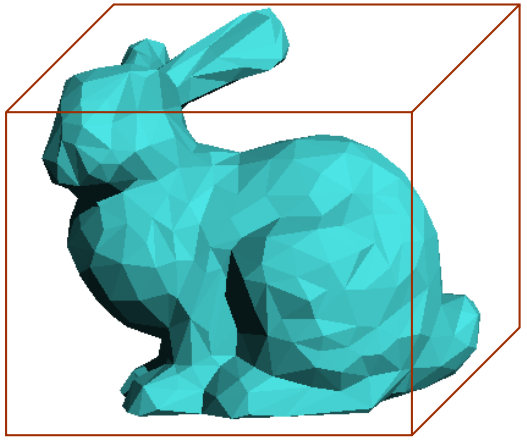
Spatial Partitioning vs. Object Partitioning

- So far: acceleration data structure subdivided space, objects (=triangles) are associated afterwards to the cells
- Now: partition the set of objects, associate a bounding volume (= subset of space) with each
- In reality, the borders between the two categories are not clear-cut!

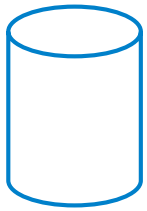
Bounding Volumes (BVs)

- General idea: approximate complex, geometric objects, or sets of objects, by some outer "hull"
- Requirements:
  - The objects must be completely inside the BV
  - More to come ...



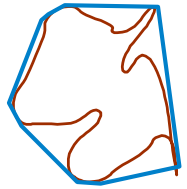
Examples of Bounding Volumes

Master thesis ...



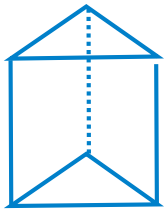
Cylinder
[Weghorst et al., 1985]

Box, AABB (R*-trees)
[Beckmann, Kriegel, et al., 1990]

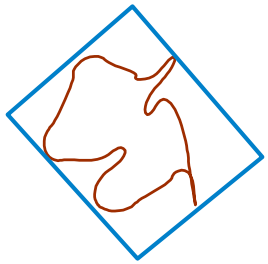


Convex hull
[Lin et. al., 2001]

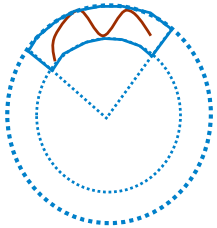
Sphere
[Hubbard, 1996]



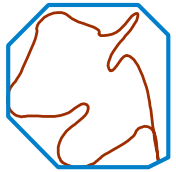
Prism
[Barequet, et al., 1996]



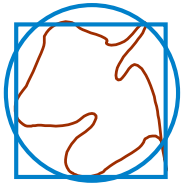
OBB (oriented bounding box)
[Gottschalk, et al., 1996]



Spherical shell
[...]



k-DOPs / Slabs
[Zachmann, 1998]

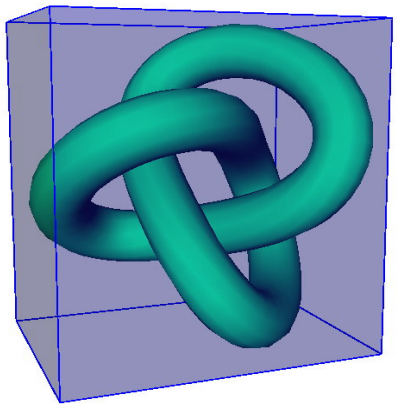


Intersection of
several, other BVs

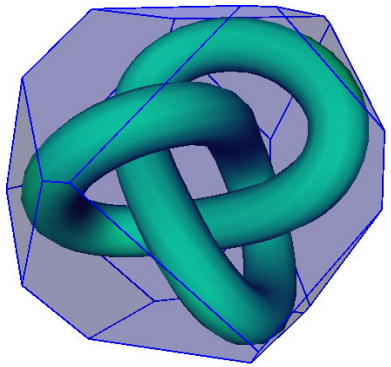
k-DOPs

- Examples:

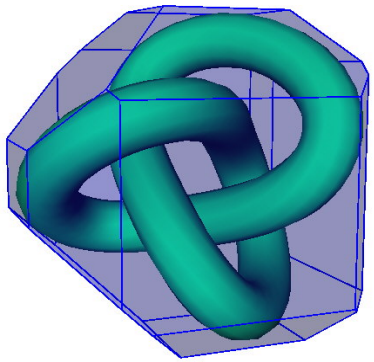
6-DOP
(AABB)



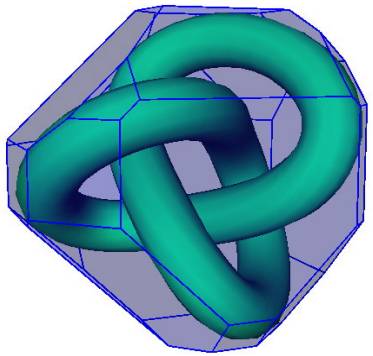
14-DOP



18-DOP



26-DOP



The Costs of BVs

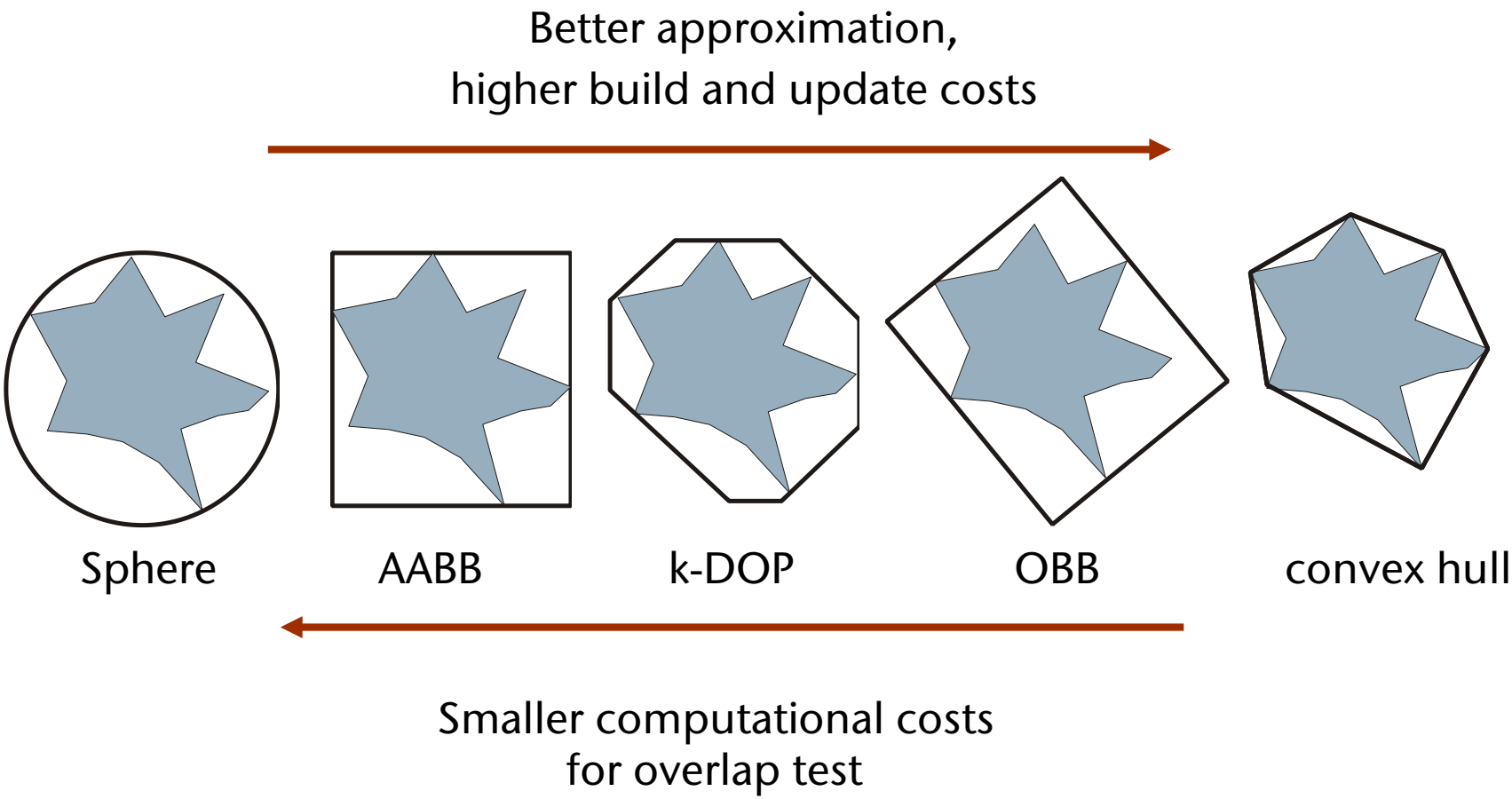
- Costs of a ray intersection with a subset of the scene, enclosed in a BV:

$$T = n \cdot B + m \cdot I$$

T = total costs
n = number of rays tested against the BV
B = costs for one ray-BV intersection test
m = number of rays that actually intersect the BV
I = costs for testing the objects in the BV

