Lecture 10
AMSC698R/CMSC878R

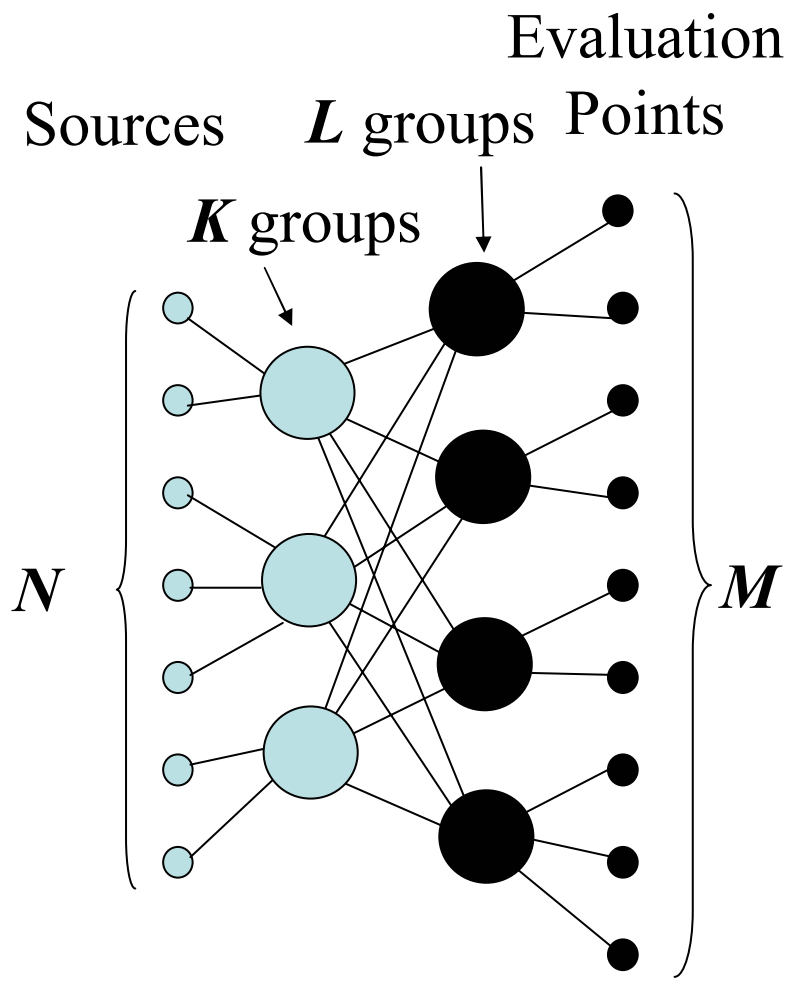
Outline

- Review
- More datastructures for FMM

SLFMM

- SLFMM achieves a complexity of $O(N^{3/2})$
- Increasing the number of levels in the hierarchy can make the algorithm approach $O(N \log N)$
- Imposes a stringent requirement on all operations
  - They have to be $o(N \log N)$ for overall complexity to stay

SLFMM



Total number of operations: $O(N+M+KL)$
For optimum $K \sim O(P^{4/3} N^{4/3})$

SLFMM Characteristics

- Group source points into clusters in "optimal way"
  - Group evaluation points into clusters
- Find the cluster center ($\mathbf{x}_*$) for each cluster
- Find distances from cluster center to points in cluster
- Find distances between clusters
- Build a *S* representation for points in each cluster

$$\Phi(\mathbf{x}_i, \mathbf{y}) = \sum_{m=1}^{p} b_m(\mathbf{x}_i, \mathbf{x}_*)\, S_m\, (\mathbf{y}-\mathbf{x}_*)$$

  - Consolidate S series from all $\mathbf{x}_i$ in the cluster
  - Find other clusters that are outside min. radius at which *S* expansion converges and translate *S* series to *R* series
  - Evaluate *R* series at the evaluation points
  - Clean-up by evaluating at points which are inside min. radius

Algorithms needed

- Clustering
- Center finding
- Distance between points (cluster centers)
- Neighbor finding
- Hierarchical division in FMM adds
  - Hierarchical clustering, center finding
  - Finding parents, children, siblings
  - Finding neighbors
- Spatial data-structures
- Also, the structures used to store these relationships require memory
- Critically determine the “break-even” point

Spatial Data Structures

- Spatial data structures have developed since late 1980s,
  - Aside: one of the founders of the field (Hanan Samet) is local
- FMM developed over the same period
  - So don't use the latest spatial data structure algorithms
- For effective "generalization" we need to use state-of-the-art spatial data structures
- Optimal data structures for FMM is an open research area
- In this course we use hierarchical division into boxes using 2^d trees
  - Use asymptotically optimal algorithms for performing operations on them
  - Related to quadtrees, octrees, etc. (but simpler)

- Constant time methods to
  - number the boxes in a way that can be generated from the coordinate
  - assign points to boxes using their binary coordinates
  - find box centers
  - Find neighboring boxes
- Use bit interleaving and bit shift
- Apply to the general *d*- dimensional case
- Storage is also minimized as most necessary relations are generated from coordinates

FMM CMSC 878R/AMSC 698R

Lecture 10

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, etc.)
  - Trading memory for speed

Ordering in *d*-dimensions. Bit interleaving.

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

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

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:

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

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

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

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

Bit Interleaving (2). Example.

