

# Representations of objects

Quadtree, octree and BSP

Meshes, half-edge and winged edge data structures

**Marcel Makovník**

Department of Algebra and Geometry  
Faculty of Mathematics, Physics and Informatics  
Comenius University in Bratislava, Slovakia

# Motivation, needs

- CAGD, displaying data, game design, entertainment
- different needs - different representations

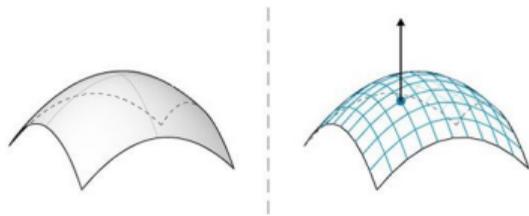


Figure: "nice" image



Figure: precise model

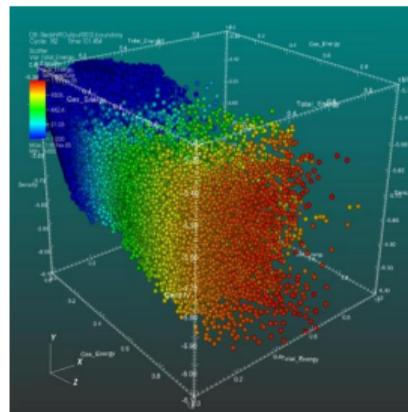
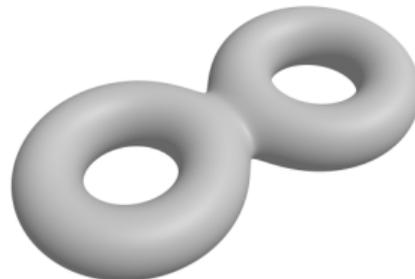
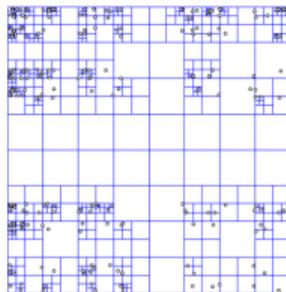
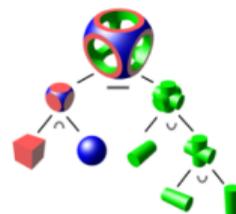
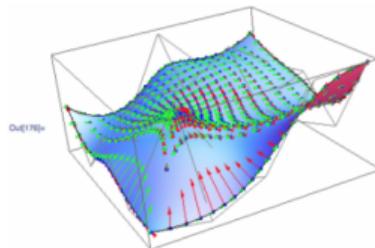
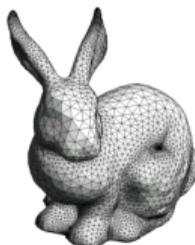


Figure: data visualization

# Types of object representations

- meshes
- splines
- CSG
- space partitioning
- implicit surfaces
- etc.

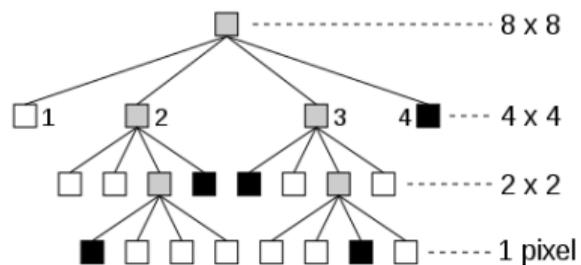
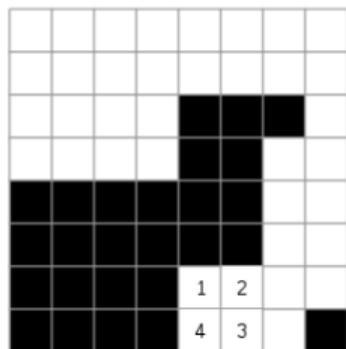


# Space partitioning

- dividing (usually Euclidean) space into two or more disjoint subsets
- usage in ray tracing (wait for course CG(2))
  
