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 $M=\tilde{N}$), 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.
  - 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: LRRL

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).

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:

$\text{RBRTLBRT} \rightarrow (\text{RB})(\text{RT})(\text{LB})(\text{RT})$

$\text{k-2 tree}$

$\text{quad-tree = } 2^2\text{-tree}$
Hierarchical Numbering (Indexing) in 2\textsuperscript{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);

  \[ \text{Number} = (N_1, N_2, \ldots, N_I), \quad N_f = 0, \ldots, 2^d - 1, \quad f = 1, \ldots, I \]

  Numbering string

**In general**:
- Universal number is a pair:
  \[ \text{UniversalNumber} = (\text{Number}, I) \]

  This number at this level

Hierarchical Numbering in 2\textsuperscript{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\textsubscript{1}=11\textsubscript{10}.
  - The small black box has the numbering string (3,1,2). So its number is 312\textsubscript{2}=54\textsubscript{10}.

  \[ \text{Number} = (2^d)^{I-1} \cdot N_1 + (2^d)^{I-2} \cdot N_2 + \ldots + 2^d \cdot N_{I-1} + N_I \]

  In general: Number at level \( I \) is:

Parent Index (Number)

- **Parent number**:
  \[ \text{Parent(Number)} = (2^d)^{I-2} \cdot N_1 + (2^d)^{I-3} \cdot N_2 + \ldots + N_{I-1} \]

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

  \[ \text{Parent}(1, 3) = \text{Parent}(23, 3) = 2_4 = 2_{10}. \]

  Parent’s universal number:

  \[ \text{Parent(Number, I)} = (\text{Parent(Number)}, I - 1). \]

  Algorithm to find the parent number:

  \[ \text{Parent(Number)} = \lfloor \text{Number}/2^d \rfloor \]

  For box \#23\textsubscript{3} (gray or black) the parent box number is 2\textsubscript{1}. 
**Children Indices (Numbers)**

Children numbering strings:

\[
\text{Children}(N_1, N_2, \ldots, N_d, N_1) = (\langle N_1, N_2, \ldots, N_d, N_1 \rangle), \quad N_{d+1} = 0, \ldots, 2^d - 1.
\]

Children numbers:

\[
\text{Children}(\text{Number}) = \langle (2^d)^{-1} \times N_1 + (2^d)^{-1} \times N_2 + \ldots + (2^d)^{-1} \times N_d + N_1 \rangle, \quad N_{d+1} = 0, \ldots, 2^d - 1
\]

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

\[
\text{Children}(111) = \text{Children}(231) = \langle 230_1, 231_1, 232_1, 233_1 \rangle = \langle 44_{10}, 45_{10}, 46_{10}, 47_{10} \rangle
\]

Children universal numbers:

\[
\text{Children}(\text{Number}, I) = \langle \text{Children}(\text{Number}), I + 1 \rangle.
\]

Algorithm to find the children numbers:

\[
\text{Children}(\text{Number}) = \langle 2^d \times \text{Number} + f \rangle, \quad f = 0, \ldots, 2^d - 1,
\]

---

**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**

\[
p = \text{bitshift}(n,-d);
\]
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 usually have symmetry of directions and can enclose that box to a cube of size $D \times \cdots \times D$, where

$$D = \max_j D_j,$$

and the corner that has minimum values of Cartesian coordinates:

$$x_{\min} = (x_{\min}, \ldots, x_{\min}).$$

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

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

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

Binary Ordering (1)

$d = 1$;

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

$$\mathbf{x} = (a_0.a_1a_2a_3 \ldots)_{10}, \quad a_j = 0, 1; \quad j = 1, 2, \ldots$$

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

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

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

$$\mathbf{x} = (b_0.b_1b_2b_3 \ldots)_2, \quad b_j = 0, 1; \quad j = 1, 2, \ldots$$

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

$$\mathbf{x} = 1 = (0.111111\ldots)_2.$$
Binary Ordering (2)
Finding the number of the box containing a given point

<table>
<thead>
<tr>
<th>Level</th>
<th>Box Size (dec)</th>
<th>Box Size (bin)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0.5</td>
<td>0.1</td>
</tr>
<tr>
<td>2</td>
<td>0.25</td>
<td>0.01</td>
</tr>
<tr>
<td>3</td>
<td>0.125</td>
<td>0.001</td>
</tr>
</tbody>
</table>

Level 1:
(0.000b:b:b...b:b...b:b) ∈ $\Box((0,0))$, (0.100b:b:b...b:b...b:b) ∈ $\Box((0,1))$, $\forall b_j = 0,1$; $j = 1,2,...$

Level 2:
(0.000b:b:b...b:b...b:b) ∈ $\Box((0,0))$, (0.100b:b:b...b:b...b:b) ∈ $\Box((0,1))$, (0.110b:b:b...b:b...b:b) ∈ $\Box((1,1))$,
$\forall b_j = 0,1$; $j = 1,2,...$

Level 3:
(0.000b:b:b...b:b...b:b) ∈ $\Box((0,0))$, (0.010b:b:b...b:b...b:b) ∈ $\Box((0,1))$, (0.100b:b:b...b:b...b:b) ∈ $\Box((1,1))$,
$\forall b_j = 0,1$; $j = 1,2,...$

We use numbering strings!

