Advanced 3D-Data Structures

Eduard Gröller, Martin Haidacher

Institute of Computer Graphics and Algorithms

Vienna University of Technology

Motivation

- For different data sources and applications different representations are necessary
- Examples:
  - 3D scanner: produces a set of spatial points which are not connected to each other
  - Computer game: Scenes and characters are usually represented as surface model consisting of many polygons
- A data structure for a certain application should be able to fulfill the necessary requirements

3D-Data Structures: Requirements

- Representation of general objects
- Exact representation of objects
- Combinations of objects
- Linear transformation
- Interaction
- Fast spatial searches
- Memory capacity
- Fast rendering

3D-Data Structures: Overview

- Point Cloud
- Wire-frame Model
- Boundary Representation
- Binary Space Partitioning Tree
- kD Tree
- Octree
- Constructive Solid Geometry Tree
- Bintree
- Grid

Point Cloud

- Object = set (list) of points
  - E.g. from a digitizer or 3D scanner
- For fast and simple preview
- Exact representation if >=1 points/pixel
  - More efficient than 1 pixel sized polygons

Operations with Point Clouds

- **Transformations**
  - Multiply the points in the point list with linear transformation matrices
- **Combinations**
  - Objects can be combined by appending the point lists to each other
- **Rendering**
  - Project and draw the points onto the image plane

Properties of Point Clouds

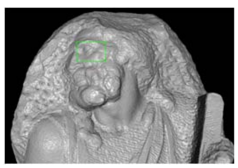
- **Advantages**
  - Fast rendering
  - Exact representation & rendering possible
  - Fast transformations
- **Disadvantages**
  - Many points (curved obj., exact representation)
  - High memory consumption
  - Limited combination operations

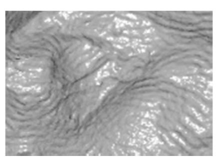
Surfels (SURFace ELementS)

- http://www.merl.com/projects/surfels/
- movies: cab, wasp, salamander with holes, salamander corrected (more movies on web page)

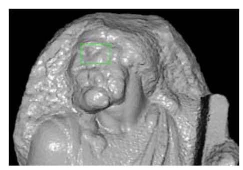
QSplat (1/2)

- 3D scan of 2.7 meter statue of St. Matthew at 0.25 mm
- 102.868.637 points
- File size: 644 MB
- Preprocessing time: 1 hour
- Demo on laptop (PII 366, 128 MB), no 3D graphics hardware
- http://graphics.stanford.edu/software/qsplat/

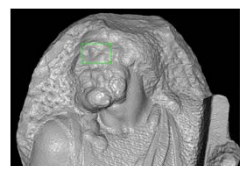




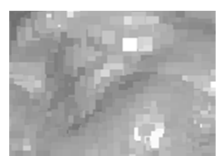
QSplat (2/2)

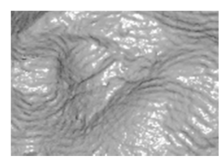


Interactive (8 frames/sec)



High quality (8 sec)



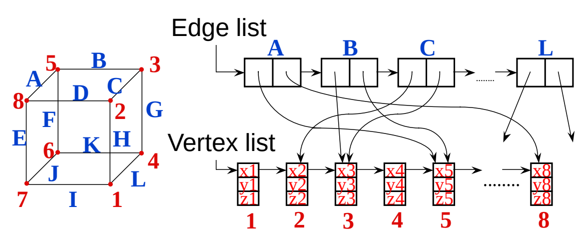


3D-Data Structures: Overview

- Point Cloud
- Wire-frame Model
- Boundary Representation
- Binary Space Partitioning Tree
- kD Tree
- Octree
- Constructive Solid Geometry Tree
- Bintree
- Grid

Wire-Frame Model

- Object is simplified to 3D lines, each edge of the object is represented by a line in the model



Operations with Wire-Frame Model

- **Transformations**
  - Multiply the points in the point list with linear transformation matrices
- **Combinations**
  - Objects can be combined by appending the point and edge lists to each other
- **Rendering**
  - Projection of all points onto image plane and drawing of edges in between