- quadtree
- octtree
- BSP tree
- $k$ -d tree

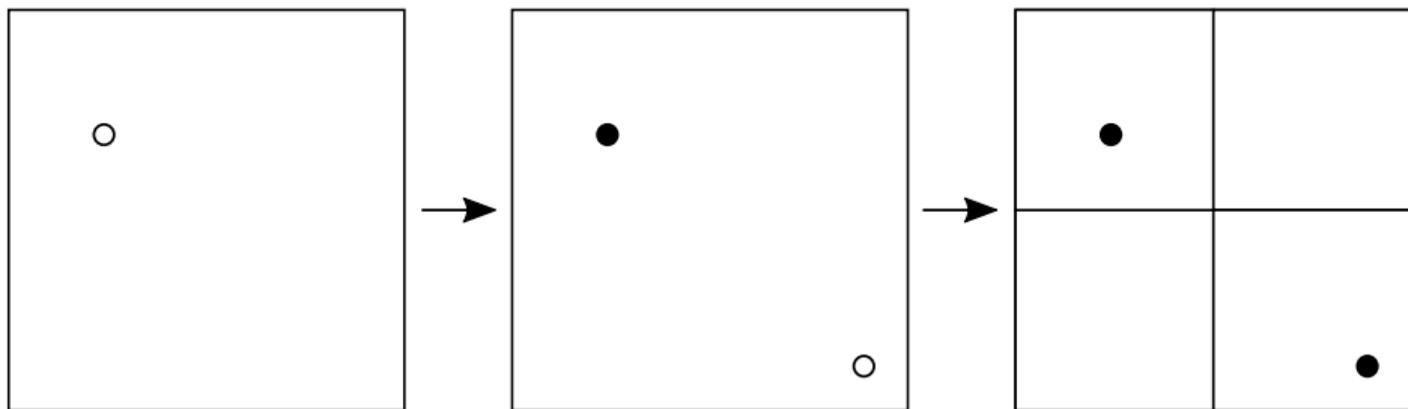
# Quadtree – definition

- tree (data structure) with each branch having exactly 4 children



## Quadtree – features

- decomposition of a space into adaptive cells (usually squares or rectangles)
- each cell has maximum capacity
- if the capacity has been exceeded, the cell is subdivided



# Quadtree – types

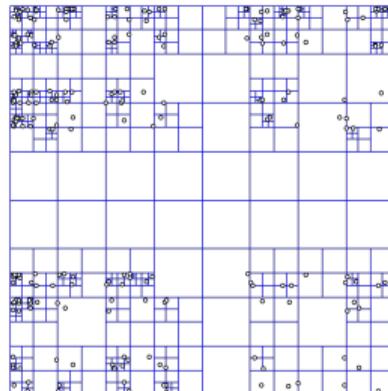
## ■ region quadtree

- max. capacity – one (or more) colour in a cell
- e.g. *image representation*



## ■ point quadtree

- max. capacity – one (or more) point in a cell
- e.g. *mesh generation*



# Quadtree – representations

- as a **tree** data structure
- as a **text string** (sometimes called *linear quadtree*)