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

$\langle Number,I\rangle = [2^l \cdot x]$

This is an algorithm for finding 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:

$Number = \langle N_1,N_2,...,N_l \rangle \cdot 2^l - \langle N_1,N_2,...,N_l \rangle$

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

$x_i(Number,I) = \langle N_1,N_2,...,N_l \rangle$

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

$x_i(Number,I) = 2^l \cdot Number + 2^{-1} = 2 \cdot \langle Number + 2^{-1} \rangle$

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

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

Solution: We have $x_i(31,5) = 2^4 \cdot 31 + 0.5 = 0.004375$.

Answer: 0.004375.

This is the algorithm!

Binary Ordering (5)
Neighbor finding

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

$x_i(\langle Neighbor(Number,I)\rangle) = x_i(Number,I) \pm 2^{-1}$.

In the binary form:

$x_i(\langle Neighbor(Number,I)\rangle) = \langle 0,N_1,N_2,...,N_l \rangle \pm \langle 0,0,0,0,1 \rangle$.

To find the number of the neighbor box we can then use the above algorithm for determining the number of the box for point $x_i$.

$Neighbor(Number,I) = \langle N_1,N_2,...,N_l,1 \pm 1 \rangle = \langle N_1,N_2,...,N_l \pm 1 \rangle = Number \pm 1$.

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 exceed 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 $x = (x_1, \ldots, x_d)$ in the $d$-dimensional unit cube can be represented in binary form

$$x_j = (b_1, b_2, b_3, \ldots, b_k), \quad b_k = 0, 1; \quad j = 1, 2, \ldots, k = 1, \ldots, d.$$ 

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:

$$x = (0, b_1b_2b_3b_4b_5b_6b_7b_8b_9b_{10}\ldots)_2.$$ 

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

$$x = (0, N_1N_2N_3\ldots N_d)_2, \quad N_j = (b_1b_2b_3b_4b_5b_6b_7b_8\ldots)_{2^j}, \quad j = 1, 2, \ldots, N_1 = 0, \ldots, 2^d - 1.$$

This maps $\mathbb{R}^d \rightarrow \mathbb{R}$, 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 $2^j$):

$$(b_1, b_2, \ldots, b_d) = (b_1b_2b_3b_4b_5b_6b_7b_8b_9b_{10}\ldots)_{2^j} \in \text{Box}((b_1b_2b_3b_4b_5b_6b_7b_8\ldots)_2).$$

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:

$$x = (0, b_1b_2b_3b_4b_5b_6b_7b_8b_9b_{10}\ldots)_2 \in \text{Box}((b_1b_2b_3b_4b_5b_6b_7b_8\ldots)_{2^1}).$$

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

$$(0, N_1N_2N_3\ldots N_dE_{d+1}E_{d+2}\ldots)_{2^l} \in \text{Box}((N_1N_2N_3\ldots N_d)_{2^l}). \quad \forall E_j = 0, \ldots, 2^d - 1; \quad j = 1, 2, \ldots.$$ 

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:

$$(0, N_1N_2N_3\ldots N_dE_{d+1}E_{d+2}\ldots)_{2^l} = (N_1N_2N_3\ldots N_dE_{d+1}E_{d+2}\ldots)_2, \quad N_1N_2N_3\ldots N_d = [(N_1N_2N_3\ldots N_d)_{2^l}]_{2^d}.$$
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 $d \cdot \ell$ bit shift:

\[(0.b_1b_2...b_{\ell}b_{\ell}b_{2\ell}...b_{2\ell}...b_{d\ell}...b_{d\ell}...b_{d\ell});2 \rightarrow (b_1b_2...b_{\ell}b_{\ell}b_{2\ell}...b_{2\ell}...b_{d\ell}...b_{d\ell}...b_{d\ell})_2;2 \]

\[ \text{Number} = (b_1b_2...b_{d\ell}b_{d\ell}b_{2d\ell}...b_{2d\ell}...b_{d^2\ell}...b_{d^2\ell})_2;2 \]

In arbitrary counting system:

\[(\text{Number}, \ell) = \left[ 2^\ell \cdot \mathbf{X} \right] \]

**Problem:** Find decimal numbers of boxes at levels 3 and 5 of the oct-tree containing point \( \mathbf{X} = (0.7681, 0.0459, 0.3042) \).