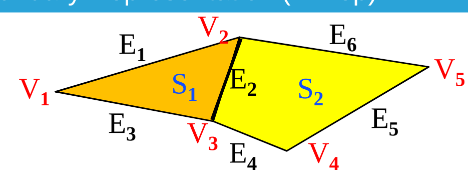
Properties of Wire-Frame Models

- **Advantages**
  - Quick rendering
  - Easy and quick transformations
  - Generation of models via digitization
- **Disadvantages**
  - High memory consumption
  - Inexact (no surfaces, no occlusion)
  - Restricted combination possibilities
  - Curves are approximated by straight lines

3D-Data Structures: Overview

- Point Cloud
- Wire-frame Model
- Boundary Representation
- Binary Space Partitioning Tree
- kD Tree
- Octree
- Constructive Solid Geometry Tree
- Bintree
- Grid

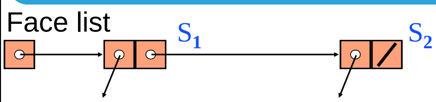
Boundary Representation (B-Rep)



vertex list	edge list	face list
V_1: x_1 y_1 z_1	E_1: V_1 V_2	S_1: E_1 E_2 E_3
V_2: x_2 y_2 z_2	E_2: V_2 V_3	S_2: E_2 E_4 E_5 E_6
V_3: x_3 y_3 z_3	E_3: V_3 V_1	
V_4: x_4 y_4 z_4	E_4: V_3 V_4	
V_5: x_5 y_5 z_5	E_5: V_4 V_5	
	E_6: V_5 V_2	

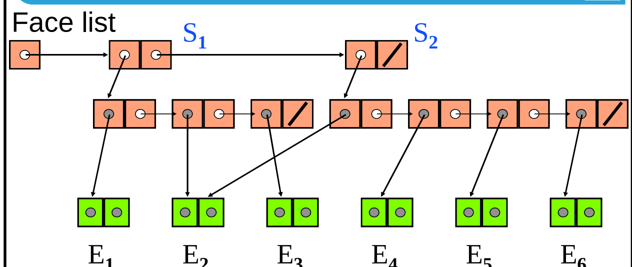
Lists for B-Reps (1/4)

Face list

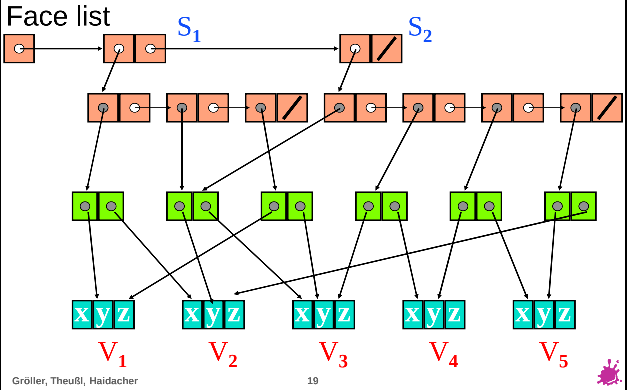


Lists for B-Reps (2/4)

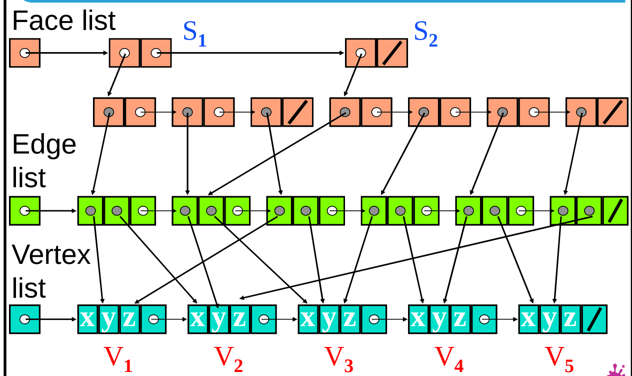
Face list



Lists for B-Reps (3/4)

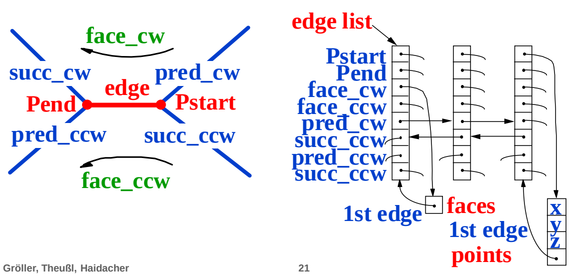


Lists for B-Reps (4/4)



