NLeSC eTEC 2020 project – TU Delft nD-pointclouds – Project meeting

Storing SFC-based point clouds with cLoI and offering access via an octree-webservice

Martijn Meijers
b.m.meijers@tudelft.nl
Delft University of Technology, The Netherlands

Monday, January 23, 2023
Delft | Faculty of Architecture and the Built Environment
Berlage Room | CET 13:00 – 17:00

More and more points collected...

- Rapid increase in data collection of Point clouds
- Many sources: Different properties
  - LiDAR: x, y, z, intensity, return number, GPS time, ...
  - Photogrammetry: x, y, z, r, g, b, quality of derived point, ...
- Which dimensions frequently queried depends on application (e.g. change detection needs time, segmentation might need normal, some applications may need accuracy of points)
- nD: multi-dimensional points

nD Point Cloud

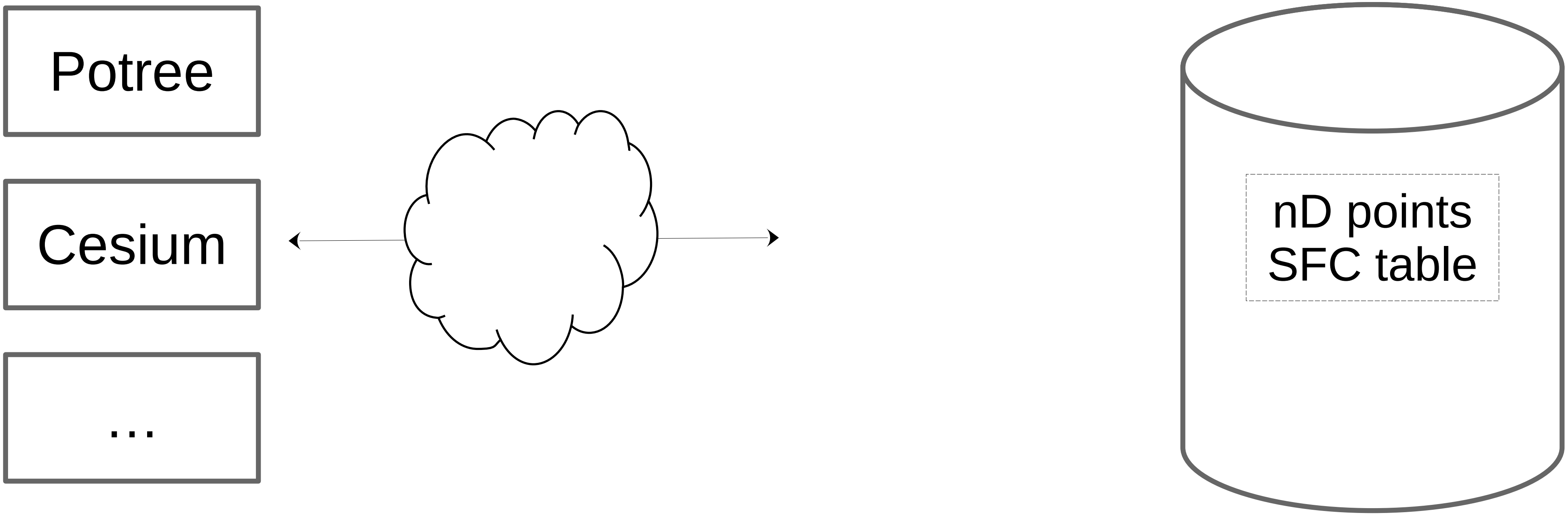
- nD-point cloud data structure result of earlier research:
  - nD points
  - stored inside database
  - access method based on Space-Filling-Curves (SFC): SFC allows mapping from nD to 1D

- Objective of this research:

  Investigate whether developed nD-point cloud data structure can be used to disseminate points to current state-of-the-art 3D point cloud web visualization applications

- Paper: http://resolver.tudelft.nl/uuid:30fc6840-8ca8-4fa4-b39f-9ea4614bfa6a
  Presented at 17th 3D GeoInfo Conference, 3DGeoinfo 2022, Sydney, Australia

PCServe – Software Architecture



Potree

Cesium

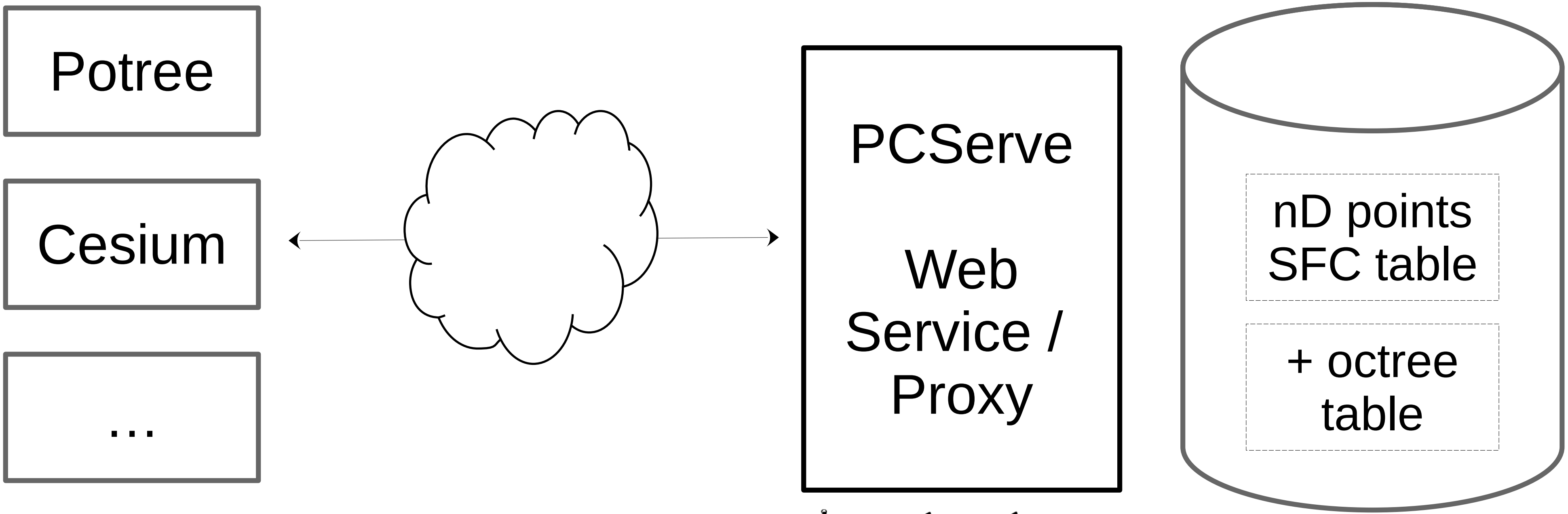
...

nD points
SFC table

WebGL

PostGIS
Spatial PostgreSQL

PCServe – Software Architecture



Potree

Cesium

…

PCServe

Web Service / Proxy

nD points SFC table

+ octree table

WebGL

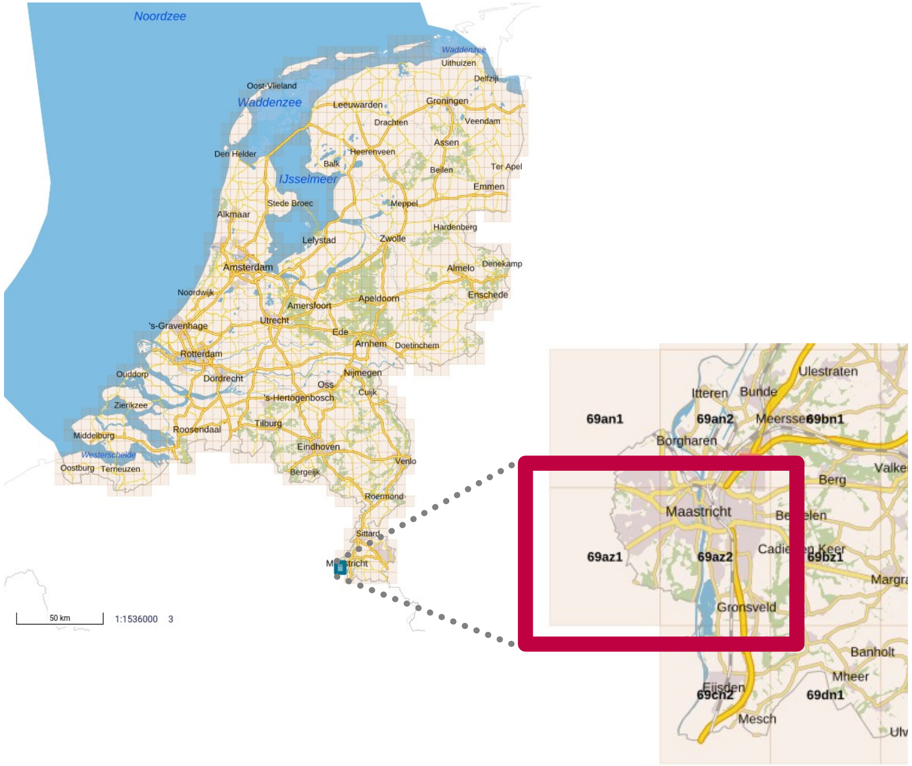
Flask
web development, one drop at a time

