



WIKIPEDIA
The Free Encyclopedia

[Main page](#)
[Contents](#)
[Featured content](#)
[Current events](#)
[Random article](#)
[Donate to Wikipedia](#)
[Wikipedia store](#)

Interaction

[Help](#)
[About Wikipedia](#)
[Community portal](#)
[Recent changes](#)
[Contact page](#)


Tools

[What links here](#)
[Related changes](#)
[Upload file](#)
[Special pages](#)
[Permanent link](#)
[Page information](#)
[Wikidata item](#)
[Cite this page](#)

Print/export

[Create a book](#)
[Download as PDF](#)
[Printable version](#)


Languages

 [Add links](#)

[Create account](#) [Log in](#)

Article [Talk](#)

[Read](#) [Edit](#) [View history](#)



Bounding interval hierarchy

From Wikipedia, the free encyclopedia

A **bounding interval hierarchy (BIH)** is a partitioning [data structure](#) similar to that of [bounding volume hierarchies](#) or [kd-trees](#). Bounding interval hierarchies can be used in high performance (or real-time) [ray tracing](#) and may be especially useful for dynamic scenes.

The BIH was first presented under the name of SKD-Trees,^[1] presented by Ooi et al., and BoxTrees,^[2] independently invented by Zachmann.

Contents [hide]

- [1 Overview](#)
- [2 Operations](#)
 - [2.1 Construction](#)
 - [2.2 Ray traversal](#)
- [3 Properties](#)
 - [3.1 Numerical stability](#)
- [4 Extensions](#)
- [5 See also](#)
- [6 References](#)
 - [6.1 Papers](#)
- [7 External links](#)

Overview [\[edit\]](#)

Bounding interval hierarchies (BIH) exhibit many of the properties of both [bounding volume hierarchies](#) (BVH) and [kd-trees](#). Whereas the construction and storage of BIH is comparable to that of BVH, the traversal of BIH resemble that of [kd-trees](#). Furthermore, BIH are also [binary trees](#) just like kd-trees (and in fact their superset, [BSP trees](#)). Finally, BIH are axis-aligned as are its ancestors. Although a more general non-axis-aligned implementation of the BIH should be possible (similar to the BSP-tree, which uses unaligned planes), it would almost certainly be less desirable due to decreased numerical stability and an increase in the complexity of ray traversal.

The key feature of the BIH is the storage of 2 planes per node (as opposed to 1 for the kd tree and 6 for an axis aligned [bounding box](#) hierarchy), which allows for overlapping children (just like a BVH), but at the same time featuring an order on the children along one dimension/axis (as it is the case for kd trees).

It is also possible to just use the BIH data structure for the construction phase but traverse the tree in a way a traditional axis aligned bounding box hierarchy does. This enables some simple speed up optimizations for large ray bundles ^[3] while keeping [memory/cache](#) usage low.

Some general attributes of bounding interval hierarchies (and techniques related to BIH) as described by ^[4] are:

- Very fast construction times
- Low memory footprint
- Simple and fast traversal
- Very simple construction and traversal algorithms
- High numerical precision during construction and traversal
- Flatter tree structure (decreased tree depth) compared to kd-trees

Operations [\[edit\]](#)

Construction [\[edit\]](#)

To construct any [space partitioning](#) structure some form of [heuristic](#) is commonly used. For this the [surface area heuristic](#), commonly used with many partitioning schemes, is a possible candidate. Another, more simplistic heuristic is the "global" heuristic described by ^[4] which only requires an [axis-aligned bounding box](#), rather than the full set of primitives, making it much more suitable for a fast construction.