Winged Edge Data Structure

- Alternative for normal hierarchical B-Rep
- Here the central element is the edge:



Operations with B-Reps (1/2)

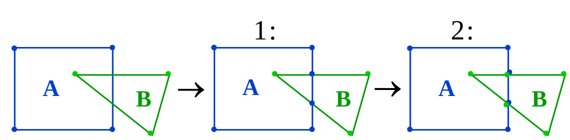
- **Transformations**
  - All points are transformed as with wire-frame model, additionally surface equations or normal vectors can be transformed
- **Rendering**
  - Hidden surface or hidden line algorithms can be used because the surfaces of the objects are known, so that the visibility can be calculated

Operations with B-Reps (2/2)

- **Combinations**
  - 1. Split the polygons of object A at the intersections with the polygons of object B
  - 2. Split the polygons of object B at ... of A
  - 3. Classify all polygons of A as "in B", "outside B" or "on the surface of B“
  - 4. Classify all polygons of B in the same way
  - 5. Remove the redundant polygons of A and B according to the operator and combine the remaining polygons of A and B

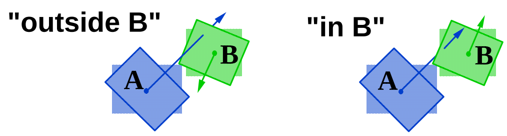
Combinations of B-Reps (1/4)

- Every polygon has a **box enclosure**
  $\Rightarrow$ simple test if polygons can intersect
- Use only **convex** polygons and produce only convex polygons as results
  $\Rightarrow$ simple intersection tests



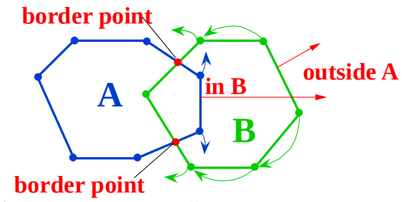
Combinations of B-Reps (2/4)

- A ray is traced in the direction of the normal vector of the polygon to be classified:
  - Ray hits no polygon of B ⇒ **"outside B"**
  - First polygon of B hit from front ⇒ **"outside B"**
  - First polygon of B hit from back ⇒ **"in B"**



Combinations of B-Reps (3/4)

- Improvement: points of A, which lie on the surface of B, are marked as border points during the dividing process (and vice versa) ⇒ only very few polygons have to be classified with the complex method



Combinations of B-Reps (4/4)

- Polygons can be removed according to tables:

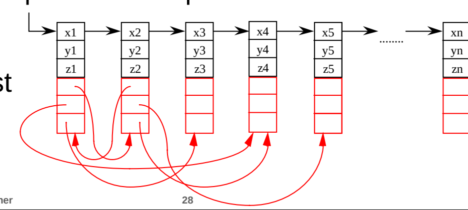
For polygons of A	op.	in B	outside B	on B (coplanar) NV equal	on B (coplanar) different
	A or B	yes	no	no	yes
	A and B	no	yes	no	yes
	A sub B	yes	no	yes	no

For polygons of B	op.	in A	outside A	on A (coplanar) NV equal	on A (coplanar) different
	A or B	yes	no	yes	yes
	A and B	no	yes	yes	yes
	A sub B	no	yes	yes	yes

Requirements on B-Reps for this Alg.

- No open (non-closed) objects
- Only convex polygons
- No double points
- Additional links in the vertex list between neighbor points with equal classification

vertex list



Partitioning of Object Surfaces

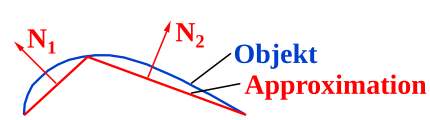
- Necessary to approximate curved surfaces
- Surfaces that **can** be parameterized:
  - E.g. free form surfaces, quadrics, superquadrics
  - partitioning of parameter space, one patch for every 2D parameter interval
- Surfaces that **cannot** be parameterized:
  - E.g. implicit surfaces, "bent" polygons ⇒ tesselation, subdivision surfaces

