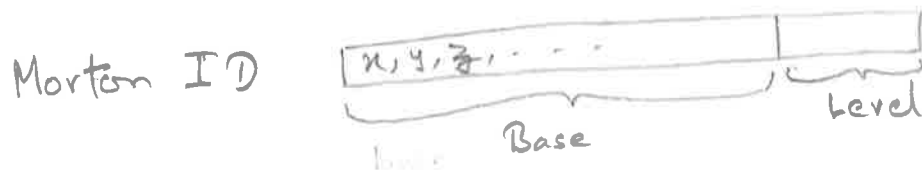


Morton Orderings

Notes:

- (1) Assume K -D space (ex: $K=2$ for quad trees, $K=3$ for octrees etc.)
- (2) Any point will need K coordinates (ex: x, y, z, \dots)
- (3) We are using binary representation of the Morton IDs
- (4) For each level in the tree, we will need 1 bit per coordinate
- (5) For L levels, we need L bits per coordinate (ex: L bits for x , L bits for y ... etc.)
- (6) For all ' K ' coordinates we need $L * K$ bits to represent a point
- (7) For L levels we need $\lceil \log L \rceil$ bits
- (8) Total size of Morton ID will be $[L * K + \lceil \log L \rceil]$ bits
- (9) For any node, given its anchor (say, lower left corner) and its level, we can form the Morton ID by bit interleaving and appending the level.
- (10) When sorted, Morton IDs of parents appear before IDs of the children (pre-order traversal)



#1 Determine ancestor

Given - nodes n_1 & n_2 at levels l_1 & l_2 respectively.

We know $l_1 < l_2$

Morton ID of n_i is $\boxed{b_i} \boxed{l_i} = MID_i$
base level

We have —

For n_1 to be an ancestor of n_2

First $(l_1 * k)$ bits of MID_1
 $==$ First $(l_1 * k)$ bits of MID_2

and,
 Remaining bits $((L * k) - (l_1 * k))$ of $MID_1 ==$ all zeros

This is an $O(1)$ computation.

Ex; from the picture, say n_1 @ (6,4) level 2
 n_2 @ (7,5) level 3

$MID_1 = \overbrace{110100}^{\text{base } b_i} \overbrace{10}^{\text{level } l_i}$

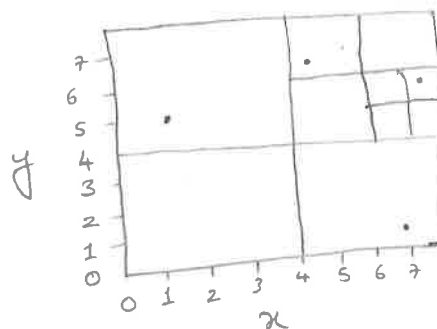
$MID_2 = 110111 \quad 11$

$k=2$ for quad tree

$l_1 = 2 \rightarrow$ So, first $l_1 * k = 4$ bits of two morton IDs are 1101
 $l_2 = 3$ And, remaining bits of MID_1 base are all zeros.

We can extend this to octrees or any other value of k

Sample Quad Tree



A point is (x, y)

Some examples from this picture —

- (a) root @ (0,0) level 0
- (b) node @ (4,4) level 1
- (c) node @ (4,4) level 2
- (d) node @ (7,5) level 3

2(a) Find colleagues of node n_1

Say n_1 is at level l_1

In the octree a node has 8 children (2^K with $K=3$)

For any node n_1 all siblings at l_1 are colleagues.

These colleagues can be enumerated by their morton IDs

If we sweep the bits $((l_1-1)*K)+1$ to (l_1*K) with values 0 to 2^K-1

we get morton IDs of all siblings.

We can then do parallel binary search on all IDs to get the corresponding nodes.

This will give $O(\log n)$ if we search on $P = 2^K - 1$ for octrees

This treatment is really for finding siblings

Colleagues can be nodes outside of siblings too

Next page treats that problem.

#2(b) Find colleagues of node n_1

* For an octree, $k=3$, a node can have at most $3^k - 1$ colleagues.

* For each node we have a measure 's', where 's' is the length of one edge of the node

(This a function of total sample space and level. With each increasing level, the size 's' halves)

Say, node n_1 @ level l_1 has size s_1 and n_1 location is (x, y, z)
possible colleagues are at
$$\left. \begin{aligned} x &= x - s_1, x, x + s_1 \\ y &= y - s_1, y, y + s_1 \\ z &= z - s_1, z, z + s_1 \end{aligned} \right\} \begin{aligned} &3 \times 3 \times 3 = 27 - 1 = 26 \\ &\text{minus 1 for } n_1 \text{ itself.} \end{aligned}$$

By definition, all neighbors should be at level l_1 .

Based on $3^k - 1$ coordinates and level number we can compute morton IDs of all possible colleagues in time $O(1)$

We can then search for each colleague using binary search in time $O(M \log N)$ where $M = 3^k - 1$ for K-D tree.

Running M binary searches in parallel will give complexity $O(\log N)$

#3 Least Common Ancestor

Given nodes n_1 & n_2
with levels $l_1 \geq l_2$

We assume, parents are at a lower level than children.

Algorithm

(1) Find ancestor of n_1 @ level l_2 and assign to n_1
if $l_1 > l_2$ else ignore.

(2) Now that both n_1 & n_2 are at level l_2 we can use the bit manipulation technique used in #1 for finding ancestors—

- (a) we can XOR morton IDs of n_1 & n_2
- (b) the string of ϕ 's at the MSB end represent MSB's that are equal between MID_1 & MID_2
- (c) These MSBs represent the bits that belong to the ancestor with remaining bits equal to all zeros.

This is an $O(1)$ computation.

#4 Rank of Children

* Rank of element a_i is
of elements in $A \leq a_i$

* Morton IDs of children is $>$ Morton ID of parent
(LSBs get added and level increases)

Given an array $A(N)$ with elements $a_i = \text{Morton ID of node } n_i$
 $\forall i = 0, \dots, N-1$

(b) array is sorted.

In the sorted array, the node after the parent can be —

① child

② sibling

③ node from different ancestor.

We will check for ancestor-child relationship between node i and all entries following i (same technique as #1)

rank_list = []

ref = $a(i)$, ref_rank = rank($a(i)$)

$i++$

while (is-ancestor(ref, $a(i)$)) {

rank_list.append(++ref_rank)

$i++$

}

// loop breaks when we run into a non-child.

// loop termination needs to accommodate size-of-array (not shown).

The loop is sequentially $O(N)$ but it can be embarrassingly parallel to approach depth $O(1)$