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.
Properties

- Representation of general objects
- Exact representation of objects
- Combinations of objects
- Linear transformation
- Interaction
- Fast spatial searches
- Memory consumption
- 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
- Exact representation if $\geq 1$ points/pixel
  - More efficient than 1 pixel sized polygons
3D model of Machu Picchu

Team from TU Wien scanned Southern train station of Vienna

http://www.tuwien.ac.at/aktuelles/news_detail/article/6203/
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)

- [Link](http://www.merl.com/publications/TR2000-10)
- Movies: cab, wasp, salamander with holes, salamander corrected
- http://graphics.stanford.edu/software/qsplat/
- 3D scan of 2.7 meter statue of St. Matthew at 0.25 mm
- 102,868,637 points
- Preprocessing time: 1 hour
- Demo on laptop (PII 366, 128 MB), no 3D graphics hardware
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.

Edge list

Vertex list

A  B  C  L

Gröller, Theußl, Haidacher
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**
- 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)

**Diagram:**
- **Vertices (V):** V1: (x1, y1, z1), V2: (x2, y2, z2), V3: (x3, y3, z3), V4: (x4, y4, z4), V5: (x5, y5, z5)
- **Simplices (S):** S1: E1 E2 E3, S2: E2 E4 E5 E6

### Tables:

<table>
<thead>
<tr>
<th>Vertex list</th>
<th>Edge list</th>
<th>Face list</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1: x1 y1 z1</td>
<td>E1: V1 V2</td>
<td>S1: E1 E2 E3</td>
</tr>
<tr>
<td>V2: x2 y2 z2</td>
<td>E2: V2 V3</td>
<td>S2: E2 E4 E5 E6</td>
</tr>
<tr>
<td>V3: x3 y3 z3</td>
<td>E3: V3 V1</td>
<td></td>
</tr>
<tr>
<td>V4: x4 y4 z4</td>
<td>E4: V3 V4</td>
<td></td>
</tr>
<tr>
<td>V5: x5 y5 z5</td>
<td>E5: V4 V5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>E6: V5 V2</td>
<td></td>
</tr>
</tbody>
</table>
Face list

\( S_1 \quad S_2 \)
Face list

\[ S_1 \quad S_2 \]

\[ E_1 \quad E_2 \quad E_3 \quad E_4 \quad E_5 \quad E_6 \]
Face list

\[ S_1 \rightarrow S_2 \]

\[ V_1 \rightarrow V_2 \rightarrow V_3 \rightarrow V_4 \rightarrow V_5 \]
Lists for B-Reps (4/4)

Face list

Edge list

Vertex list

S_1

S_2

V_1

V_2

V_3

V_4

V_5

Gröller, Theußl, Haidacher
Alternative for normal hierarchical B-Rep

Here the central element is the edge:

- **Pstart**
- **Pend**
- **pred_cw**
- **succ_cw**
- **face_cw**
- **face_ccw**

**Edge List**:
- **Pstart**
- **Pend**
- **face_cw**
- **face_ccw**
- **pred_cw**
- **pred_ccw**
- **succ_cw**
- **succ_ccw**

**Faces**

1st edge

1st edge points

Gröller, Theußl, Haidacher
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
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 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
  - Tessellation, subdivision surfaces
Tesselation

- Divide polygons in smaller polygons (triangles) until the approximation is exact enough
- Normal vector criterion as termination condition:
  \[ N_1 \cdot N_2 \geq 1 - \varepsilon \]
- Normal vectors of neighboring polygons are similar:
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** ⇒ simple test if polygons can intersect
- Use only **convex** polygons and produce only convex polygons as results ⇒ simple intersection tests
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"
Improvement: points of A, which lie on the surface of B, are marked as border points during the dividing process (and vice versa) \(\Rightarrow\) only very few polygons have to be classified with the complex method.
Polygons can be removed according to tables:

<table>
<thead>
<tr>
<th>op.</th>
<th>in B</th>
<th>outside B</th>
<th>on B (co-planar)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>NV equal</td>
</tr>
<tr>
<td>A or B</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>A and B</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>A sub B</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

For polygons of A

<table>
<thead>
<tr>
<th>op.</th>
<th>in A</th>
<th>outside A</th>
<th>on A (co-planar)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>NV equal</td>
</tr>
<tr>
<td>A or B</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>A and B</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>A sub B</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