- Goal: minimize T
- Consequence: 2 incompatible requirements on BVs:
  - BVs should be simple (e.g., sphere or box) = small costs for ray tests, B; downside: number of ray hits, m, is usually large
  - BVs should be compact (e.g., exact, convex hull) = small m; downside: intersection costs, B, are high

- Qualitative comparison:

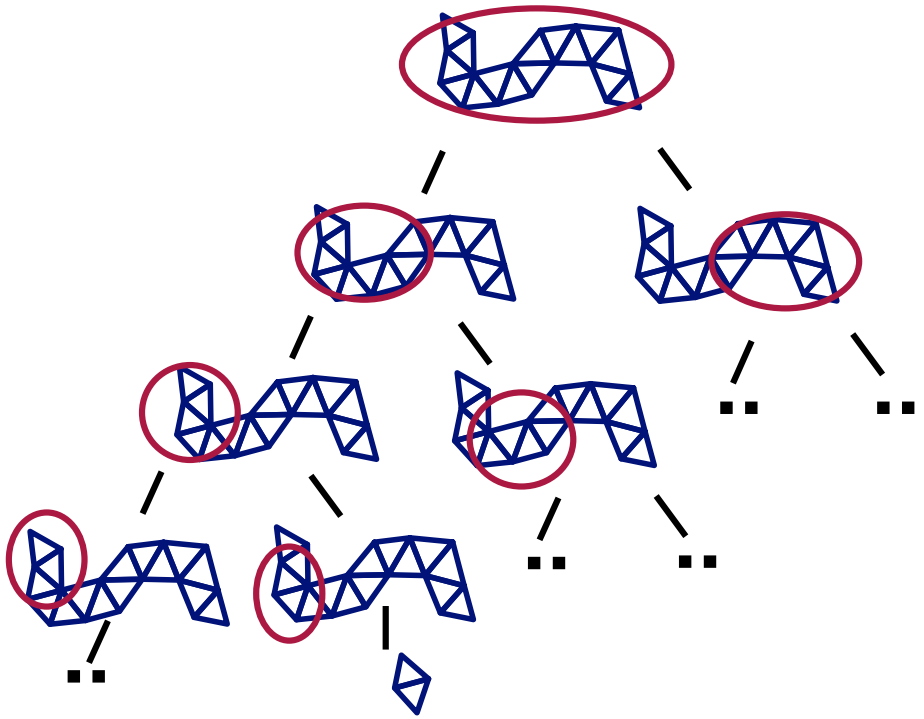


The Bounding Volume Hierarchy (BVH)

- Definition:
  A BVH over a set of primitives, $\mathcal{P}$, is a tree where each node are associated
  - a subset of $\mathcal{P}$; and
  - a BV $\mathcal{B}$, that encloses all primitives in the subset.
- Remark:
  - Often, we use the BV as a synonym for the node in the BVH
  - Primitives are usually *stored only* at child nodes
    - Feel free to experiment; exceptions can make sense
  - Most of the time, primitives are partitioned, i.e., if $\mathcal{P}$ is the set of primitives associated with a node, and $\mathcal{P}_i$ are the subsets of primitives associated with the children, then

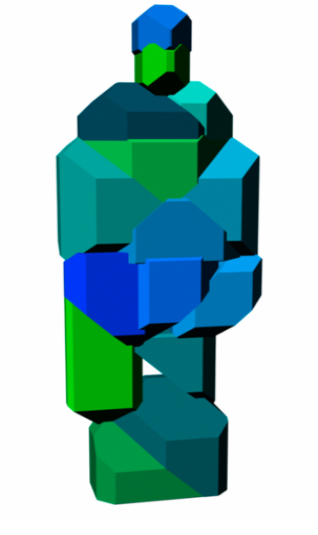
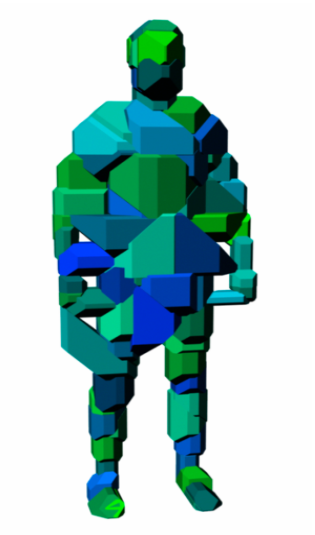
$$\mathcal{P} = \mathcal{P}_1 \dot{\cup} \ldots \dot{\cup} \mathcal{P}_n$$

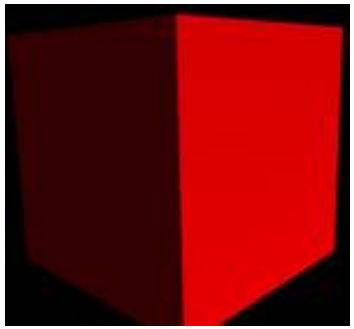
- Schematic example:

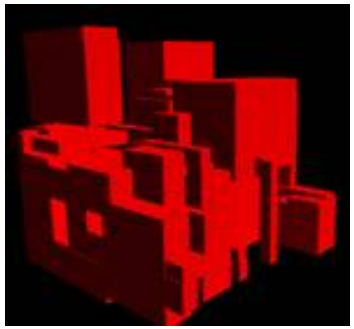
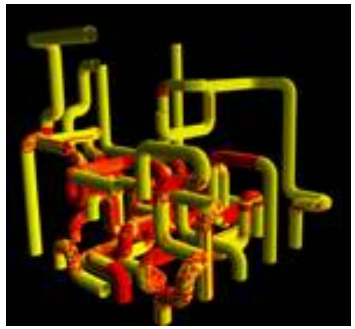


- Parameters:
  - The kind of BV used
  - *"Arity"* (degree of the nodes)
  - Stopping criterion (in particular, number of triangles per leaf)
  - Criterion for partitioning the primitives (guiding the construction)