Tesselation

- Divide polygons in smaller polygons (triangles) until the approximation is exact enough
- Normal vector criterion as termination condition:

$$\mathbf{N_1 \cdot N_2 \geq 1-\varepsilon}$$

- Normal vectors of neighboring polygons are similar:



Properties of B-Reps

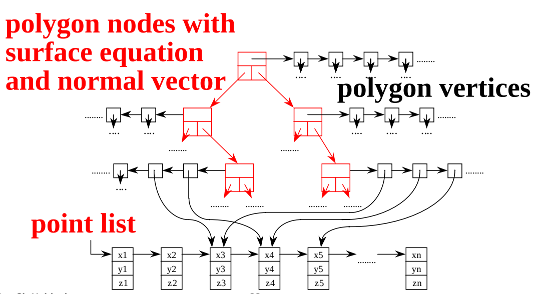
- **Advantages**
  - General representation
  - Generation of models via digitization
  - Transformations are easy and fast
- **Disadvantages**
  - High memory requirement
  - Combinations are relatively costly
  - Curved objects must be approximated

3D-Data Structures: Overview

- Point Cloud
- Wire-frame Model
- Boundary Representation
- Binary Space Partitioning Tree
- kD Tree
- Octree
- Constructive Solid Geometry Tree
- Bintree
- Grid

Binary Space Partitioning Tree

- Special B-Rep for quick rendering with visibility
  - Especially of static scenes



Binary Space Partitioning Tree

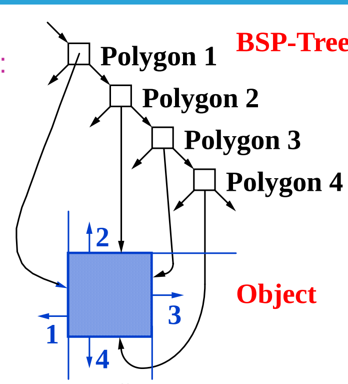
- The base plane of the polygon in a node partitions space in two halves:
  - In front of and behind the polygon
- **Left subtree** of the node: contains only polygons that are **in front of** the basis plane
- **Right subtree** of the node: contains only polygons that are **behind** the basis plane
- Polygons that lie in both halves are divided by the base plane into two parts

Generation of BSP Trees

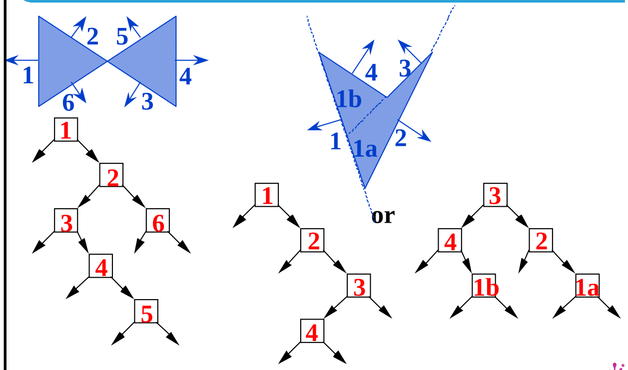
- Convex objects: BSP tree is linear list
- Else: conversion B-Rep $\Rightarrow$ BSP tree
- Algorithm:
  - 1. Find the polygon who's plane **intersects the fewest** other polygons and cut these in two
  - 2. Divide the polygon list in **two sets**: in front of that plane / behind that plane
  - 3. The polygon found in 1. is the **root** of the BSP tree, the left and the right subtrees can be generated **recursively** (from two "halves")

BSP Example

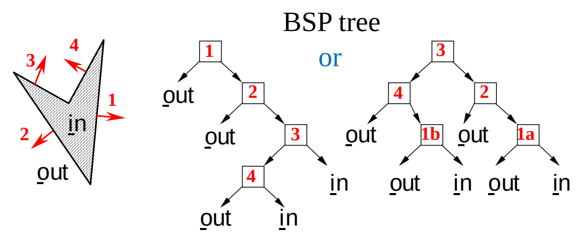
2D example:



More BSP Examples



BSP Trees as Solids



- Left empty trees represent outside space
- Right empty trees represent inside volumes

Operations with BSP Trees (1/2)

