MAIT 627 Fast Multipole Methods

Lecture 6

Outline

- Data Structures and FMM
- Hierarchical Space Subdivision
  - Binary Trees, Quad-trees, Oct-trees;
  - k-d trees;
  - 2^d-trees. Definitions.
- Hierarchical Numbering (Indexing)
- Spatial Ordering
  - Scaling.
  - Binary ordering.
  - Ordering in d-dimensions.
  - Bit interleaving and deinterleaving
  - Algorithms for finding the box index, box center, parents, children, and neighbors.
  - Examples.

Data Structures and FMM

- Spatial grouping techniques are employed in various versions of the FMM (SL_FMM, RML_FMM, AML_FMM, etc.)
- All major components of the FMM are interrelated: truncated factorization, error bounds, translation operations, and spatial grouping.

Data Structures and FMM (2)

- Since the complexity of FMM should not exceed $O(N^2)$ (at $M \sim N$), data organization should be provided for efficient numbering, search, and operations with these data.
- Some naive approaches can utilize search algorithms that result in $O(N^2)$ complexity of the FMM (and so they kill the idea of the FMM).
- In d-dimensions $O(N \log N)$ complexity for operations with data can be achieved.
- Some caution is needed while utilizing standard (library) routines on sets. If using such routines always check the complexity of the algorithm!
- Understanding of complexity of each FMM procedure is crucial. Normally it is worth to develop your own library for operations with FMM data (using standard routines as their complexity is satisfactory).

Data Structures and FMM (3)

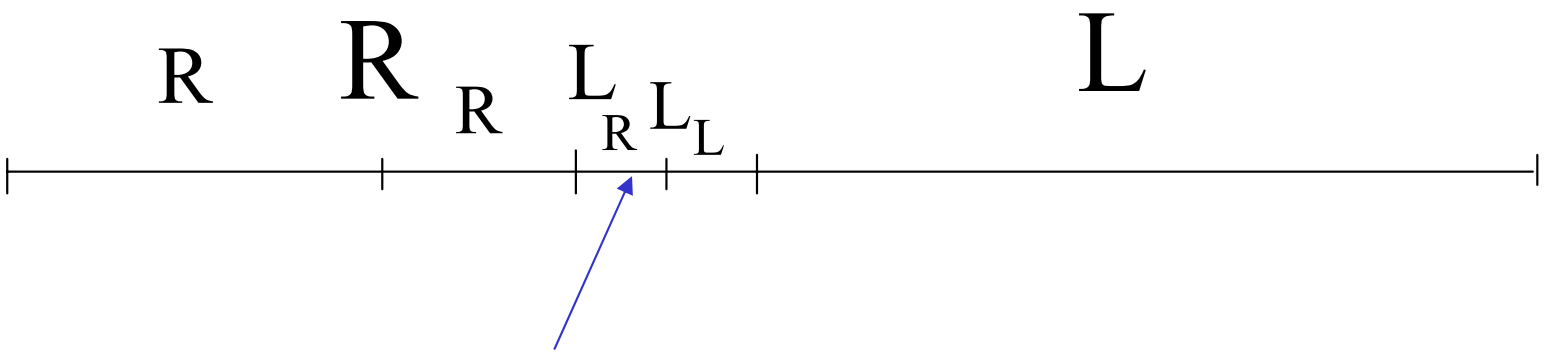
- Approaches include:
  - Data preprocessing
    - Sorting
    - Building lists (such as neighbor lists): requires memory, potentially can be avoided;
    - Building and storage of trees: requires memory, potentially can be avoided;
  - Operations with data during the FMM algorithm
    - Operations on data sets;
    - Search procedures.
- Preferable algorithms:
  - Avoid unnecessary memory usage;
  - Use fast (constant and logarithmic) search procedures;
  - Employ bitwise operations;
  - Can be parallelized.

Hierarchical Space Subdivision

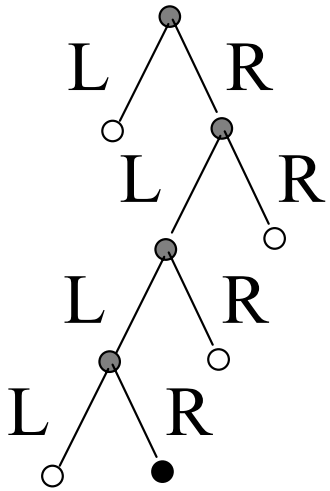
Historically:

- Binary trees (1D), Quad-trees (2D), Oct-trees (3D);
- For dimensions larger than 3 (sometimes for d=2 and 3 also) k-d trees.
- We will consider a concept of 2^d-tree:
  - d=1 – binary;
  - d=2 – quadtree;
  - d=3 – octtree;
  - d=4 – hexatree;
  - and so on..

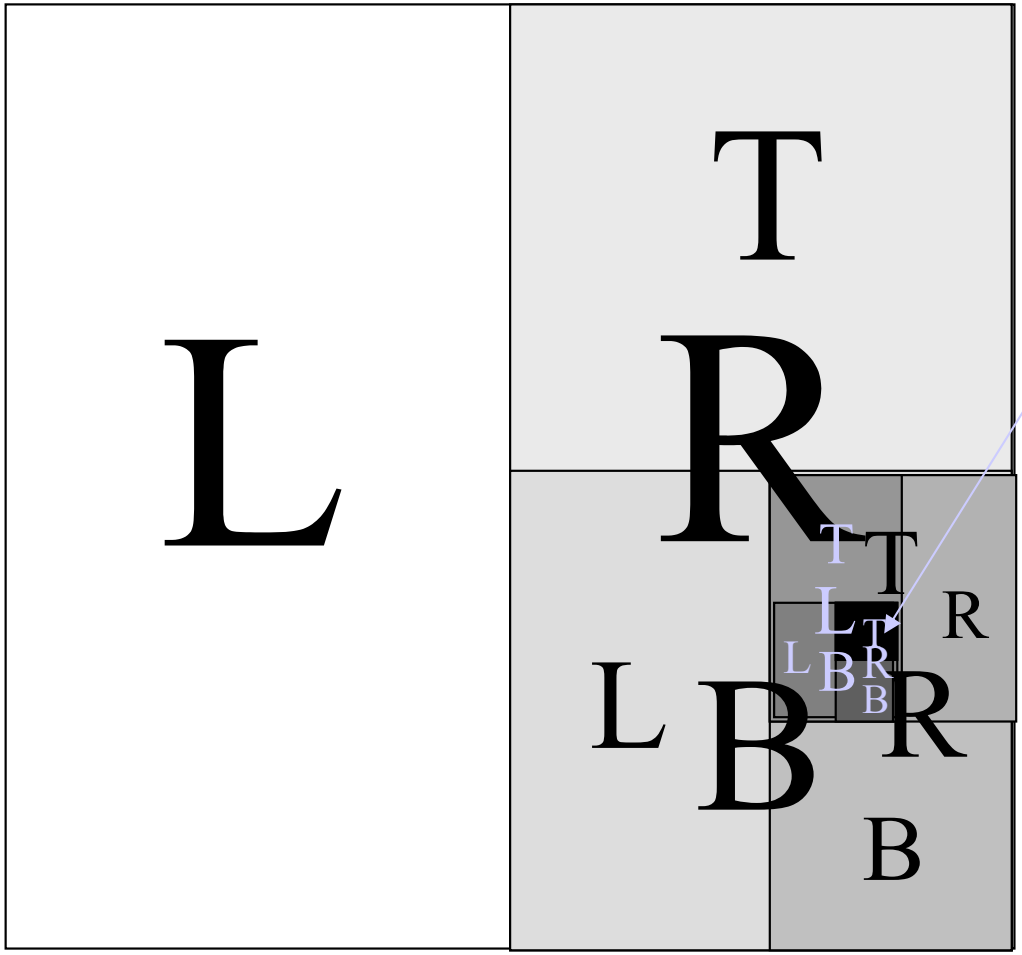
Binary Trees



Target segment coordinates:
RLLR

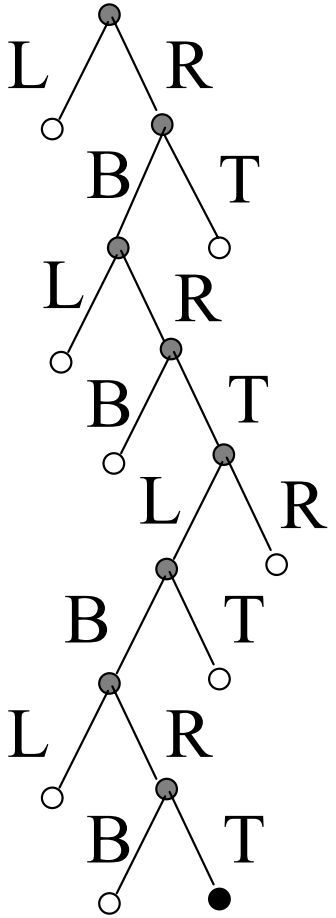


What are k-d trees?



Target box coordinates:

RBRTLBRT

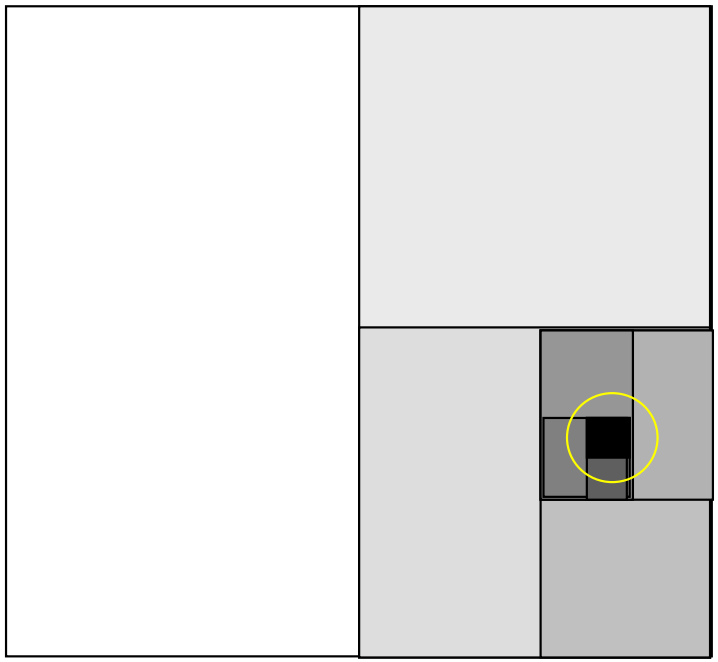


Like a binary tree!