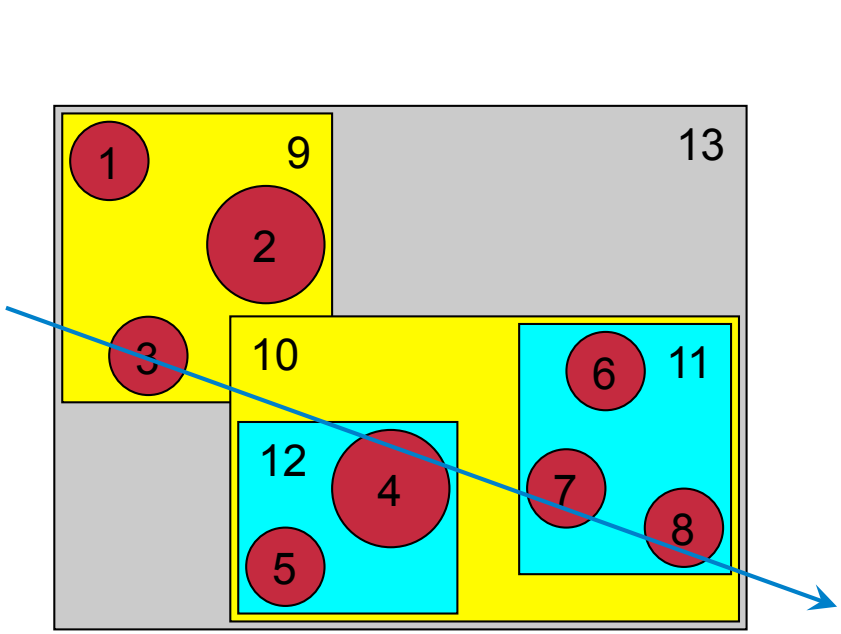
Examples

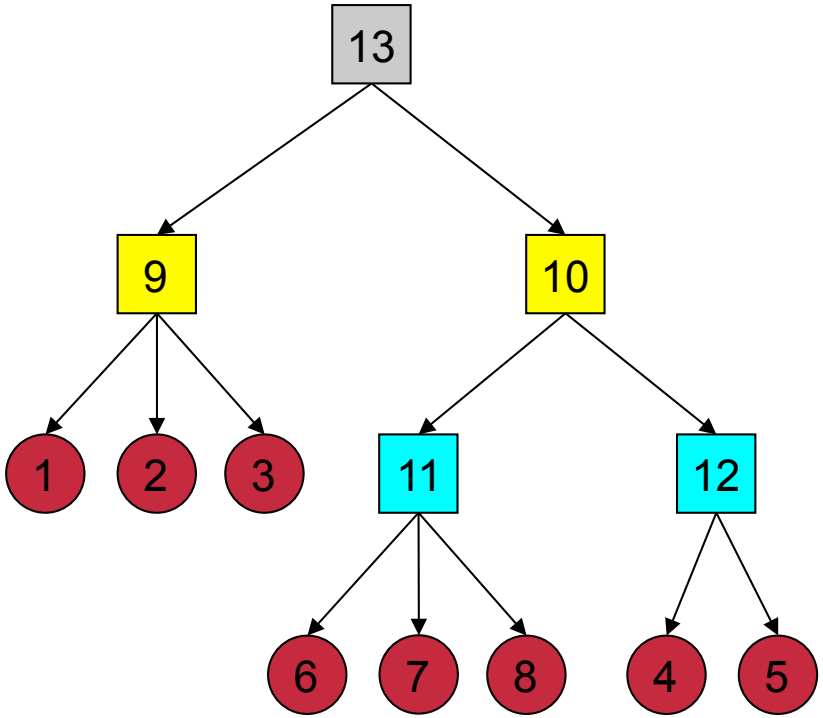



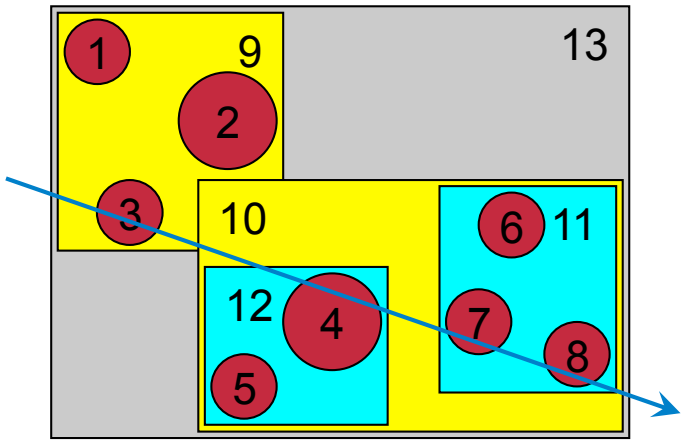



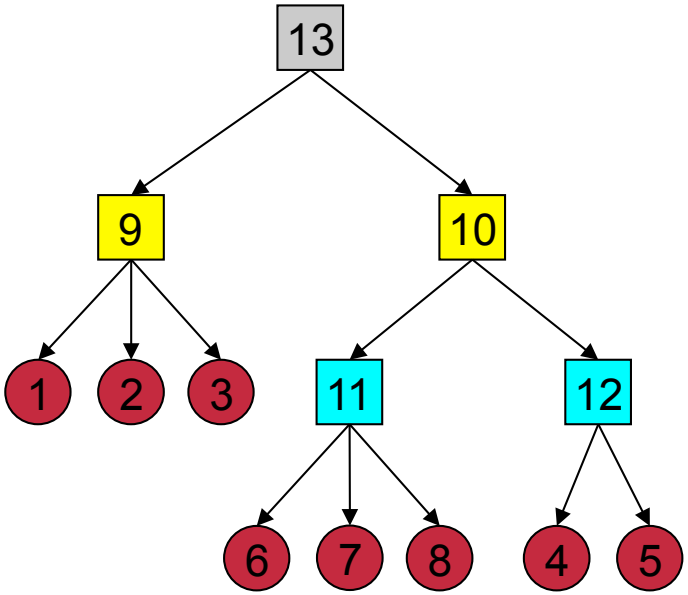



Example for the Traversal of a BVH with a Ray









- Test 13 ⟶ yes
  - Test 9 ⟶ yes
    - Test 1 ⟶ no
    - Test 2 ⟶ no
    - Test 3 ⟶ yes
  - Test 10 ⟶ yes, but intersection point is farther away

➢ Result: only 3 instead of 8 tests with objects, plus 3 tests with BVs

- Question: why did we start with BV 9?

A Better Hierarchy Traversal

- Problem: the order by which nodes are visited with pure depth-first search (DFS) depends *only* on the topology of the tree
- Better: consider the spatial layout of the BV's, too
- Criterion: distance between origin of ray and intersection with BV (*estimated distance*)
- Consequence: should not use simple recursion / stack any more
- Use **priority queue**

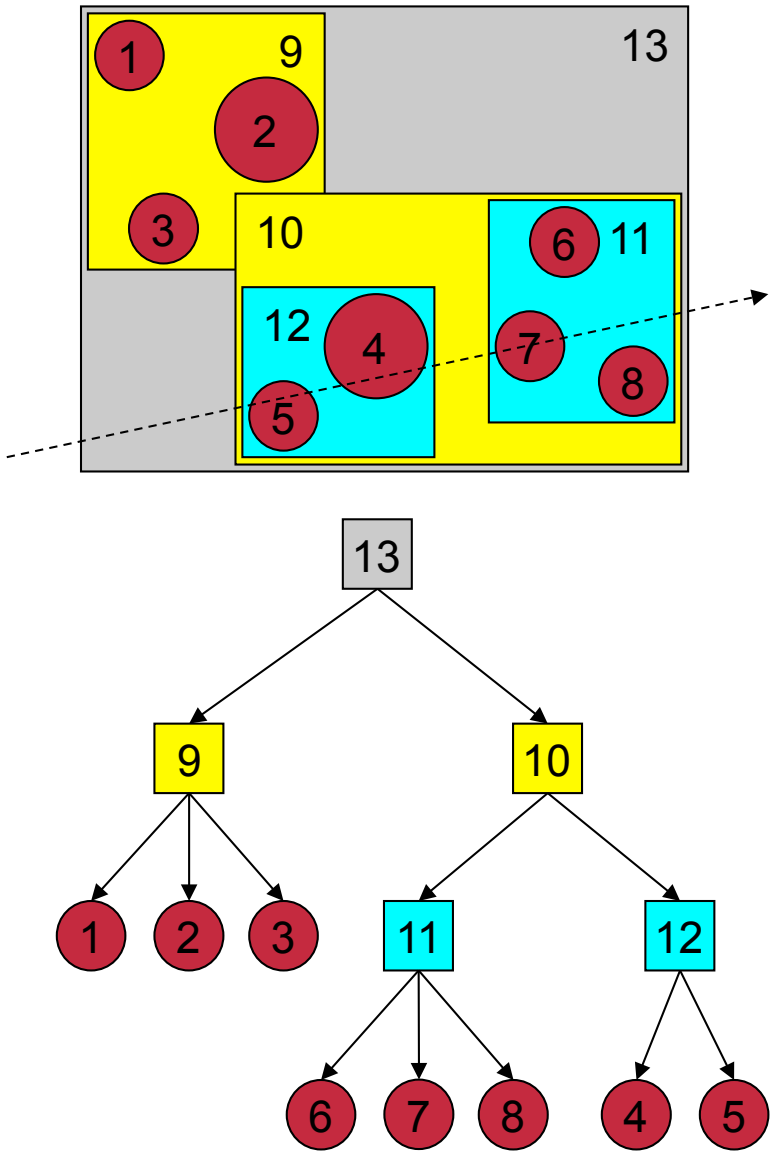
Algorithm

- Maintain a p-queue
  - Contains all BVs that still need to be visited
  - Sorted by their distance from ray origin (along ray)

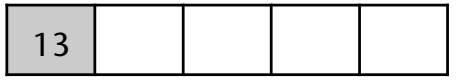
```
Pqueue q ← init with root
closest_hit = ∞
while q not empty:
  node ← extract front from q          // = nearest BV
  if dist(node) <= closest_hit:        // else: skip this subtree
    if node is leaf:
      intersect ray with all polygons in node
      update closest_hit, if any polygon is closer
    else                               // inner node
      forall children of node:
        if ray intersects child:
          insert child in q with its distance
```

- Efficient implementation of a p-queue: heap
- Insertion of an element, and extracting the front $\longrightarrow$ O(log n)

Example



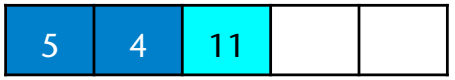
- Test with 13 ⟶ yes, insert



- Pop front of queue ⟶ 13
  - Test with 9 ⟶ no
  - Test with 10 ⟶ yes, insert

- Pop front of queue ⟶ 10
  - Test with 11 ⟶ yes
  - Test with 12 ⟶ yes

- 12 herausnehmen
  - Schnitt mit 4 → Ja
  - Schnitt mit 5 → Ja



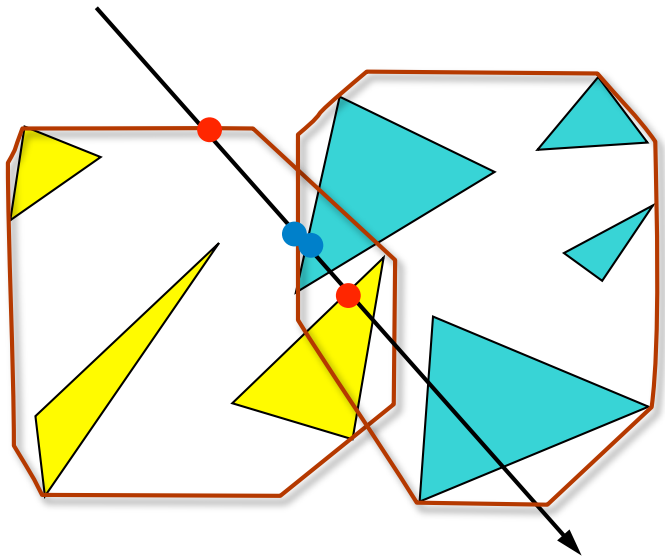
- 5 herausnehmen, Test mit Primitiv
- 6 herausnehmen, Test mit Primitiv

- 11 herausnehmen ...