For polygons of B
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
3D-Data Structures: Overview

- Point Cloud
- Wire-frame Model
- Boundary Representation
- Binary Space Partitioning Tree
- kD Tree
- Octree
- Constructive Solid Geometry Tree
- Bintree
- Grid
Special B-Rep for quick rendering with visibility

- Especially of static scenes

**polygon nodes with surface equation and normal vector**

Point list

Gröller, Theußl, Haidacher
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
Generate BSP Tree:

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")
More BSP Examples

or

Gröller, Theußl, Haidacher
BSP Trees as Solids

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

BSP tree

Gröller, Theußl, Haidacher
Convex objects: BSP tree is a linear list
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)
Disadvantages

- Curved objects must be approximated
- Only convex polygons
- High memory cost
Operations with BSP Trees (1/2)

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

```plaintext
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;
```

Gröller, Theußl, Haidacher
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)
The structure of one tree has to act as structure for the result $\Rightarrow$ one tree has to be included into the other
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:
    
    \[
    C_{left} = A_{out} \text{ op } B_{out} \quad \quad C_{right} = A_{in} \text{ op } B_{in}
    \]

Gröller, Theußl, Haidacher
Combination of BSP Trees: $\bigcup$

Gröller, Theußl, Haidacher
# Simple BSP Node Combination Rules

<table>
<thead>
<tr>
<th>op</th>
<th>A</th>
<th>B</th>
<th>A op B</th>
</tr>
</thead>
<tbody>
<tr>
<td>or</td>
<td>inhom.</td>
<td>full</td>
<td>full</td>
</tr>
<tr>
<td></td>
<td>inhom.</td>
<td>empty</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>full</td>
<td>inhom.</td>
<td>full</td>
</tr>
<tr>
<td></td>
<td>empty</td>
<td>inhom.</td>
<td>B</td>
</tr>
<tr>
<td>and</td>
<td>inhom.</td>
<td>full</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>inhom.</td>
<td>empty</td>
<td>empty</td>
</tr>
<tr>
<td></td>
<td>full</td>
<td>inhom.</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>empty</td>
<td>inhom.</td>
<td>empty</td>
</tr>
<tr>
<td>sub</td>
<td>inhom.</td>
<td>full</td>
<td>empty</td>
</tr>
<tr>
<td></td>
<td>inhom.</td>
<td>empty</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>full</td>
<td>inhom.</td>
<td>–B</td>
</tr>
<tr>
<td></td>
<td>empty</td>
<td>inhom.</td>
<td>empty</td>
</tr>
</tbody>
</table>
Combination of BSP Trees: $\cup$

A → a → b → c → d

B → 1 → 2 → 3 → 4

C → a\text{out} → b → c → d\text{out}

1\text{in} → B\text{in} → 3\text{in} → 4\text{out}

2

Gröller, Theußl, Haidacher
Combination of BSP Trees: \( \bigcap \)

\[ A_{out} \bigcap B_{out} \quad A_{in} \bigcap B_{in} \]

\[ A_{out} \bigcap B_{out} = \emptyset \]
Combination of BSP Trees: \( \cap \)

A diagram illustrates the combination of three BSP trees labeled A, B, and C. The diagram includes nodes marked with letters (a, b, c, d) and directions (in, out) indicating the flow of data. The text states that \( b_{in} \) and \( c_{in} \) are empty.

Gröller, Theußl, Haidacher
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

Gröller, Theußl, Haidacher
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 into 8 subspaces
- Each subspace is either empty, full or further divided
- The subdivision stops when an object can be represented accurate enough
Octree Example

\[ G(\text{WWWBWBGBWMB}) \]
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 algorithm is fast

- **Disadvantages**
  - High storage consumption for approximated objects
  - Transformations are not trivial in general
  - General objects cannot be represented exactly

Gröller, Theußl, Haidacher
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

Gröller, Theußl, Haidacher
3D-Data Structures: Overview

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

Gröller, Theußl, Haidacher
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

Object

CSG-Tree

MINUS

OR

Block
Transformation

Block
Transformation

Zylinder
Transformation

Gröller, Theußl, Haidacher
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
- $\Rightarrow$ Hierarchical grid