python

PostGIS
Spatial PostgreSQL

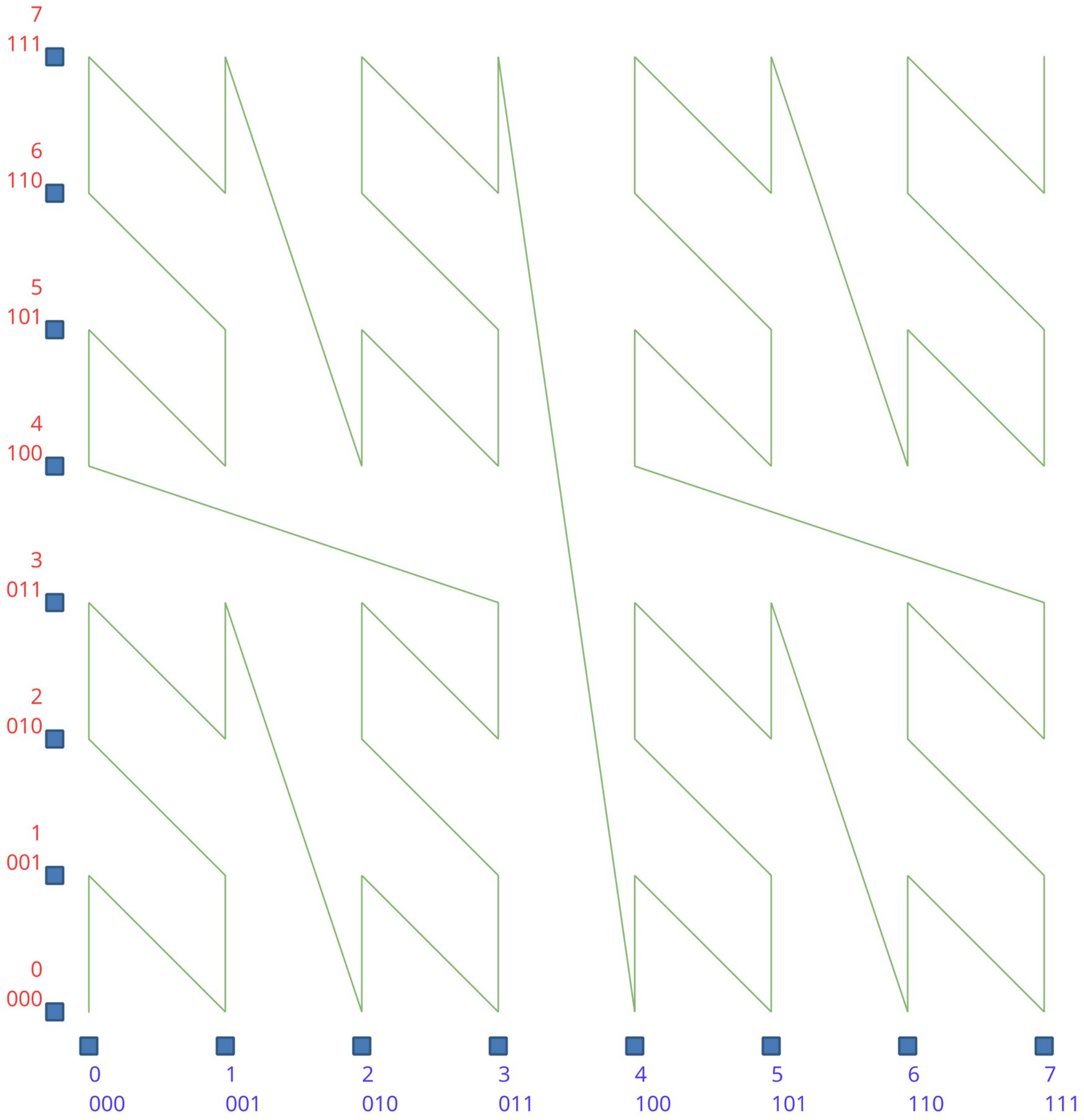
Experiment

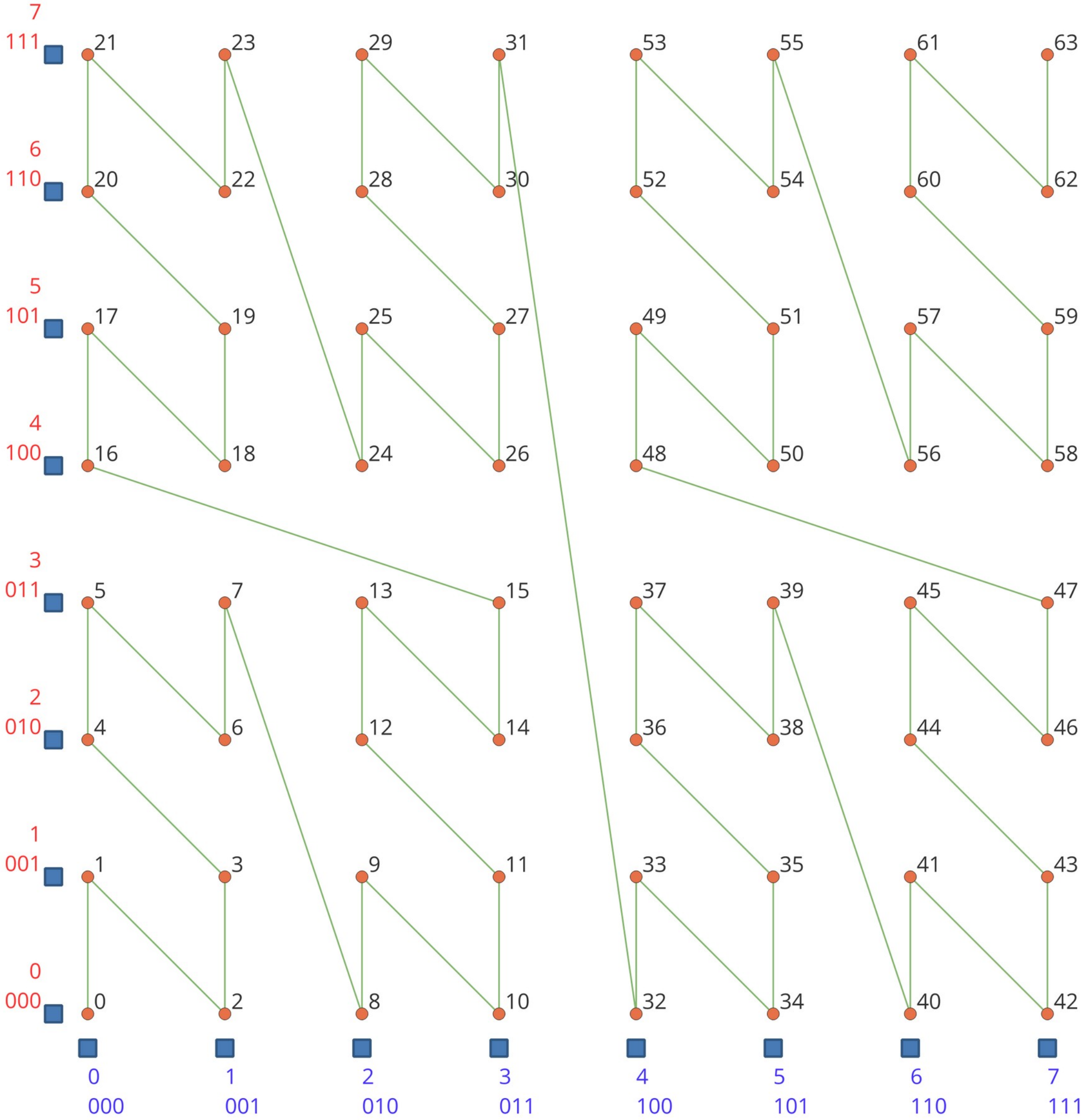
- Use Dutch LiDAR height data set (AHN3)
- Load as nD point cloud structure:
  SFC key – points table
  sorted by SFC (in PostgreSQL)
- Create virtual Octree (octree table)
- Use PCServe (web service) and Potree as visualization client
- Measure performance:
  - Disk / Response size
  - Time of construction
  - Time of data retrieval



nD Point cloud structure – Background
Space Filling Curve (SFC)