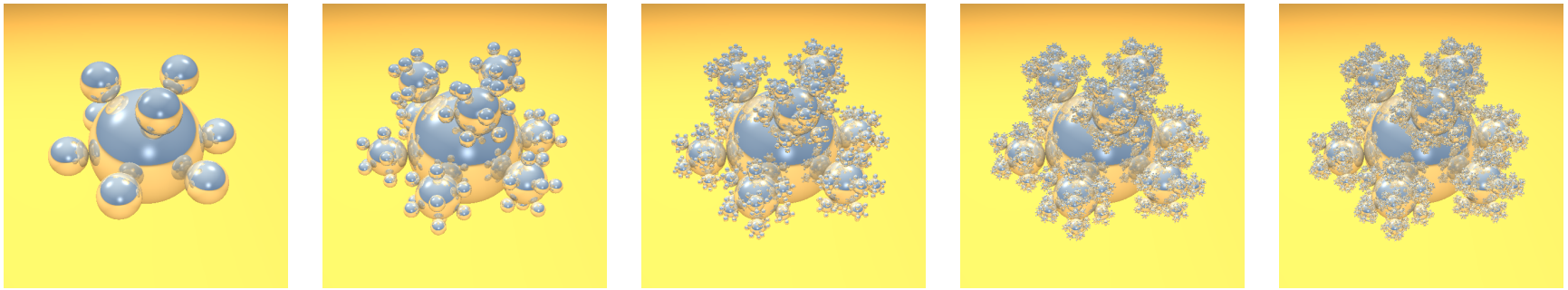
Remarks

- We don't need a complete ordering among the BV's in the priority queue, because in each step, we only need to extract the BV that has the *closest* intersection (among all others in the queue); hence the heap

- **Warning:** the closest ray-BV intersection and the closest ray-primitive intersection can occur in different BV's!



How Much Do We Gain?



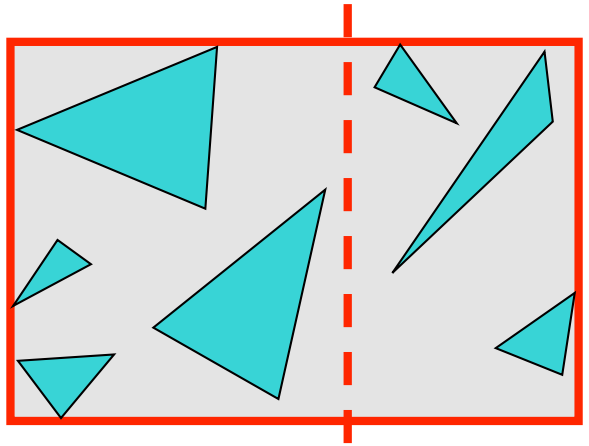
Number of spheres	10	91	820	7381	66430
Brute-force	2.5	11.4	115.0	2677.0	24891.0
Goldsmith/ Salmon BVH	2.3	2.8	4.1	5.5	7.4

Rendering times in seconds, Athlon XP 1900+
(Markus Geimer)

The Construction of BV Hierarchies

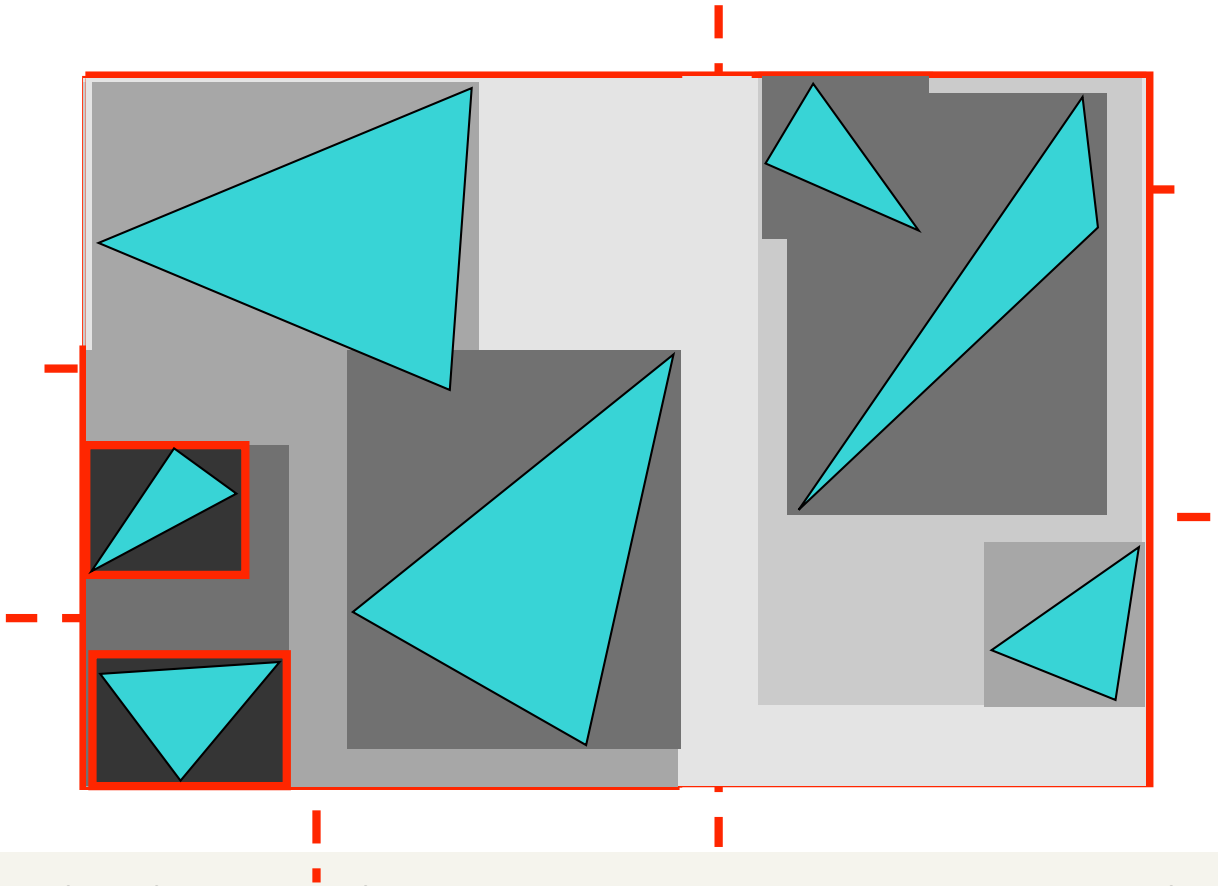
- There are many possible principles:

1. Given by modeling process (e.g., in form of scene graph)
2. Bottom-up:
   - Recursively combine objects/BV's and enclose in (larger) BV
   - Problem: how to choose the objects/BV's to be combined?
3. Top-down:
   - Partition the set of primitives recursively
   - Problem: how to partition the set?
4. Iterative Insert:
   - Heuristic developed by Goldsmith/Salmon



