



**RUNDER TISCH GIS E.V.**

Das effiziente GIS-Netzwerk von Forschung, Wirtschaft und Verwaltung

# Towards a relational database Space Filling Curve (SFC) interface specification for managing nD-PointClouds

13-3-2019

Peter van Oosterom,  
Martijn Meijers, Edward Verbree, Haicheng Liu and Theo Tijssen

Münchner GI-Runde, 14-15 March 2019

# Why a DBMS approach?

- today's common practice: specific file format (LAS, LAZ, ZLAS,...) with specific tools (libraries) for that format
- specific files formats are sub-optimal data management:
  - multi-user (access and some update)
  - scalability (not nice to process 60.000 AHN2 files)
  - integrate data (types: vector, raster, admin)
- 'work around' could be developed, but that's building own DBMS
- no reason why point cloud can not be supported efficient in DBMS
- perhaps 'mix' of both: use file format for the PC blocks
- point clouds are a **bit similar to raster data**:  
sampling nature, huge volumes, relatively static
- point clouds are a **bit similar to vector data**:  
arbitrary xyz locations and may have other attributes

# Standardization of point clouds?

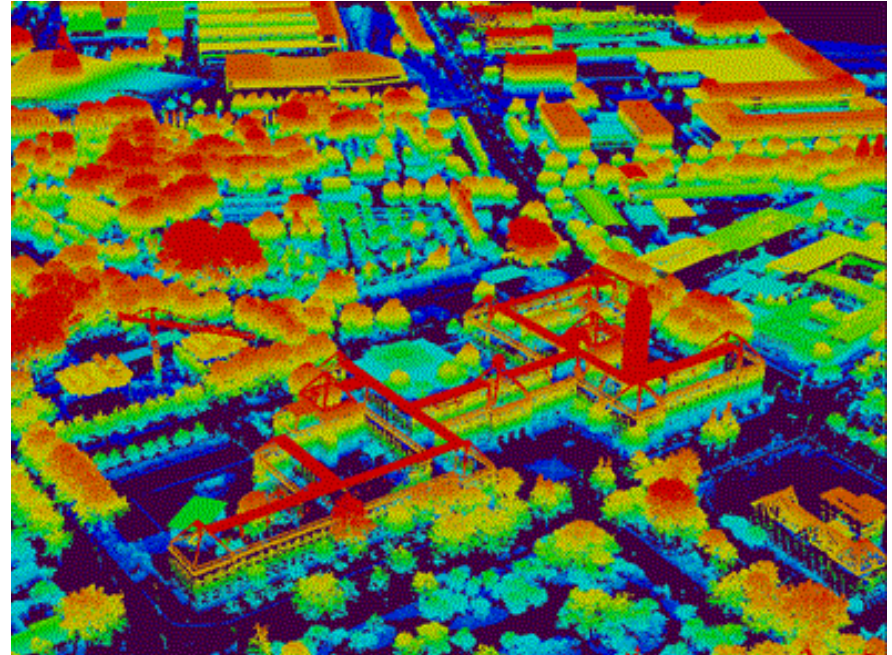


- ISO/OGC spatial data:
  - at **abstract/generic level**, 2 types of spatial representations: features and coverages
  - at next level (**ADT level**), 2 types: vector and raster, but perhaps points clouds should be added
  - at implementation/ **encoding level**, many different formats (for all three data types)
- nD-PointCloud:
  - points in nD space and not per se limited to x,y,z (n ordinates of point which may also have m attributes)
  - make fit in future ISO 19107
  - note: nD point clouds are very generic; e.g. also cover moving object point data: x,y,z,t (id) series.



# Overview

1. Motivation: nD-PointCloud
2. Space Filling Curves
3. Operations
4. DBMS interface
5. Conclusion



acknowledgements: based on joint work with  
Oscar Martinez-Rubi, Mike Horhammer, Stella Psomadaki,  
Xuefeng Guan, Jippe van der Maaden, Simon van Oosterom, ...