Lebesgue / Morton curve

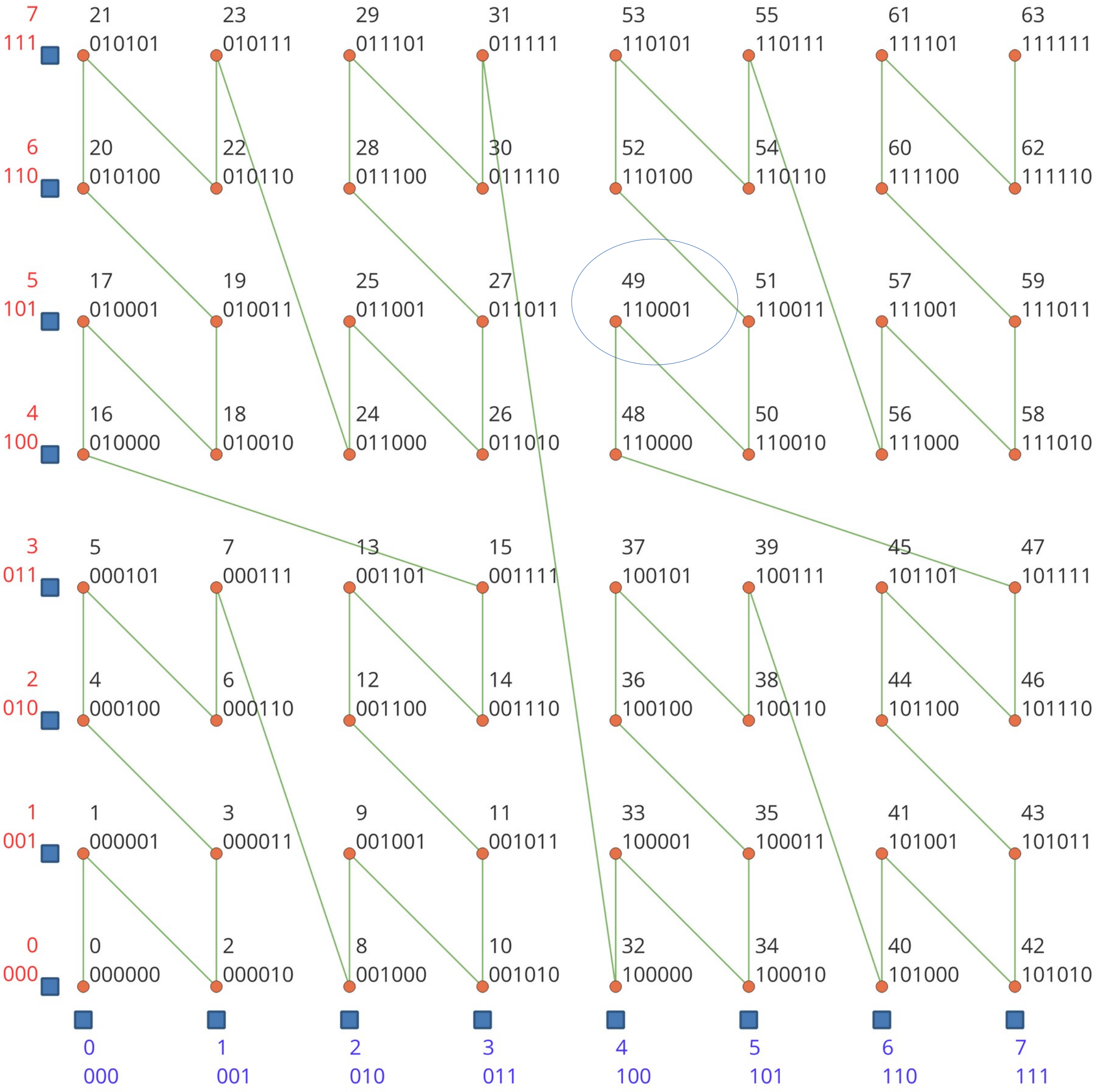



7
111
21
23
29
31
53
55
61
63
6
110
20
22
28
30
52
54
60
62
5
101
17
19
25
27
49
51
57
59
4
100
16
18
24
26
48
50
56
58
3
011
5
7
13
15
37
39
45
47
2
010
4
6
12
14
36
38
44
46
1
001
1
3
9
11
33
35
41
43
0
000
0
2
8
10
32
34
40
42
0
000
1
001
2
010
3
011
4
100
5
101
6
110
7
111

(4, 5) 49

↕ ↕

(100, 101) ↔ 11 00 01



(4, 5)

↕

(100, 101) ↔ $\underline{11}$ 00 01

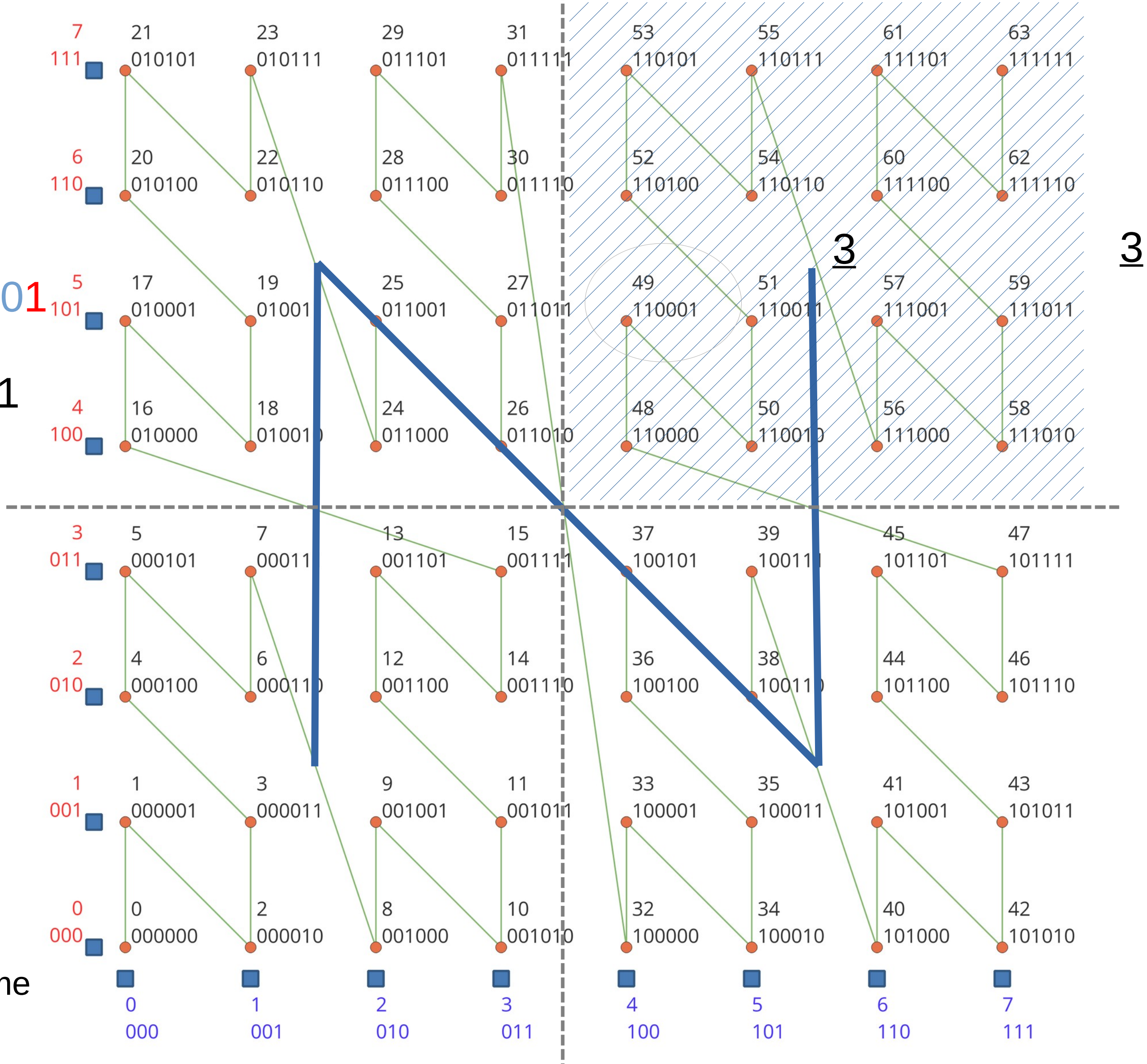
49

↕

$\underline{11}$ 00 01

↕

$\underline{3}$, 0, 1



Upper part Y dim

$\underline{3}$ … … ↔ $\underline{11}$ .. ..