The general construction scheme for a BIH:

- calculate the scene bounding box
- use a heuristic to choose one axis and a split plane candidate perpendicular to this axis
- sort the objects to the left or right child (exclusively) depending on the bounding box of the object (note that objects intersecting the split plane may either be sorted by its overlap with the child volumes or any other heuristic)
- calculate the maximum bounding value of all objects on the left and the minimum bounding value of those on the right for that axis (can be combined with previous step for some heuristics)
- store these 2 values along with 2 bits encoding the split axis in a new node
- continue with step 2 for the children

Potential heuristics for the split plane candidate search:

- Classical: pick the longest axis and the middle of the node bounding box on that axis
- Classical: pick the longest axis and a split plane through the median of the objects (results in a leftist tree which is often unfortunate for ray tracing though)
- Global heuristic: pick the split plane based on a global criterion, in the form of a regular grid (avoids unnecessary splits and keeps node volumes as cubic as possible)
- Surface area heuristic: calculate the surface area and amount of objects for both children, over the set of all possible split plane candidates, then choose the one with the lowest costs (claimed to be optimal, though the cost function poses unusual demands to proof the formula, which can not be fulfilled in real life. also an exceptionally slow heuristic to evaluate)

Ray traversal [\[edit\]](#)

The traversal phase closely resembles a kd-tree traversal: One has to distinguish 4 simple cases, where the ray

- just intersects the left child
- just intersects the right child
- intersects both children
- intersects neither child (the only case not possible in a kd traversal)

For the third case, depending on the ray direction (negative or positive) of the component (x, y or z) equalling the split axis of the current node, the traversal continues first with the left (positive direction) or the right (negative direction) child and the other one is pushed onto a [stack](#).

Traversal continues until a leaf node is found. After intersecting the objects in the leaf, the next element is popped from the stack. If the stack is empty, the nearest intersection of all pierced leaves is returned.

It is also possible to add a 5th traversal case, but which also requires a slightly complicated construction phase. By swapping the meanings of the left and right plane of a node, it is possible to cut off empty space on both sides of a node. This requires an additional bit that must be stored in the node to detect this special case during traversal. Handling this case during the traversal phase is simple, as the ray

- just intersects the only child of the current node or
- intersects nothing

Properties [\[edit\]](#)

Numerical stability [\[edit\]](#)

All operations during the hierarchy construction/sorting of the triangles are min/max-operations and comparisons. Thus no triangle clipping has to be done as it is the case with kd-trees and which can become a problem for triangles that just slightly intersect a node. Even if the kd implementation is carefully written, numerical errors can result in a non-detected intersection and thus rendering errors (holes in the geometry) due to the missed ray-object intersection.

Extensions [\[edit\]](#)

Instead of using two planes per node to separate geometry, it is also possible to use any number of planes to create a n-ary BIH or use multiple planes in a standard binary BIH (one and four planes per node were already proposed in [\[4\]](#) and then properly evaluated in [\[5\]](#)) to achieve better object separation.






See also [\[edit\]](#)

- [Axis-aligned bounding box](#)

- [Bounding volume hierarchy](#)
- [kd-tree](#)

References [\[edit\]](#)

Papers [\[edit\]](#)

1. [^] Nam, Beomseok; Sussman, Alan. [A comparative study of spatial indexing techniques for multidimensional scientific datasets](#) 
2. [^] Zachmann, Gabriel. [Minimal Hierarchical Collision Detection](#) 
3. [^] Wald, Ingo; Boulos, Solomon; Shirley, Peter (2007). [Ray Tracing Deformable Scenes using Dynamic Bounding Volume Hierarchies](#) 
4. [^] ^{[a](#)} ^{[b](#)} ^{[c](#)} Wächter, Carsten; Keller, Alexander (2006). [Instant Ray Tracing: The Bounding Interval Hierarchy](#) 
5. [^] Wächter, Carsten (2008). [Quasi-Monte Carlo Light Transport Simulation by Efficient Ray Tracing](#) 

External links [\[edit\]](#)

- BIH implementations: [Javascript](#) 

| |
|--|
| Categories: Geometric data structures 3D computer graphics Trees (data structures) |
|--|

This page was last modified on 16 March 2015, at 15:56.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#). Wikipedia® is a registered trademark of the [Wikimedia Foundation, Inc.](#), a non-profit organization.

[Privacy policy](#) [About Wikipedia](#) [Disclaimers](#) [Contact Wikipedia](#) [Developers](#) [Mobile view](#)

