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

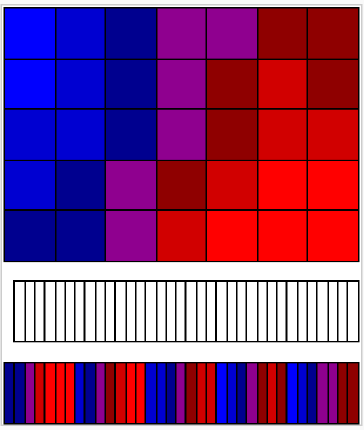
Print/export
Create a book
Download as PDF
Printable version

Languages
Српски / srpski
Edit links

Implicit *k*-d tree

From Wikipedia, the free encyclopedia
(Redirected from Implicit kd-tree)

An **implicit *k*-d tree** is a *k*-d tree defined implicitly above a rectilinear grid. Its split planes' positions and orientations are not given explicitly but implicitly by some recursive splitting-function defined on the hyperrectangles belonging to the tree's nodes. Each inner node's split plane is positioned on a grid plane of the underlying grid, partitioning the node's grid into two subgrids.


Construction and storage of a 2D implicit max *k*d-tree using the grid median splitting-function. Each cell of the rectilinear grid has one scalar value from low (bright blue) to high (bright red) assigned to it. The grid's memory footprint is indicated in the lower line. The implicit max *k*d-tree's predefined memory footprint needs one scalar value less than that. The storing of the node's max values is indicated in the upper line.

Contents [hide]

Nomenclature and references [edit]

The terms "min/max *k*-d tree" and "implicit *k*-d tree" are sometimes mixed up. This is because the first publication using the term "implicit *k*-d tree" [1] did actually use explicit min/max *k*-d trees but referred to them as "implicit *k*-d trees" to indicate that they may be used to ray trace implicitly given iso surfaces. Nevertheless, this publication used also slim *k*-d trees which are a subset of the implicit *k*-d trees with the restriction that they can only be built over integer hyperrectangles with sidelengths that are powers of two. Implicit *k*-d trees as defined here have recently been introduced, with applications in computer graphics.[2][3] As it is possible to assign attributes to implicit *k*-d tree nodes, one may refer to an implicit *k*-d tree which has min/max values assigned to its nodes as an "implicit min/max *k*-d tree".

Construction [edit]

Implicit *k*-d trees are in general not constructed explicitly. When accessing a node, its split plane orientation and position are evaluated using the specific splitting-function defining the tree. Different splitting-functions may result in different trees for the same underlying grid.

Splitting-functions [edit]

Splitting-functions may be adapted to special purposes. Underneath two specifications of special splitting-function classes.

- **Non-degenerated splitting-functions** do not allow the creation of degenerated nodes (nodes whose corresponding integer hyperrectangle's volume is equal zero). Their corresponding implicit *k*-d trees are full binary trees, which have for *n* leaf nodes *n - 1* inner nodes. Their corresponding implicit *k*-d trees are **non-degenerated implicit *k*-d trees**.
- **complete splitting-functions** are non-degenerated splitting-functions whose corresponding implicit *k*-d tree's leaf nodes are single grid cells such that they have one inner node less than the amount of gridcells given in the grid. The corresponding implicit *k*-d trees are **complete implicit *k*-d trees**.

A complete splitting function is for example the **grid median splitting-function**. It creates fairly balanced implicit *k*-d trees by using *k*-dimensional integer hyperrectangles *hyprec[2][k]* belonging to each node of the implicit *k*-d tree. The hyperrectangles define which gridcells of the rectilinear grid belong to their corresponding node. If the volume of this hyperrectangle equals one, the corresponding node is a single grid cell and is therefore not further subdivided and marked as leaf node. Otherwise the hyperrectangle's longest extend is chosen as orientation *o*. The corresponding split plane *p* is positioned onto the grid plane that is closest to the hyperrectangle's grid median along that orientation.

Split plane orientation *o*:

```
o = min{argmax(i = 1 ... k: (hyprec[1][i] - hyprec[0][i]))}
```

Split plane position *p*:

```
p = roundDown((hyprec[0][o] + hyprec[1][o]) / 2)
```

Assigning attributes to implicit *k*-d tree nodes [edit]

An obvious advantage of implicit *k*-d trees is that their split plane's orientations and positions need not to be stored explicitly.

But some applications require besides the split plane's orientations and positions further attributes at the inner tree nodes. These attributes may be for example single bits or single scalar values, defining if the subgrids belonging to the nodes are of interest or not. For complete implicit *k*-d trees it is possible to pre-allocate a correctly sized array of attributes and to assign each inner node of the tree to a unique element in that allocated array.