**Solution:** First we convert the coordinates of the point to binary format, where we can keep only 5 digits after the point (maximum level in 5), so \( \mathbf{X} = (0.11000, 0.00001, 0.01100) \). Second, we form a single mixed number \( \mathbf{X} = 0.100101001000001 \). Performing $3 \cdot 3 = 9$ bit shift and taking integer part we have \( \text{Number}_3 = 100101001000001 \); $2^9 = 512$. Performing $3 \cdot 5 = 15$ bit shift we obtain \( \text{Number}_5 = 1001010010000010 \) = 19010.

**Answer:** #297 and #19010.

---

Bit Deinterleaving

Convert the box number at level $\ell$ into binary form:

\[ \text{Number} = (b_1b_2...b_{d\ell}b_{d\ell}b_{2d\ell}...b_{2d\ell}...b_{d^2\ell}...b_{d^2\ell})_2; \]

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

\[ \text{Number}_1 = (b_1b_2...b_{2\ell})_2; \]
\[ \text{Number}_2 = (b_{\ell}b_{2\ell}...b_{3\ell})_2; \]
\[ \vdots \]
\[ \text{Number}_d = (b_{d\ell}b_{2d\ell}...b_{d^2\ell})_2; \]

---

Bit deinterleaving (2). Example.

\[ \text{Number} = 76893_{10} \]

It is OK that the first group is incomplete

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

\[
\begin{align*}
\text{Number}_3 &= 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ = 111_2 = 7_{10} \\
\text{Number}_2 &= 1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0 \ = 111010_2 = 58_{10} \\
\text{Number}_1 &= 0 \ 1 \ 0 \ 0 \ 1 \ = 1001_2 = 9_{10}
\end{align*}
\]

---

Finding the center of a given box.

Coordinates of the box center in binary form are

\[ x_{k\ell}(\text{Number}, \ell) = (b_{k\ell}b_{(k+1)\ell}...b_{d\ell})_2, \quad k = 1, ..., d. \]

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

\[ x_{k\ell}(\text{Number}, \ell) = 2^{\ell-k} \cdot \text{Number}_{k+1} \cdot 2^{\ell-k} \cdot \left( \text{Number}_{k+1} + \frac{1}{2} \right), \quad k = 1, ..., d. \]

**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:

\[ \text{Number}_5 = 1001; \quad \text{Number}_4 = 101; \quad \text{Number}_3 = 101 \]

We have then

\[ x_{55}(533, 5) = 2^0 \cdot (1 + 0.5) = 0.4875, \quad x_{45}(533, 5) = 2^3 \cdot (2 + 0.5) = 0.078125, \quad x_{35}(533, 5) = 2^5 \cdot (9 + 0.5) = 0.296875. \]

**Answer:** $x_5 = (0.4875, 0.078125, 0.296875)$. 

Neighbor Finding

Step 1: Deinterleaving:

\[ \text{Number} \rightarrow \langle \text{Number}_1, ..., \text{Number}_d \rangle \]

Step 2: Shift of the coordinate numbers

\[ \text{Number}_i' = \text{Number}_i + 1, \quad \text{Number}_j' = \text{Number}_j - 1, \quad k = 1, ..., d, \]

and formation of sets:

\[ s_k = \begin{cases} \langle \text{Number}_i', \text{Number}_j', \text{Number}_k' \rangle, & \text{Number}_k' = 0, 2^l - 1 \\ \langle \text{Number}_i', \text{Number}_j' \rangle, & \text{Number}_k' = 0 \\ \langle \text{Number}_i', \text{Number}_k' \rangle, & \text{Number}_j' = 2^l - 1. \end{cases} \]

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 the case when all \( n_k = \text{Number}_k \) simultaneously for all \( k = 1, ..., d \), since this case corresponds to the box itself.

Spatial Data Structuring

Definitions

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:

\[ C = \langle x_1, x_2, ..., x_d \rangle, \quad x_j \in [0, 1) \times [0, 1) \times ... \times [0, 1) \subset \mathbb{R}^d, \quad j = 1, ..., N. \]

Data Set:

We call a collection \( C = \langle x_1, x_2, ..., x_d \rangle \) “data set”, if \( \forall i \neq j, \, \text{dist}(x_i, x_j) \neq 0 \), where \( \text{dist}(x_i, x_j) \) denotes distance between \( x_i \) and \( x_j \).