Upper part X dim

SFC key also ‘address’ / materialized path in 2^n tree

Here *n* = 2, but works same for higher *n* (3, 4, 5, ...)

(4, 5)

↕

(100, 101) ↔ 11 00 01

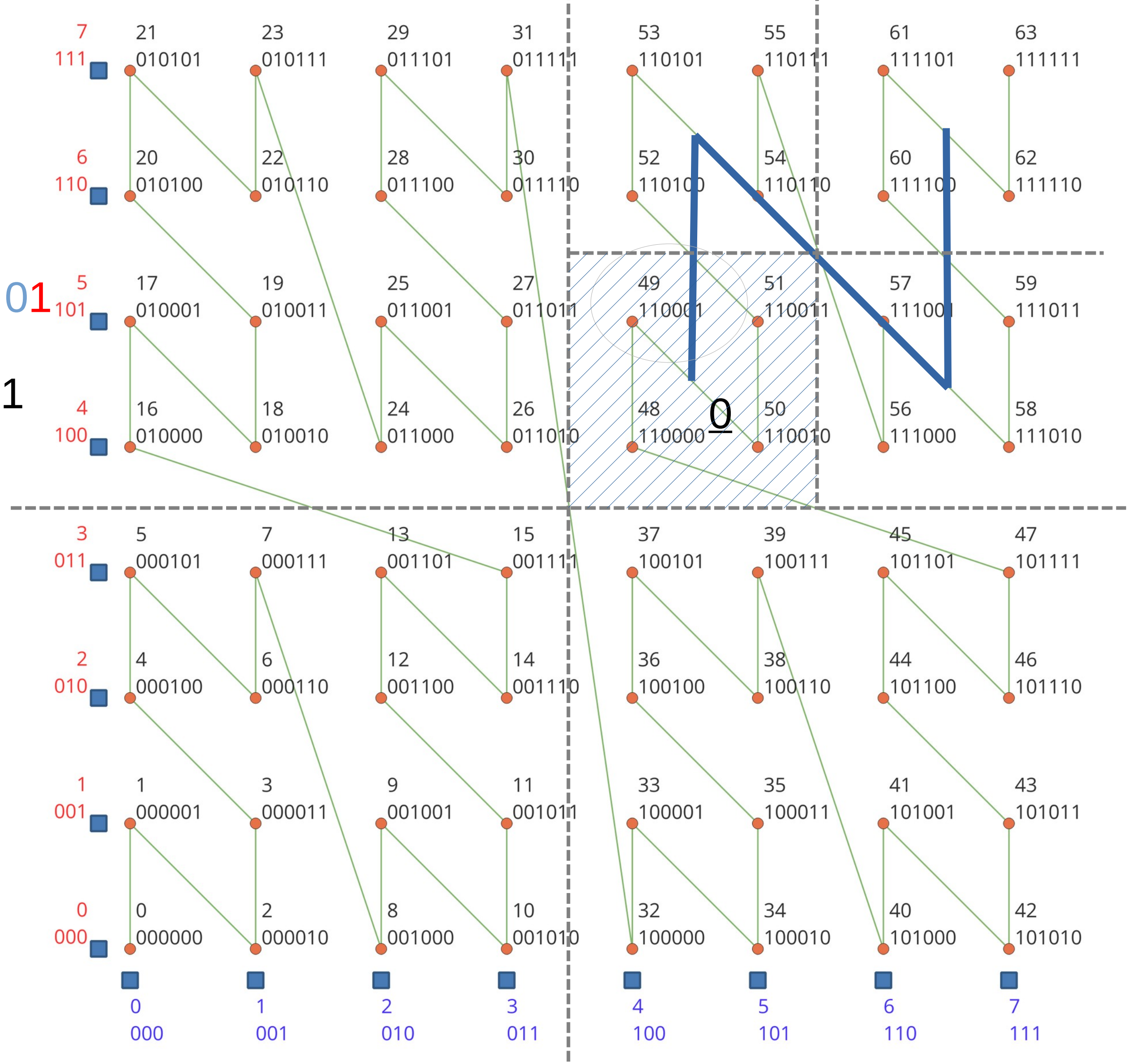
49

↕

11 00 01

↕

3, 0, 1



	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
7 111	21 010101	23 010111	29 011101	31 011111	53 110101	55 110111	61 111101	63 111111
6 110	20 010100	22 010110	28 011100	30 011110	52 110100	54 110110	60 111100	62 111110
5 101	17 010001	19 010011	25 011001	27 011011	49 110001	51 110011	57 111001	59 111011
4 100	16 010000	18 010010	24 011000	26 011010	48 110000	50 110010	56 111000	58 111010
3 011	5 000101	7 000111	13 001101	15 001111	37 100101	39 100111	45 101101	47 101111
2 010	4 000100	6 000110	12 001100	14 001110	36 100100	38 100110	44 101100	46 101110
1 001	1 000001	3 000011	9 001001	11 001011	33 100001	35 100011	41 101001	43 101011
0 000	0 000000	2 000010	8 001000	10 001010	32 100000	34 100010	40 101000	42 101010

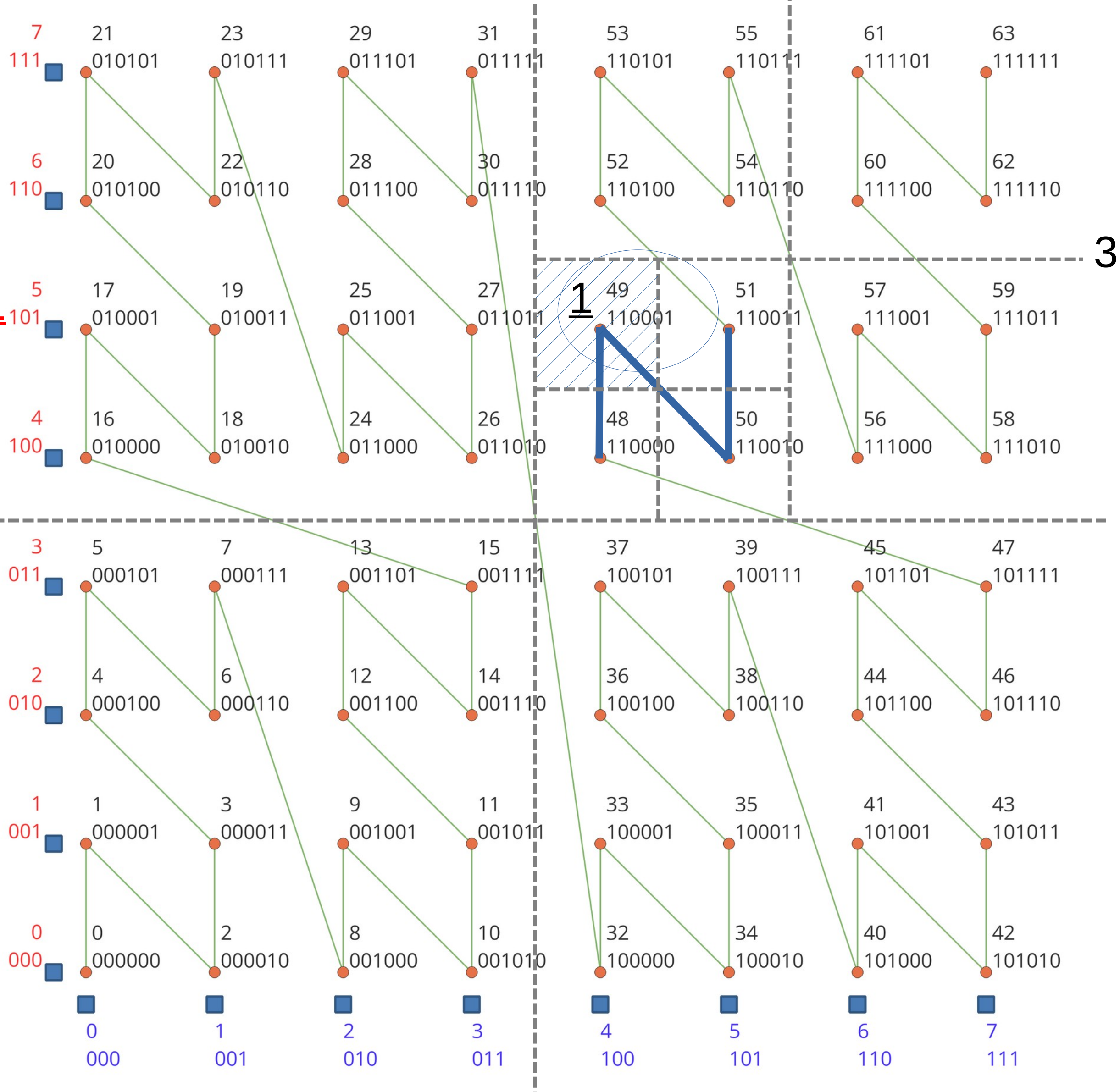
Lower part Y dim

3, 0, ... ↔ 11 00 ..

