

FMM CMSC 878R/AMSC 698R

Lecture 11

Outline

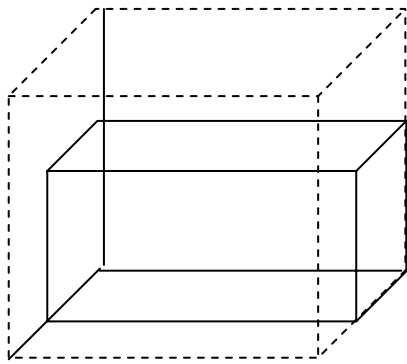
- Ordering in d-dimensions
 - Bit interleaving
 - Children ordering
 - Bit deinterleaving
 - Neighbor and other search algorithms
- Spatial data structuring
 - Separability of space
 - Threshold level of subdivision
 - Data sorting
 - Binary search
 - About some operations on sets (union, difference, intersection)

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 usually have symmetry of directions and can enclose that box to a cube of size $D \times \dots \times D$, where

$$D = \max_d D_d,$$

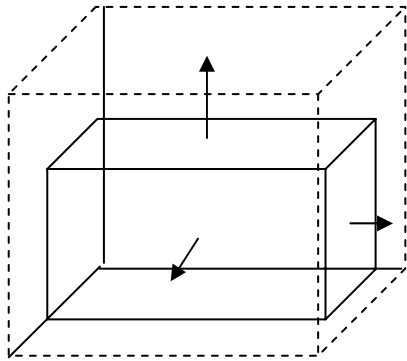
and the corner that has minimum values of Cartesian coordinates:

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

This cube then can be mapped to the unit cube $[0, 1] \times \dots \times [0, 1]$ 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

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

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

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

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 $\bar{\mathbf{x}} \in [0, 1] \times \dots \times [0, 1]$ 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

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

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

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

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

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

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

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

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:

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

Level 2:

$$(0.00b_1b_2b_3\dots)_2 \in \text{Box}((0,0)), \quad (0.01b_1b_2b_3\dots)_2 \in \text{Box}((0,1)),$$


$$(0.10b_1b_2b_3\dots)_2 \in \text{Box}((1,0)), \quad (0.11b_1b_2b_3\dots)_2 \in \text{Box}((1,1)),$$

$$\forall b_j = 0, 1; \quad j = 1, 2, \dots,$$

Level l :

$$(0.N_1N_2\dots N_l b_1b_2b_3\dots)_2 \in \text{Box}((N_1, N_2, \dots, N_l)), \quad \forall b_j = 0, 1; \quad j = 1, 2, \dots$$

We use numbering strings !



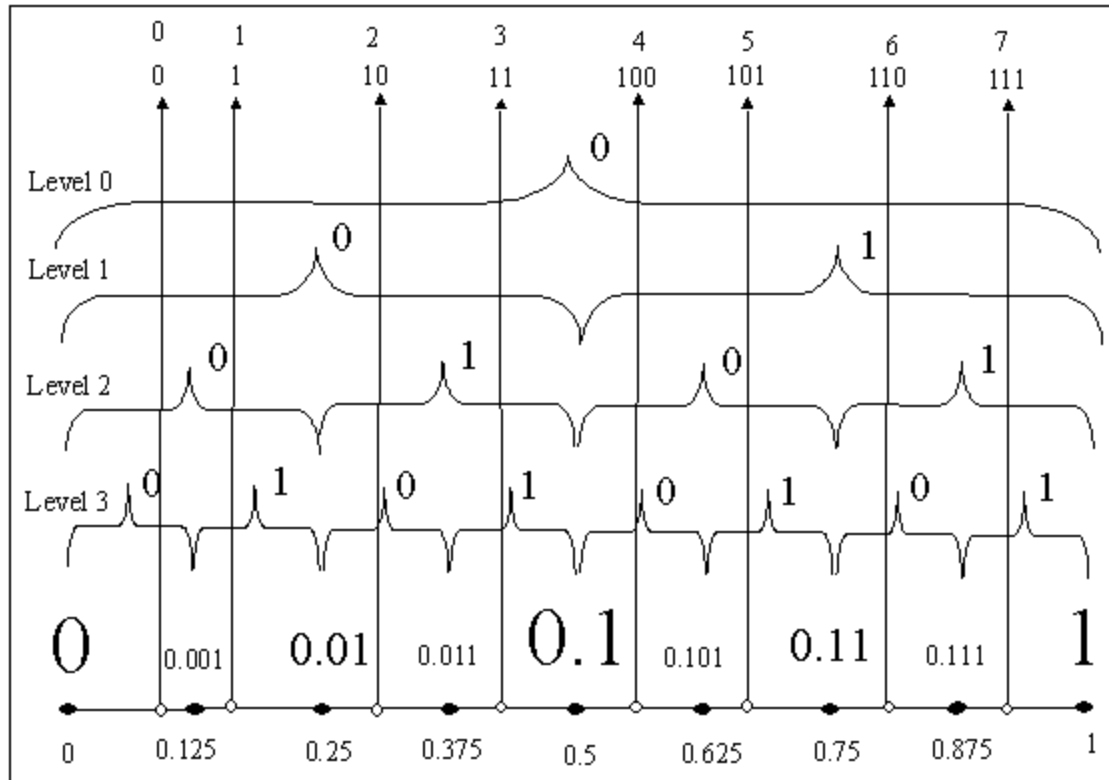
Binary Ordering (3)

