Advanced 3D-Datastructures

3D-Datastructures: Requirements

- general objects
- exact representation of objects
- generation of models via digitization
- combinations
- linear transformation
- interaction
- memory consumption
- fast rendering

3D-Datastructures: Overview

- Point Cloud
- Wireframe
- Boundary Representation
- Binary Space Partitioning Tree
- Extended Octree
- Bintree
- Grid

Point Cloud

Object = set (list) of points

- e.g. from a digitizer
- 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

Point Cloud Properties

- **advantages**
  - fast rendering
  - exact representation & rendering possible
  - fast transformations
  - generation of models via digitization
- **disadvantages**
  - many points (curved obj., exact representation)
  - high memory consumption
  - limited combination operations

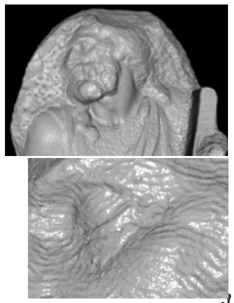
Surfels (SURFace ELementS)

http://www.merl.com/projects/surfels/

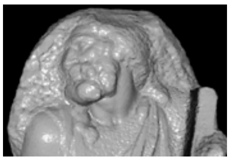
movies: cab, wasp, salamander with holes, salamander corrected (more movies on web page)

QSplat (1)

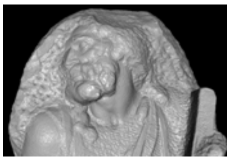
- 3D scan of 2.7 meter statue of St. Matthew at 0.25 mm
- 102,868,637 points
- File size: 644 MB
- Preprocessing time: 1 hour
- Demo on laptop (PII 366, 128 MB), no 3D graphics hardware
- http://graphics.stanford.edu /software/qsplat/



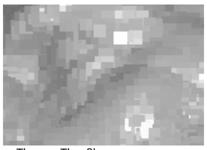
Qsplat (2)

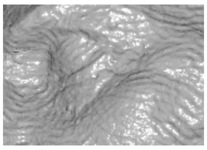


Interactive (8 frames/sec)



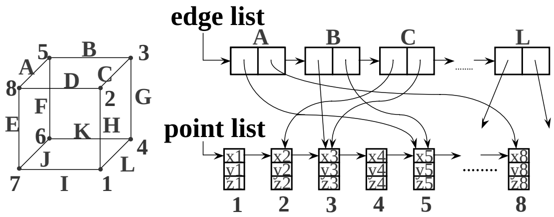
High quality (8 sec)





Wireframe

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



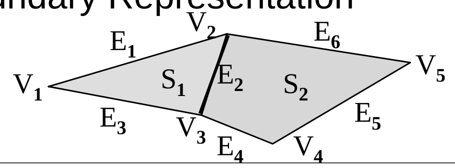
Operations with Wire-Frame Model

- **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

Wire-Frame Model Properties

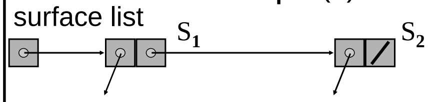
- **advantages**
  - quick rendering
  - easy and quick transformations
  - generation of models via digitization
- **disadvantages**
  - high memory consumption
  - inexact (no surfaces, no occlusion)
  - restricted combination possibilities
  - curves are approximated by straight lines

Boundary Representation

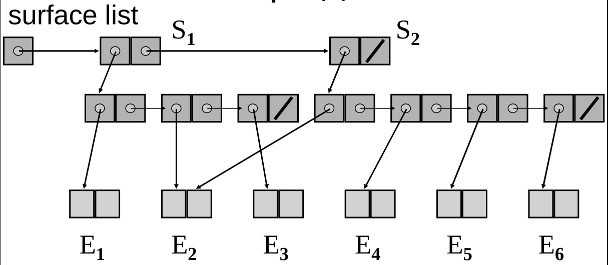


vertex list	edge list	surface list
V_1: x_1 y_1 z_1	E_1: V_1 **V_2**	**S_1:** E_1 **E_2** E_3
V_2: x_2 y_2 z_2	**E_2: V_2 V_3**	**S_2:** **E_2** E_4 E_5 E_6
V_3: x_3 y_3 z_3	E_3: **V_3** V_1	
V_4: x_4 y_4 z_4	E_4: **V_3** V_4	
V_5: x_5 y_5 z_5	E_5: V_4 V_5	
	E_6: V_5 **V_2**	

