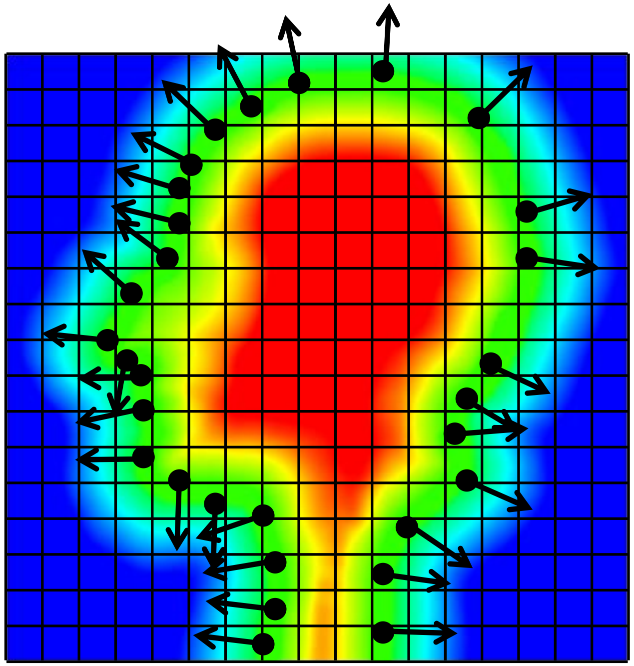
Assignment 2

Morton Codes & Hashbased Octrees

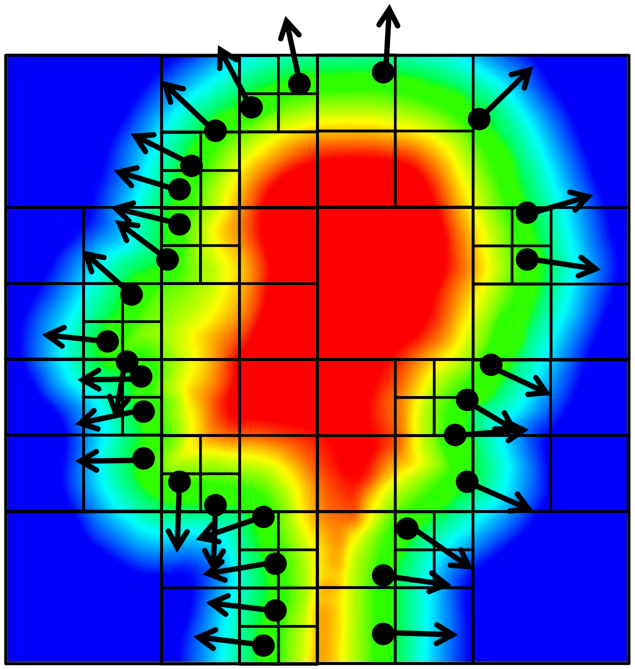
Octree

- Uniform grid inefficient
  - Cubic complexity (in 3D)
  - Many empty cells



Octree http://en.wikipedia.org/wiki/Octree

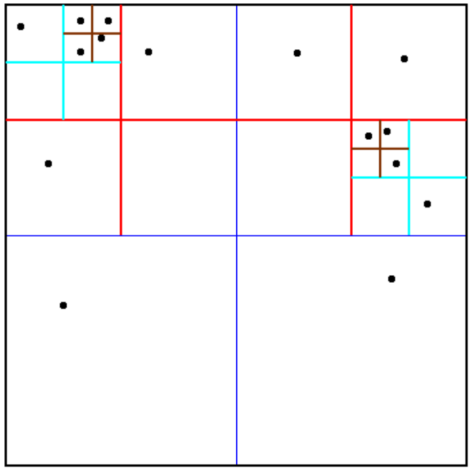
- Hierarchically split volume into 8 octants
- Visualization in 2D (quadtree)

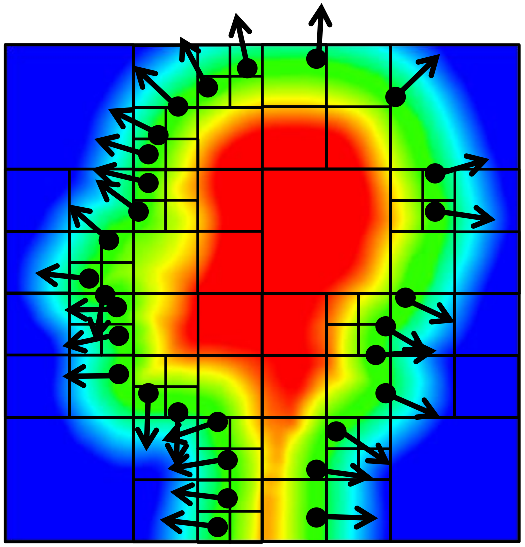


Octree with maximum M=5 levels

Octree

- Applications
  - Acceleration structure
  - Next neighbor retrieval
  - Adaptive resolution





Octree http://en.wikipedia.org/wiki/Octree

Splitting criteria

1. Until maximum level
   - Number of maximum levels M
   - Resolution up to $2^{M\text{-}1}$ x $2^{M\text{-}1}$ x $2^{M\text{-}1}$
   - Typically M=9, larger for complex objects
2. Maintain minimum samples per node
   - Typically 1 to 5
   - For noisier inputs 5 to 10