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

$$(0.N_1N_2\dots N_l b_1 b_2 b_3 \dots)_2 \rightarrow (N_1 N_2 \dots N_l . b_1 b_2 b_3 \dots)_2; \quad N_1 N_2 \dots N_l = [(N_1 N_2 \dots N_l . b_1 b_2 b_3 \dots)_2].$$

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

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:

$$Number = (N_1N_2\dots N_l)_2 \rightarrow (0.N_1N_2\dots N_l)_2,$$

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

$$\bar{x}_c(Number, l) = (0.N_1N_2\dots N_l1)_2.$$

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:

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

In the binary form:

The size of the box at level l

$$\bar{x}_c(\text{Neighbor}((\text{Number}, l))) = (0.N_1N_2\dots N_l1)_2 \pm \left(\underbrace{0.0\dots01}_l \right)_2,$$

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 :

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

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 $\bar{\mathbf{x}} = (\bar{x}_1, \dots, \bar{x}_d)$ in the d -dimensional unit cube can be represented in binary form

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

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:

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

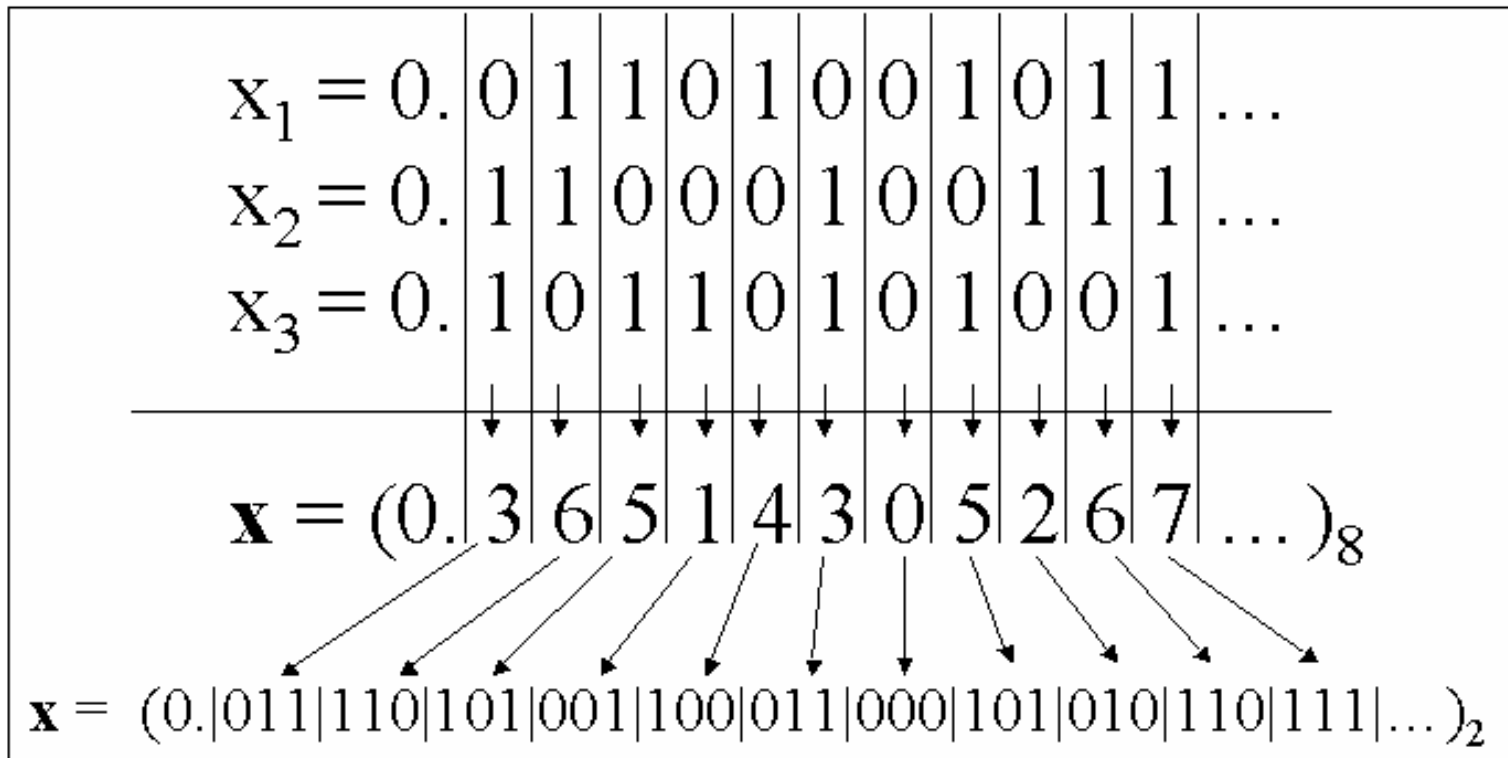
This number can be rewritten in the system with base 2^d :