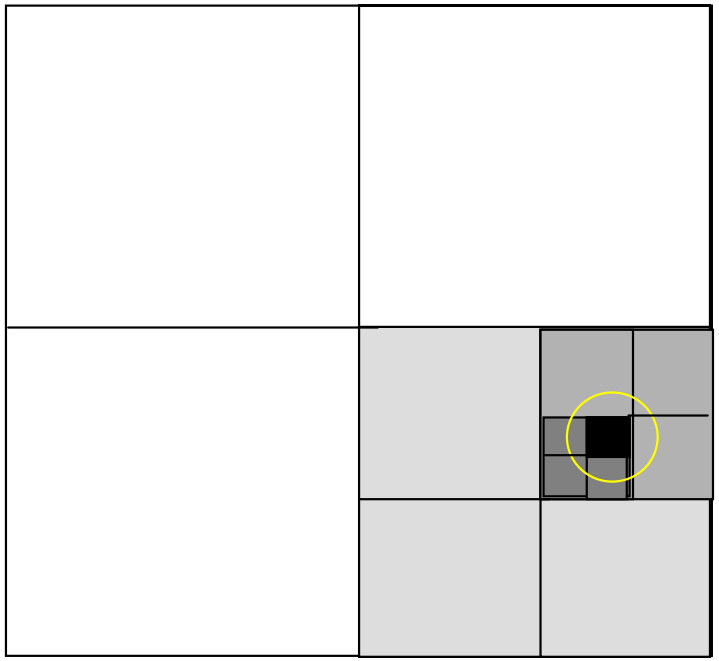
Why 2^d-trees?

It is possible to use k-d trees in the FMM
(while in this course we mainly focus on 2^d-trees)

k-d tree



2^d-tree

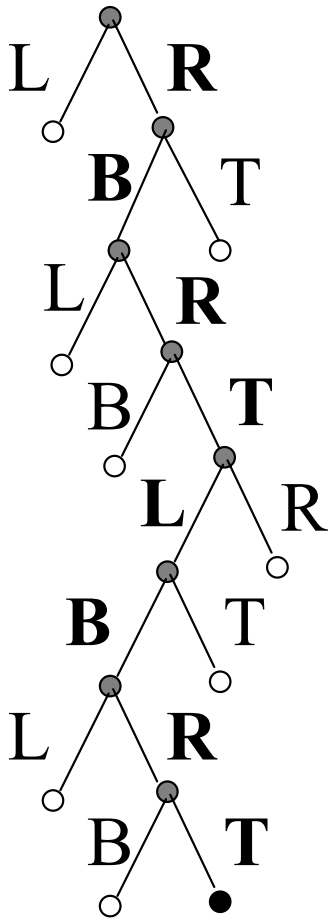


Simply, because
It is convenient in the FMM to enclose a cube into a sphere with easy determination of the neighborhood of similar boxes.

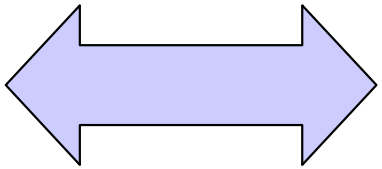
It is easy to convert k-d tree into 2^d-tree and back (for cubic boxes)

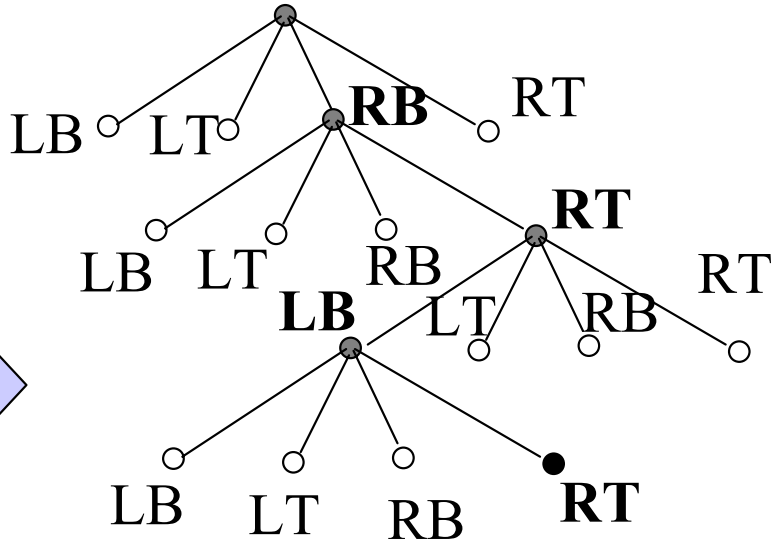
Target box coordinates:

RBRTLBRT $\rightarrow$(RB)(RT)(LB)(RT)