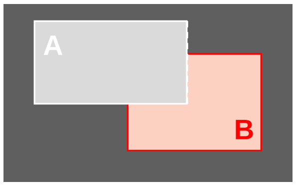
- **Rendering**
  - BSP trees are very good for fast rendering
  - Painter´s Algorithm:

```
IF eye is in front of a (in A+)
THEN BEGIN draw all polygons of A-;
       draw a;
       draw all polygons of A+ END
ELSE BEGIN draw all polygons of A+;
       (draw a);
       draw all polygons of A- END;
```


Operations with BSP Trees (2/2)

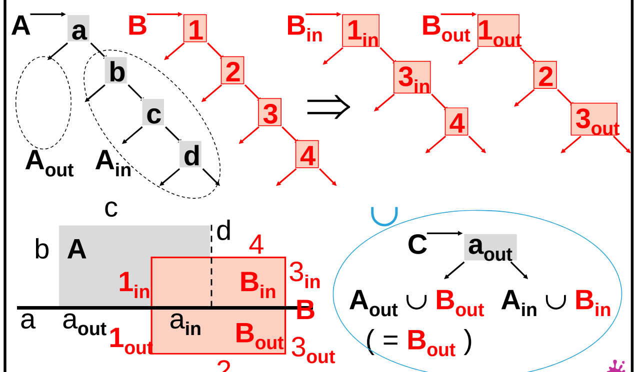
- **Transformations**
  - Points, plane equation and normal vector have to be transformed
- **Combinations**
  - Perform combination with B-Rep, then generate BSP tree
  - Combine BSP trees directly (faster)

Combination of BSP Trees



The structure of one tree has to act as structure for the result ⇒ one tree has to be included into the other

Combination of BSP Trees: ∪



BSP Algorithm for A op B = C:

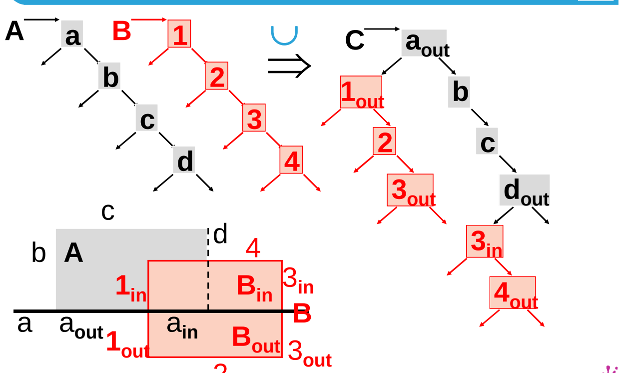
- A or B homogeneous (full or empty) $\Rightarrow$ simple rules
- Else:
  - 1. Divide root polygon a of A at object B in a_{in}, a_{out}
  - 2. Root node c of C: if op="and" then c:=a_{in} else c:=a_{out} (with its plane)
  - 3. Divide B at plane of a in B_{in}, B_{out}
  - 4. Recursive evaluation of the subtrees:

$C_{left}=A_{out}$ op B_{out} $\quad$ $C_{right}=A_{in}$ op B_{in}

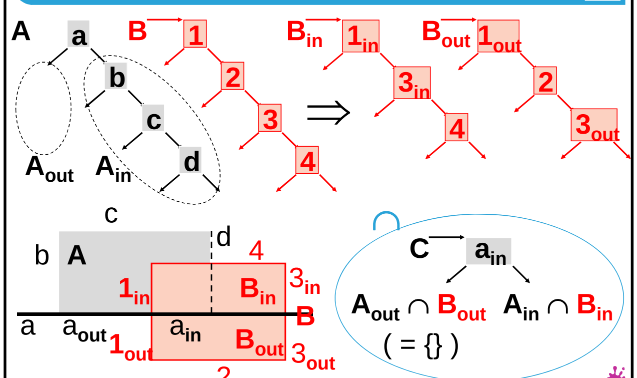
Simple BSP Node Combination Rules

op	A	B	A op B
or	inhom.	full	full
	inhom.	empty	A
	full	inhom.	full
	empty	inhom.	B
and	inhom.	full	A
	inhom.	empty	empty
	full	inhom.	B
	empty	inhom.	empty
sub	inhom.	full	empty
	inhom.	empty	A
	full	inhom.	–B
	empty	inhom.	empty

