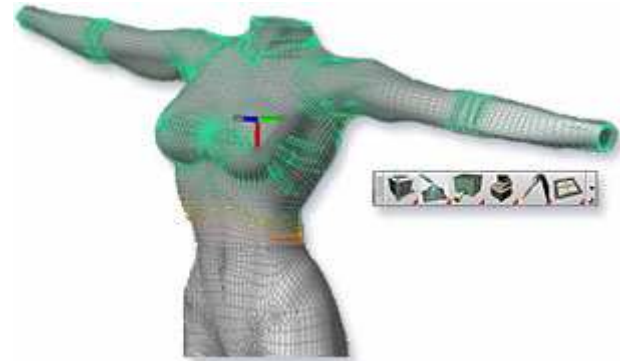
	Vrchol	Hrana	Stena
Vrchol	<b>VV</b>	<b>VE</b>	<b>VF</b>
Hrana	<b>EV</b>	<b>EE</b>	<b>EF</b>
Stena	<b>FV</b>	<b>FE</b>	<b>FF</b>

- hľadanie susednosti na viacero krokov
- hľadanie spojitosti dvoch prvkov na povrchu

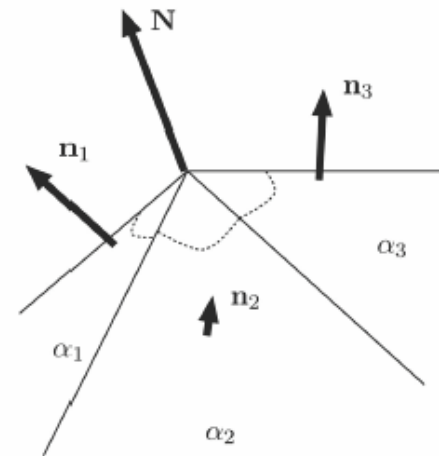
# Použitie topologických algoritmov



prerozdeľovacie plochy



modelovanie povrchu



výpočty na povrchu

# 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
- $O(n.m)$  pamäte,  $2.m$  informácií

# Matica susednosti

- 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



# Zoznam okolí vrcholov

- pre každý vrchol zoznam vrcholov, ktoré s ním susedia
- chýba informácia o stenách (oblastiach)
- vyhľadávanie iba susedov

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

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

```
struct Vertices
{
    vector<Vertex*> vertices;
}
```

# Zoznam hrán

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

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

```
struct Vertices
{
    vector<Vertex*> vertices;
}
```

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

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

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

```
struct Edges
{
    vector<Edge*> Edges;
}
```

# Zoznam stien

- v štruktúre vrcholy a steny
- možnosť pridania hrán → viacero spôsobov reprezentácie
- postupnosť vrcholov hrán v stene → orientácia

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

```
struct Face
{
    vector<Vertex*> vertices;
}
```

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

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

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

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

# Zoznam stien 2

- 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

# 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;
}
```



# FF pre zoznam

```
MeshFF(Face* face, Mesh* mesh)
{
    for (int i = 0; i < mesh->faces.size(); i++)
    {
        Face* aux_face = mesh->faces[i];

        if (aux_face == face)
            continue;

        for (int j = 0; j < aux_face->vertices.size(); j++)
        {
            Vertex* aux1 = aux_face->vertices[j];
            Vertex* aux2 = aux_face->vertices[(j + 1) % aux_face->vertices.size()];
            for (int k = 0; k < face->vertices.size(); k++)
            {
                Vertex* v1 = face->vertices[k];
                Vertex* v2 = face->vertices[(k + 1) % face->vertices.size()];
                if ((v1 == aux1 && v2 == aux2) || (v1 == aux2 && v2 == aux1))
                    result.add(aux_face);
            }
        }
    }
    return result;
}
```

**koniec (-:**

`florek@sccg.sk`