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

$$x_1 = 0.\,0\,1\,1\,0\,1\,0\,0\,1\,0\,1\,1\ldots$$
$$x_2 = 0.\,1\,1\,0\,0\,0\,1\,0\,0\,1\,1\,1\ldots$$
$$x_3 = 0.\,1\,0\,1\,1\,0\,1\,0\,1\,0\,0\,1\ldots$$

$$\mathbf{x} = (0.\,3\,6\,5\,1\,4\,3\,0\,5\,2\,6\,7\ldots)_8$$

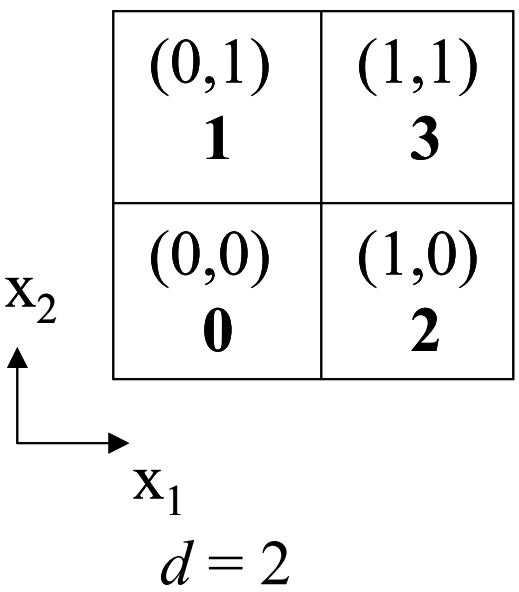
$$\mathbf{x} = (0.|011|110|101|001|100|011|000|101|010|110|111|\ldots)_2$$

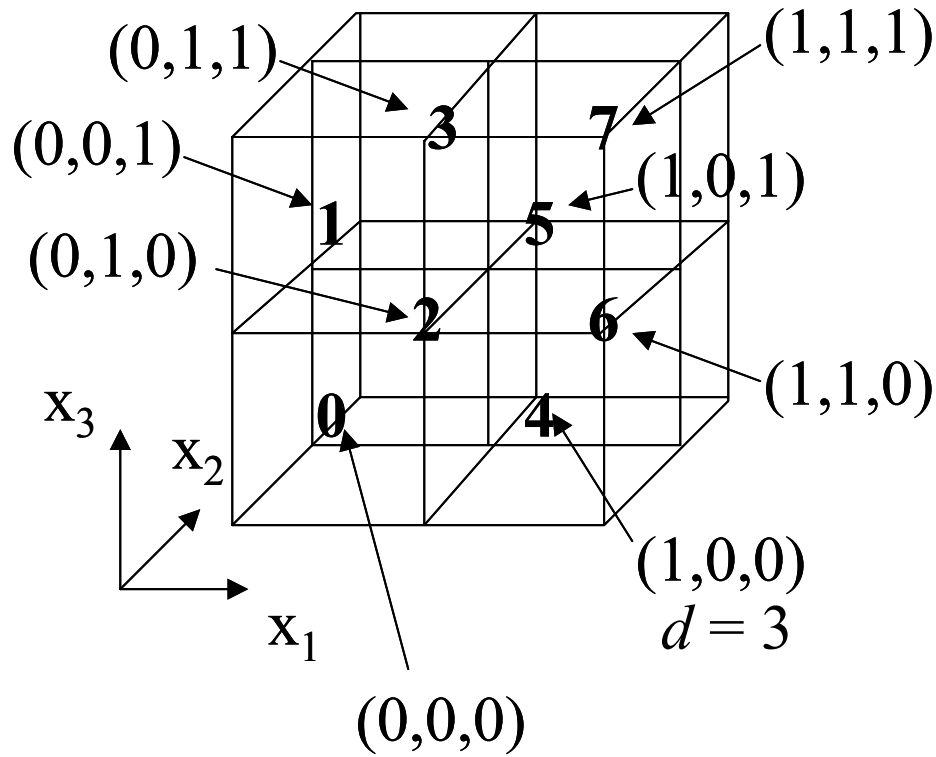
Convention for Children Ordering.

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):

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

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





Finding the number (index) 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:

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

Finding the number (index) 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:

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

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

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.

$\text{Number} = 76893_{10}$

It is OK that the first group is incomplete

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

$$100101100010111 01_2$$

$$\begin{cases} \text{Number}_3 = 0\ 0\ 0\ 1\ 1\ 1 = 111_2 = 7_{10} \\ \text{Number}_2 = 1\ 1\ 1\ 0\ 1\ 0 = 111010_2 = 58_{10} \\ \text{Number}_1 = 0\ 1\ 0\ 0\ 1 = 1001_2 = 9_{10} \end{cases}$$

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:

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

Step 2: Shift of the coordinate numbers

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

and formation of sets:

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

The set of neighbor generating numbers is then

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

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

Neighbor Finding (2). Example.

$Number_2$								
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

$Number_1$

$26_{10} = 11010_2$

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

- Needed for FMM
  - Understanding of Neighborhoods;
  - Understanding Error Bounds and Limits;
  - Understanding of Complexity (e.g. what Log N complexity is in practice);
  - Understanding of Types of Variables to be Employed;
  - Efficient Algorithms.

So, some basic philosophy first…

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:

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

Data Set.

We call a collection $\mathsf{C} = \{\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2, \ldots, \bar{\mathbf{x}}_N\}$ "*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 $\mathsf{C} = \{\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2, \ldots, \bar{\mathbf{x}}_N\}$ "*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

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

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

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):

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

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:

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

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

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

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

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

so

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

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

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.