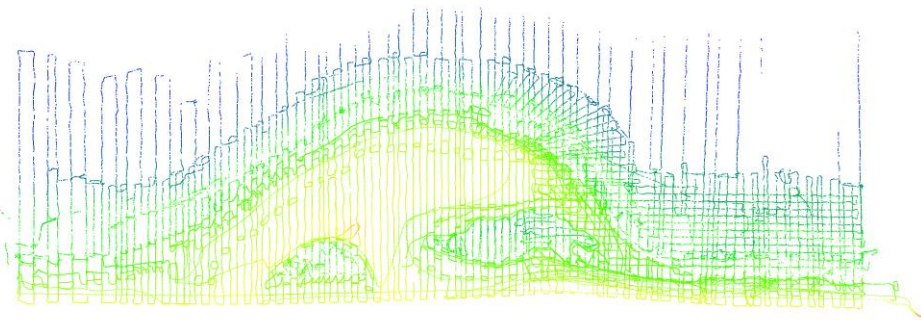
# Motivation nD-PointCloud in DBMS

- point cloud data sets are often used for monitoring
  - dynamic point clouds
  - time added as additional organizing dimension
- organizing point cloud data in LoD's/importance levels is an approach to manage large data sets
  - LoD: discrete (multi-scale) or continuous (vario-scale)
  - scale treated as additional organizing dimension
- how to manage higher dimensional point clouds (4D, 5D)

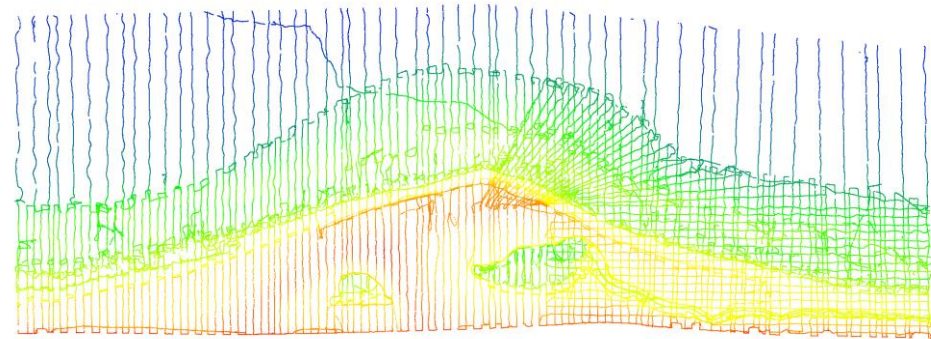
# Dynamic Point Clouds

- point clouds are generated every day, hour, minute
- repeated scans of the same area → dynamic
- time as selective as the spatial component or needed in integrated space – time selections
- current DBMS solutions designed for static point clouds
- management is still a challenge
- example **Sand Engine**, time series Dutch coast, Deltares (see Psomadaki et al, 2016) MSc thesis TU Delft.