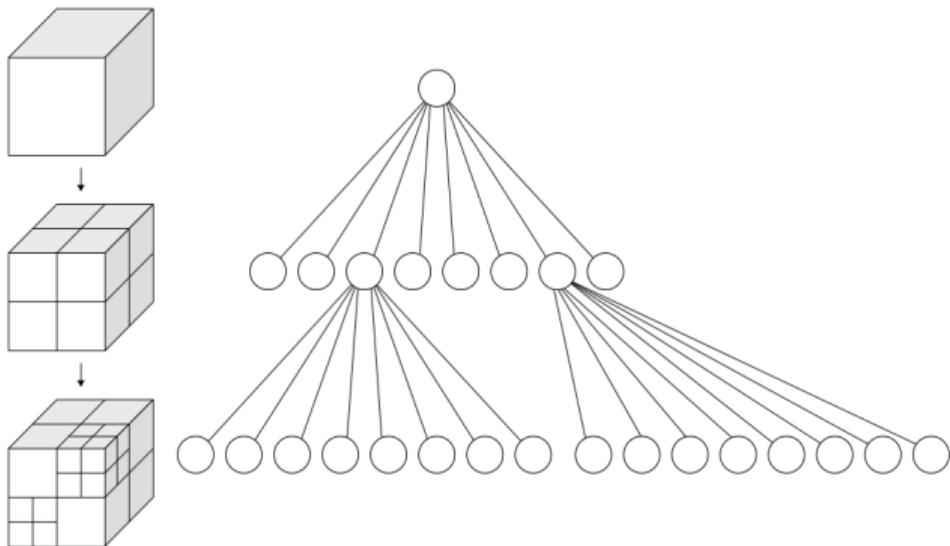
## Exercise

*Record the following monochromatic image using quadtree in both tree and line notation:*



# Octree

- each branch has exactly 8 children
- one subdivision step creates 8 octants
- 3D analogy of quadtree
- e.g. *collision detection in game design*



## Binary space partitioning – BSP

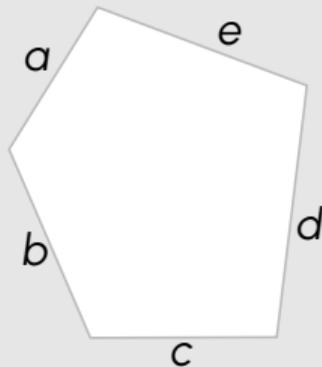
- binary tree
- one subdivision step divides a space into convex sets by **hyperplanes**
- generalization of both quadtree and octree
- store and retrieve spatial information about primitives
- static geometry
- e.g. *painter's algorithm*, *CSG - operations with shapes*

## BSP – generation

- scene is divided into two until partitioning satisfies one or more criteria
- for points (???), for polygons (???)

### Exercise

*Generate BSP tree for a pentagon.  
What is the optimal hyperplane for  
the initial subdivision?*



# Vertex, edge and face

## Definition

Let  $\mathcal{V} = \{V_1, \dots, V_n\}$ ,  $V_i \in \mathbb{R}^3$ ,  $i = 1, \dots, n$ . Then  $V_i$  is called a **vertex** and  $\mathcal{V}$  is called a **set of vertices**.

## Definition

Denote  $\mathcal{E} \subseteq \mathcal{V}^{[2]}$ , where  $\mathcal{V}^{[2]} = \{[P_i, P_j] \in \mathcal{V}^2 \mid P_i \neq P_j\}$  and each  $e = [V_i, V_j] \in \mathcal{E}$  satisfies  $[V_j, V_i] = e$ . Then  $e$  is called an **edge** and  $\mathcal{E}$  is called a **set of edges**.