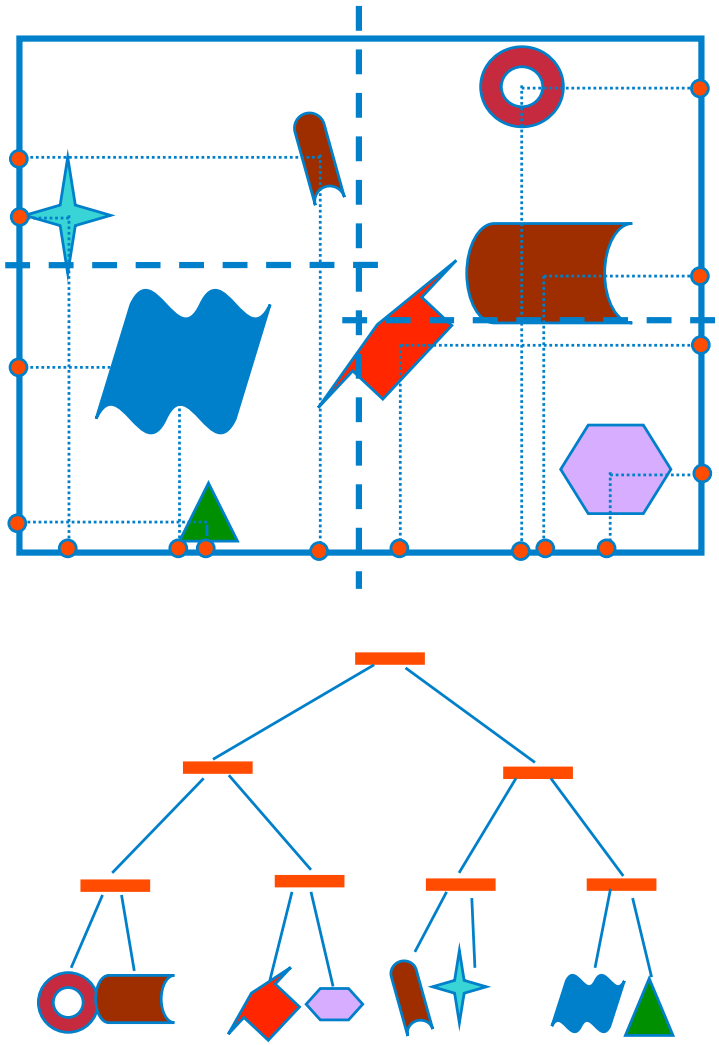
Example for the Top-Down Construction of a BVH

- Enclose each object (= primitives) by an elementary BV (e.g., AABB)
- In the following, work only with those elementary BVs
- Partition the set of objects in two sub-sets
- Recurse



Simplest Heuristic for Top-Down Construction: Median Cut

1. Construct elementary BVs around all objects
2. Sort all objects according to their "center" along the x-axis
3. Partition the scene along the *median* on the x-axis; assign half of the objects to the left and the right sub-tree, resp.
   1. Variant: cyclically choose a different axis on each level
   2. Variant: choose the axis with the longest extent
4. Repeat 1-3 recursively
   - Terminate, when a node contains less than *n* objects



A Better BVH Construction Method

- Given a set of polygons, what is their optimal partitioning? (optimal with respect to raytracing performance)

- Employ the Surface-Area-Heuristic (SAH): partition B such that

$$C(B) = \text{Area}(B_1) \cdot N(B_1) + \text{Area}(B_2) \cdot N(B_2)$$

  attains its minimum

- Optimum could be achieved by exhaustive search:

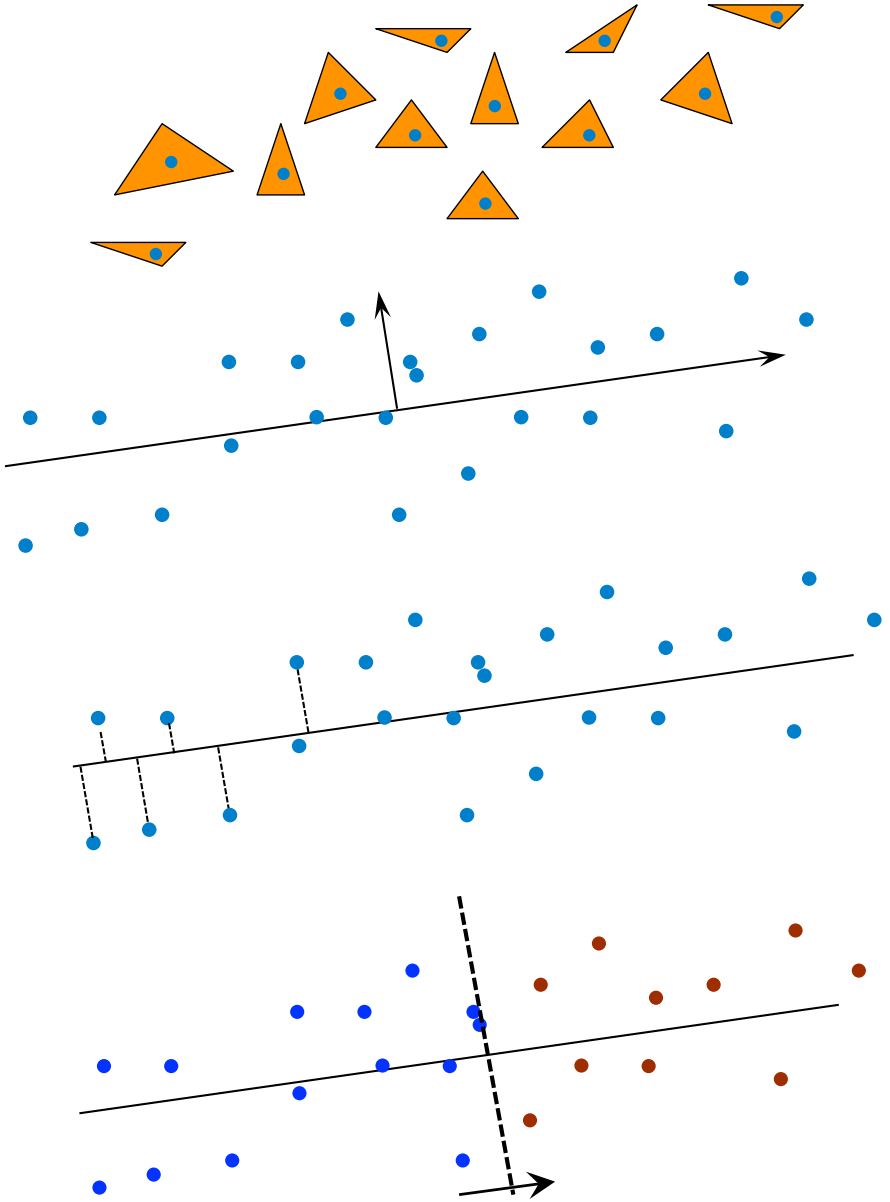
$$C(B) = \min_{B' \in \mathcal{P}(B)} C(B', B \backslash B')$$

  - Not practical

- Current "best" way: use method similar to kd-tree construction

Heuristic Method to Achieve Good BVHs

1. Represent all polygons by their midpoint
2. Calculate axis of largest extent (using PCA)
3. Project all midpoints onto that axis and sort
4. Search minimum of $C(B)$ by plane sweep



- Running time:

$$T(n) = T(\alpha n) + T((1-\alpha)n) + O(n \log n)$$

$$\in O(n \log^2 n)$$

where α is the proportion of polygons that end up in the "left" child BV, and assuming α is bounded (e.g., between 0.1 and 0.9)

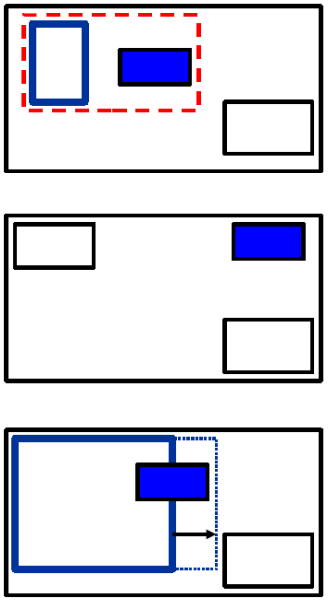
- Remarks:
  - Stopping criteria are the same as for the kd-tree
  - Top-down methods usually lead to better BVHs than iterative ones

Iterative Insert

[Goldsmith & Salmon, 1987]