2011

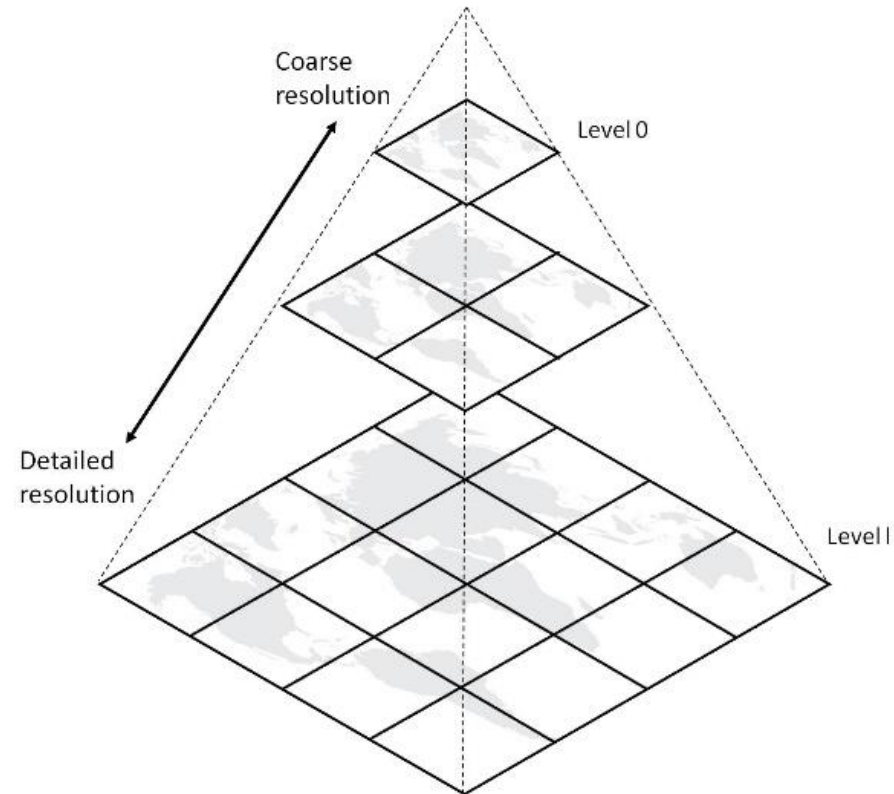
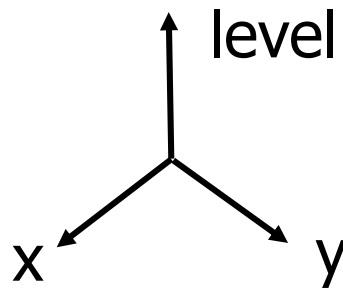


2015



# Scale as dimension

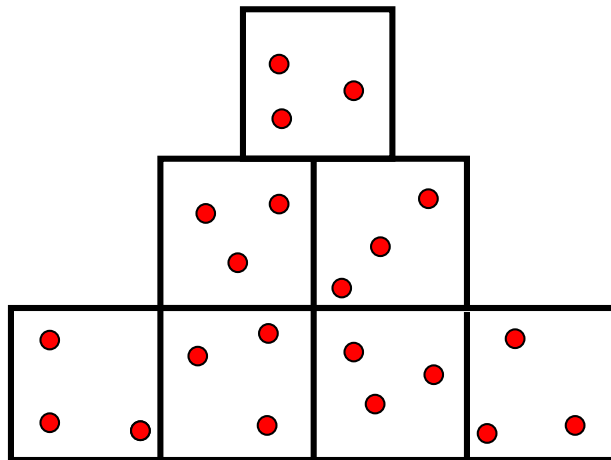
- less obvious than time
- **data pyramids**  
(Level of Detail/ Multi-scale)
- well-known from raster data
- results in discrete number of levels (multi-scale)
- level could be considered as additional dimension



# Point cloud data pyramid

- overview queries just want top-subset
- detailed queries part of bottom-subset
- organize in data pyramid

2D schematic view, data blocks....