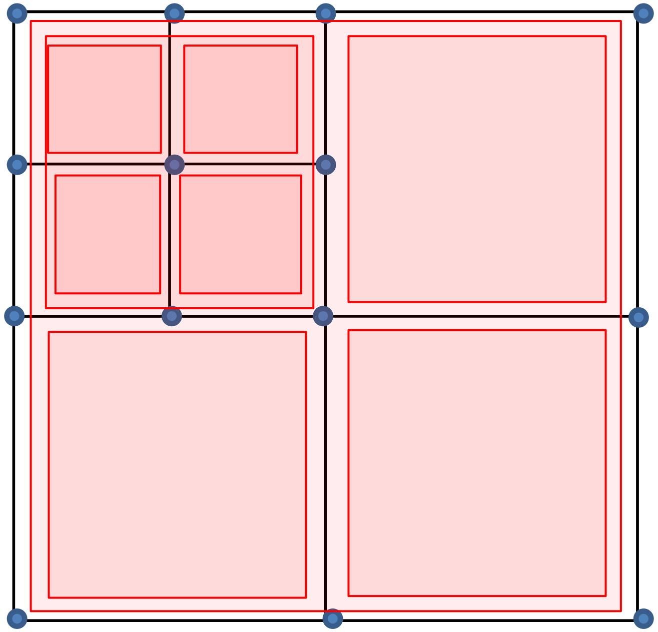
Hashtable based Octree

- Pointerless data structure with hashing
- Very fast & flexible navigation through the tree

- Following work by Lewiner et al. http://zeus.mat.puc-rio.br/tomlew/pdfs/fastdualoctree_sgp.pdf

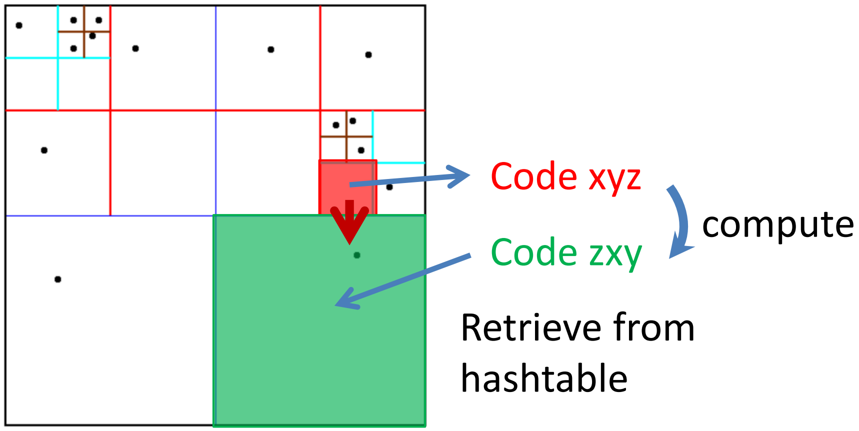
Hashbased Octree

- Octree Cells & Octree Vertices are stored in Hashtables, the hashes are given by morton codes



Hashbased octree

- Navigation: compute target hash and retrieve target cell instead of following links.