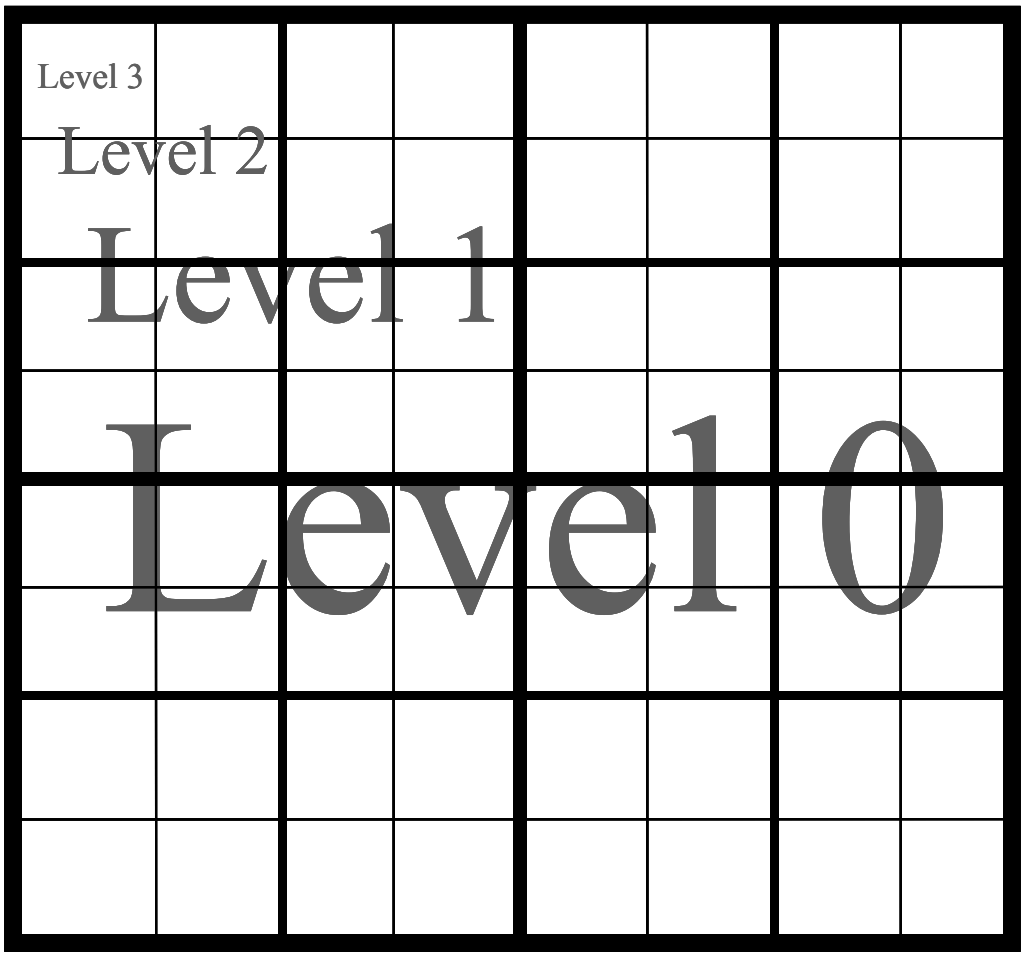
k-2 tree

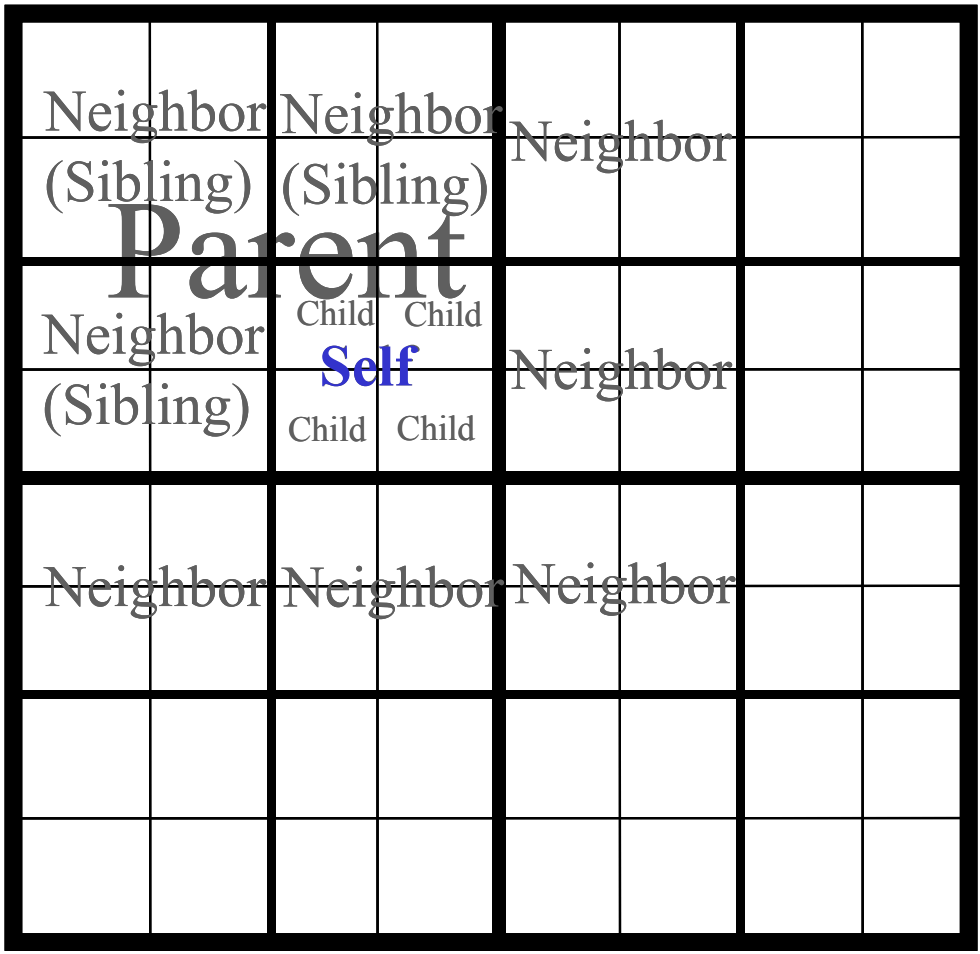




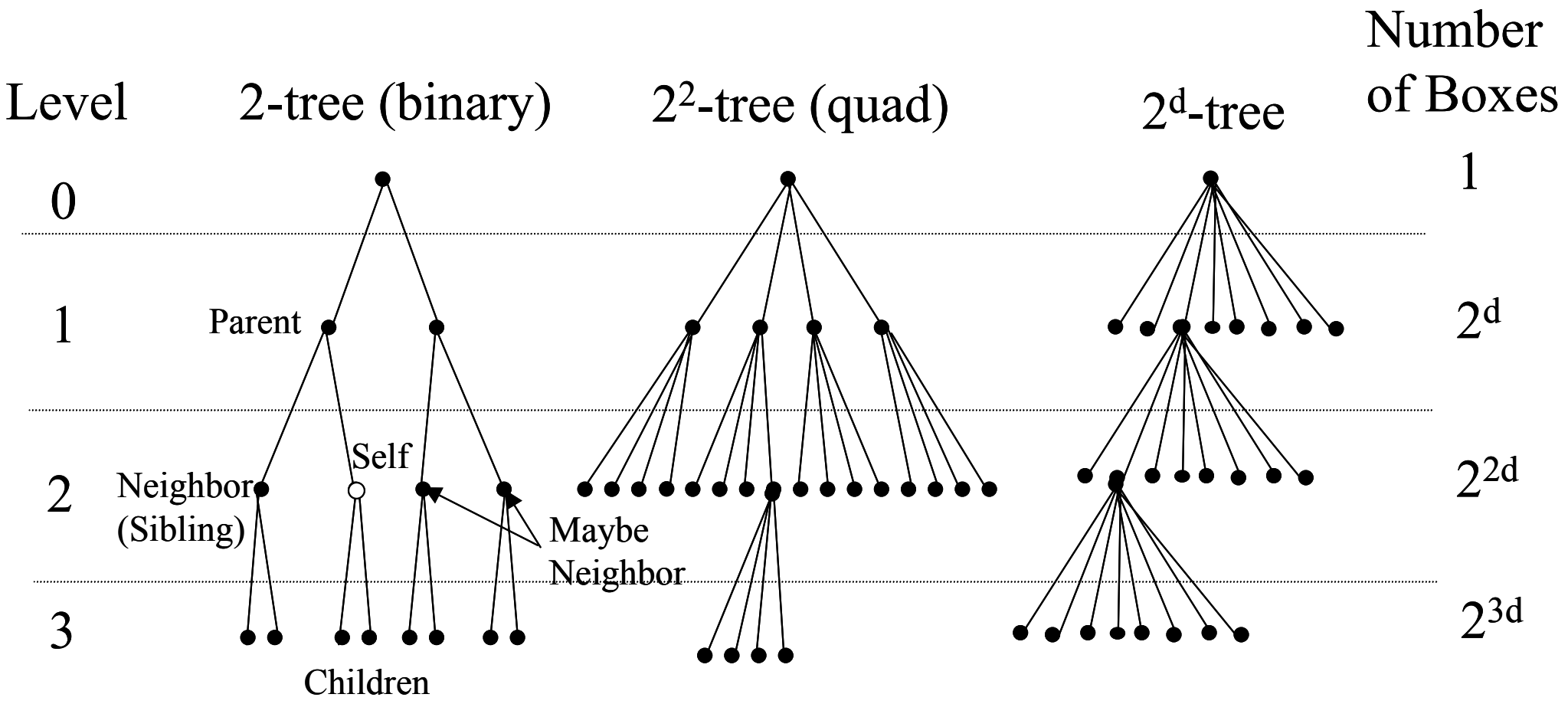
quad-tree = 2^2-tree

Hierarchy in 2^d-tree

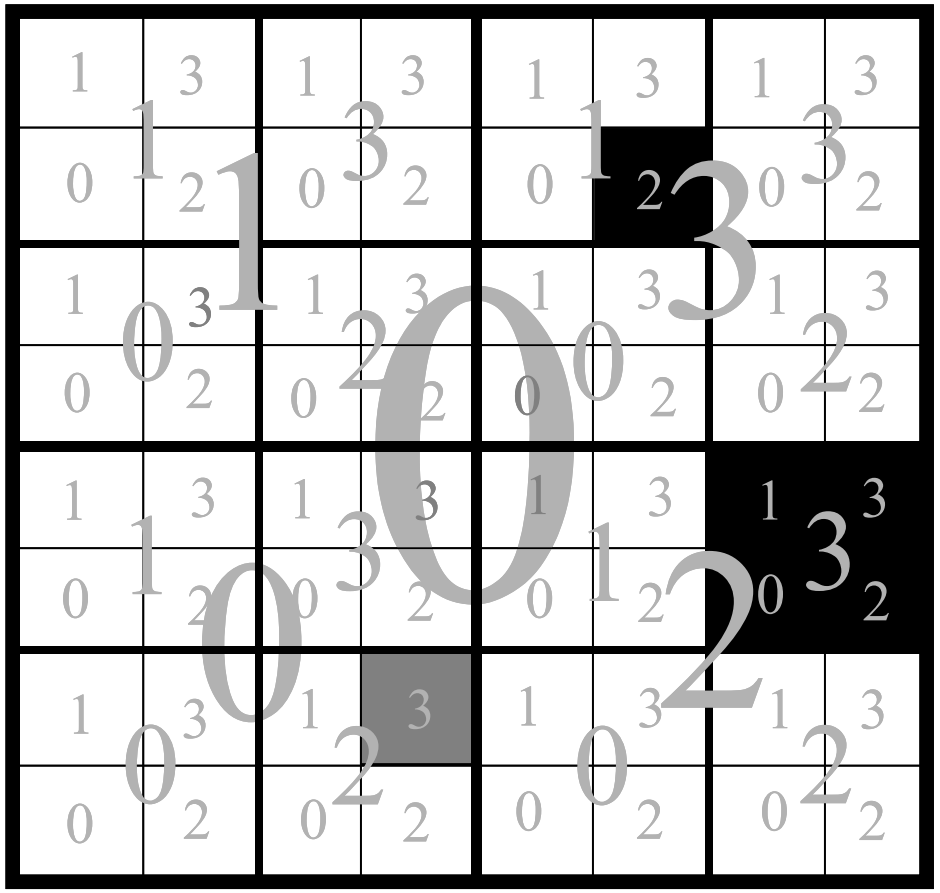




2^d-trees


Level
2-tree (binary)
2^2-tree (quad)
2^d-tree
Number of Boxes
0
1
2
3
1
2^d
2^{2d}
2^{3d}
Parent
Self
Neighbor (Sibling)
Maybe Neighbor
Children

Hierarchical Numbering (Indexing) in 2^d-trees. Numbering (Indexing) String.



Numbering in quad-tree

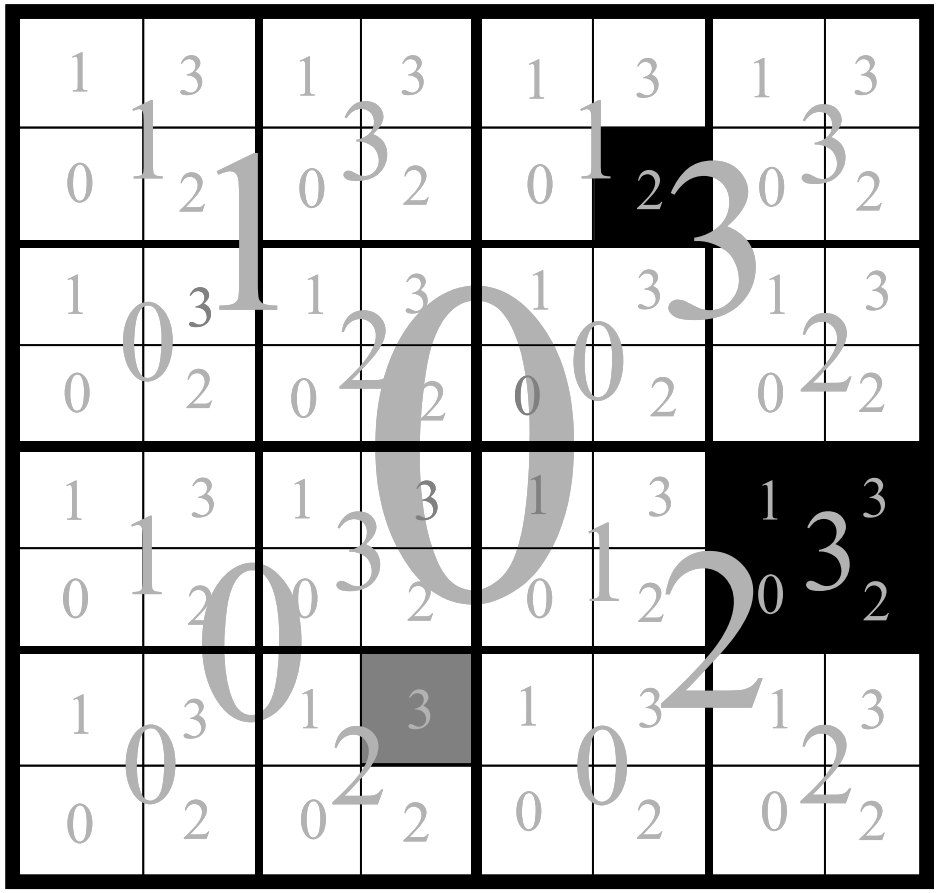
The large black box has the numbering string (2,3);

The small black box has the numbering string (3,1,2);

$$Number = (N_1, N_2, \ldots, N_l), \quad N_j = 0, \ldots, 2^d - 1, \quad j = 1, \ldots, l,$$

Numbering string

Hierarchical Numbering in 2^d-trees. Number at the Level.