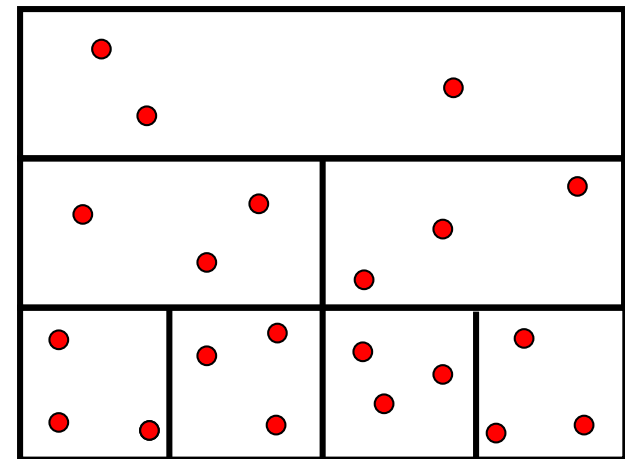


stretched over domain

LoD 2

LoD 1

LoD 0



density  
low



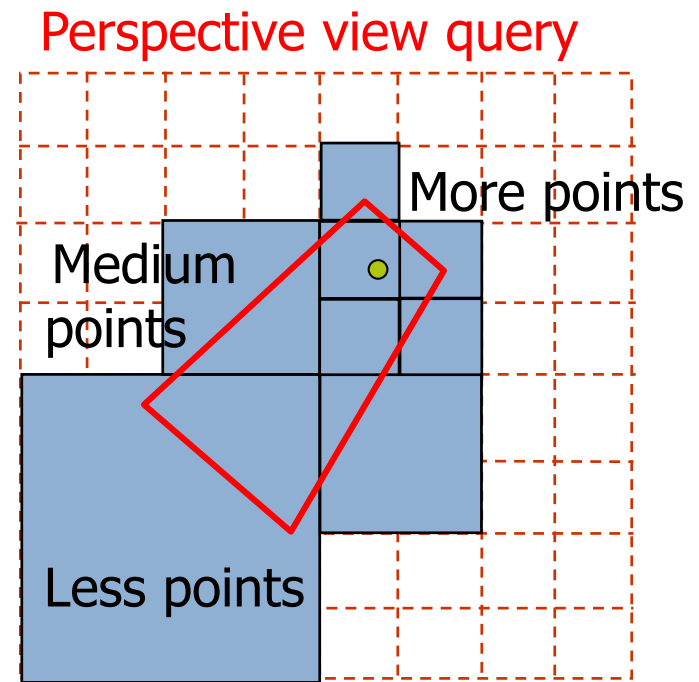
high

every next higher level, density  $2^k$  times less (2D  $\rightarrow 4$ , 3D  $\rightarrow 8$ )