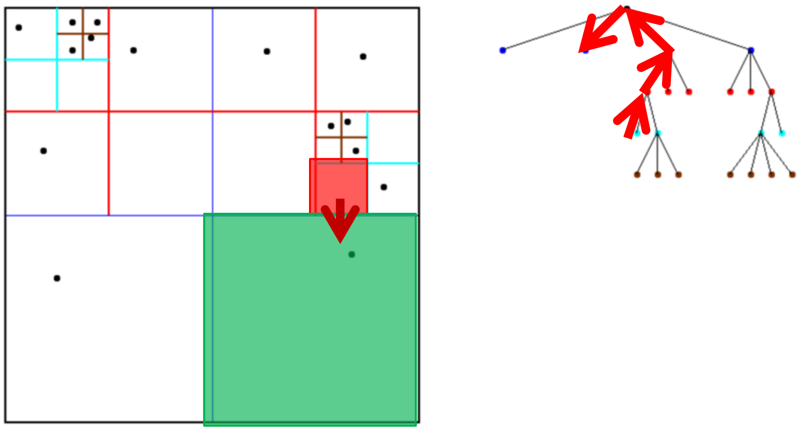
Code xyz

compute

Code zxy

Retrieve from hashtable

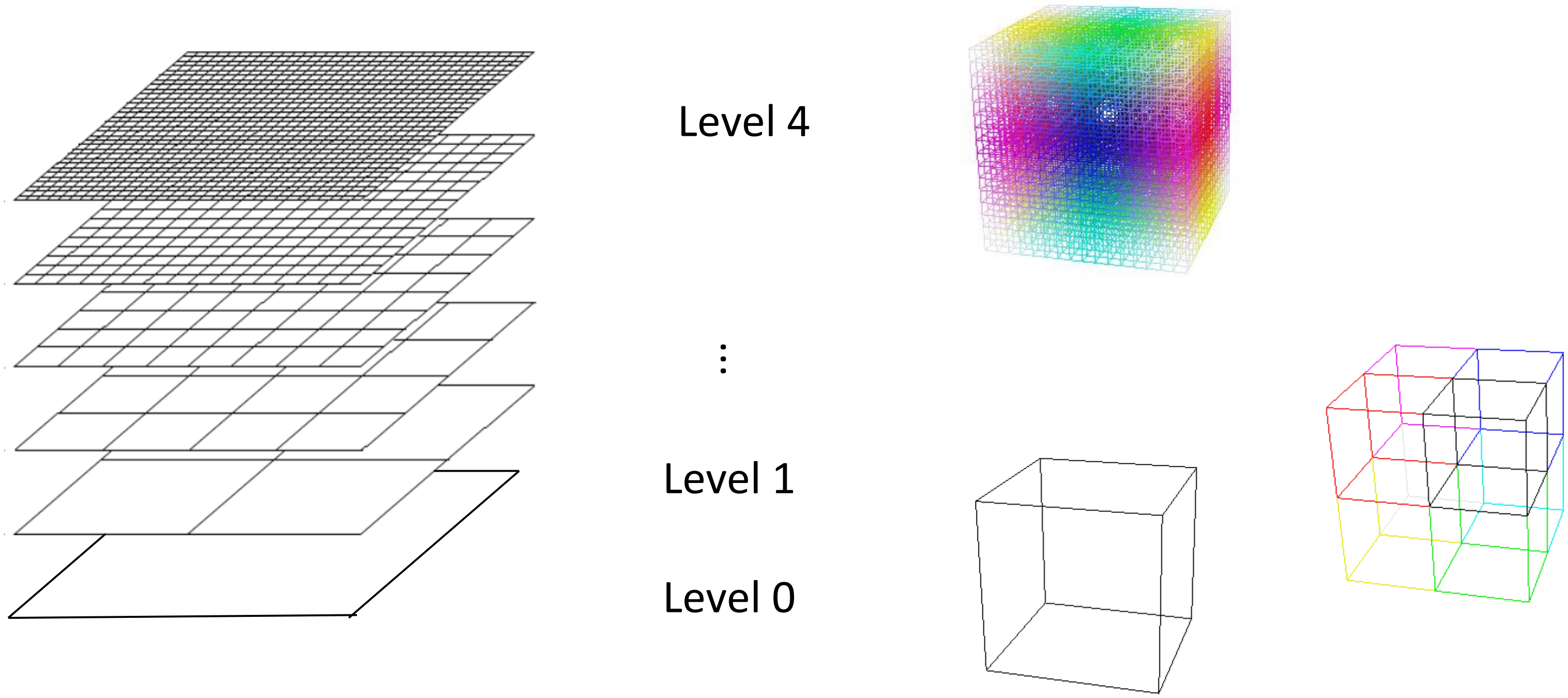
Hash based



Link based

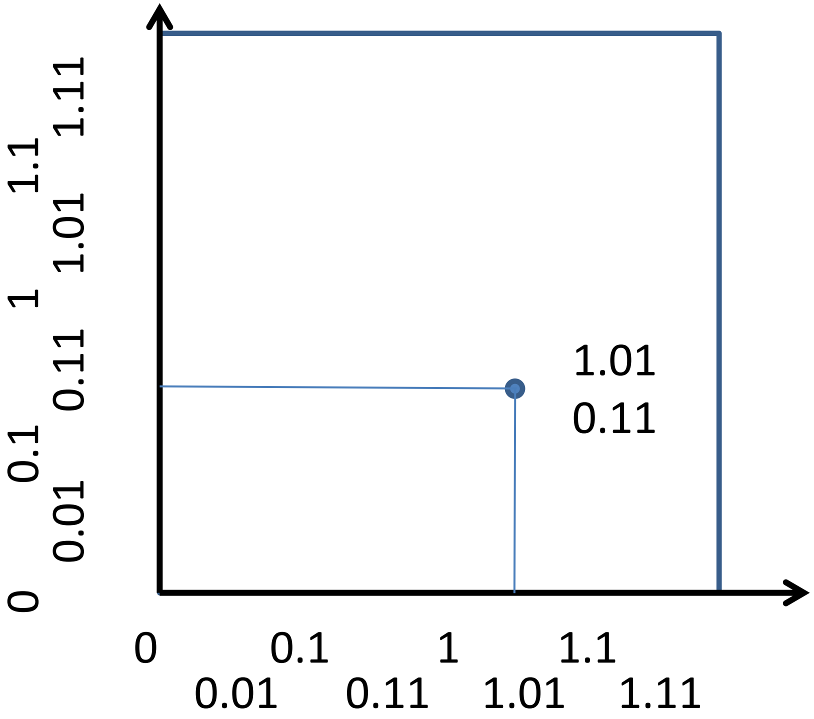
Morton Codes

- Integer that uniquely encodes level and position of a cell in a multigrid



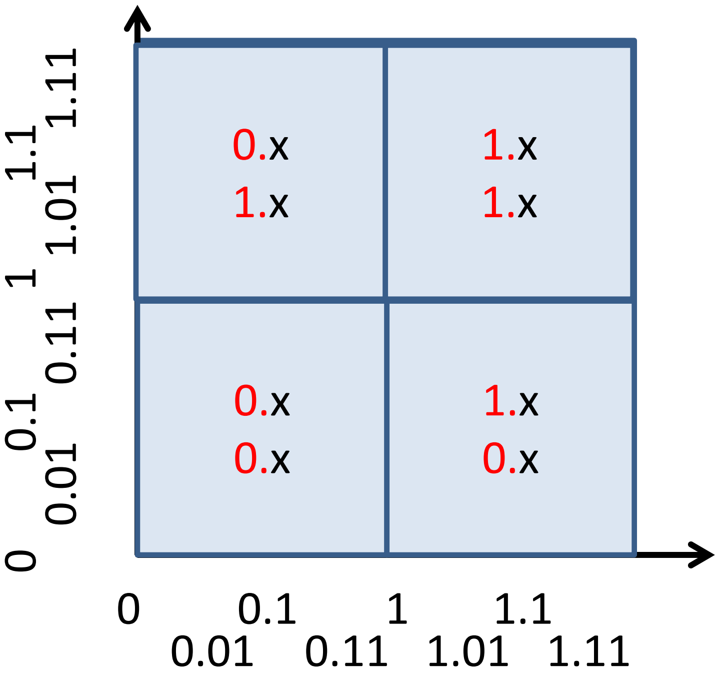
Morton codes

- Assign a coordinate system expressed with fractions in a binary system