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

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.

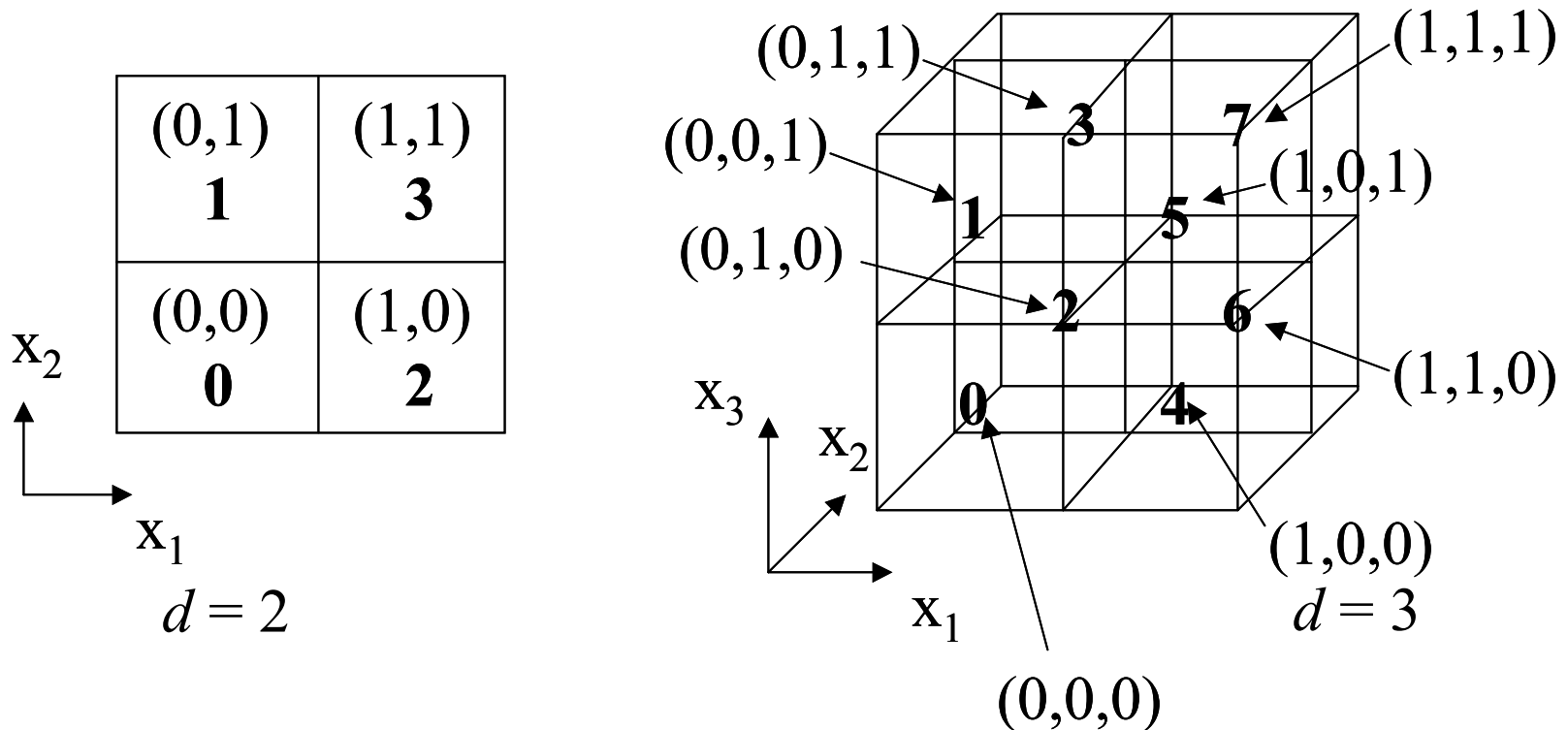


Ordering in d -dimensions (3). Convention for Children Ordering.

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

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

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:

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

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

$$(0.N_1N_2\dots N_l c_1c_2c_3\dots)_{2^d} \in \text{Box}((N_1, N_2, \dots, N_l)_{2^d}), \quad \forall c_j = 0, \dots, 2^d - 1; \quad j = 1, 2, \dots$$

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:

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

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:

$$(0.b_{11}b_{21}\dots b_{d1}b_{12}b_{22}\dots b_{d2}\dots b_{1l}b_{2l}\dots b_{dl}b_{\dots})_2 \rightarrow (b_{11}b_{21}\dots b_{d1}b_{12}b_{22}\dots b_{d2}\dots b_{1l}b_{2l}\dots b_{dl}.b_{\dots})_2;$$
$$\text{Number} = (b_{11}b_{21}\dots b_{d1}b_{12}b_{22}\dots b_{d2}\dots b_{1l}b_{2l}\dots b_{dl})_2.$$