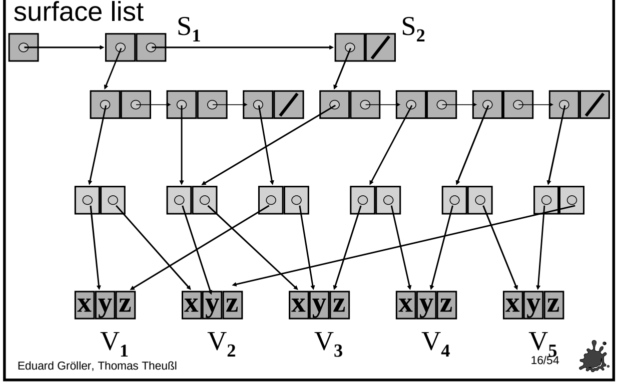
Lists for B-Reps (1)



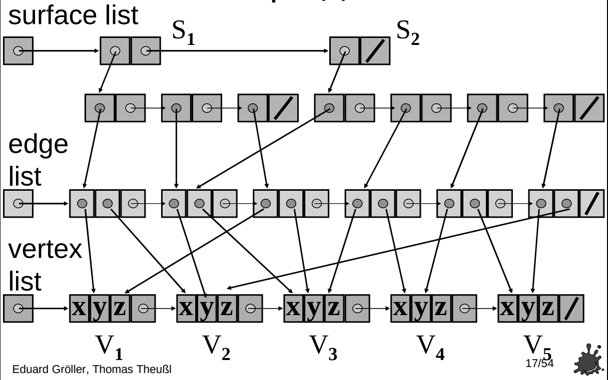
Lists for B-Reps (2)



Lists for B-Reps (3)



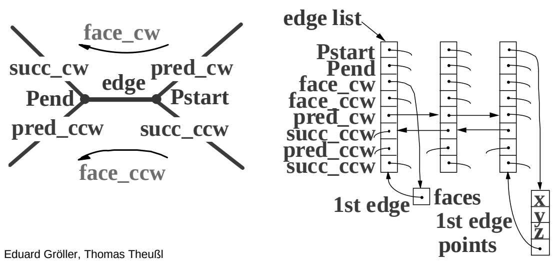
Lists for B-Reps (4)



Winged Edge Data Structure

alternative for normal hierarchical B-Rep.

here the central element is the edge:



Operations with B-Reps (1)

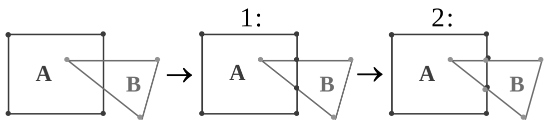
- **transformations**
  all points are transformed as with wire-frame model, additionally surface equations or normal vectors can be transformed

Operations with B-Reps (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, steps 1. and 2.

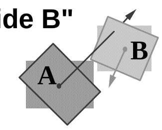


- every polygon has a box enclosure
  ⇒ simple test if polygons can intersect
- use only convex polygons and produce only convex polygons as results
  ⇒ simple intersection tests

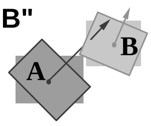
Combinations of B-Reps, steps 3. and 4.

- 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"

"outside B"

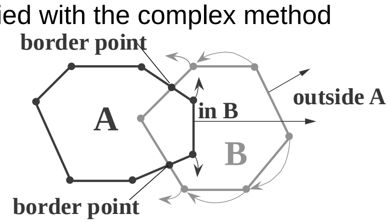


"in B"



Combinations of B-Reps, steps 3. and 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, step 5.

polygons can be removed according to the tables:

for polygons of A

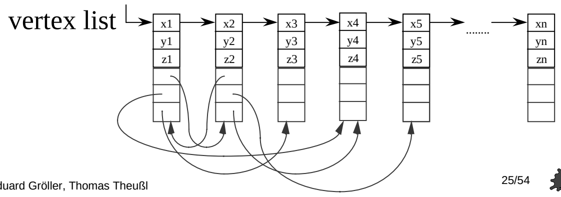
op.	in B	outside B	on B (coplanar) NV equal	on B (coplanar) different
A or B	yes	no	no	yes
A and B	no	yes	no	yes
A sub B	yes	no	yes	no

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.

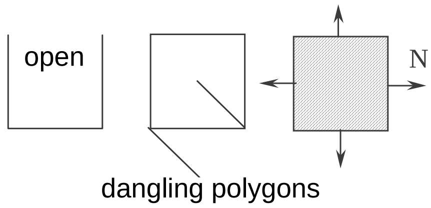
- no open (non-closed) objects
- only convex polygons
- no double points
- additional links in the vertex list between neighbor points with equal classification



Representation of Solids in a B-Rep

a B-Rep allows also polygon sets which represent no solid object.

therefore: definition of a semantic



Operations with B-Reps (3)

- **rendering of B-Rep objects**
  hidden surface or hidden line algorithms can be used because the surfaces of the objects are known, so that the visibility can be calculated.