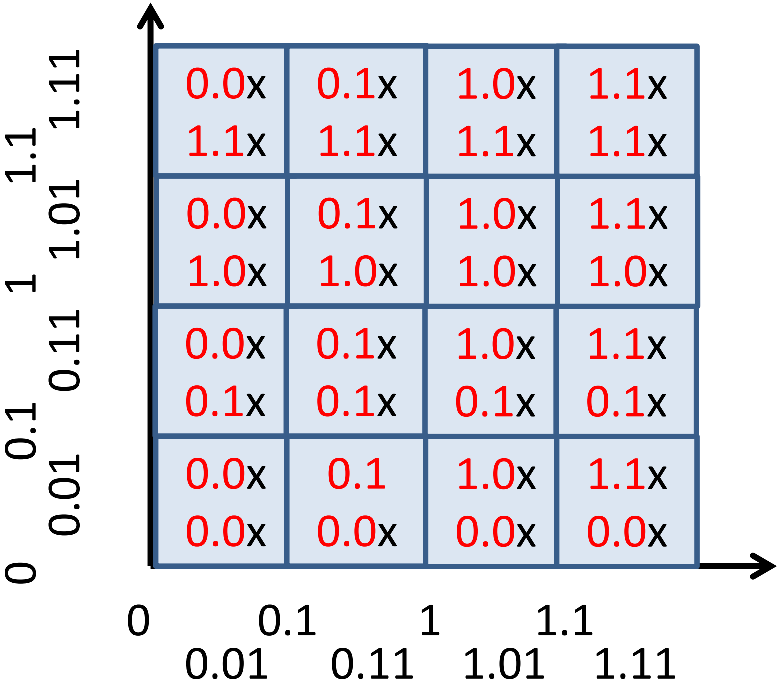


Morton codes

- Coordinates in a multigrid cell share first bits:



0.x 1.x	1.x 1.x
0.x 0.x	1.x 0.x



0.0x 1.1x	0.1x 1.1x	1.0x 1.1x	1.1x 1.1x
0.0x 1.0x	0.1x 1.0x	1.0x 1.0x	1.1x 1.0x
0.0x 0.1x	0.1x 0.1x	1.0x 0.1x	1.1x 0.1x
0.0x 0.0x	0.1 0.0x	1.0x 0.0x	1.1x 0.0x

Morton codes

- Morton codes of multigrid cells:
  - Swizzeling coordinates + 1 delimiter
  - Our convention
    - 1 xy xy xy xy
    - 1 xyz xyz xyz xyz in 3D