In arbitrary counting system:

$$(\text{Number}, l) = \lceil 2^{dl} \cdot \bar{\mathbf{x}} \rceil.$$

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 $(\text{Number}, 3) = 100101001_2 = 297$. Performing $3 \cdot 5 = 15$ bit shift we obtain $(\text{Number}, 5) = 100101001000010_2 = 19010$.

Answer: #297 and #19010.

Bit Deinterleaving

Convert the box number at level l into binary form

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

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

$$\textit{Number}_1 = (b_{11}b_{12}\dots b_{1l})_2.$$

$$\textit{Number}_2 = (b_{21}b_{22}\dots b_{2l})_2.$$

...

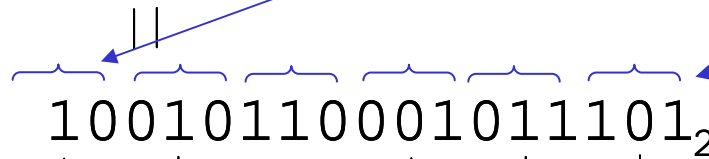
$$\textit{Number}_d = (b_{d1}b_{d2}\dots b_{dl})_2.$$

Bit deinterleaving (2). Example.

Number = 76893_{10}

It is OK that the first group is incomplete

To break the number into groups of d bits start from the last digit!



{	Number ₃	=	↓	0	↓	0	↓	0	↓	1	↓	1	↓	1	=	111_2	=	7_{10}
	Number ₂	=	↓	1	↓	1	↓	1	↓	0	↓	1	↓	0	=	111010_2	=	58_{10}
	Number ₁	=	↓	0	↓	1	↓	0	↓	0	↓	0	↓	1	=	1001_2	=	9_{10}

Finding the center of a given box.