Numbering in quad-tree

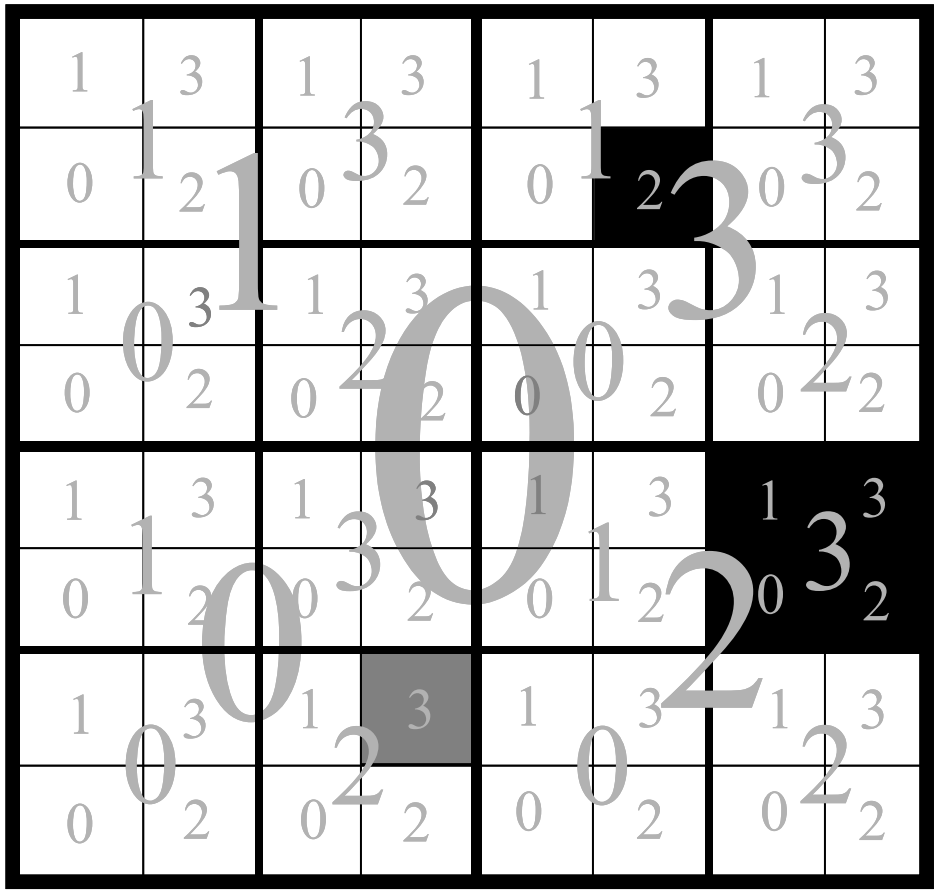
The large black box has the numbering string (2,3). So its number is $23_4=11_{10}$.

The small black box has the numbering string (3,1,2). So its number is $312_4=54_{10}$.

In general: Number at level l is:

$$Number = (2^d)^{l-1} \bullet N_1 + (2^d)^{l-2} \bullet N_2 + \ldots + 2^d \bullet N_{l-1} + N_l.$$

Hierarchical Numbering in 2^d-trees. Universal Number.



Numbering in quad-tree

The large black box has the numbering string (2,3). So its number is $23_4=11_{10}$ at level 2

The small gray box has the numbering string (0,2,3). So its number is $23_4=11_{10}$ at level 3.

In general: Universal number is a pair:

$$UniversalNumber = (Number, l)$$

This number at this level

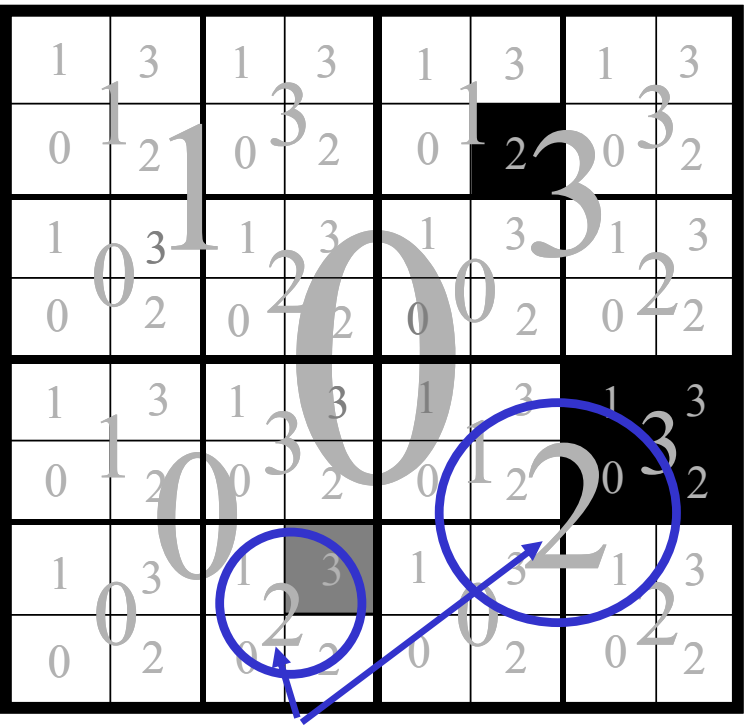
Parent Number

Parent numbering string:

$$Parent(N_1, N_2, \ldots, N_{l-1}, N_l) = (N_1, N_2, \ldots, N_{l-1}).$$

Parent number:

$$Parent(Number) = (2^d)^{l-2} \cdot N_1 + (2^d)^{l-3} \cdot N_2 + \ldots + N_{l-1}.$$



Parent number does not depend on the level of the box! E.g. in the quad-tree at any level

$$Parent(11_{10}) = Parent(23_4) = 2_4 = 2_{10}.$$

Parent's universal number:

$$Parent((Number, l)) = (Parent(Number), l-1).$$

Algorithm to find the parent number:

$$Parent(Number) = \lceil Number/2^d \rceil$$

For box #23_4 (gray or black) the parent box number is 2_4.

Children Numbers

Children numbering strings:

$$Children(N_1,N_2,\ldots,N_{l-1},N_l) = \{(N_1,N_2,\ldots,N_{l-1},N_l,N_{l+1})\}, \quad N_{l+1} = 0,\ldots,2^d-1.$$

Children numbers:

$$Children(Number) = \left\{(2^d)^l \bullet N_1 + (2^d)^{l-1} \bullet N_2 + \ldots + (2^d) \bullet N_l + N_{l+1}\right\}, \quad N_{l+1} = 0,\ldots,2^d-1.$$

Children numbers do not depend on the level of the box! E.g. in the quad-tree at any level:

$$Children(11_{10}) = Children(23_4) = \{230_4,231_4,232_4,233_4\} = \{44_{10},45_{10},46_{10},47_{10}\}$$

Children universal numbers:

$$Children((Number,l)) = (Children(Number),l+1).$$

Algorithm to find the children numbers:

$$Children(Number) = \{2^d \bullet Number + j\}, \quad j = 0,\ldots,2^d-1,$$

A couple of examples:

Problem: Using the above numbering system and decimal numbers find parent box number for box #5981 in oct-tree.

Solution: Find the integer part of division of this number by 8. [5981/8] = 747.

Answer: #747.

Problem: Using the above numbering system and decimal numbers find children box numbers for box #100 in oct-tree.

Solution: Multiply this number by 8 and add numbers from 0 to 7.

Answer: ##800, 801, 802, 803, 804, 805, 806, 807.

Can it be even faster?

YES!
USE BITSHIFT PROCEDURES!

(HINT: Multiplication and division by 2^d
are equivalent to d-bit shift.)

Matlab Program for Parent Finding

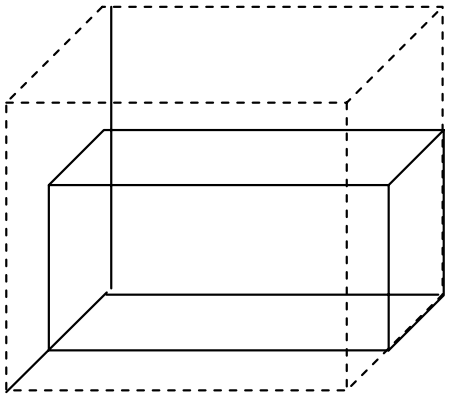
```
p = bitshift(n,-d);
```

Spatial Ordering

These algorithms of Parent and Children finding
are beautiful (O(1)), but how about neighbor finding?

Also we need to find box center coordinates…

The answer is SPATIAL ORDERING.



Scaling

In physics-based problems ($d = 1, 2, 3$) we usualy have symmetry of directions and can enclose that box to a cube of size $D \times \ldots \times D$, where

$$D = \max_d D_d,$$

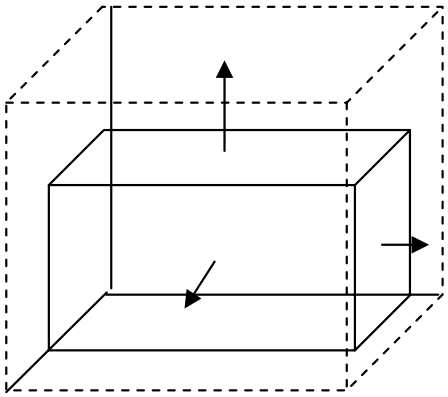
and the corner that has minimum values of Cartesian coordinates:

$$\mathbf{x}_{\min} = (x_{1,\min}, \ldots, x_{d,\min}).$$

This cube then can be mapped to the unit cube $[0, 1] \times \ldots \times [0, 1]$ by the shift of the origin and scaling:

$$\bar{\mathbf{x}} = \frac{\mathbf{x} - \mathbf{x}_{\min}}{D},$$

where $\mathbf{x}$ are true Cartesian coordinates of any point in the cube, and $\bar{\mathbf{x}}$ are normalized coordinates of such a point.