0.10x
0.01x

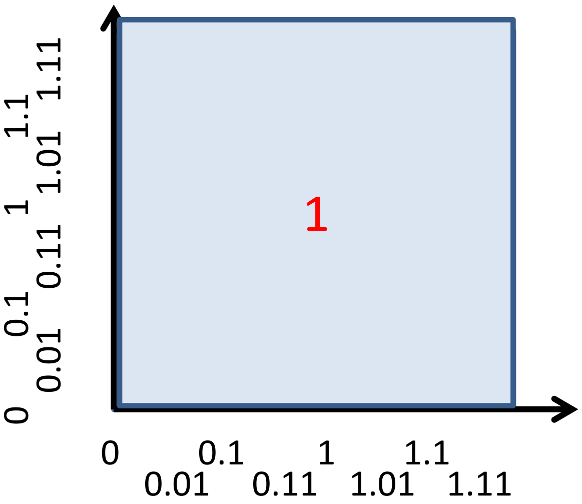
1 00 10 01

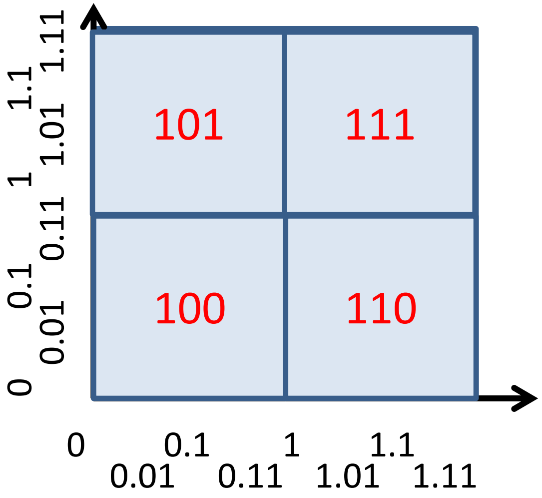
Morton codes

- 1 Delimiter:
  - 0000, 00000000 = 0
  - With delimiter: 10000 vs 100000000

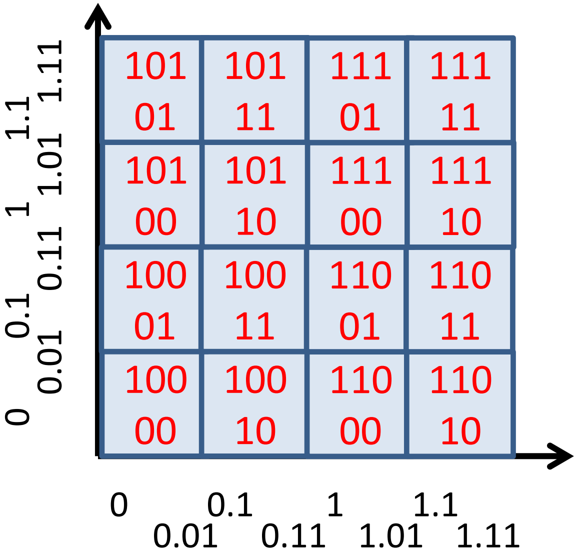
Morton Codes

- Final Morton codes of the first three levels





101	111
100	110



101 01	101 11	111 01	111 11
101 00	101 10	111 00	111 10
100 01	100 11	110 01	110 11
100 00	100 10	110 00	110 10

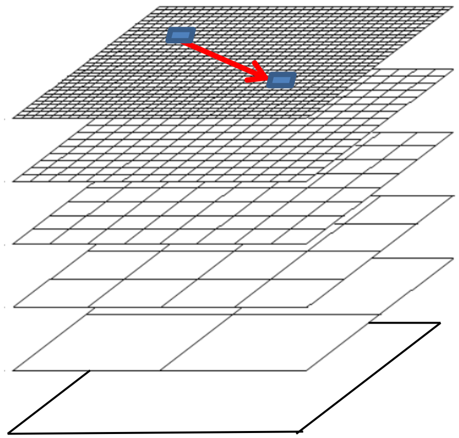
Navigation

- Via morton code manipulations
- Go to parent cell: simple bit shift
  - C>>3 in R^3, C>>2 in R^2.
- Go to neighbors....

Navigation (‚brute force‘)

- From cell to cell, for a given difference vector

$$\searrow = \begin{pmatrix} 1.010 \\ -0.110 \end{pmatrix}$$



1 01 00 11 01 00

unswizzle

$$\begin{pmatrix} 0.0100 \\ 1.0110 \end{pmatrix} + \begin{pmatrix} 1.010 \\ -0.110 \end{pmatrix} = \begin{pmatrix} 1.1000 \\ 0.1010 \end{pmatrix}$$

reswizzle

1 10 11 10 01 00

- Unswizzeling/reswizzeling inefficient.

Navigation (good)

- Bitwise manipulations
- Dilated addition/subtraction:
  - Ignore the bits where a mask is 0.

$$x\bar{+}y = \underbrace{((\underbrace{(x|\sim mask)}_{\text{Set masked bits 1}} + \underbrace{y \mathbin{\&} mask}_{\text{Ignore masked bits}}) \mathbin{\&} mask)}_{\text{Computes the correct value for the bits denoted by mask}} \underbrace{|(x \mathbin{\&} \sim mask)}_{\text{Reset other bits to original value}} .$$

$$x\bar{-}y = ((\underbrace{(x \mathbin{\&} mask)}_{\text{Set masked bits 0}} - y \mathbin{\&} mask) \mathbin{\&} mask)|(x \mathbin{\&} \sim mask).$$

  - Intuition: masked bits are set to 1 with addition, to 0 in subtraction, such that added/subtracted bits carry to the next unmasked bit

Navigation (good)

- Example (try it out with pen and paper)

$$x = 0b\ 0\underline{00}\ 0\underline{11}\ 1\underline{11}$$
$$y = 0b\ 1\underline{11}$$
$$mask = 0b\ 100\ 100\ 100$$
$$x\bar{+}y = 0b\ 0\underline{00}\ 1\underline{11}\ 0\underline{11}$$

Legend: $\underline{\quad}$ ignored bits

- When dropping the last term of the formula, the masked bits are set to zero

$$x\tilde{+}y = \underbrace{((\underbrace{(x|\sim mask)}_{\text{Set masked bits 1}} + \underbrace{y\ \&\ mask}_{\text{Ignore masked bits}})\ \&\ mask)}_{\text{Computes the correct value for the bits denoted by mask}}$$

$$x\tilde{-}y = ((\underbrace{(x\ \&\ mask)}_{\text{Set masked bits 0}} - y\ \&\ mask)\ \&\ mask).$$

$$x = 0b\ 000\ 011\ 111$$
$$y = 0b\ 111$$
$$mask = 0b\ 100\ 100\ 100$$

$$x\tilde{+}y = 0b\ 0\underline{00}\ 1\underline{00}\ 0\underline{00}$$

Navigation (good)