Optional

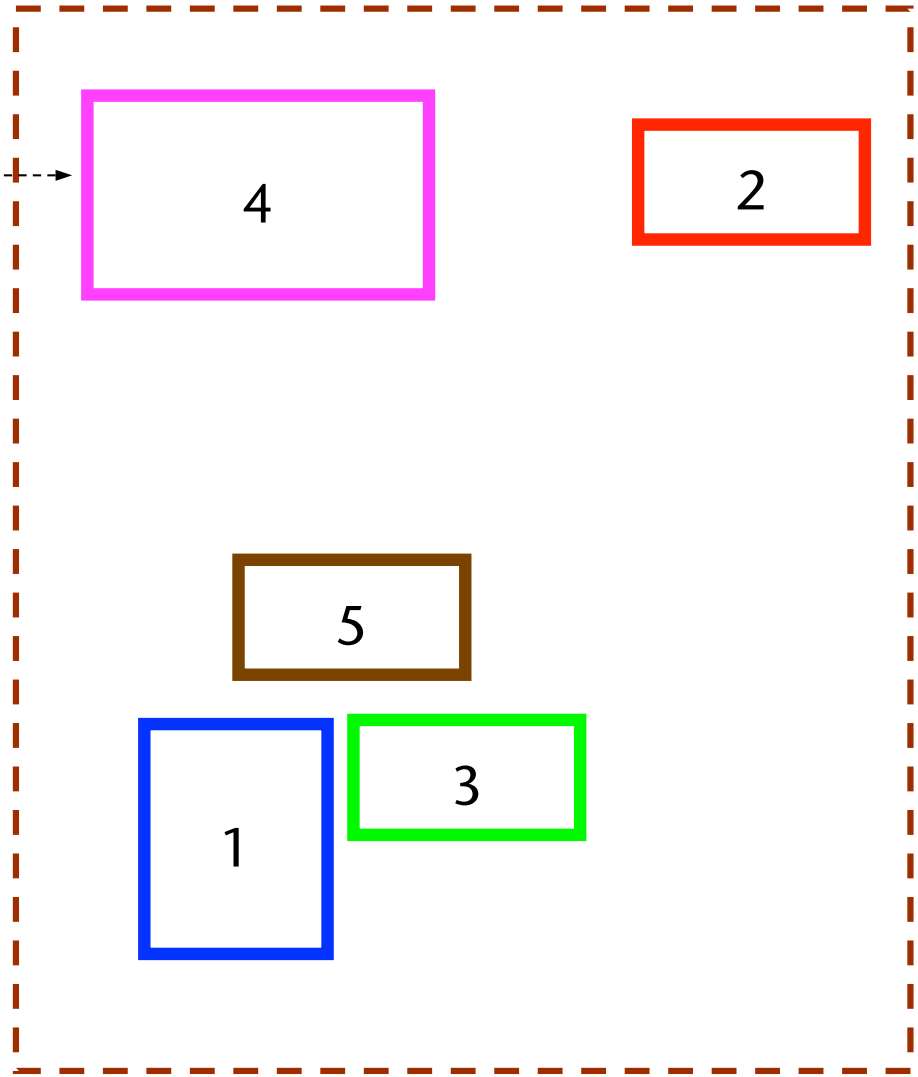
- Start with an empty root node
- Iteratively insert one triangle after another into the BVH, possibly thereby extending the BVH:
  - Let the triangle "sift" to the bottom of the BVH
    - Vergrößere dabei ggf. das BV der Knoten
  - Ist das Dreieck an einem Blatt angekommen →
    - Ersetze das Blatt durch einen inneren Knoten
    - füge das neue und das alte Dreieck als dessen Kinder an
  - Steht man an einem inneren Knoten → treffe eine der folgenden Entscheidungen:
    - füge das Dreieck am aktuellen (inneren) Knoten an
    - lasse das Dreieck in den linken / rechten Teilbaum sickern



Beispiel für Goldsmith und Salmon

Optional

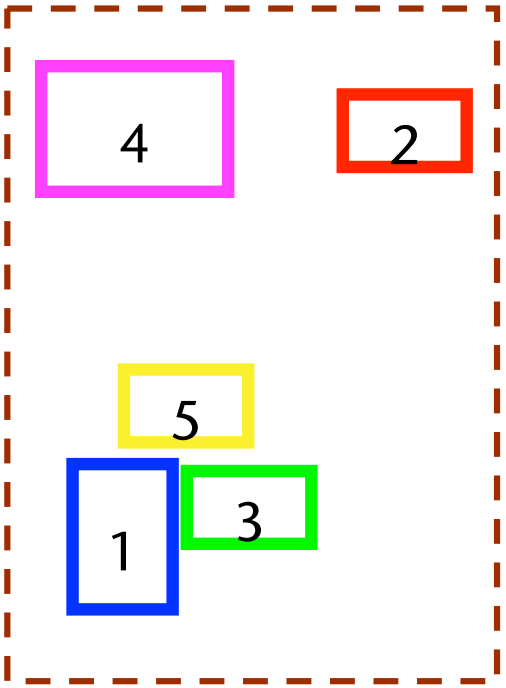
- Szene vor der Erzeugung der Hierarchie
- Jedes Objekt wird durch sein Bounding Volume umgeben
- Das gestrichelte Viereck ist die gesamte Szene