Lower part X dim

(4, 5) ↕ (100, 101) ↔ 11 00 $\underline{01}$

49 ↕ 11 00 $\underline{01}$ ↕ 3, 0, $\underline{1}$



Higher part Y dim

3, 0, $\underline{1}$ ↔ 11 00 $\underline{01}$

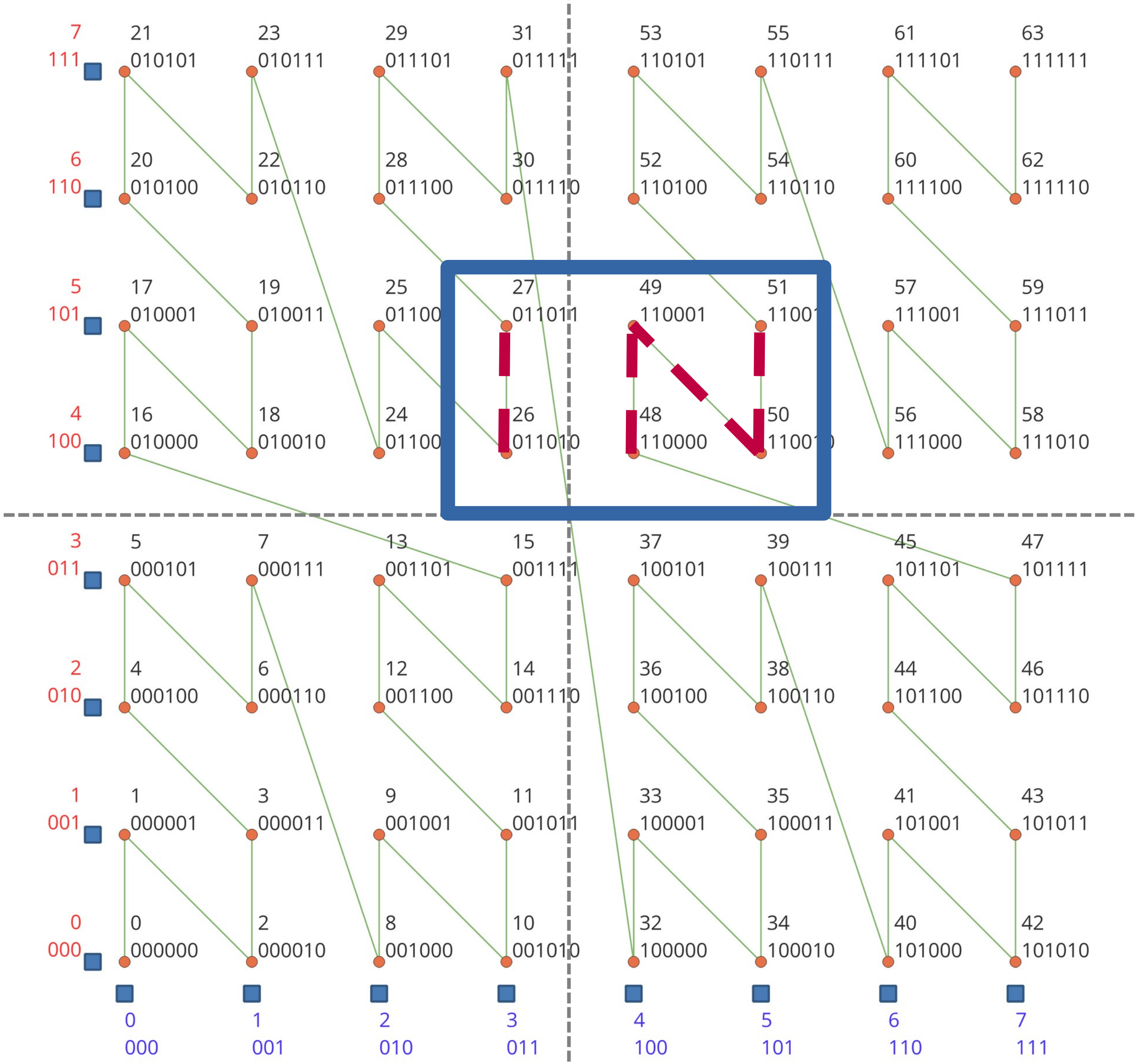
Lower part X dim

SFC key also ‘address’ / materialized path in 2^n tree

Here *n* = 2, but works same for higher *n* (3, 4, 5, ...)

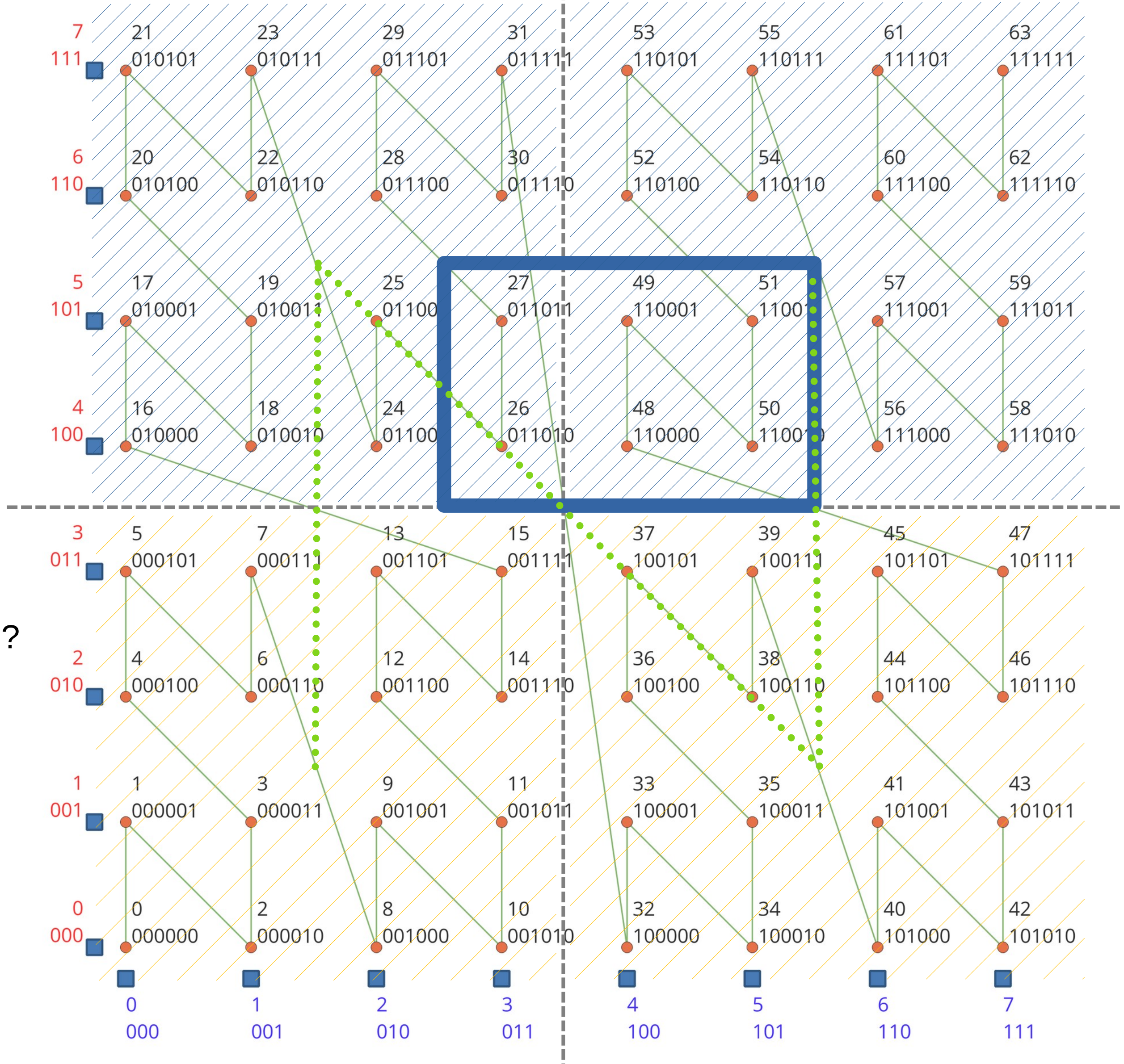
nD Point cloud structure – Background
Space Filling Curve (SFC) – Query Approach

Query:
Box((3,4), (5,5))



Query:
Box((3,4), (5,5))

Ranges:
[16, 32)
[48, 64)



Query box fully contains SFC 2^n tree node?