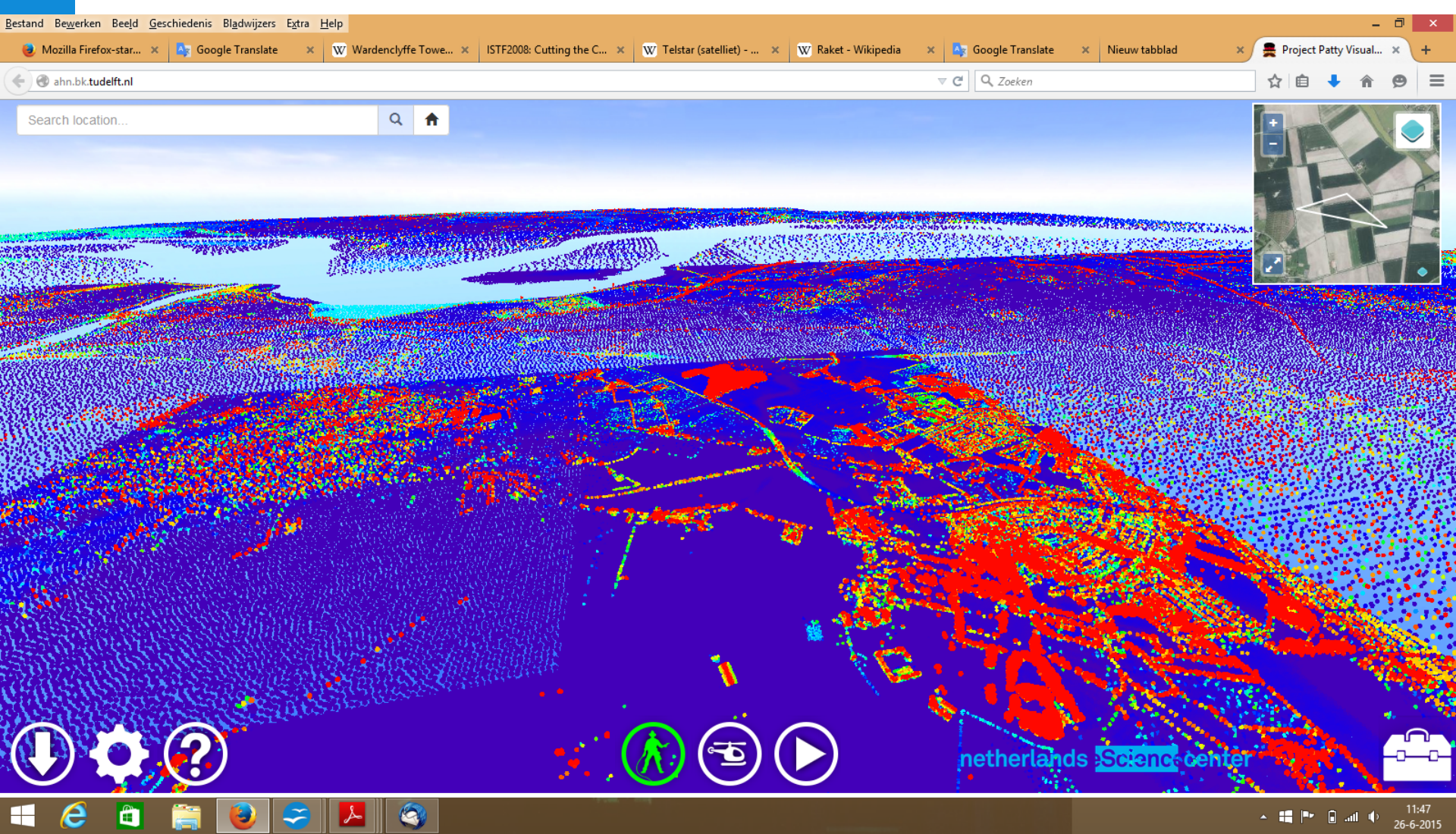
# Data pyramid/multi-scale

- allows fast spatial searching including LoD selection
- the further away from viewer the lesser points selected (i.e. the higher level blocks/points)
- drawbacks:
  1. discrete number of levels
  2. bottom-up filling, unbalanced top
  3. point random assigned to level



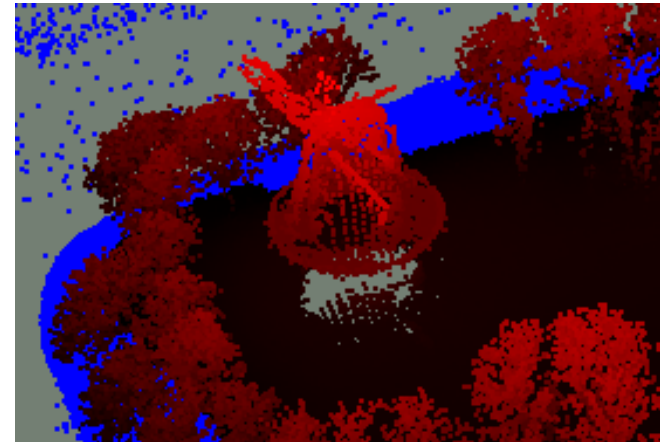
# Discrete LoD's are visible...

<http://ahn2.pointclouds.nl>: 640.000.000.000 points on-line 3D viewer



# Vario-scale for point cloud data

- lesson from vario-scale research: **add one continuous dimension** to the geometry to represent scale (2D data vario-scale represented by 3D geometry)
- apply this to point cloud data
  1. compute the imp value (bonus slides)
  2. add this as dimension, either  $x, y, \text{imp}$  (z and others attributes) or  $x, y, z, \text{imp}$  (and others as attributes)
  3. Cluster/index the 3D or 4D point
  4. Define perspective view selections, view frustum with one more dimension: the further, the higher imp's