Scaling (2)

In case if a 2^d-tree data structure is applied for parametric studies, where each parameter has its own scale D_j the mapping of the original box

$$[x_{1,\min}, x_{1,\max}] \times \ldots \times [x_{d,\min}, x_{d,\max}], \quad x_{j,\max} - x_{j,\min} = D_j, \quad j = 1, \ldots, d$$

to the unit cube $[0,1] \times \ldots \times [0,1]$ can be also easily performed by scaling in each dimension as:

$$\bar{x}_j = \frac{x_j - x_{j,\min}}{D_j}, \quad j = 1, \ldots, d, \quad \bar{\mathbf{x}} = (\bar{x}_1, \ldots, \bar{x}_d).$$

Further we will work only with a unit cube, assuming that such scaling is performed and if necessary any point $\mathbf{x}$ in the original d-dimensional space can be found from given $\bar{\mathbf{x}} \in [0,1] \times \ldots \times [0,1]$ and back, for any $\mathbf{x}$ its image $\bar{\mathbf{x}}$ can be found.

When scaling like this, don't forget about deformation of the domains for your R and S expansions!

Binary Ordering (1)

$d = 1$:

All $\bar{x} \in [0,1]$ naturally ordered and can be represented in decimal system as

$$\bar{x} = (0.a_1a_2a_3...)_{10}, \quad a_j = 0,...,9; \quad j = 1,2,...$$

Note that the point $\bar{x} = 1$ can be written not only $\bar{x} = 1.0000...$, but also as

$$\bar{x} = 1 = (0.999999....)_{10}$$

We also can represent any point $\bar{x} \in [0,1]$ in binary system as

$$\bar{x} = (0.b_1b_2b_3...)_2, \quad b_j = 0,1; \quad j = 1,2,...$$

By the same reasons as for decimal system the point $\bar{x} = 1$ can be written as

$$\bar{x} = 1 = (0.111111....)_2.$$

Binary Ordering (2)
Finding the number of the box containing a given point

Level	Box Size (dec)	Box Size (bin)
0	1	1
1	0.5	0.1
2	0.25	0.01
3	0.125	0.001
...	...	...

Level 1:

$$(0.0b_1b_2b_3\ldots)_2 \in Box((0)), \quad (0.1b_1b_2b_3\ldots)_2 \in Box((1)), \quad \forall b_j = 0,1; \quad j = 1,2,\ldots,$$

Level 2:

$$(0.00b_1b_2b_3\ldots)_2 \in Box((0,0)), \quad (0.01b_1b_2b_3\ldots)_2 \in Box((0,1)),$$
$$(0.10b_1b_2b_3\ldots)_2 \in Box((1,0)), \quad (0.11b_1b_2b_3\ldots)_2 \in Box((1,1)),$$
$$\forall b_j = 0,1; \quad j = 1,2,\ldots,$$

We use numbering strings !

Level l:

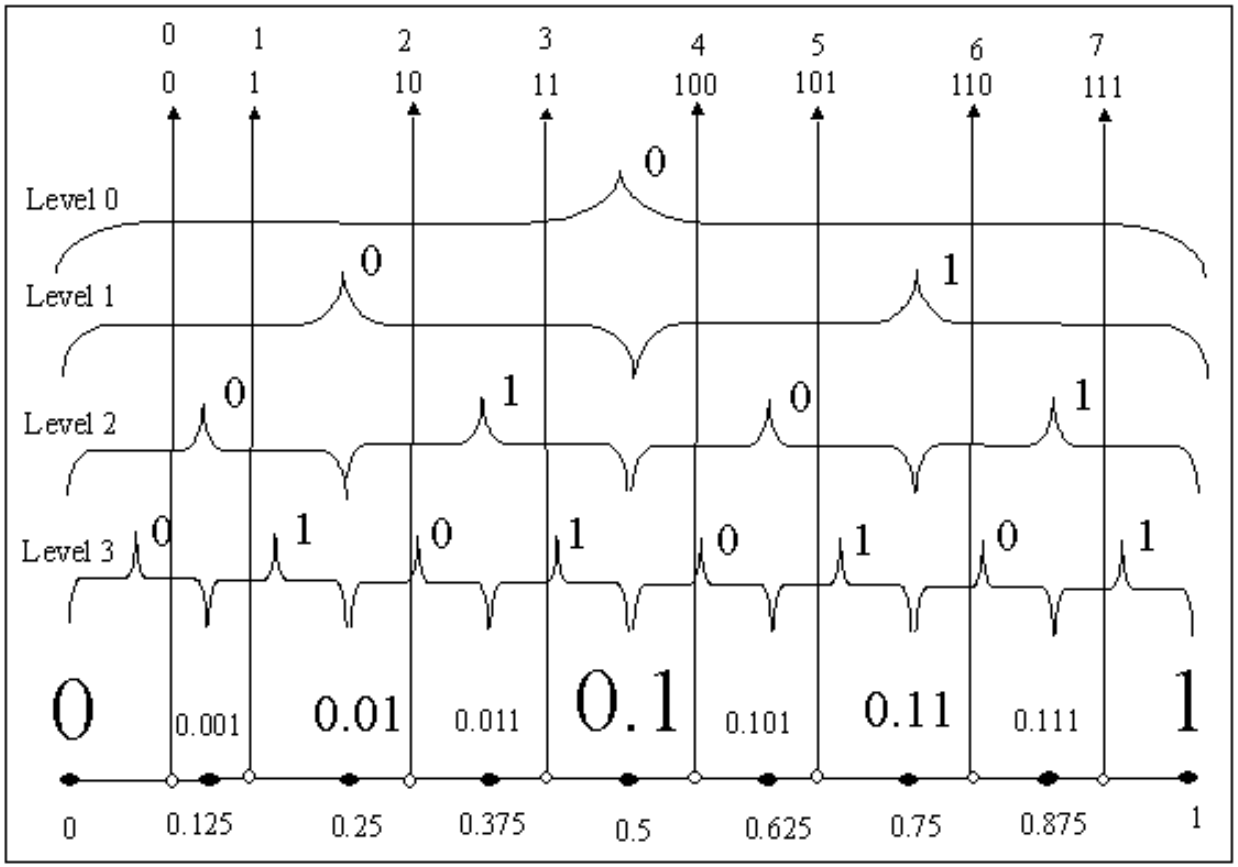
$$(0.N_1N_2\ldots N_lb_1b_2b_3\ldots)_2 \in Box((N_1,N_2,\ldots,N_l)), \quad \forall b_j = 0,1; \quad j = 1,2,\ldots$$

Binary Ordering (3)
Finding the number of the box containing a given point (2)

$$(0.N_1N_2\ldots N_lb_1b_2b_3\ldots)_2 \rightarrow (N_1N_2\ldots N_l.b_1b_2b_3\ldots)_2;\quad N_1N_2\ldots N_l = [(N_1N_2\ldots N_l.b_1b_2b_3\ldots)_2].$$

$$(Number, l) = \left[2^l \cdot \bar{x}\right].$$

This is an algorithm for finding of the box number at level l (!)



Binary Ordering (4)
Finding the center of a given box.

For box number *Number* at level *l* the left boundary can be found by *l* –bit shift:

$$Number = (N_1 N_2 ... N_l)_2 \rightarrow (0.N_1 N_2 ... N_l)_2,$$

Add 1 as an extra digit (half of the box size), so we have for the center of the box at level *l* :

$$\bar{x}_c(Number, l) = (0.N_1 N_2 ... N_l 1)_2.$$

This procedure also can be written in the form that does not depend on the counting system:

$$\bar{x}_c(Number, l) = 2^{-l} \cdot Number + 2^{-l-1} = 2^{-l} \cdot (Number + 2^{-1}).$$

since addition of one at position $l+1$ after the point in the binary system is the same as addition of 2^{-l-1}.

Problem: Find the center of box #31 (decimal) at level 5 of the binary tree.
Solution: We have $\bar{x}_c(31,5) = 2^{-5} \cdot (31 + 0.5) = 0.984375$.
Answer: 0.984375.

This is the algorithm!

Binary Ordering (5)
Neighbor finding

In the binary tree each box has 2 neighbors, except the boxes that have boundaries $\bar{x} = 0$ and $\bar{x} = 1$. The centers of the neighbor boxes:

$$\bar{x}_c(Neighbor((Number, l))) = \bar{x}_c(Number, l) \pm 2^{-l}.$$

The size of the box at level l

In the binary form:

$$\bar{x}_c(Neighbor((Number, l))) = (0.N_1N_2...N_l1)_2 \pm \left(0.\underbrace{0..01}_{l}\right)_2,$$

To find the number of the neighbor box we can then use the above algorithm for determining the number of the box for point $\bar{x}_c$:

$$Neighbor((Number, l)) = [N_1N_2...N_l.1 \pm 1] = N_1N_2...N_l \pm 1 = Number \pm 1.$$

If the neighbor number at level l equal 2^l or -1 we drop this box from the neighbor list.

Problem: Find all neighbors of box #31 (decimal) at level 5 of the binary tree.

Solution: The neighbors should have numbers $31 - 1 = 30$ and $31 + 1 = 32$. However, $32 = 2^5$, which exceeds the number allowed for this level. Thus, only box #30 is the neighbor.

Answer: #30.

This is the algorithm!

Ordering in *d*-dimensions (1). Bit Interleaving.