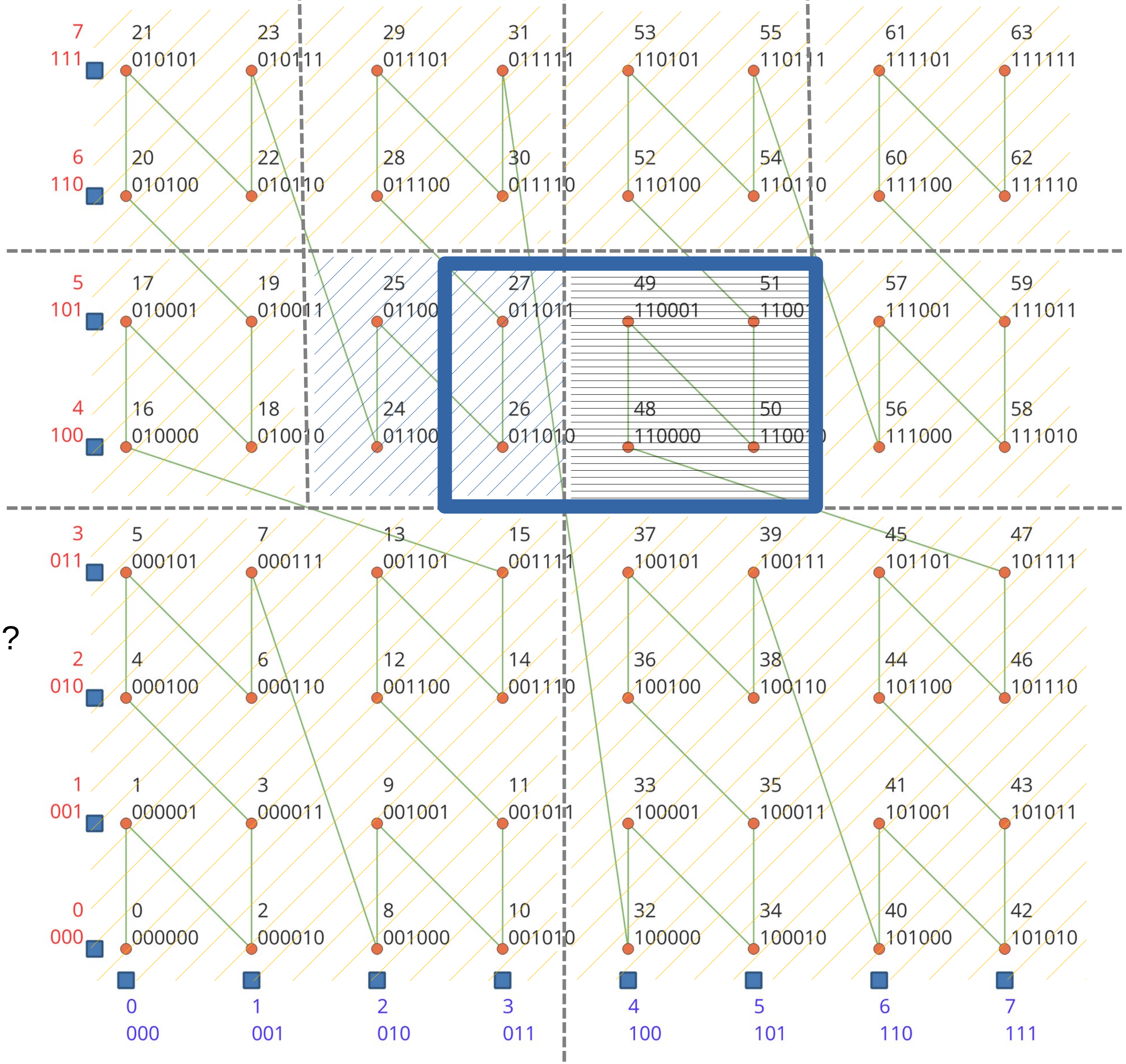
▨ Overlap: refine possible

▨ No overlap

▤ Fully contained

Query:
Box((3,4), (5,5))

Ranges:
[24, 28)
[48, 52)



	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
7 111	21 010101	23 010111	29 011101	31 011111	53 110101	55 110111	61 111101	63 111111
6 110	20 010100	22 010110	28 011100	30 011110	52 110100	54 110110	60 111100	62 111110
5 101	17 010001	19 010011	25 01100	27 011011	49 110001	51 1100	57 111001	59 111011
4 100	16 010000	18 010010	24 01100	26 011010	48 110000	50 1100	56 111000	58 111010
3 011	5 000101	7 000111	13 001101	15 001111	37 100101	39 100111	45 101101	47 101111
2 010	4 000100	6 000110	12 001100	14 001110	36 100100	38 100110	44 101100	46 101110
1 001	1 000001	3 000011	9 001001	11 001011	33 100001	35 100011	41 101001	43 101011
0 000	0 000000	2 000010	8 001000	10 001010	32 100000	34 100010	40 101000	42 101010

Query box fully contains SFC 2^n tree node?

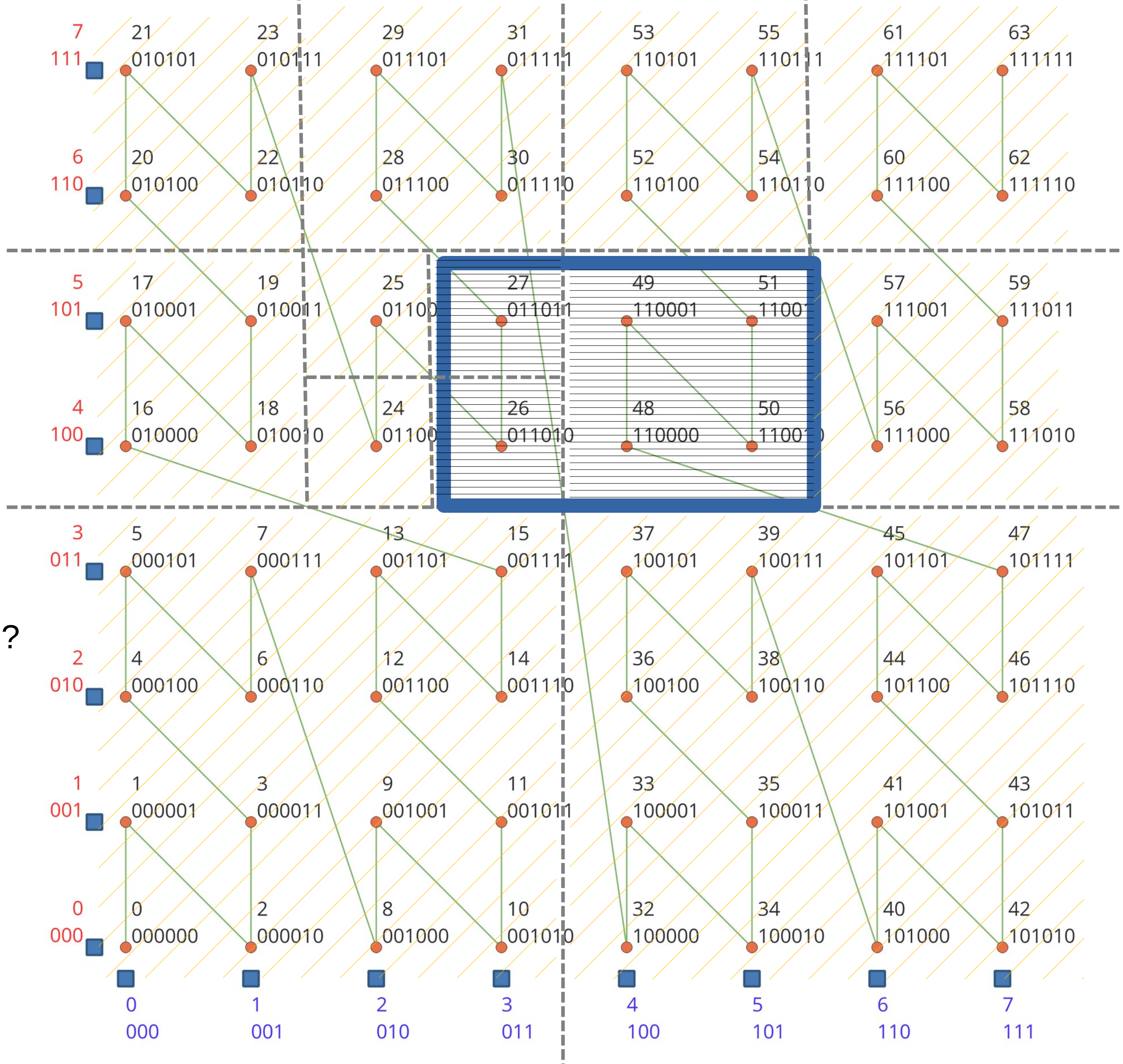
▨ Overlap: refine possible

▨ No containment

▤ Fully contained

Query
Box((3,4), (5,5))

Ranges:
[26, 27)
[27, 28)
[48, 52)



Query box fully contains SFC 2^n tree node?