- For the navigation, manipulate x bits, y bits and z bits separately:

$$\begin{aligned} & \quad\; xyz\ xyz\ xyz \\ x &= 0b1\ 000\ 101\ 111 \\ y &= 0b011 \\ & \quad\; xyz\ xyz\ xyz \\ mask_x &= 0b\ 100\ 100\ 100 \\ mask_y &= 0b\ 010\ 010\ 010 \\ mask_z &= 0b\ 001\ 001\ 001 \end{aligned}$$

Addition

$$Result = (((x| \sim mask_x) + (y\&mask_x))\&mask_x)|$$
$$(((x| \sim mask_y) + (y\&mask_y))\&mask_y)|$$
$$(((x| \sim mask_z) + (y\&mask_z))\&mask_z)$$

Subtraction

$$Result = (((x\&mask_x) - (y\&mask_x))\&mask_x)|$$
$$(((x\&mask_y) - (y\&mask_y))\&mask_y)|$$
$$(((x\&mask_z) - (y\&mask_z))\&mask_z)$$

Navigation (good)

- To navigate through morton codes:
  - Use swizzeled difference vector (no delimiter)

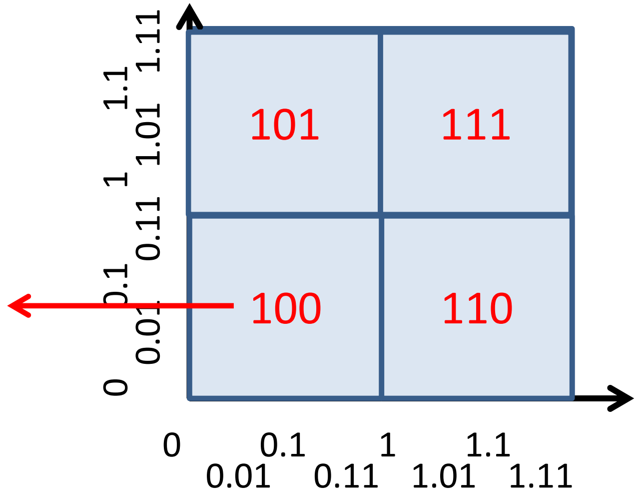
$$y = \begin{pmatrix}1.010\\0.110\end{pmatrix} \quad 10\;01\;11\;00$$

  - And use bit magic to modify xyz bits of a morton code separately.

$$\begin{gathered}Result = (((x| \sim mask_x) + (y\&mask_x))\&mask_x)|\\(((x| \sim mask_y) + (y\&mask_y))\&mask_y)|\\(((x| \sim mask_z) + (y\&mask_z))\&mask_z)\end{gathered}$$

Check for Overflows!

- A level k morton code +/- a difference vector should produce a level k Morton code.



Subtraction =>
000
INVALID code

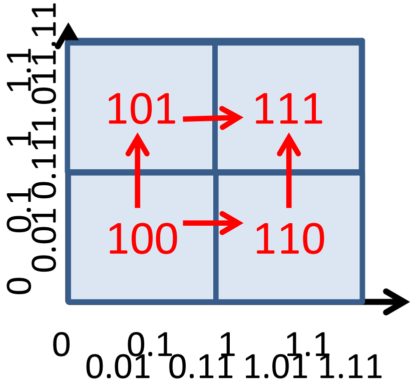
- *C = 0b1 xyz xyz … xyz* is a level k Morton code iff (C >> (3*k)) == 0b1

Separate Addition and Subtratcion

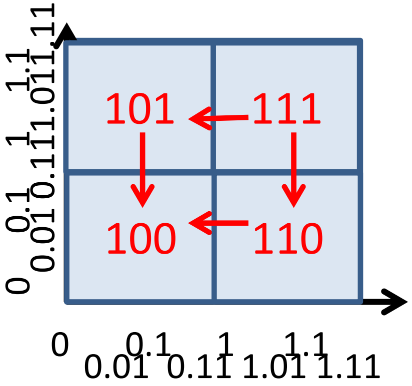
- Addition & subtraction need to be handled separately with dilated addition / subtraction

$$y = \begin{pmatrix} 1.010 \\ -0.110 \end{pmatrix}$$

- In the Java code: write separate methods to compute positive & negative neighbors