Coordinates of a point $\bar{\mathbf{x}} = (\bar{x}_1, ..., \bar{x}_d)$ in the *d*-dimensional unit cube can be represented in binary form

$$\bar{x}_k = (0.b_{k1}b_{k2}b_{k3}...)_2, \quad b_{kj} = 0,1; \quad j = 1,2,..., \quad k = 1,...,d.$$

Instead of having *d* numbers characterizing each point we can form a single binary number that represent the same point by ordered mixing of the digits in the above binary representation (this is also called *bit interleaving*), so we can write:

$$\bar{\mathbf{x}} = (0.b_{11}b_{21}...b_{d1}b_{12}b_{22}...b_{d2}...b_{1j}b_{2j}...b_{dj}...)_2.$$

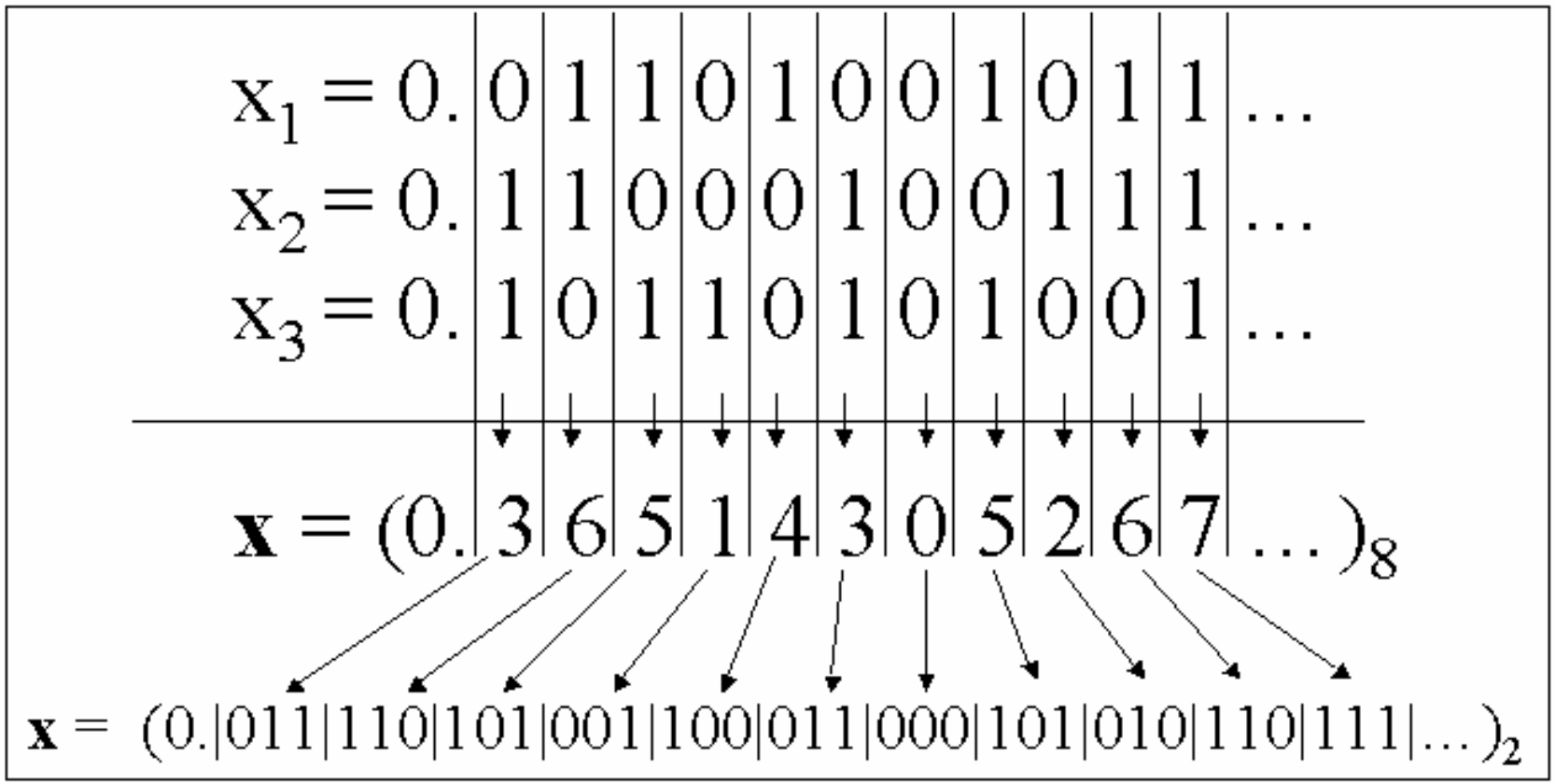
This number can be rewritten in the system with base 2^d:

$$\bar{\mathbf{x}} = (0.N_1N_2N_3...N_j...)_{2^d}, \quad N_j = (b_{1j}b_{2j}...b_{dj})_2, \quad j = 1,2,..., \quad N_j = 0,...,2^d - 1.$$

This maps $\mathbf{R}^d \rightarrow \mathbf{R}$, where coordinates are ordered naturally!

Ordering in *d*-dimensions (2). Bit Interleaving (2). Example.

Consider 3-dimensional space, and an oct-tree.

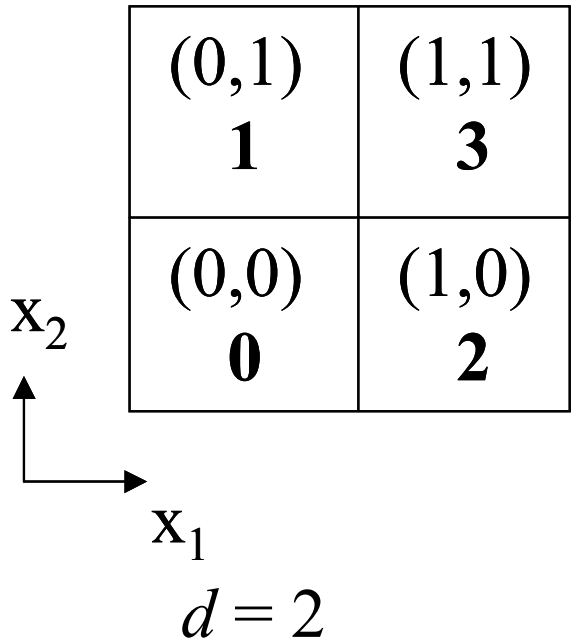


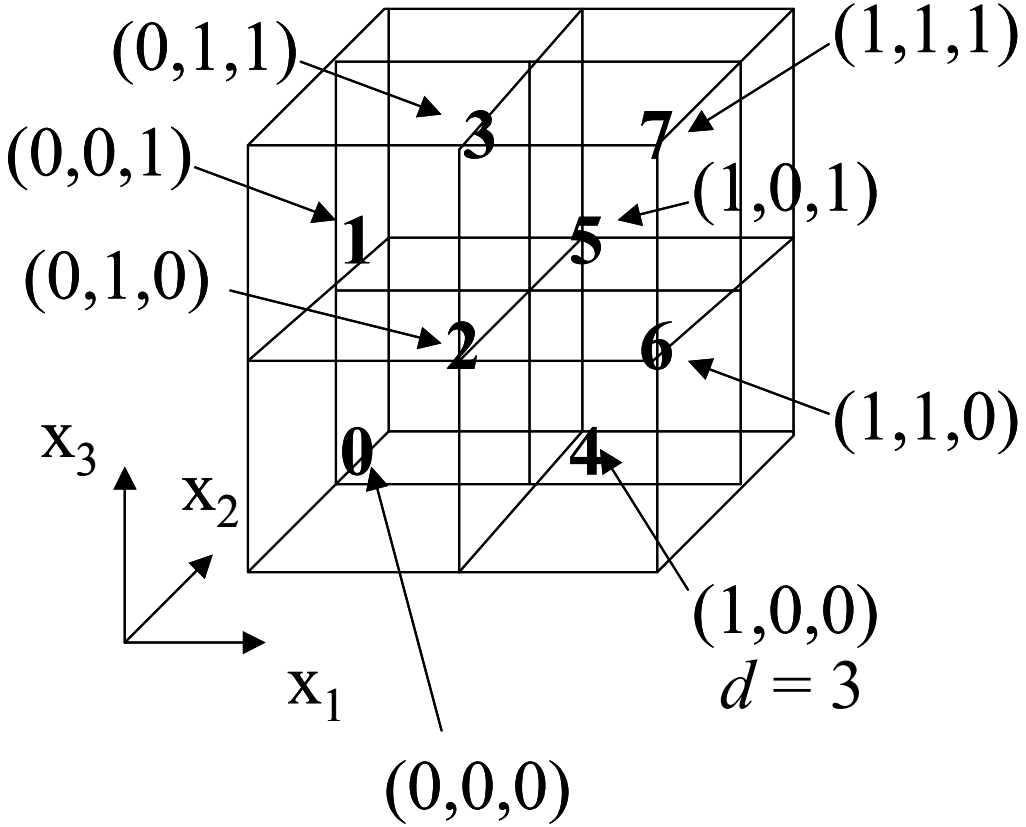
Ordering in *d*-dimensions (3). Convention for Children Ordering.

Any binary string of length d can be converted into a a single number (binary or in some other counting system, e.g. with the base 2^d):

$$(b_1, b_2, ..., b_d) \rightarrow (b_1 b_2 ... b_d)_2 = N_{2^d}.$$

This provides natural numbering of 2^d children of the box.





Ordering in *d*-dimensions (4). Finding the number of the box containing a given point.

Level 1:

$$\bar{\mathbf{x}} = (0.b_{11}b_{21}...b_{d1}b_{12}b_{22}...b_{d2}...b_{1j}b_{2j}...b_{dj}...)_2 \in Box((b_{11}b_{21}...b_{d1})_2) = Box((N_1)_{2^d}),$$

Let us use 2^d-based counting system. Then we can find the box containing a given point at Level l :

$$(0.N_1N_2...N_lc_1c_2c_3...)_{2^d} \in Box((N_1,N_2,...,N_l)_{2^d}), \quad \forall c_j = 0,...,2^d-1; \quad j = 1,2,...$$

Therefore to find the number of the box at level l to which the given point belongs we need simply shift the 2^d number representing this point by l positions and take the integer part of this number:

$$(0.N_1N_2...N_lc_1c_2c_3...)_{2^d} \rightarrow (N_1N_2...N_l.c_1c_2c_3...)_{2^d}; \quad N_1N_2...N_l = [(N_1N_2...N_l.b_1b_2b_3...)_{2^d}].$$

Ordering in d-dimensions (5). Finding the number of the box containing a given point (2). Algorithm and Example.

This procedure also can be performed in binary system by $d \cdot l$ bit shift:

$$(0.b_{11}b_{21}...b_{d1}b_{12}b_{22}...b_{d2}...b_{1l}b_{2l}...b_{dl}b...)_2 \to (b_{11}b_{21}...b_{d1}b_{12}b_{22}...b_{d2}...b_{1l}b_{2l}...b_{dl}.b...)_2;$$

$$Number = (b_{11}b_{21}...b_{d1}b_{12}b_{22}...b_{d2}...b_{1l}b_{2l}...b_{dl})_2.$$

In arbitrary counting system:

$$(Number, l) = \left[2^{dl} \cdot \bar{\mathbf{x}}\right].$$

Problem: Find decimal numbers of boxes at levels 3 and 5 of the oct-tree containing point $\bar{\mathbf{x}} = (0.7681, 0.0459, 0.3912)$.