Coordinates of the box center in binary form are

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

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

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

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

$$\bar{x}_{1,c}(533, 5) = 2^{-5} \cdot (1 + 0.5) = 0.04875, \quad \bar{x}_{2,c}(533, 5) = 2^{-5} \cdot (2 + 0.5) = 0.078125,$$

$$\bar{x}_{3,c}(533, 5) = 2^{-5} \cdot (9 + 0.5) = 0.296875.$$

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

Neighbor Finding

Step 1: Deinterleaving:

$$\text{Number} \rightarrow \{\text{Number}_1, \dots, \text{Number}_d\}$$

Step 2: Shift of the coordinate numbers

$$\text{Number}_k^+ = \text{Number}_k + 1, \quad \text{Number}_k^- = \text{Number}_k - 1, \quad k = 1, \dots, d,$$

and formation of sets:

$$s_k = \begin{cases} \{\text{Number}_k^-, \text{Number}_k, \text{Number}_k^+\}, & \text{Number}_k \neq 0, 2^l - 1 \\ \{\text{Number}_k, \text{Number}_k^+\}, & \text{Number}_k = 0. \\ \{\text{Number}_k^-, \text{Number}_k\}, & \text{Number}_k = 2^l - 1. \end{cases} \quad k = 1, \dots, d.$$

The set of neighbor generating numbers is then

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

where each n_k can be any element of s_k , except of the case when all $n_k = \text{Number}_k$ simultaneously for all $k = 1, \dots, d$, since this case corresponds to the box itself.

Neighbor Finding (2). Example.

7	21	23	29	31	53	55	61	63
6	20	22	28	30	52	54	60	62
5	17	19	25	27	49	51	57	59
4	16	18	24	26	48	50	56	58
3	5	7	13	15	37	39	45	47
2	4	6	12	14	36	38	44	46
1	1	3	9	11	33	35	41	43
0	0	2	8	10	32	34	40	42
	0	1	2	3	4	5	6	7

x_2 ↑
 ↓ x_1

Number₁

$$26_{10} = 11010_2$$

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

$$1101, 11000, 11001, 1111, 11011, 100101, 110000, 110001$$

$$= 13, 24, 25, 15, 27, 37, 48, 49$$

Spatial Data Structuring

Data Collection.

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

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

Data Set.

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

Non-Separable (Multi-entry) Data Collection.

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

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

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 $N \geq 2$ at level

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

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

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

Note: in case if \mathbf{C} is a data set of power $N \geq 2$, then at level $L_{th}(\mathbf{C})$ we will have maximum one data point per box, and at $L < L_{th}(\mathbf{C})$ 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 $Bit_{\max} < L_{th}(C)$ 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 \mathbb{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 $3^{-10} \approx 1.7 \cdot 10^{-5}$ 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 \mathbf{C} . Each point can be then indexed (or numbered):

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

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

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

$$(v_1, v_2, \dots, v_N) \rightarrow (v_{i_1}, v_{i_2}, \dots, v_{i_N}), \quad v_{i_1} \leq v_{i_2} \leq \dots \leq v_{i_N}.$$

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

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

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

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

so

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

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 $[L_{\text{available}} = \text{BitMax}/d]$).
- 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

$i = 0, m = s,$

while $m < N$

$i = i + 1, m = m + 1;$

$a = \text{Interleaved}(v(\mathbf{ind}(i)));$

$b = \text{Interleaved}(v(\mathbf{ind}(m)));$

$j = \text{Bit}_{\max} + 1$

while $a \neq b$

$j = j - 1;$

$a = \text{Parent}(a);$

$b = \text{Parent}(b);$

$l_{\max} = \max(l_{\max}, j);$

end;

end;

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: $C = A \setminus B$

Intersection: $C = A \cap B$

Union: $C = A \cup B$

Let $Pow(A) = N$, $Pow(B) = M$, $N \geq M$,

Then the complexity for sorted input/output:

$$A \setminus B : N$$

$$A \cap B : \min(N, M \log N)$$

$$A \cup B : N$$

Operations

$$Neighbors(W; n, l) = NeighborsAll(n, l) \cap W, \quad W = X, Y,$$

$$Children(W; n, l) = ChildrenAll(n, l) \cap W, \quad W = X, Y.$$

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