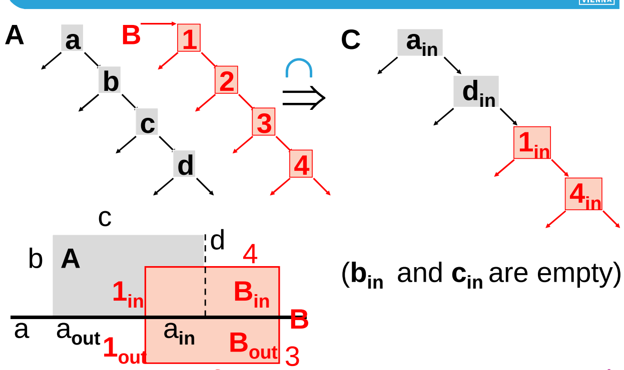
Combination of BSP Trees: ∪



Combination of BSP Trees: ∩



Combination of BSP Trees: ∩



(b_{in} and c_{in} are empty)

Properties of BSP Trees

- **Advantages**
  - Fast rendering
  - Fast transformation
  - Combinations faster than for B-Reps
  - General representation
  - Generation of models via digitization
  - Tree structure (fast search)

Properties of BSP Trees

- **Disadvantages**
  - Curved objects must be approximated
  - Only convex polygons
  - High memory cost

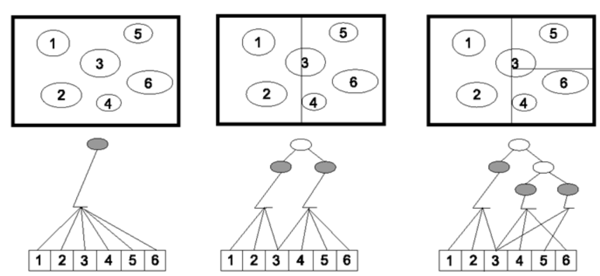
3D-Data Structures: Overview

- Point Cloud
- Wire-frame Model
- Boundary Representation
- Binary Space Partitioning Tree
- kD Tree
- Octree
- Constructive Solid Geometry Tree
- Bintree
- Grid

kD Tree

- Special case of BSP Tree
- Only axes-aligned partitioning planes => specified by one value
- Partitioning direction specified either implicitly (pre-defined order) or explicitly
- 1D Tree ⇔ binary tree

kD Tree Example: 2-D Tree



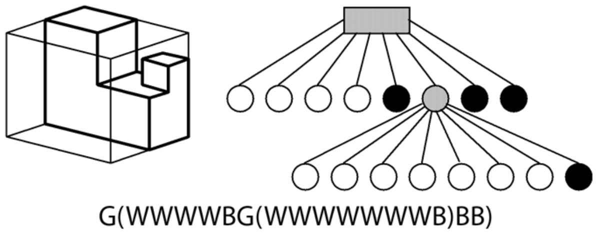
3D-Data Structures: Overview

- Point Cloud
- Wire-frame Model
- Boundary Representation
- Binary Space Partitioning Tree
- kD Tree
- Octree
- Constructive Solid Geometry Tree
- Bintree
- Grid

Octree

- Used to represent solid volumetric objects
- Each node is subdivided in 8 subspaces
- Each subspace is either empty, full or further divided
- The subdivision stops when an object can be represented accurate enough

Octree Example



Operations with Octrees

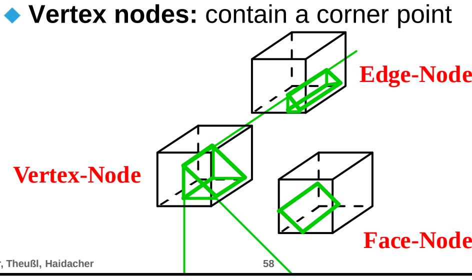
- **Transformations**
  - Hard to implement; easy: rotations of 90°
- **Combinations**
  - Can easily be done by logical operations; both octrees must be adapted to each other to have the same depth in each subspace
- **Rendering**
  - The octree is rendered depending on the view direction starting with the subspace farthest away from the viewer