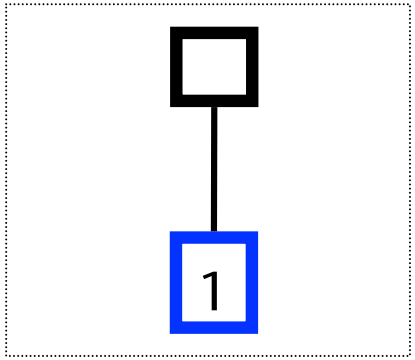
1. Iteration

Optional

Gegenwärtiger Baum

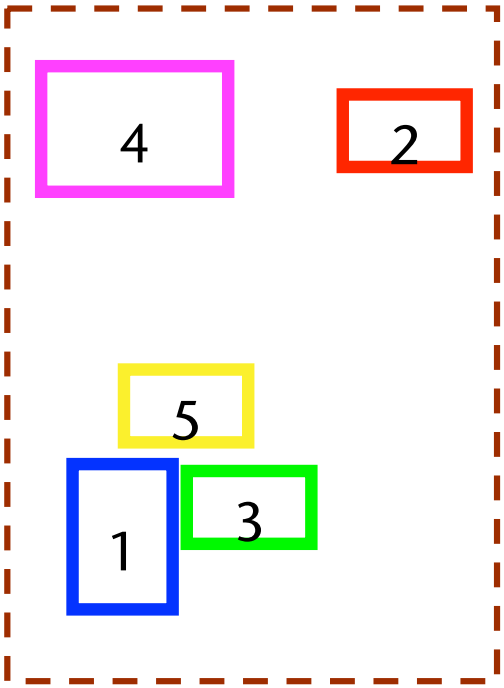


Möglichkeiten

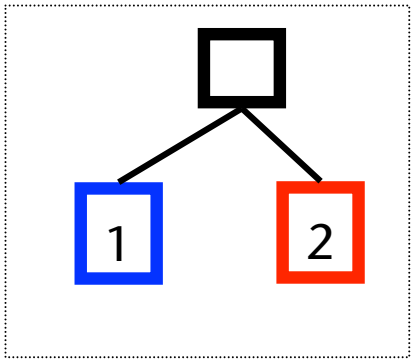


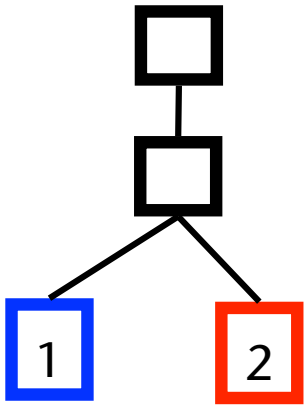
2. Iteration

Gegenwärtiger Baum



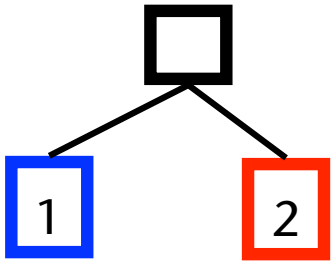
Möglichkeiten

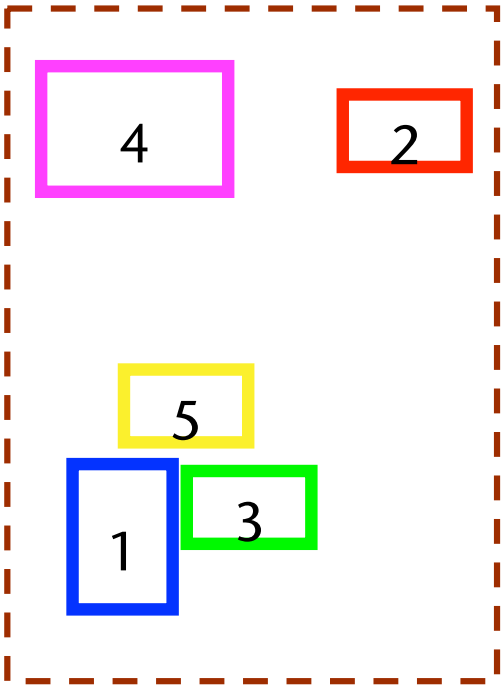




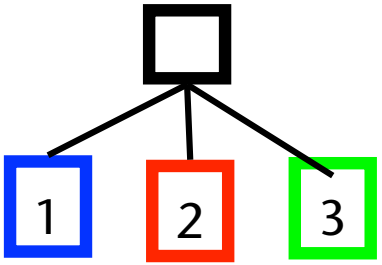
3. Iteration

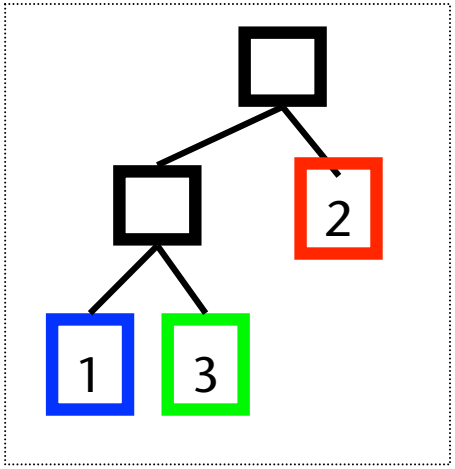
Gegenwärtiger Baum

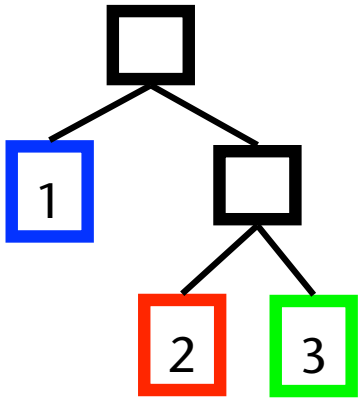




Möglichkeiten

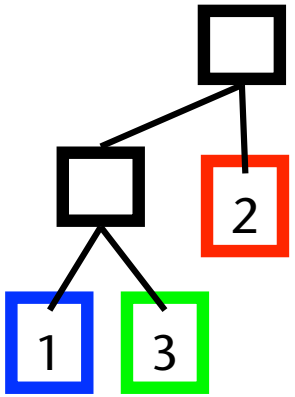


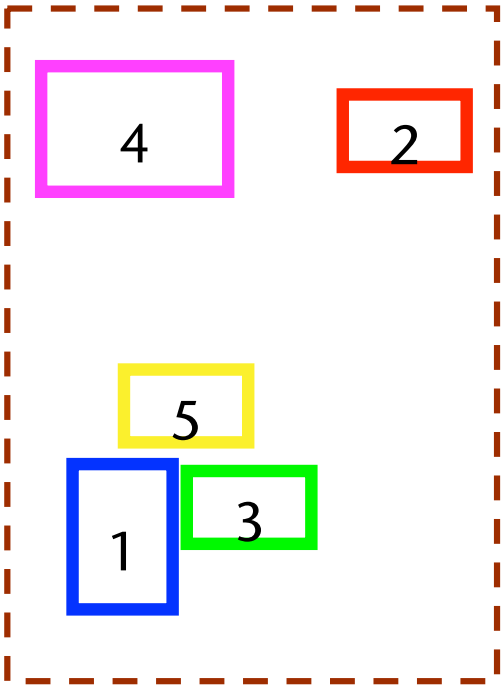




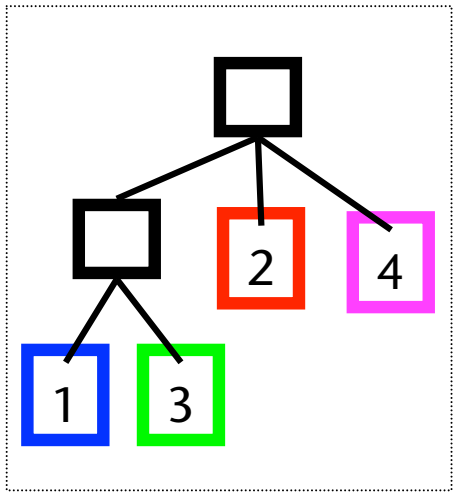
4. Iteration

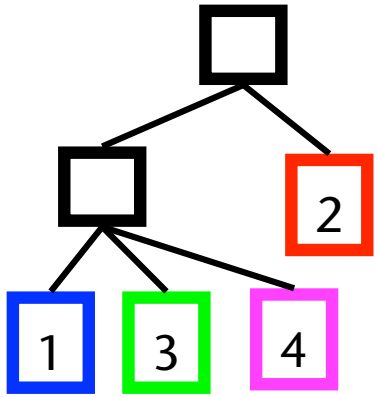
Gegenwärtiger Baum

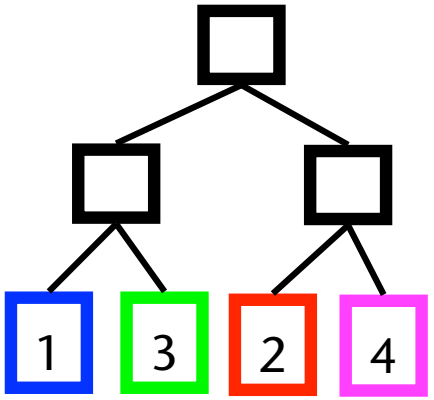




Möglichkeiten

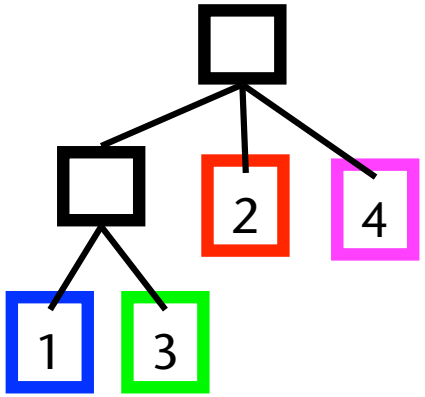


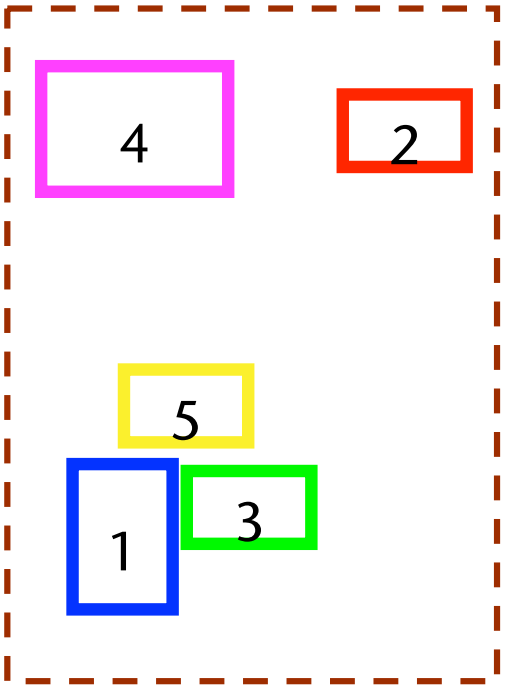




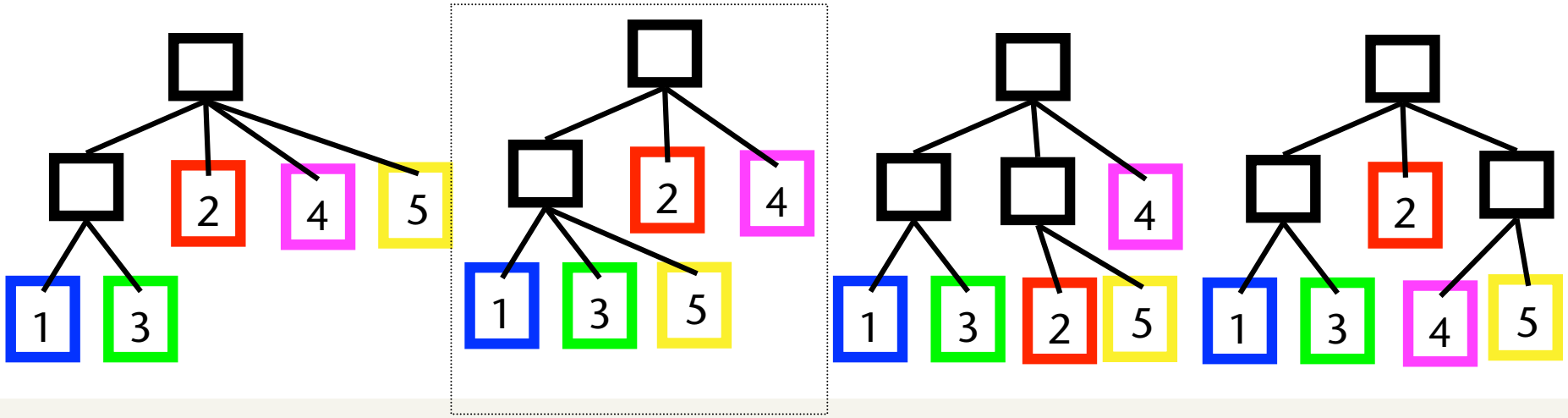
5. Iteration

Gegenwärtiger Baum



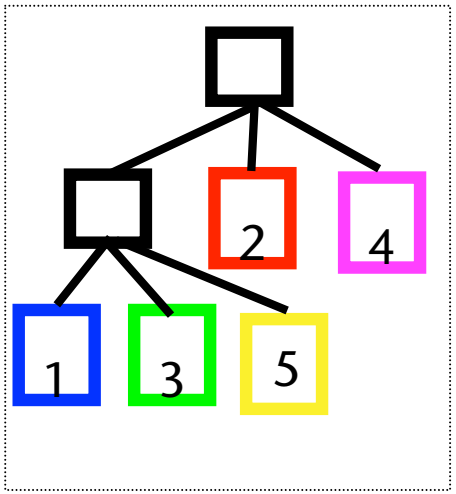