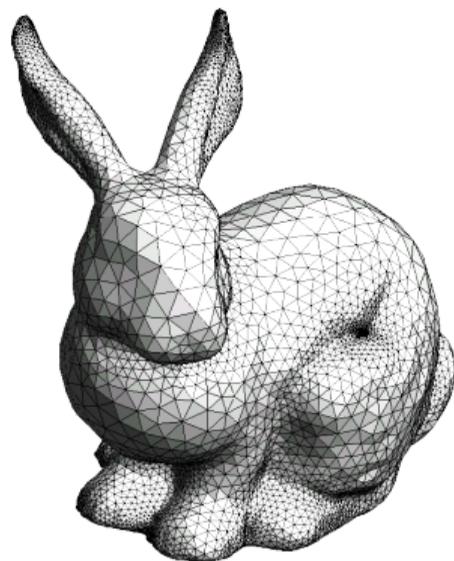
## Definition

Let  $\mathcal{F} \subseteq \mathcal{V}^{[n]}$ ,  $n \geq 3$ , where  $\mathcal{V}^{[n]} = \{[V_{i_1}, V_{i_2}, \dots, V_{i_n}] \in \mathcal{V}^n \mid V_{i_j} \neq V_{i_k}, j \neq k\}$ . The set  $\mathcal{F}$  is called a **set of faces**. Its element  $\mathbf{F} = [V_{i_1}, V_{i_2}, \dots, V_{i_n}] \in \mathcal{F}$  is called a **face** and conforms the condition, that all permutations of  $V_{i_1}, V_{i_2}, \dots, V_{i_n}$  represent the same face  $\mathbf{F}$ .

## Definition

Denote  $\mathcal{M} = \{\mathcal{V}, \mathcal{E}, \mathcal{F}\}$ .  $\mathcal{M}$  is called a **polygon mesh**. The corresponding sets of vertices, edges and faces are denoted by  $\mathcal{V}(\mathcal{M})$ ,  $\mathcal{E}(\mathcal{M})$  and  $\mathcal{F}(\mathcal{M})$ , respectively.

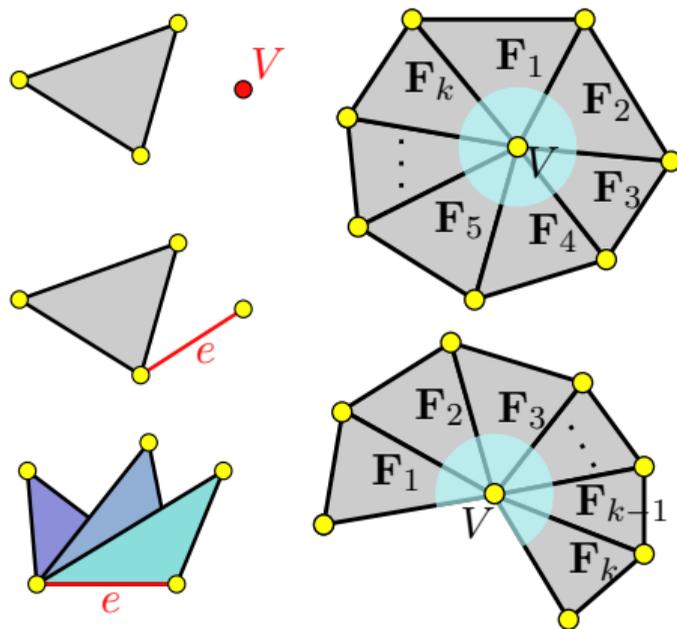
A mesh is meant to be a **2-manifold**, meaning each vertex on a mesh  $\mathcal{M}$  has a neighbourhood that is homeomorphic to the Euclidean space  $\mathbb{E}^2$ .



# Mesh

To avoid non-manifold meshes, additional conditions are required:

- 1 each vertex  $V \in \mathcal{V}$  lies on some of the edges from  $\mathcal{E}$
- 2 each edge  $e \in \mathcal{E}$  lies on some of the faces from  $\mathcal{F}$
- 3 each edge  $e \in \mathcal{E}$  belongs to at most two faces from  $\mathcal{F}$
- 4 for each face  $[P_i, P_j, P_k] \in \mathcal{F}$  we have  $[P_i, P_j], [P_i, P_k], [P_j, P_k] \in \mathcal{E}$ ,
- 5 for each vertex  $V \in \mathcal{V}$  and all faces  $\mathbf{F}_1, \dots, \mathbf{F}_k \in \mathcal{F}$  sharing  $V$ , one of the following statements applies:
  - $\mathbf{F}_1, \dots, \mathbf{F}_k$  create a simple cycle around  $V$
  - $\mathbf{F}_1, \dots, \mathbf{F}_k$  create a simple sequence around  $V$



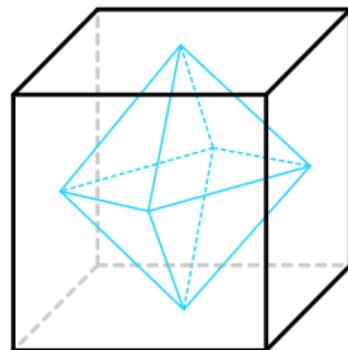
# Mesh

## Definition

**Boundary of a mesh** is the union of all edges belonging to exactly one face. The mesh is said to be **closed**, if it has no boundary.