Properties of Octrees

- **Advantages**
  - Combinations are easy to implement
  - Spatial search is fast due to the tree structure
  - Rendering algortihm is fast
- **Disadvantages**
  - High storage consumption for approximated objects
  - Transformations are not trivial in general
  - General objects cannot be represented exactly

Extended Octrees

- Additional node types:
  - **Face nodes:** contain a surface
  - **Edge nodes:** contain an edge
  - **Vertex nodes:** contain a corner point

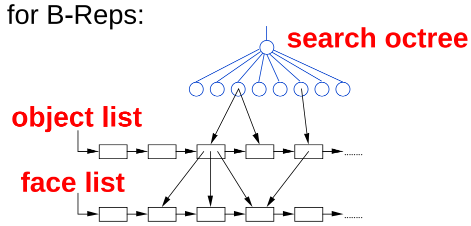


Generation of Extended Octrees

- 1. Generate B-Rep
- 2. Divide point and surface list at the subdivision planes into 8 sets
- 3. For each octant:
  - Point and surface lists empty ⇒ **full or empty**
  - Only one vertex ⇒ **vertex** node
  - Only one surface ⇒ **face** node
  - Only two surfaces ⇒ **edge** node
  - Else: subdivide recursively

Octree as Spatial Directory

- Octree as search structure for objects in other representations
- E.g. for B-Reps:



octree of low depth is sufficient

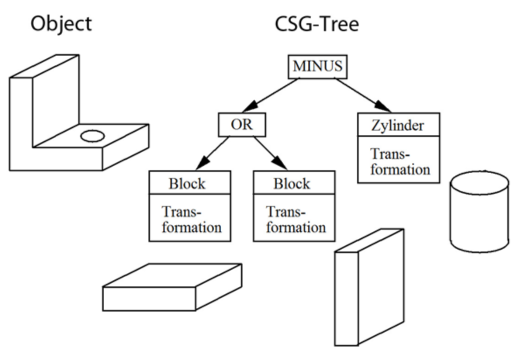
3D-Data Structures: Overview

- Point Cloud
- Wire-frame Model
- Boundary Representation
- Binary Space Partitioning Tree
- kD Tree
- Octree
- Constructive Solid Geometry Tree
- Bintree
- Grid

Constructive Solid Geometry Tree

- A Constructive Solid Geometry (CSG) Tree consists of simple primitives, transformations and logical operations
- Useful to describe complex objects with a small number of primitives
- Examples for primitives
  - Cube
  - Sphere
  - Cylinder

CSG Tree Example



Operations with CSG Trees

- **Transformations**
  - An object is transformed by adding the transformation to the transformation of each primitive
- **Combinations**
  - Two objects are simple combined by adding them as children in a new tree
- **Rendering**
  - Needs to be converted into a B-Rep or it is rendered with raytracing

Properties of CSG Trees

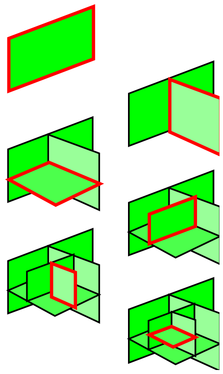
- **Advantages**
  - Minimal storage consumption
  - Combinations and transformations are simple
  - Objects can be represented exactly
  - Tree structure (fast search)
- **Disadvantages**
  - Cannot be rendered directly; slow rendering
  - Model generation cannot be done through digitization of real objects

3D-Data Structures: Overview

- Point Cloud
- Wire-frame Model
- Boundary Representation
- Binary Space Partitioning Tree
- kD Tree
- Octree
- Constructive Solid Geometry Tree
- Bintree
- Grid

Bintree

- 3D Tree
- Subdivision order xyzxyz...
- Choose separation plane for optimized (irregular) subdivision
- Fewer nodes than octree



3D-Data Structures: Overview

- Point Cloud
- Wire-frame Model
- Boundary Representation
- Binary Space Partitioning Tree
- kD Tree
- Octree
- Constructive Solid Geometry Tree
- Bintree
- Grid

Grid

- Regular subdivision
- Directly addresses cells
- Simple neighborhood finding O(1)
  - E.g. for ray traversal
- Problem:
  - Too few/many cells
  - ⇒ **Hierarchical grid**