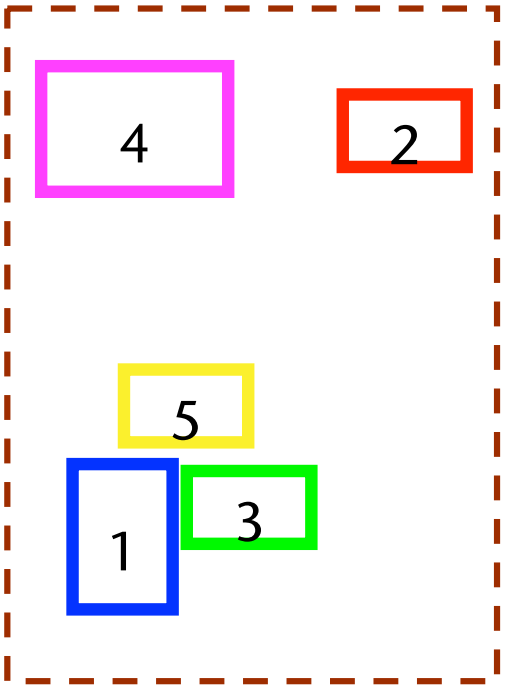
Möglichkeiten



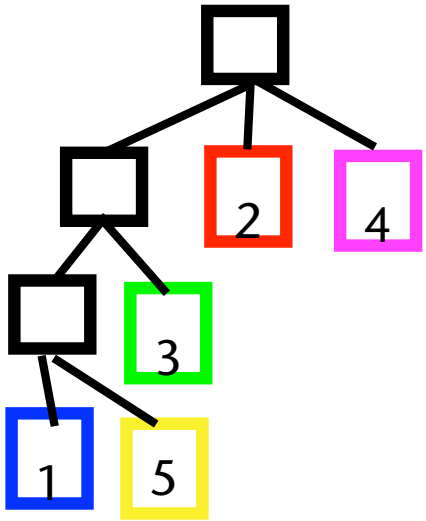
5. Iteration

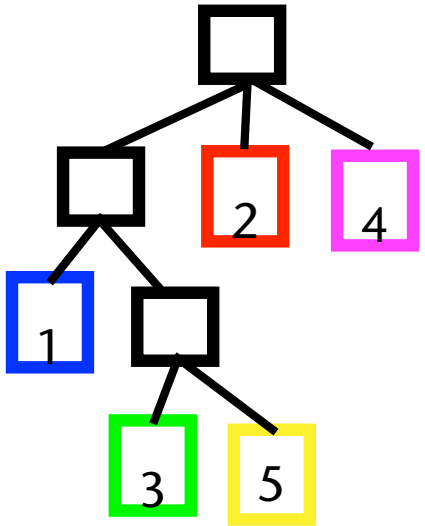
Gegenwärtiger Baum





Möglichkeiten





Bemerkungen

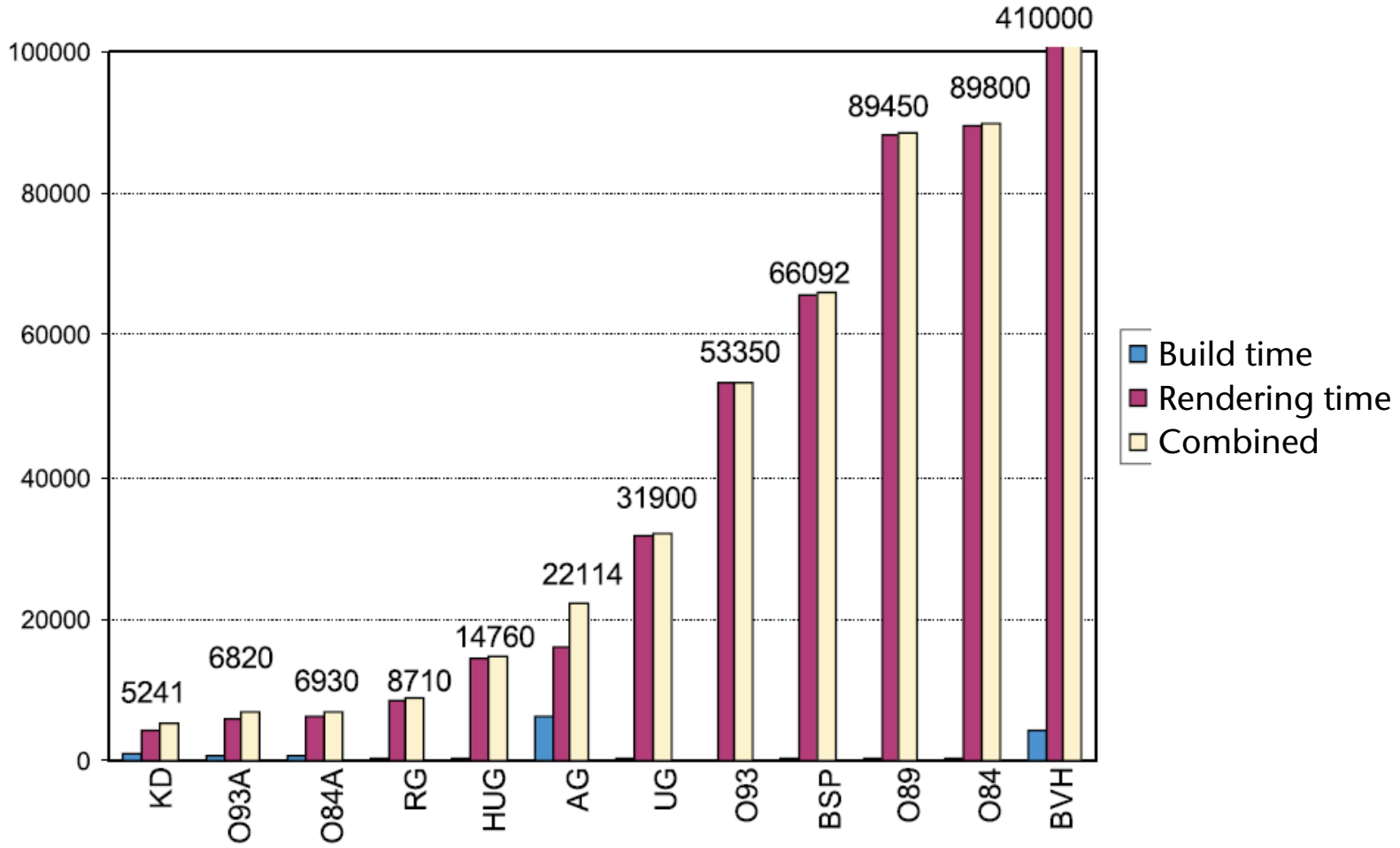
Optional

- Die Reihenfolge, in der die Objekte eingefügt werden, hat einen sehr großen Einfluss darauf, wie gut der Baum wird
- Goldsmith/Salmon experimentierten mit:
  - Reihenfolge wie im geladenen Modell
  - zufällig (shuffled)
  - Sortiert entlang einer Koordinatenachse

Zahl der Schnitt-Berechnungen pro Strahl bei verschiedenen Testszenen

User Supplied	5.94	19.9	12.9	10.1	32.0	63.2
Sorted	6.53	20.0	15.9	13.3	32.0	55.2
Average Shuffled	6.21	19.9	14.3	9.4	40.5	44.8
Best Shuffled	5.94	19.9	12.4	8.7	36.7	42.4
Worst Shuffled	6.32	19.9	17.4	18.3	48.2	47.2

Comparison of Acceleration Data Strucures [Havran, 2001(?)]



- Applies only to raytracing
- Is not definitive, since new techniques are emerging all the time