## Definition

The mesh is **connected** if the dual graph is connected.



# Polygon mesh – properties

## ■ Euler's formula

$$\#V - \#E + \#F = 2(1 - g),$$

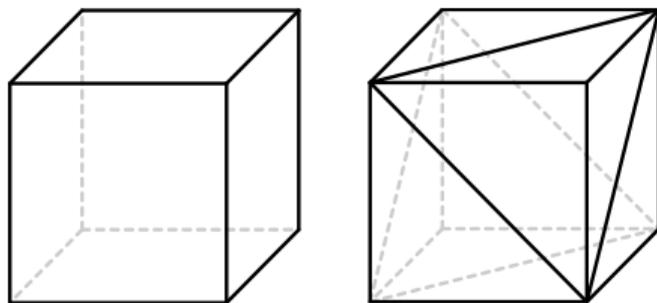
$g$  – genus (number of handles),  $\#V, \#E, \#F$  – number of vertices, edges and faces, respectively

For triangle meshes:

## ■ number of vertices, faces and edges

$$\#F \approx 2\#V, \quad \#E \approx 3\#V$$

## ■ average **vertex valence** (number of incident edges) is 6



# How to represent a mesh?

- the efficiency and memory consumption depends on the data structure
- we need to take into account
  - **topological requirements** – *2-manifold meshes/complex edges or singular vertices, triangle/polygon meshes etc.*
  - **algorithmic requirements** – *just render the mesh, local neighbourhoods of vertices/edges/faces, static/changing geometry etc.*

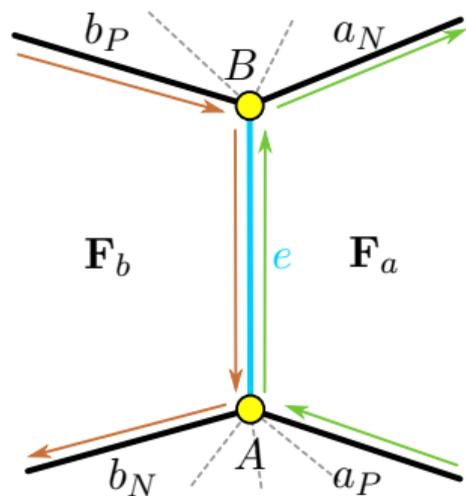
# Face-based data structures

Vertices
$x_1, y_1, z_1$
...
$x_v, y_v, z_v$

Triangles
$i_{11}, i_{12}, i_{13}$
...
...
...
$i_{F1}, i_{F2}, i_{F3}$

- vertices  $\longleftrightarrow$  coordinates
- triangles  $\longleftrightarrow$  indices
- static data
- formats (.obj, .off), VRML and OpenGL
- no information about additional connectivity
- traversal is expensive

# Edge-based data structures (winged-edge)



## Vertex

coinciding edge  
coordinates

## Faces

coinciding edges

## Edges

coinciding vertices

coinciding faces

prev. and next edge  
of the left traverse

prev. and next edge  
of the right traverse

- connectivity via edges
- clockwise ordering
- fast traversal through edges, vertices and faces
- case distinctions

# Halfedge-based data structure

Vertex
halfedge pointing to the vertex
coordinates

Faces
coinciding halfedge

Edges
vertex it points to
adjacent face
next halfedge
previous halfedge
opposite halfedge

- edge is split into two halfedges
- consistent counter-clockwise ordering
- only for 2-manifolds
  
- C++ library – **OpenMesh**