Overlap:
refine possible

No containment

Fully contained

nD Point cloud structure – Background
Continuous Level of Importance (cLoI)

Continuous Level of Importance

- No Level of Detail mechanism for measured points
- ‘Level organisation thinking’: not so many points in overview level (e.g. for 2D), then next level 4 times (= 2^{nDims}) more points (with 3D → 8 times)
- Add per point: Continuous Level of Importance (cLoI)*
  Continuous (float: e.g. 0.1, 0.2, 3.5, 3.6) values instead of Discrete values (int, e.g. 0, 0, 3, 3): Refinement of integer levels.
- Can be cheaply generated by pseudo-random value *U* [0, 1)
  with *L* largest / max level we need, and *n* nature of data (e.g. *n* = 2 for 2.5D surface scan):

$$I = 1/n \log_2 (U (2^{n(L+1)} - 1) + 1)$$

* More details in:
Van Oosterom et al., 2022, “Organising and visualizing point clouds with continuous levels of detail”, ISPRS J Photogr. + Remote Sensing, http://dx.doi.org/10.1016/j.isprsjprs.2022.10.004

nD point cloud in Postgres

- Read points from .laz: x, y, z (and additional attributes)
- For each point:
  - cloi = compute_cLoI(nDims, maxLevel)
  - compute_key(x, y, z, cloi) → SFC key
- Load SFC key (and attributes) to table inside database
- Sort & Cluster table
  - Create B-tree index on SFC key column
  - Cluster on this index (sorts table physically)
- Create histogram* for efficient querying
  (store approximate count per 2^n-tree node upto certain depth)

* More info on nD-Histogram: Haicheng Liu, “nD-PointCloud Data Management”, PhD thesis, TU Delft, June 2022, Chapters 4 & 5.
http://resolver.tudelft.nl/uuid:9f380f03-5842-45a0-87d4-4a8372e88dd5

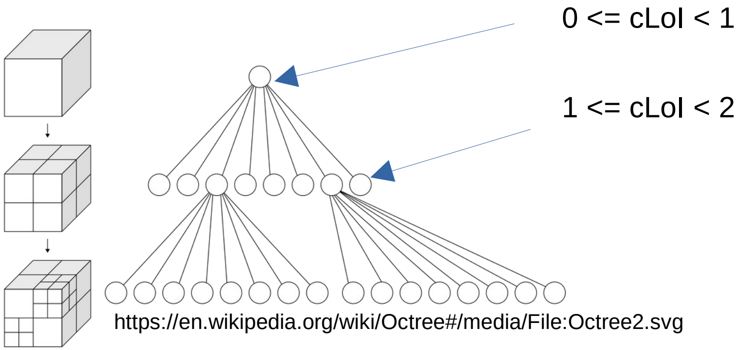
PCServe

3D Point cloud Web visualization

- Potree, Cesium (in browser, WebGL)
- Expect to find groups of points organised as Octree server-side
- Two options:
    1. Adapt client (make it aware of our nD point cloud structure)
    2. <u>Model Octree on top of nD point cloud structure</u>
- For this research: Use option 2) and make virtual Octree available via web service: PCServe (acting as ‘octree’ proxy)

Preprocessing for PCServe: derive Octree

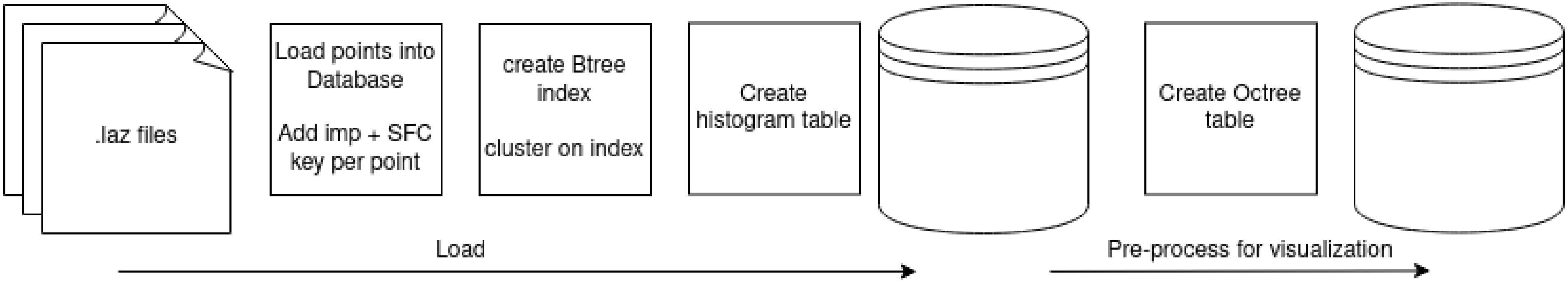
- Given sorted nD point cloud data table loaded in database;
  4D SFC key (x, y, z, cLoI)
- Create *virtual* Octree – i.e. structure of octree nodes + query ranges to get points from nD point cloud table
- Using SFC query mechanism to determine if Octree node has points
- Store all found Octree nodes in Octree table: (node ID, 4D node geometry, SFC ranges, point count)



https://en.wikipedia.org/wiki/Octree#/media/File:Octree2.svg

- Algorithm:
  - Root node of Octree spans complete domain (xyz)-volume && cLoI value: 0 <= cLoI < 1
  - Translate this x-y-z-imp nD geometry into SFC ranges and query for number of points
  - If points in root node: split (xyz)-volume in 8 childs and use cLoI value for first level child nodes: 1 <= cLoI < 2
  - Repeat until no more points occur or when we reach cLoI max

PCServe: DB preparation (summary)



.laz files

Load points into Database

Add imp + SFC key per point

create Btree index

cluster on index

Create histogram table

Create Octree table

Load

Pre-process for visualization

PCServe: From DB to web client

- Octree structure: Can be retrieved from Octree table
- Point data: Given a node id:
  - retrieve SFC ranges for node from Octree table
  - use these SFC ranges to get points from nD points table (indexed with SFC key)
- Serialization format:
  As .laz file or as Potree’s own binary format

Results

nD point cloud data structure

- Input:
  - Point count: 651’481’021
  - .laz: 3’219MB

- Output (table + Btree):
  - 80’521 MB (x25)

- 85 min to create + load nD point cloud structure
- 19 min to create histogram
- Total: 1 ¾ hour (104 min)