Solution: First we convert the coordinates of the point to binary format, where we can keep only 5 digits after the point (maximum level is 5), so $\bar{\mathbf{x}} = (0.11000, 0.00001, 0.01100)_2$. Second, we form a single mixed number $\bar{\mathbf{x}} = 0.100101001000010_2$. Performing $3 \cdot 3 = 9$ bit shift and taking integer part we have $(Number, 3) = 100101001_2 = 297$. Performing $3 \cdot 5 = 15$ bit shift we obtain $(Number, 5) = 100101001000010_2 = 19010$.

Answer: #297 and #19010.

Bit Deinterleaving

Convert the box number at level l into binary form

$$Number = (b_{11}b_{21}...b_{d1}b_{12}b_{22}...b_{d2}...b_{1l}b_{2l}...b_{dl})_2.$$

Then we decompose this number to d numbers that will represent d coordinates:

$$Number_1 = (b_{11}b_{12}...b_{1l})_2.$$

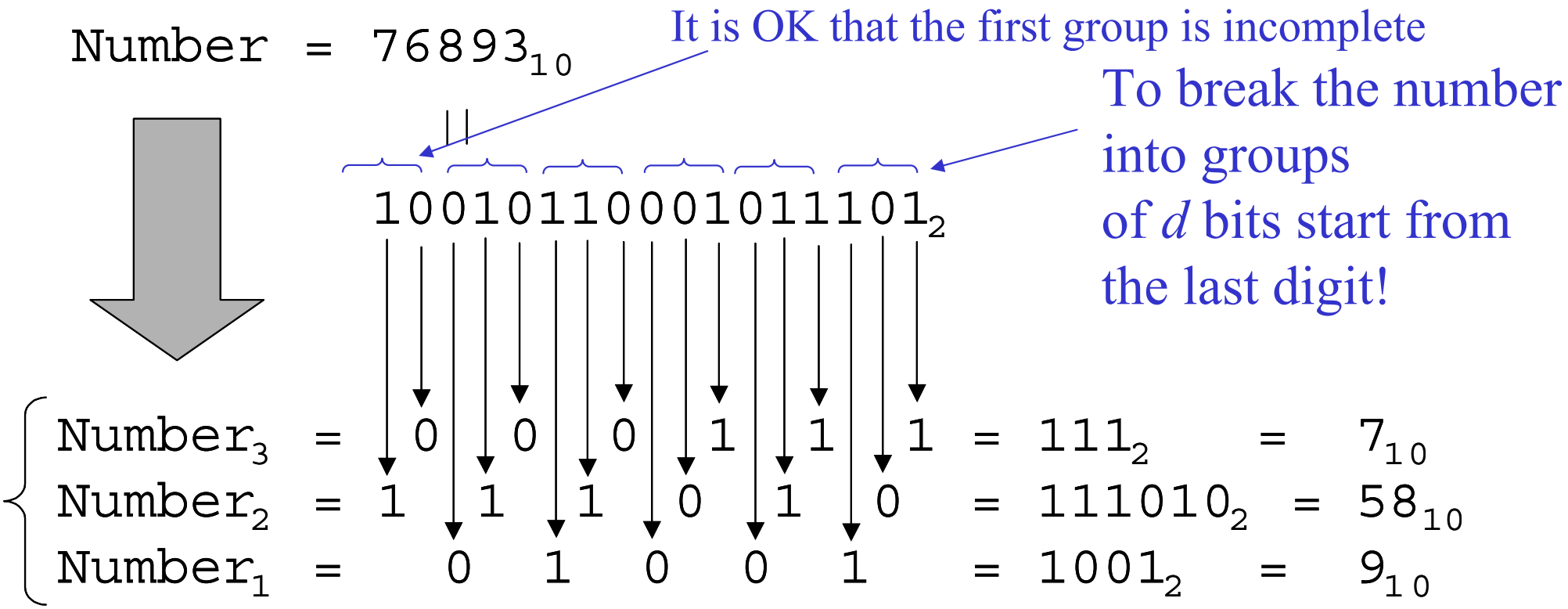
$$Number_2 = (b_{21}b_{22}...b_{2l})_2.$$

$$...$$

$$Number_d = (b_{d1}b_{d2}...b_{dl})_2.$$

.

Bit deinterleving (2). Example.



Finding the center of a given box.

Coordinates of the box center in binary form are

$$\bar{x}_{k,c}(Number, l) = (0.b_{k1}b_{k2}...b_{kl}1)_2, \quad k = 1,...,d.$$

or in the form that does not depend on the counting system:

$$\bar{x}_{k,c}(Number, l) = 2^{-l} \cdot Number_k + 2^{-l-1} = 2^{-l} \cdot \left(Number_k + \frac{1}{2}\right), \quad k = 1,...,d.$$

Problem: Find the center of box #533 (decimal) at level 5 of the oct-tree.

Solution: Converting this number to the bit string we have $533_{10} = 1000010101_2$. Retrieving the digits of three components from the last digit of this number we obtain: $Number_3 = 1001_2 = 9_{10}$, $Number_2 = 10_2 = 2_{10}$, $Number_1 = 1_2 = 1_{10}$. We have then $\bar{x}_{1,c}(533,5) = 2^{-5} \cdot (1 + 0.5) = 0.04875$, $\bar{x}_{2,c}(533,5) = 2^{-5} \cdot (2 + 0.5) = 0.078125$, $\bar{x}_{3,c}(533,5) = 2^{-5} \cdot (9 + 0.5) = 0.296875$.

Answer: $\bar{\mathbf{x}}_c = (0.04875, 0.078125, 0.296875)$.

Neighbor Finding

Step 1: Deinterleaving:

$$Number \rightarrow \{Number_1, ..., Number_d\}$$

Step 2: Shift of the coordinate numbers

$$Number_k^+ = Number_k + 1, \quad Number_k^- = Number_k - 1, \quad k = 1, ..., d,$$

and formation of sets:

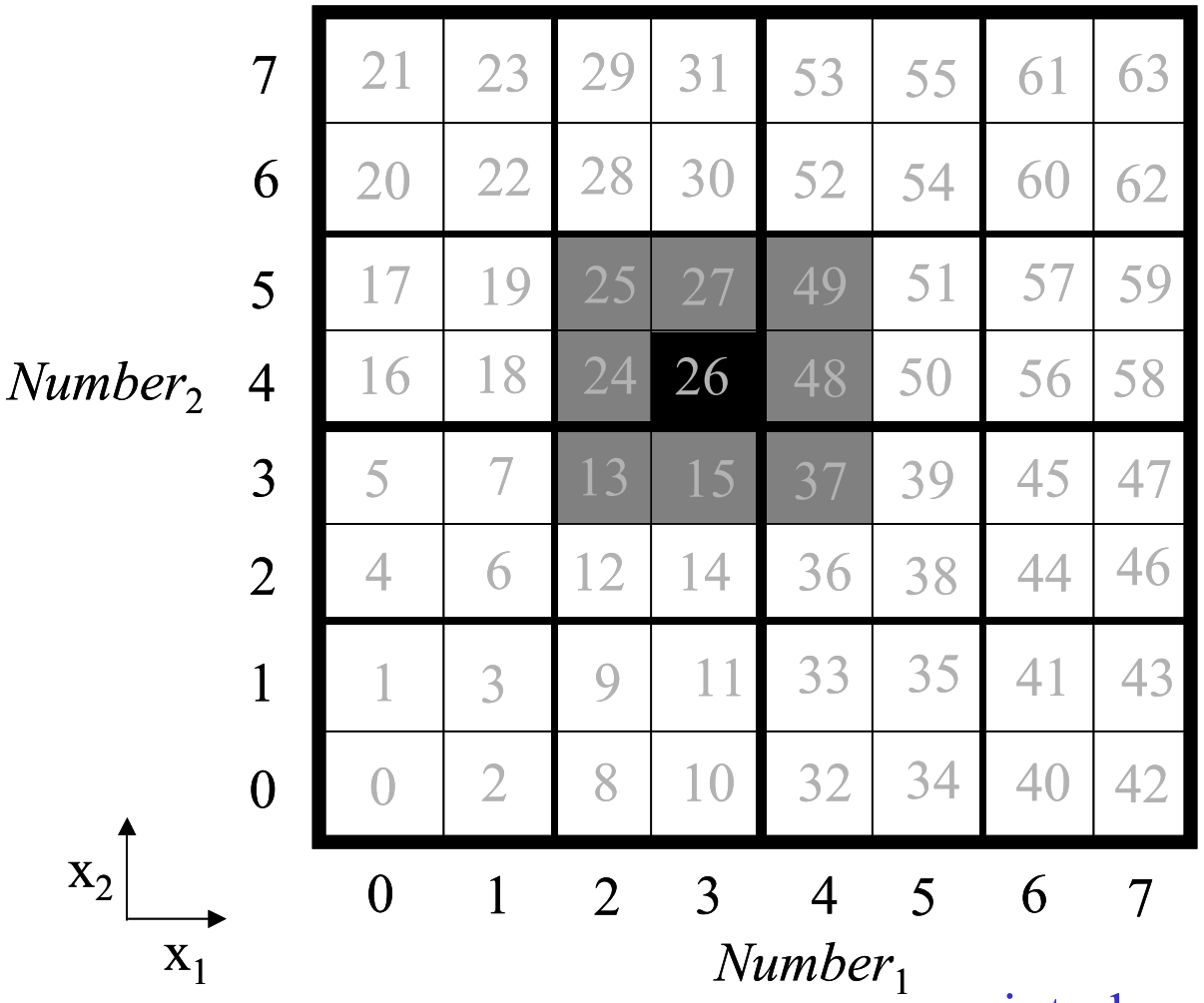
$$s_k = \begin{cases} \{Number_k^-, Number_k, Number_k^+\}, & Number_k \neq 0, 2^l - 1 \\ \{Number_k, Number_k^+\}, & Number_k = 0. \\ \{Number_k^-, Number_k\}, & Number_k = 2^l - 1. \end{cases} \qquad k = 1, ..., d.$$

The set of neighbor generating numbers is then

$$n = (n_1, ..., n_d), \quad n_k \in s_k, \quad k = 1, ..., d.$$

where each n_k can be any element of s_k , except of the case when all $n_k = Number_k$ simultaneously for all $k = 1, ..., d,$ since this case corresponds to the box itself.