Non-Separable (Multi-entry) Data Collection:

We call a collection \( C = \langle x_1, x_2, ..., x_d \rangle \) “non-separable data collection”, if \( \exists i \neq j, \, \text{dist}(x_i, x_j) = 0 \).

(By this definition a non-separable data collection cannot be uniquely ordered using distance function \( \text{dist}(x_i, x_j) \)).

Some properties of \( 2^d \)-tree hierarchy.

Theorem: Let \( \text{dist}(x_i, x_j) = |x_i - x_j| \). Then for any data collection \( C \) at level \( L \geq \frac{1}{d} \log_2 N \) 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 \( \text{dist}(x_i, x_j) = |x_i - x_j| \). For any data set \( S \) of power \( N > 2 \) at level

\[ L > \log_2 \frac{\text{dist}(x_i, x_j)}{D_{\text{max}}}. \]

where

\[ D_{\text{max}} = \max \{ |x_i - x_j| \}, \quad i, j = 1, ..., N. \]

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

Proof: The theorem is proved if \( L > \log_2 \text{dist}(x_i, x_j)/D_{\text{max}} \).
Threshold Level

We call level \( L_0(C) \) “threshold level” of data collection \( C \) if the maximum number of data points in a box for any level of subdivision \( L > L_0(C) \) is the same as for \( L_0(C) \) and differs from \( L_0(C) \) for any \( L < L_0(C) \).

Note: in case if \( C \) is a data set of power \( N \geq 2 \), then at level \( L_0(C) \) we will have maximum one data point per box, and at \( L < L_0(C) \) there exists at least 1 box containing 2 or more data points.

Spatial Data Sorting

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

\[ v = (v_1, v_2, \ldots, v_D), \quad v_i = \text{Number}(x_i, L), \quad i = 1, \ldots, N, \]

where \( \text{Number} \) can be determined using the algorithm described in the previous sections. The array \( v \) then can be sorted for \( O(N \log N) \) operations:

\[ (v_1, v_2, \ldots, v_D) \rightarrow (v_{i_1}, v_{i_2}, \ldots, v_{i_D}), \quad v_{i_1} \leq v_{i_2} \leq \ldots \leq v_{i_D}, \]

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

\[ \text{ind} = (i_1, i_2, \ldots, i_D), \]

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

\[ v(i) = v_{i_k}, \quad \text{ind}(j) = i_j, \quad v(\text{ind}(j)) = v_{i_j}, \quad i_j = 1, \ldots, N, \]

so

\[ v(\text{ind}) = (v_{i_1}, v_{i_2}, \ldots, v_{i_D}). \]

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 \( L_{\text{available}} = \frac{\text{BitMax}}{d} \)).
- In the hierarchical 2d-tree subdivision the sorted list will remain sorted at any level \( L < L_{\text{available}} \). So the data ordering is required only one time.

Some Practical Issues Related to Spatial Ordering

- If the type of data used allows to keep \( \text{Bit}_{\text{max}} \) bits to represent each coordinate of a data point, then the maximum available level of space subdivision is \( L_{\text{max}} \). If it happens that \( \text{Bit}_{\text{max}} < L_0(C) \) 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 \( \text{dist} \), we have:

\[ \exists \varepsilon > 0, \quad \text{dist}(x_i, x_j) = 0 \text{ if } |x_i - x_j| < \varepsilon. \]

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 \( |d| \). 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 \( 3^{-10} = 1 \times 10^{-10} \) such points cannot be uniquely ordered using the oct-tree. For larger dimensions extended types for integers utilized in hierarchical numbering should be defined.
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

\[ i = 0, \quad m = s, \quad l_{max} = 0; \]
\[ \text{while } m < N \]
\[ i = i + 1, \quad m = m + 1; \]
\[ a = \text{Interleaved}(s, \text{ind}(i)); \]
\[ b = \text{Interleaved}(s, \text{ind}(m)); \]
\[ j = BH_{max} + 1 \]
\[ \text{while } a \neq b \]
\[ j = j - 1; \]
\[ a = Parent(a); \]
\[ b = Parent(b); \]
\[ l_{max} = \min(l_{max}, j); \]
\[ \text{move} \quad \text{end}; \]
\[ \text{end}; \]

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 $\text{Pow}(A) = N$, $\text{Pow}(B) = M$, $N \geq M$.

Then the complexity for sorted input/output:

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

Operations

- $\text{Neighbors}(W, n, l) = \text{NeighborsAll}(n, l) \cap W$, $W = X, Y$
- $\text{Children}(W, n, l) = \text{ChildrenAll}(n, l) \cap W$, $W = X, Y$

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 (SIS, SIR, RIR).
- Create a data structure, which allows you efficiently retrieve translation operator data.