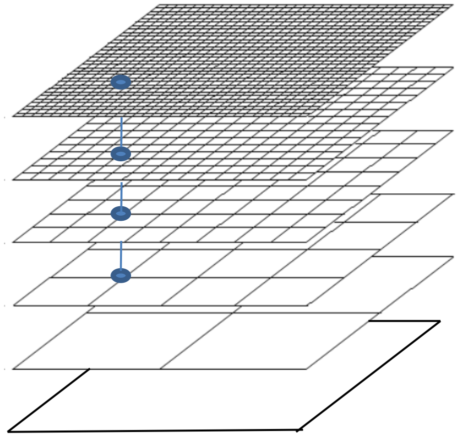
Positive negihbor



Negative neighbor

Vertex Codes

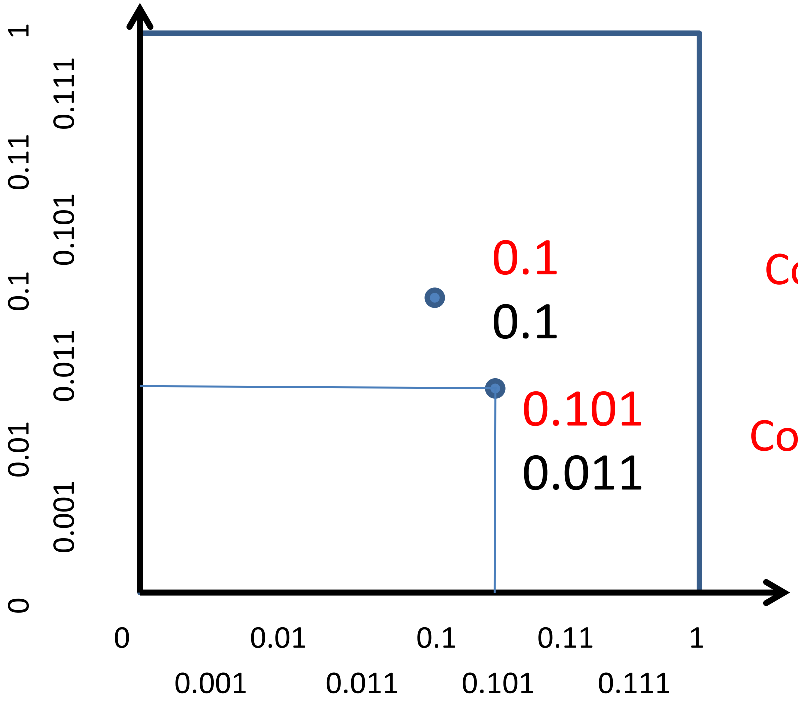
- Vertices do not belong to one fixed level, but to all levels above some level.



- To get a unique code: Restict maximal multigrid layer and use highest level Morton codes for all vertices.
  - Equal to padding lower level code with zeros

Vertex Codes

- Vertex code intuition: 1 prefix, swizzle together coordinates & pad to the maximal lenght.

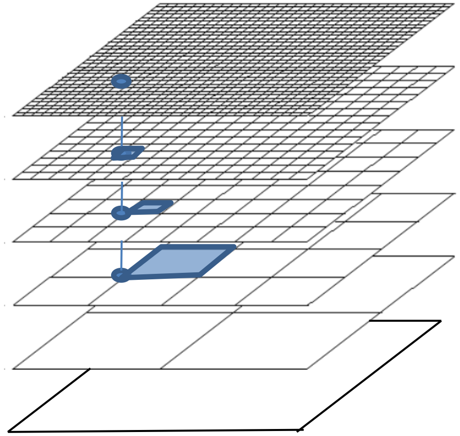


Code: 100000000

Code: 100100111

Vertex Codes (Navigation)

- Vertex -> cell and cell -> vertex:
  - Unpadded vertex code = cell code
  - Padded cell code = vertex code



Vertex code

0b1 00 11 00 00 00

Cell code

0b1 00 11 00 00 00

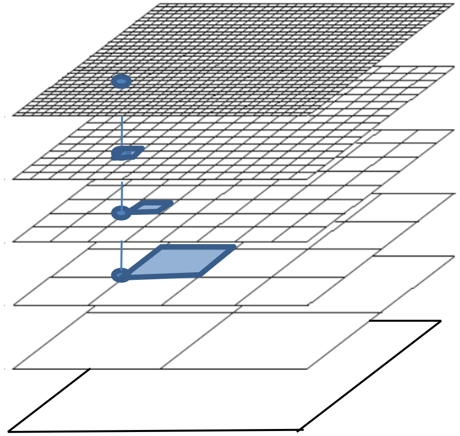
0b1 00 11 00 00

0b1 00 11 00

0b1 00 11

Vertex Codes (Navigation)

- Navigation:
  - Unpad code to appropriate cell level
  - Cell to cell navigation
  - Repad

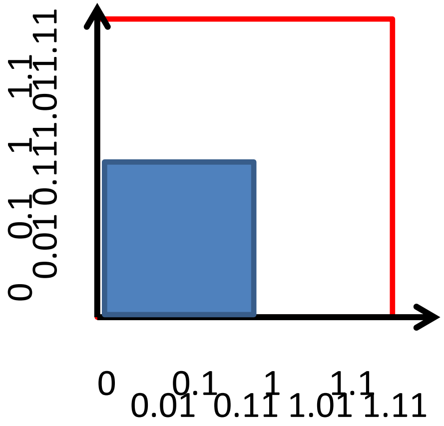


Vertex code
0b1 00 11 00 00 00

Cell code
0b1 00 11 00 00 00
0b1 00 11 00 00
0b1 00 11 00
0b1 00 11

Hashbased Octree

- Octree: subset of multigrid
- Problem:
  - because of 1-delimiter: we cannot assign vertex codes on the red boundary



  - Solution: use the codes in first quadrant only.