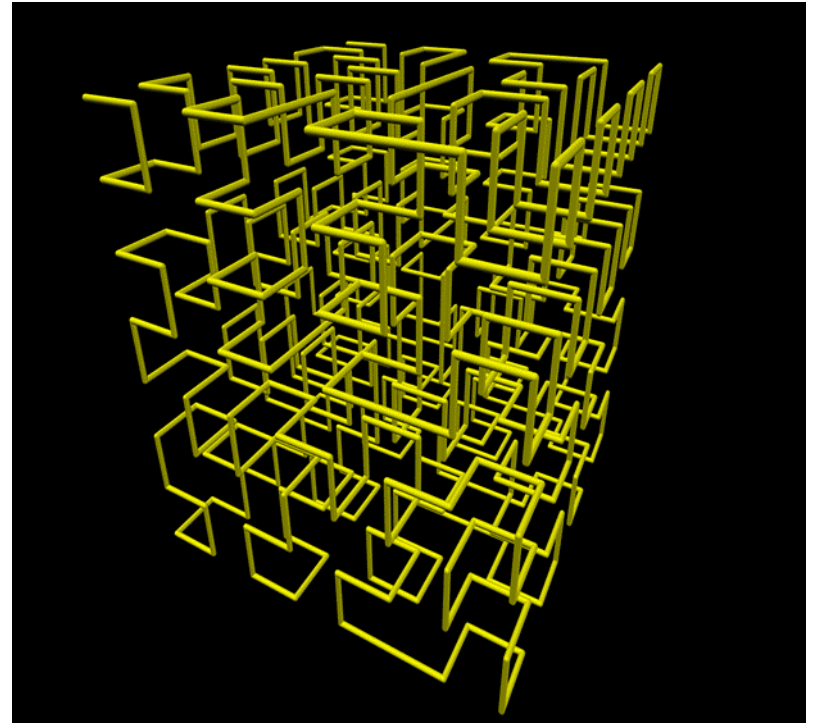


# Representations of space, time, scale ...after grid/voxel or object/vector

- new 3<sup>rd</sup> representation: nD-PointCloud (nD-PC)
- many scientific domains (spatial): geography, medicine, physics, astronomy, hydrology, architecture, archaeology, arts, CAD, social media/ moving objects, gaming...
- deep integration space/time/scale
  1. more efficient, store, exchange, compute
  2. more functionality (smooth zoom/ analysis)
- nD-PC in whole processing chain: acquisition, DBMS, analysis, simulation, dissemination, visualization,...
- BIG spatial data: 35 trillion points (in astronomy, geo-info)

# Overview

1. Motivation
2. Space Filling Curves
3. Operations
4. DBMS interface
5. Conclusion



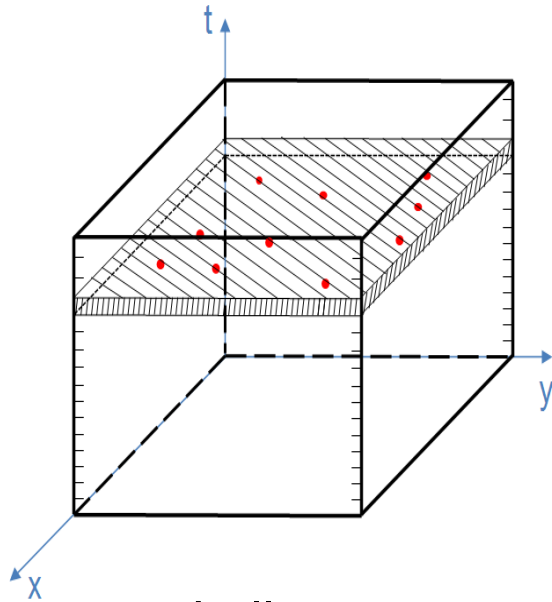
# nD-PC data management

- management of nD-PC data, starts by defining
  - dimensions (and their roles/priorities in the points)
  - associated attributes
- dimensions are main drivers for data organization, clustering, indexing, subdivision (for parallel processing), compression, blocking/ caching and streaming of data
- various data management options possible, now focus on
  - *integrate dimension values in 1 value via Space Filling Curve (SFC): Morton, Hilbert, and relation to quadtree ( $2^n$ -tree)*