Virtual Octree construction

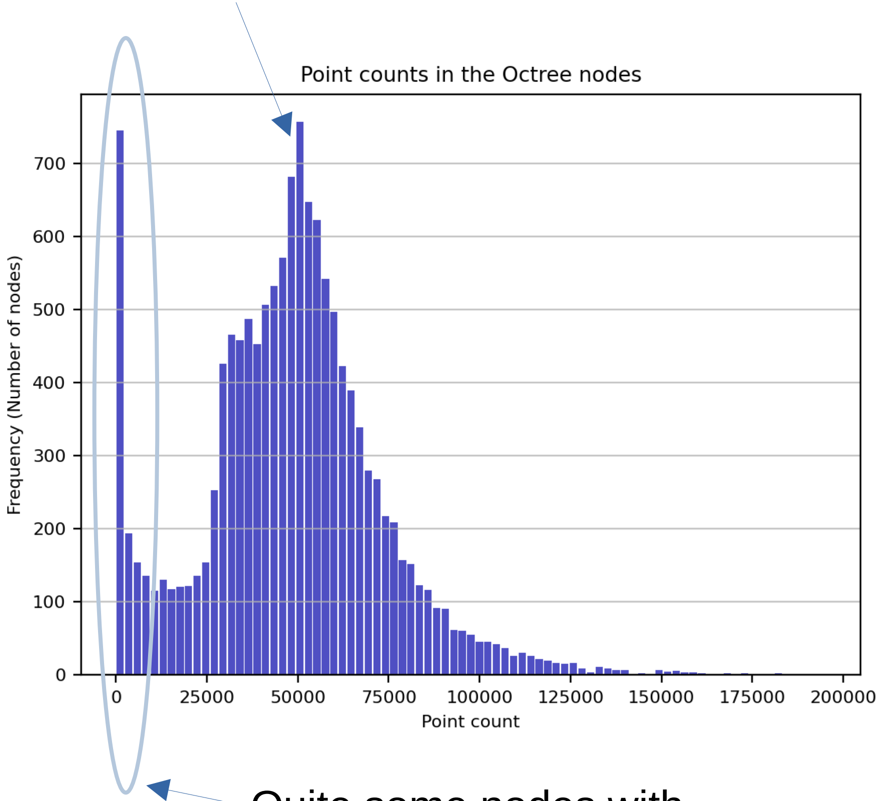
- Time to construct: 1 hour total
  - 17 min (range generation)
  - 43 min (query for point count in Octree nodes)

Virtual Octree layout

level	oct count	point count
0	1	30 119
1	7	116 422
2	24	450 523
3	74	1 909 593
4	226	7 631 203
5	745	30 542 745
6	2 615	122 157 155
7	9 809	488 604 847
total	13 501	651 442 607

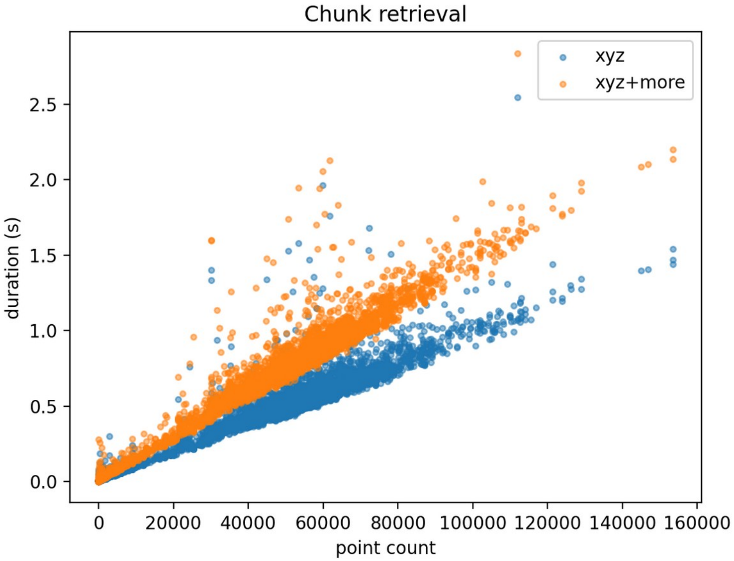
factor ~4x

Average of ~50k points per node



Quite some nodes with little number of points

Octree chunk retrieval



Averages for responses made by PCServe

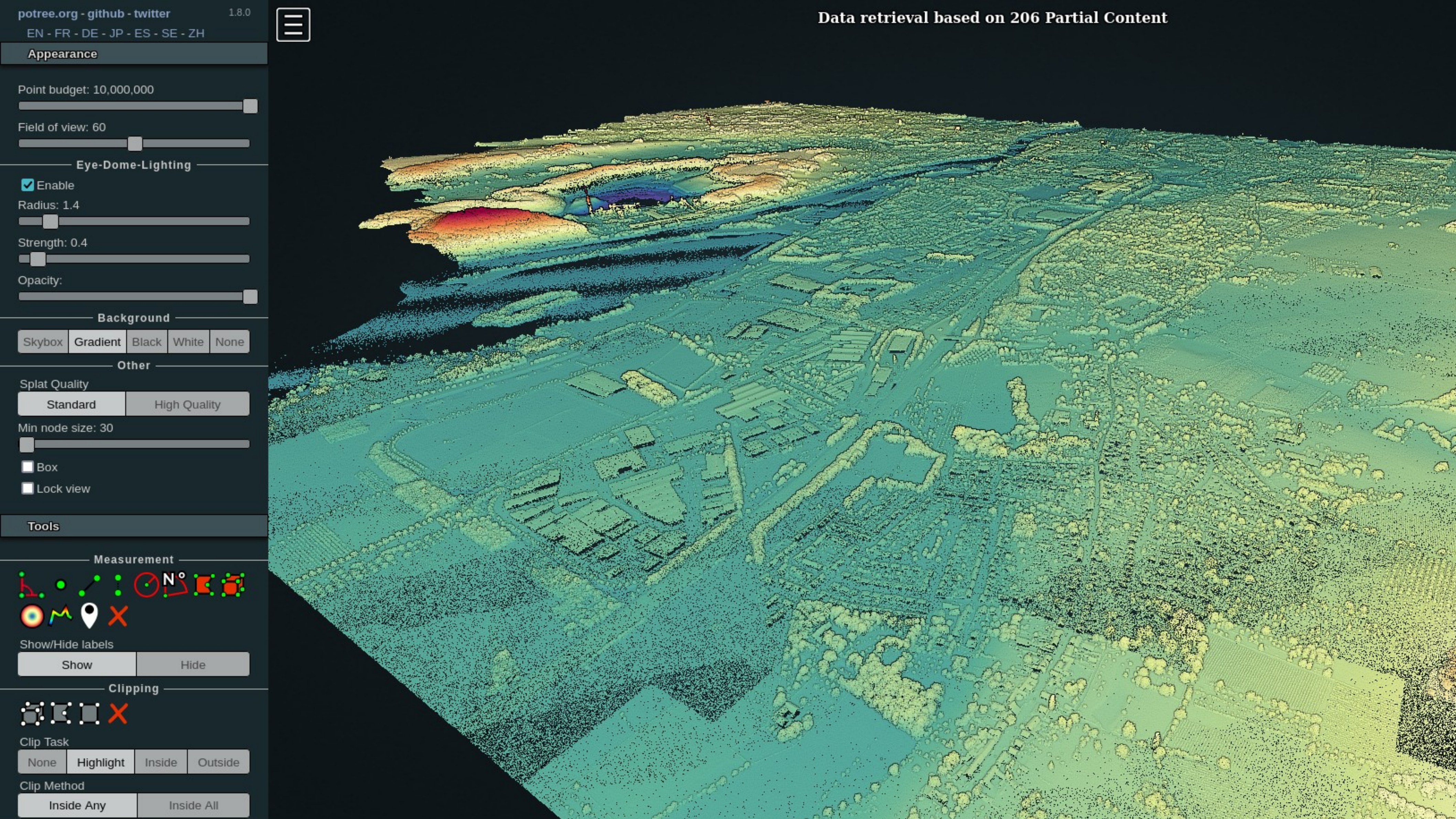
	xyz	xyz+more
Query duration (ms)	329 (72.7%)	416 (69.6%)
Serialize duration (ms)	123 (27.3%)	181 (30.4%)
Total duration (ms)	452	598
Point count	44 555	38 981

Prototype / Demo

- Prototype available at:

  http://ahn2.pointclouds.nl/potree-sfc/

Data retrieval based on 206 Partial Content



potree.org - github - twitter
1.8.0
EN - FR - DE - JP - ES - SE - ZH
Appearance

Point budget: 10,000,000

Field of view: 60

☑ Enable

Radius: 1.4

Strength: 0.4

Opacity:

Min node size: 30

□ Box

□ Lock view

Conclusions

Conclusion

- Presented nD point cloud structure + PCServe
- PCServe acts as proxy to nD SFC pointcloud in database
- ‘Eat own dogfood’: Possible to use the nD point cloud data structure for web visualization
- Used Potree renderer: Interactive visualization possible

Currently working on

- Engineering: parallel pre-processing, optimizing nD point cloud data structure (less disk size consumption – parquet files seem promising, but outside DBMS)
- More clients: Next to Potree also Cesium, QGIS (via COPC)
- Other spatial access structure for use in renderer (‘option 1’): Follow SFC curve (no need to ‘proxy’ requests)
- Use cLoI inside renderer directly (decide which points (not) to show in 3D view depending on eye of observer)
- Study same principle with other dimensions
(where dims != x, y, z, cloi → e.g. time, classification, quality)

Questions?

- Martijn Meijers
  b.m.meijers@tudelft.nl
  https://www.tudelft.nl/en/staff/b.m.meijers/
  tel. (+31) 15 278 56 42