Codes in Hashbased Octree

	Morton Code:	level
root cell	0b1000	1
general cell	$0b1\,\underbrace{000\ xyz\ xyz \cdots xyz}_{3k\ bits}$	k

	Morton Code:	maxLevel	minLevel
vertex	$0b1\ \underbrace{\underbrace{xyz \cdots xyz}_{3 \cdot minLevel\ bits}\ 000 \cdots 000}_{tree.depth \cdot 3\ bits}$	$\max_{vertex \in cell} cell.level$	$\min_{vertex \in cell} cell.level$

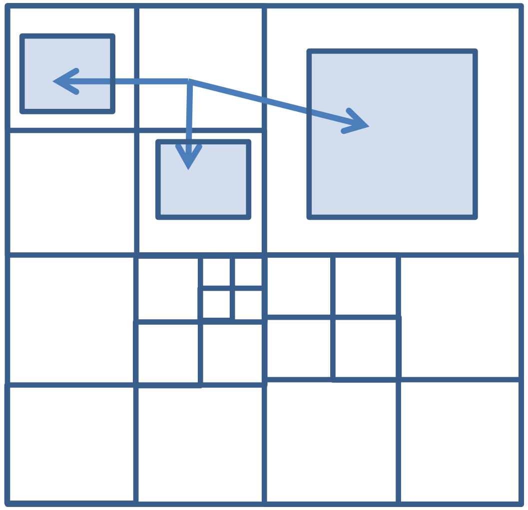
Navigation in the Hashoctree

Find neighbors

1. Compute theoretical Morton code of neighbor
2. Check existence of the associated vertex or cell, if inexistent, decide:
    - retrieve next existing vertex/cell, or
    - return null.

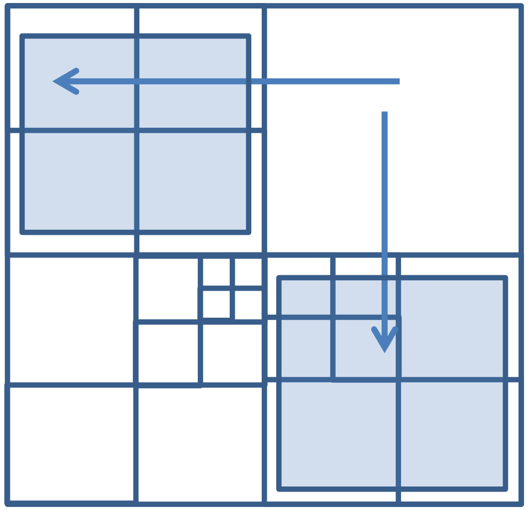
Navigation cell-> cell

- Neighbor cells: find a neighbor cell of less or equal level



Navigation cell-> cell

- Neighbor cells: find a neighbor cell of less or equal level

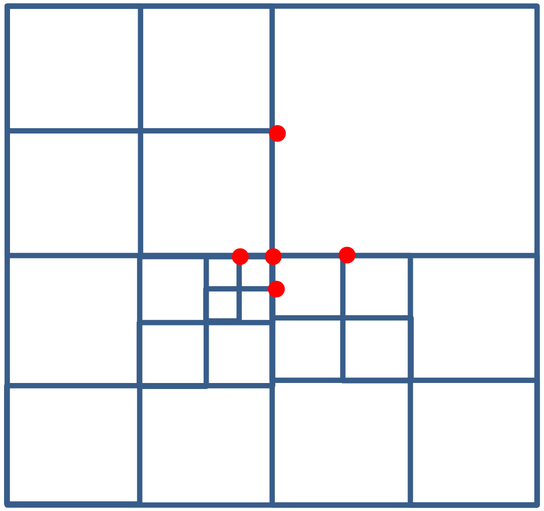


Navigation cell-> cell

- Compute the negihbor code on the same level
- Iterate until the level 0 is reached:
  - Check if the cell exists
  - If not: go to the parent.

Navigation Vertex -> Vertex

- Neighbors: closest vertex in some direction



- Unpad to finest level cell code, compute neighbor cell code, repad, check existence. Iterate up to minimum level of vertex

Bit Arithmetic

- ~ not. | or. >> rightshift. << leftshift
- 64 bit long:
  - First bit decides about positive, negative.
  - Negatives are coded as ~ (positive - 1).
    - -4 = ~(3) = ~(0b000...0011) = 0b111...1100
    - -1 = 0b(111111...111)
    - ~(-1 << k) = 0b(00...011111) can be a handy mask

Summary

- Dilated Operations:

$$x\tilde{+}y = \underbrace{((\underbrace{(x| \sim mask)}_{\text{Set masked bits 1}} + \underbrace{y\ \&\ mask}_{\text{Ignore masked bits}})\ \&\ mask)}_{\text{Computes the correct value for the bits denoted by mask}}$$

$$x\tilde{-}y = ((\underbrace{(x\ \&\ mask)}_{\text{Set masked bits 0}} - y\ \&\ mask)\ \&\ mask).$$

$$\begin{aligned} Result = {} & (((x| \sim mask_x) + (y\&mask_x))\&mask_x)| \\ & (((x| \sim mask_y) + (y\&mask_y))\&mask_y)| \\ & (((x| \sim mask_z) + (y\&mask_z))\&mask_z) \end{aligned}$$

- Morton Codes:

	Morton Code:	level
root cell	0b1000	1
general cell	$0b1\,\underbrace{000\ xyz\ xyz\cdots\ xyz}_{3k\ bits}$	k

	Morton Code:	maxLevel	minLevel
vertex	$0b1\ \underbrace{\underbrace{xyz\ \cdots\ xyz}_{3\cdot minLevel\ bits}\ 000\cdots000}_{tree.depth\cdot 3\ bits}$	$\max_{vertex\in cell} cell.level$	$\min_{vertex\in cell} cell.level$

Summary

- I am paid to help
- And use the forum ☺

Questions?