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

Partitioning of Object Surfaces

necessary to approximate curved surfaces

- surfaces that ***can*** be parametrized:
  e.g. free form surfaces, quadrics, superquadrics
  partitioning of parameter space,
  one patch for every 2D parameter interval
- surfaces that ***cannot*** be parametrized:
  e.g. implicit surfaces, "bent" polygons
  ⇒ tesselation, subdivision surfaces

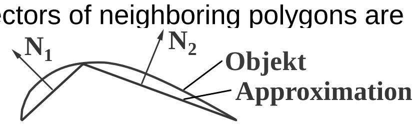
Tesselation

divide polygons in smaller polygons (triangles) until the approximation is exact enough.

normal vector criterion as termination condition:

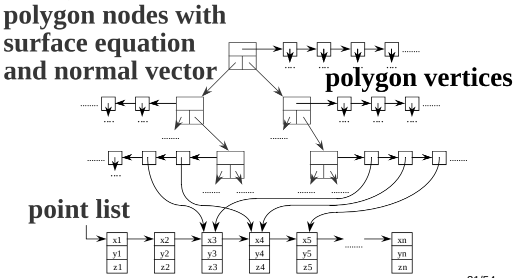
$$\mathbf{N_1 \cdot N_2 \geq 1-\varepsilon}$$

normal vectors of neighboring polygons are similar:



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

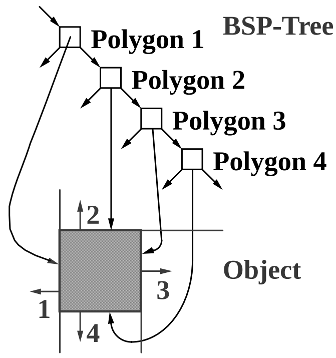
- convex objects: BSP tree is linear list
- else: conversion B-Rep ⇒ BSP tree

Algorithm:

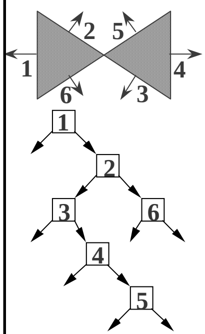
1. find the polygon who's plane intersects the fewest other polygons and cut these in two
2. divide the polygon list in two sets:
   in front of that plane / behind that plane
3. the polygon found in 1. is the root of the BSP tree, the left and the right subtrees can be generated recursively (from two "halves")

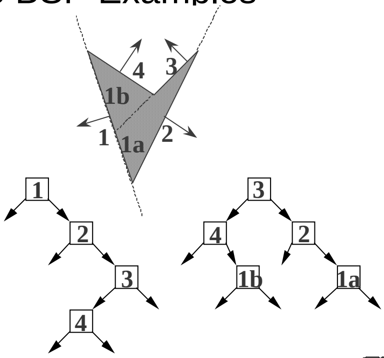
BSP Example

2D example:

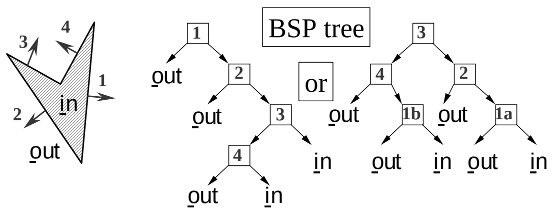


More BSP Examples





BSP Trees as Solids

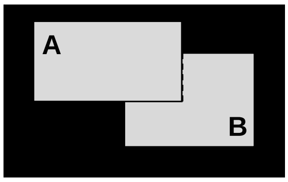


left empty trees represent outside space,
right empty trees represent inside volumes

Operations with BSP Trees (1)

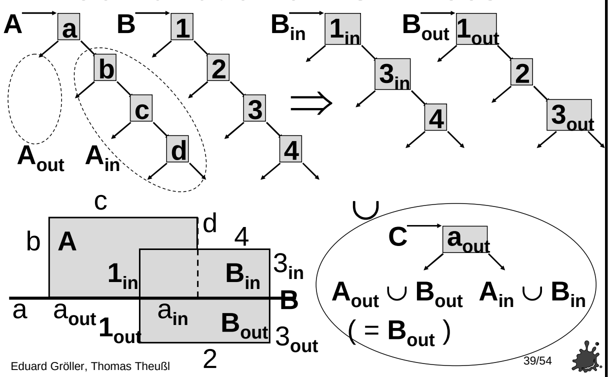
- **transformations**
  points, plane equation and normal vector have to be transformed
- **combinations**
  - perform combination with B-Rep, then generate BSP tree
  - combine BSP trees directly (faster)

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:

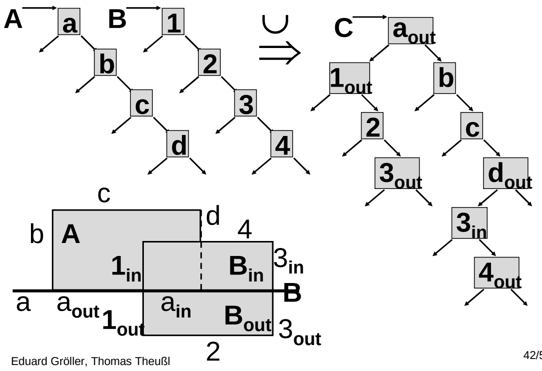
- A or B homogeneous (full or empty)
  ⇒ simple rules
- else:

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} C_{right}=A_{in} op B_{in}

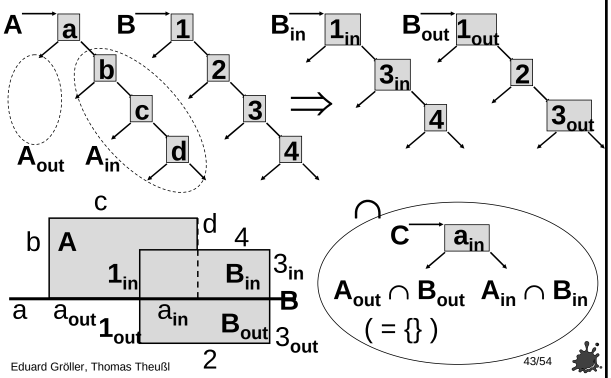
Simple BSP Node Combination Rules

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**

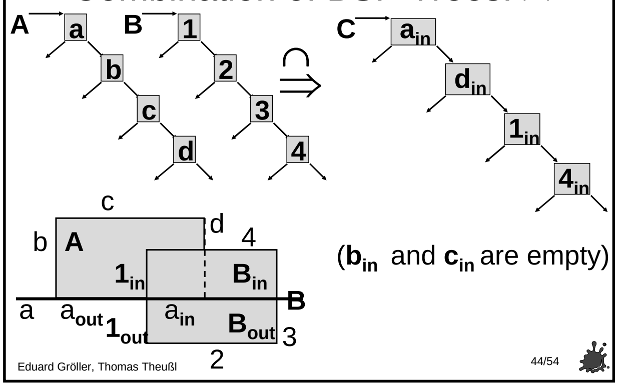
Combination of BSP Trees: ∪



Combination of BSP Trees: ∩



Combination of BSP Trees: ∩



Operations with BSP Trees (3)

- **rendering of BSP objects**

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;
```


BSP Tree Properties

- **advantages**
  - fast rendering
  - fast transformation
  - combinations faster than for B-Reps
  - general representation
  - generation of models via digitization

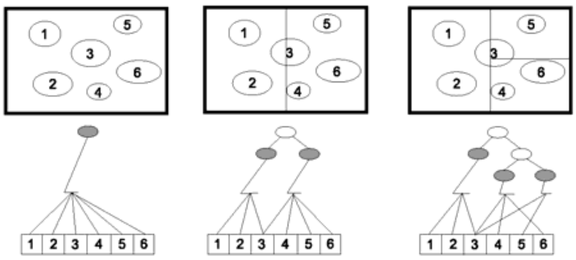
BSP Tree Properties

- **disadvantages**
  - curved objects must be approximated
  - only convex polygons
  - high memory cost

k-D 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
- 1-D Tree ⇔ binary tree

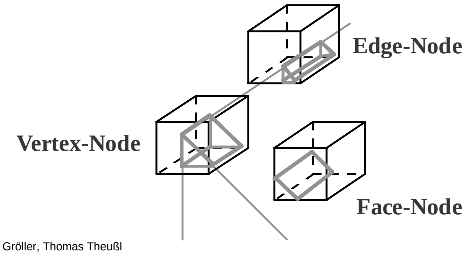
k-D Tree Example: 2-D Tree



Extended Octrees

additional node types:

- **face nodes** contain a surface
- **edge nodes** contain an edge
- **vertex nodes** contain a corner point



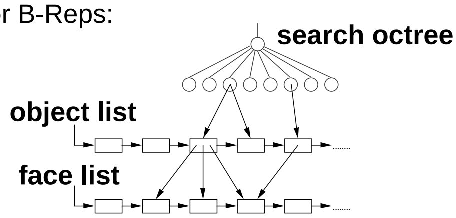
Generation of Extended Octrees

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

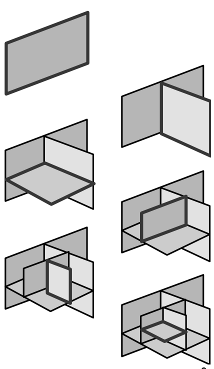
e.g. for B-Reps:



octree of low depth is sufficient

Bintree

- 3-D Tree
- subdivision order xyzxyz...
- choose separation plane for optimized (irregular) subdivision
- fewer nodes than octree



Grid

- regular subdivision
- directly addressable cells
- simple neighbour finding O(1),
  e.g., for ray traversal
- problem:
  too few/many cells
  ⇒ **Hierarchical Grid**