The amount of gridcells in the grid is equal the volume of the integer hyperrectangle belonging to the grid. As a complete implicit *k*-d tree has one inner node less than grid cells, it is known in advance how many attributes need to be stored. The relation "*Volume of integer hyperrectangle to inner nodes*" defines together with the complete splitting-function a recursive formula assigning to each split plane a unique element in the allocated array. The corresponding algorithm is given in C-pseudo code underneath.

```
// Assigning attributes to inner nodes of a complete implicit k-d tree

// create an integer help hyperrectangle hyprec (its volume vol(hyprec) is equal the
amount of leaves)
int hyprec[2][k] = { { 0, ..., 0 }, { length_1, ..., length_k } };
// allocate once the array of attributes for the entire implicit k-d tree
attr *a = new attr[volume(hyprec) - 1];

attr implicitKdTreeAttributes(int hyprec[2][k], attr *a)
{
 if(vol(hyprec) > 1) // the current node is an inner node
 {
   // evaluate the split plane's orientation o and its position p using the
underlying complete split-function
   int o, p;
   completeSplittingFunction(hyprec, &o, &p);
   // evaluate the children's integer hyperrectangles hyprec_l and hyprec_r
   int hyprec_l[2][k], hyprec_r[2][k];
   hyprec_l       = hyprec;
   hyprec_l[1][o] = p;
   hyprec_r       = hyprec;
   hyprec_r[0][o] = p;
   // evaluate the children's memory location a_l and a_r
   attr* a_l = a + 1;
   attr* a_r = a + vol(hyprec_l);
   // evaluate recursively the children's attributes c_l and c_r
   attr c_l = implicitKdTreeAttributes(hyprec_l, a_l);
   attr c_r = implicitKdTreeAttributes(hyprec_r, a_r);
   // merge the children's attributes to the current attribute c
   attr c = merge(c_l, c_r);
   // store the current attribute and return it
   a[0] = c;
   return c;
 }
 // The current node is a leaf node. Return the attribute belonging to the
corresponding gridcell
 return attribute(hyprec);
}
```

It is worth mentioning that this algorithm works for all rectilinear grids. The corresponding integer hyperrectangle does not necessarily have to have sidelengths that are powers of two.

Applications [edit]

Implicit max-*k*-d trees are used for ray casting isosurfaces/MIP (maximum intensity projection). The attribute assigned to each inner node is the maximal scalar value given in the subgrid belonging to the node. Nodes are not traversed if their scalar values are smaller than the searched iso-value/current maximum intensity along the ray. The low storage requirements of the implicit max *k*d-tree and the favorable visualization complexity of ray casting allow to ray cast (and even change the isosurface for) very large scalar fields at interactive framerates on commodity PCs. Similarly an implicit min/max kd-tree may be used to efficiently evaluate queries such as terrain line of sight.[4]

Complexity [edit]

Given an implicit *k*-d tree spanned over an *k*-dimensional grid with *n* gridcells.

- Assigning attributes to the nodes of the tree takes $\mathrm{O}(kn)$ *time.*
- Storing attributes to the nodes takes $\mathrm{O}(n)$ memory.
- Ray casting iso-surfaces/MIP an underlying scalar field using the corresponding implicit max *k*-d tree takes roughly $\mathrm{O}(\log(n))$ time.

See also [edit]

- *k*-d tree
- min/max *k*-d tree

References [edit]

1. ^ Ingo Wald, Heiko Friedrich, Gerd Marmitt, Philipp Slusallek and Hans-Peter Seidel "Faster Isosurface Ray Tracing using Implicit KD-Trees" IEEE Transactions on Visualization and Computer Graphics (2005)
2. ^ Matthias Groß, Carsten Lojewski, Martin Bertram and Hans Hagen "Fast Implicit *k*-d Trees: Accelerated Isosurface Ray Tracing and Maximum Intensity Projection for Large Scalar Fields" CGIM07: Proceedings of Computer Graphics and Imaging (2007) 67-74
3. ^ Matthias Groß (PhD, 2009) Towards Scientific Applications for Interactive Ray Casting
4. ^ Bernardt Duvenhage "Using An Implicit Min/Max KD-Tree for Doing Efficient Terrain Line of Sight Calculations" in "Proceedings of the 6th International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa", 2009.

v · t · e	Tree data structures [hide]
Search trees (dynamic sets/associative arrays)	2–3 · 2–3–4 · AA · (a,b) · AVL · B · B+ · B* · B^x · (Optimal) Binary search · Dancing · HTree · Interval · Order statistic · (Left-leaning) Red-black · Scapegoat · Splay · T · Treap · UB · Weight-balanced
Heaps	Binary · Binomial · Fibonacci · Leftist · Pairing · Skew · Van Emde Boas
Tries	Hash · Radix · Suffix · Ternary search · X-fast · Y-fast
Spatial data partitioning trees	BK · BSP · Cartesian · Hilbert R · *k*-d (**implicit *k*-d**) · M · Metric · MVP · Octree · Priority R · Quad · R · R+ · R* · Segment · VP · X
Other trees	Cover · Exponential · Fenwick · Finger · Fusion · Hash calendar · iDistance · K-ary · Left-child right-sibling · Link/cut · Log-structured merge · Merkle · PQ · Range · SPQR · Top

Categories: Computer graphics data structures | Trees (data structures)