Neighbor Finding (2). Example.



$26_{10} = 11010_2$

deinterleaving

$(11,100)_2 = (3,4)_{10}$

generation of neighbors

(2,3) ,(2,4),(2,5),(3,3),
(3,5),(4,3),(4,4),(4,5)
=
(10,11),(10,100),(10,101),
(11,11),(11,101),(100,11),
(100,100),(100,101)

interleaving

1101,11000,11001,1111,11011,100101,110000,110001
= 13, 24, 25, 15, 27, 37, 48, 49

Definitions

Spatial Data Structuring

Data Collection.

Scaling and mapping finite d-dimensional data into a unit d-dimensional cube yields in a *collection* C of N points distributed inside such a cube:

$$\mathsf{C} = \{\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2, ..., \bar{\mathbf{x}}_N\}, \quad \bar{\mathbf{x}}_i \in [0,1) \times [0,1) \times ... \times [0,1) \subset \mathsf{R}^d, \quad i = 1, ..., N.$$

Data Set.

We call a collection $\mathsf{C} = \{\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2, ..., \bar{\mathbf{x}}_N\}$ "*data set*", if $\forall i \neq j$, $dist(\bar{\mathbf{x}}_i, \bar{\mathbf{x}}_j) \neq 0$, where $dist(\bar{\mathbf{x}}_i, \bar{\mathbf{x}}_j)$ denotes distance between $\bar{\mathbf{x}}_i$ and $\bar{\mathbf{x}}_j$.

Non-Separable (Multi-entry) Data Collection.

We call a collection $\mathsf{C} = \{\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2, ..., \bar{\mathbf{x}}_N\}$ "*non-separable data collection*", if $\exists i \neq j$, $dist(\bar{\mathbf{x}}_i, \bar{\mathbf{x}}_j) = 0$.

(By this definition a non-separable data collection cannot be uniquely ordered using distance function $dist(\bar{\mathbf{x}}_i, \bar{\mathbf{x}}_j)$).

Some properties of 2^d-tree hierarchy.

Theorem: Let $dist(\bar{\mathbf{x}}_i, \bar{\mathbf{x}}_j) = |\bar{\mathbf{x}}_i - \bar{\mathbf{x}}_j|$. Then for any data collection C at level

$$L > \frac{1}{d} \log_2 N$$

of 2^d-tree hierarchical space subdivision there exist boxes that do not contain points from C. We call such boxes "*empty*" or "*zero*" boxes.

Proof. The number of boxes at level L is 2^{Ld}, which is larger than N, at $L > \frac{1}{d} \log_2 N$.

Theorem: Let $dist(\bar{\mathbf{x}}_i, \bar{\mathbf{x}}_j) = |\bar{\mathbf{x}}_i - \bar{\mathbf{x}}_j|$. For any data set S of power $N \geqslant 2$ at level

$$L > \log_2 \frac{d^{1/2}}{\bar{D}_{\min}}, \quad \text{where}$$

$$\bar{D}_{\min} = \min_{i \neq j} |\bar{\mathbf{x}}_i - \bar{\mathbf{x}}_j|, \quad i,j = 1,...,N,$$

of 2^d-tree hierarchical space subdivision each box contains not more than 1 data point.

Proof. The the main diagonal of the box at level L is $d^{1/2} 2^{-L}$. This is smaller than $\bar{D}_{\min}$ if $L > \log_2 (d^{1/2} / \bar{D}_{\min})$.

Threshold Level

We call level $L_{th}(\mathsf{C})$ "*threshold level*" of data collection C if the maximum number of data points in a box for any level of subdivision $L > L_{th}(\mathsf{C})$ is the same as for $L_{th}(\mathsf{C})$ and differs from $L_{th}(\mathsf{C})$ for any $L < L_{th}(\mathsf{C})$.

Note: in case if C is a data set of power $N \geqslant 2$, then at level $L_{th}(\mathsf{C})$ we will have maximum one data point per box, and at $L < L_{th}(\mathsf{C})$ there exists at least 1 box containing 2 or more data points.

Some Practical Issues Related to Spatial Ordering

- If the type of data used allows to keep $Bit_{\max}$ bits to represent each coordinate of a data point, then the maximum available level of space subdivision is $Bit_{\max}$. If it happens that $Bit_{\max} < L_{th}(\mathsf{C})$ then C is non-separable in machine representation (by definition we always have a box containing at least 2 data points).

Indeed, limited (discrete) representation of numbers results in discrete distance between the points. If this distance is denoted as *dist*, we have

$$\exists \epsilon > 0, \quad dist(\bar{\mathbf{x}}_i, \bar{\mathbf{x}}_j) = 0 \text{ if } |\bar{\mathbf{x}}_i - \bar{\mathbf{x}}_j| < \epsilon.$$

And a data collection, which is a data set in norm of R^d, is a non-separable data collection in the norm of machine representation.

- Box numbering in d-dimensions using interleaving shows that at level L the number of bits required for box number is Ld. This may cause severe restrictions on use of standard types for representation of integers. E.g. if the max number of bits for integer is 31 and $d = 3$, L cannot exceed 10. This means that if the difference between coordinates of data points is less than $3^{-10} \approx 1.7 \bullet 10^{-5}$ such points cannot be uniquely ordered using the oct-tree. For larger dimensions extended types for integers utilized in hierarchical numbering should be defined.

Spatial Data Sorting

Consider data collection **C**. Each point can be then indexed (or numbered):

$$\mathbf{v} = (v_1, v_2, ..., v_N), \quad v_i = Number(\bar{\mathbf{x}}_i, L), \quad i = 1, ..., N,$$

where *Number* can be determined using the algorithm described in the previous sections.

The array **v** then can be sorted for $O(N \log N)$ operations:

$$(v_1, v_2, ..., v_N) \to (v_{i_1}, v_{i_2}, ..., v_{i_N}), \quad v_{i_1} \leqslant v_{i_2} \leqslant ... \leqslant v_{i_N}.$$

using standart sorting algorithms. These algorithms also return the permutation index (other terminology can be permutation vector or pointer vector) of length N :

$$\mathbf{ind} = (i_1, i_2, ..., i_N),$$

that can be stored in the memory. In terms of memory usage the array **v** should not be rewritten and stored again, since **ind** is a pointer and

$$\mathbf{v}(i) = v_i, \quad \mathbf{ind}(j) = i_j, \quad \mathbf{v}(\mathbf{ind}(j)) = \mathbf{v}(i_j) = v_{ij}, \quad i,j = 1, ..., N,$$

so

$$\mathbf{v}(\mathbf{ind}) = (v_{i_1}, v_{i_2}, ..., v_{i_N}).$$

Spatial Data Sorting (2)

- Before sorting represent your data with maximum number of bits available (or intended to use). This corresponds to maximum level $L_{\text{available}}$ available (say $[L_{\text{available}} = BitMax/d]$.
- In the hierarchical 2^d-tree space subdivision the sorted list will remain sorted at any level $L < L_{\text{available}}$. So the data ordering is required only one time.

After data sorting we need to find the maximum level of space subdivision that will be employed

In Multilevel FMM two following conditions can be mainly considered:

- At level L_{max} each box contains not more than s points (s is called clustering or grouping parameter)
- At level L_{max} the neighborhood of each box contains not more than q points.

The threshold level determination algorithm in $O(N)$ time

$i = 0,\ m = s,\ \ l_{\max} = 0;$
while $m < N$
  $i = i + 1,\ m = m + 1;$
  $a = Interleaved(v(\mathbf{ind}(i));$
  $b = Interleaved(v(\mathbf{ind}(m));$
  $j = Bit_{\max} + 1$
  while $a \neq b$
    $j = j - 1;$
    $a = Parent(a);$
    $b = Parent(b);$
    $l_{\max} = \max(l_{\max}, j);$
  end;
end;

move

s is the clustering parameter

Binary Search in Sorted List

- Operation of getting non-empty boxes at any level L (say neighbors) can be performed with O(log$N$) complexity for any fixed d.
    - It consists of obtaining a small list of all neighbor boxes with O(1) complexity and
    - Binary search of each neighbor in the sorted list at level L is an $O(Ld)$ operation.
    - For small L *and* d this is almost $O(1)$ procedure.

Operations on Sets

Difference: $\mathbb{C} = \mathbb{A} \setminus \mathbb{B}$
Intersection: $\mathbb{C} = \mathbb{A} \cap \mathbb{B}$
Union: $\mathbb{C} = \mathbb{A} \cup \mathbb{B}$

Let $Pow(\mathbb{A}) = N, \quad Pow(\mathbb{B}) = M, \quad N \geqslant M,$

Then the complexity for sorted input/output:

$$\mathbb{A} \setminus \mathbb{B} : N$$
$$\mathbb{A} \cap \mathbb{B} : \min(N, M \log N)$$
$$\mathbb{A} \cup \mathbb{B} : N$$

Operations

$$Neighbors(\mathbb{W}; n, l) = NeighborsAll(n, l) \cap \mathbb{W}, \quad \mathbb{W} = \mathbb{X}, \mathbb{Y},$$
$$Children(\mathbb{W}; n, l) = ChildrenAll(n, l) \cap \mathbb{W}, \quad \mathbb{W} = \mathbb{X}, \mathbb{Y}.$$

are $O(\log N)$ operations for minimum memory requirements and $O(1)$ for sufficiently large memory.

Practical recommendations for the FMM

- Even though you can write an $O(\log N)$ algorithm for getting all necessary information – don't do that, if you have sufficient memory.
- Instead at the step of setting the data structure precompute and store all box indices (sources and receivers), Children (sources and receivers), E2 and E4 neighbor lists for all receivers (neighbor sources for receivers).
- If memory is sufficient, precompute and store all translation operators (S|S, S|R, R|R).
- Create a data structure, which allows you efficiently retrieve translation operator data.