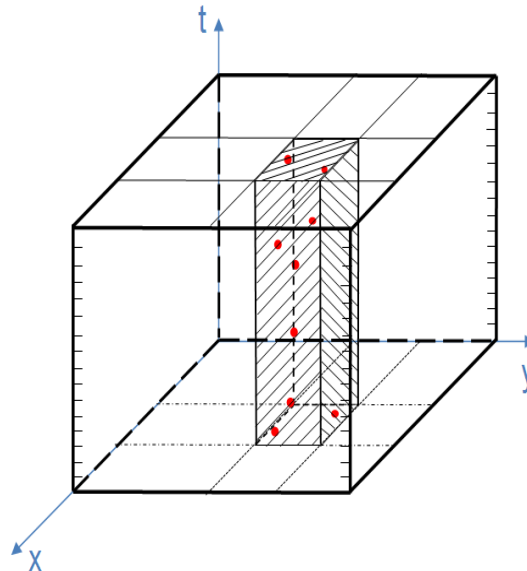


# Different clustering scheme's for space-time (or space-scale) cube

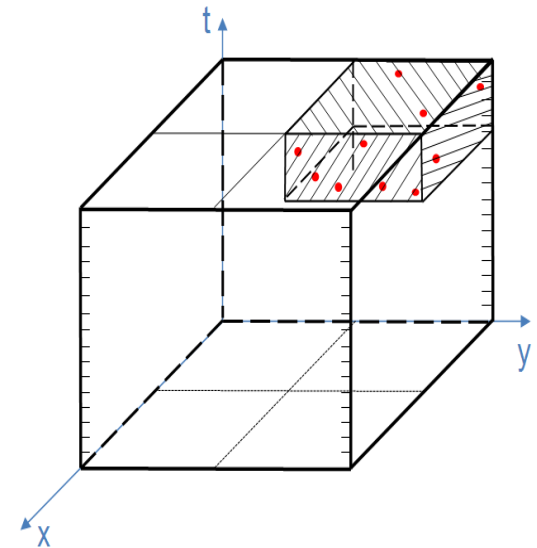
16x16x1



4x4x16



8x8x4



- challenge increases for higher dimensional hyper-cubes:
  - 4D: 2D space-time-scale, 3D space-time, 3D space-scale
  - 5D: 3D space-time-scale

# nD-PC data management

- modelling theory for nD point cloud data
- tools to support modelers, developers and users in point cloud data organization design decisions for (given 1. data sets and 2. required functionalities in applications):
  - what are the dimensions,
  - what are the attributes,
  - what type of organization: Morton-code/ kd-tree/ nD simplices-part,
  - what relative scale of various dimensions,
  - parameters such as clustering/ blocking size,
  - what compression,
  - what approach and level of parallelism (incl. hardware aspects),

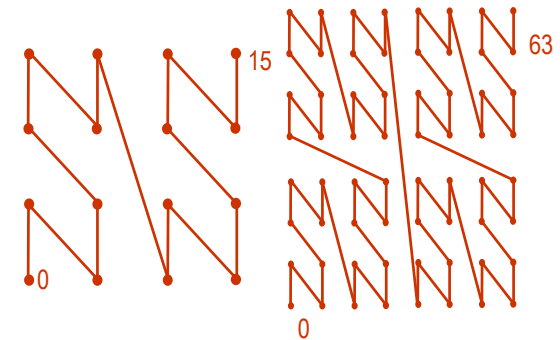
→ Modeling workbench



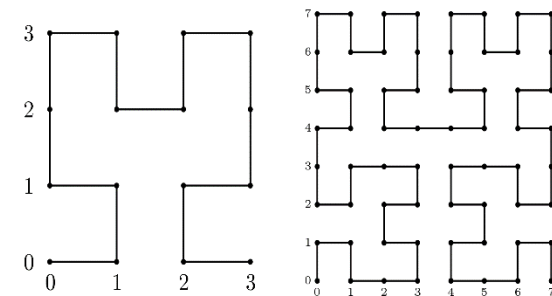
# Space Filling Curves (SFCs)

- apply linear ordering to a multidimensional domain (spatial clustering)
- organize a flat table efficiently
- full resolution keys: avoid storing  $x,y[,z] + t/l$   
→ recovered from SFC key
- use Index Organized Table  
(data stored in the B-Tree index)
- queries need to be re-written to SFC-ranges,  
benefit from spatial clustering → efficient
- SFCs based on hyper-cubes
  - Morton/Hilbert both **nD and quadrant recursive**
  - Consider relative scaling of dimensions
  - Space reserved on the hypercube for future data

Morton (Peano)



Hilbert



# SQL DDL for index organized table

- Oracle:

```
CREATE TABLE PC_demo (hm_code NUMBER PRIMARY KEY)  
    ORGANIZATION INDEX;
```

- PostgreSQL, pseudo solution, not dynamic (better available?):

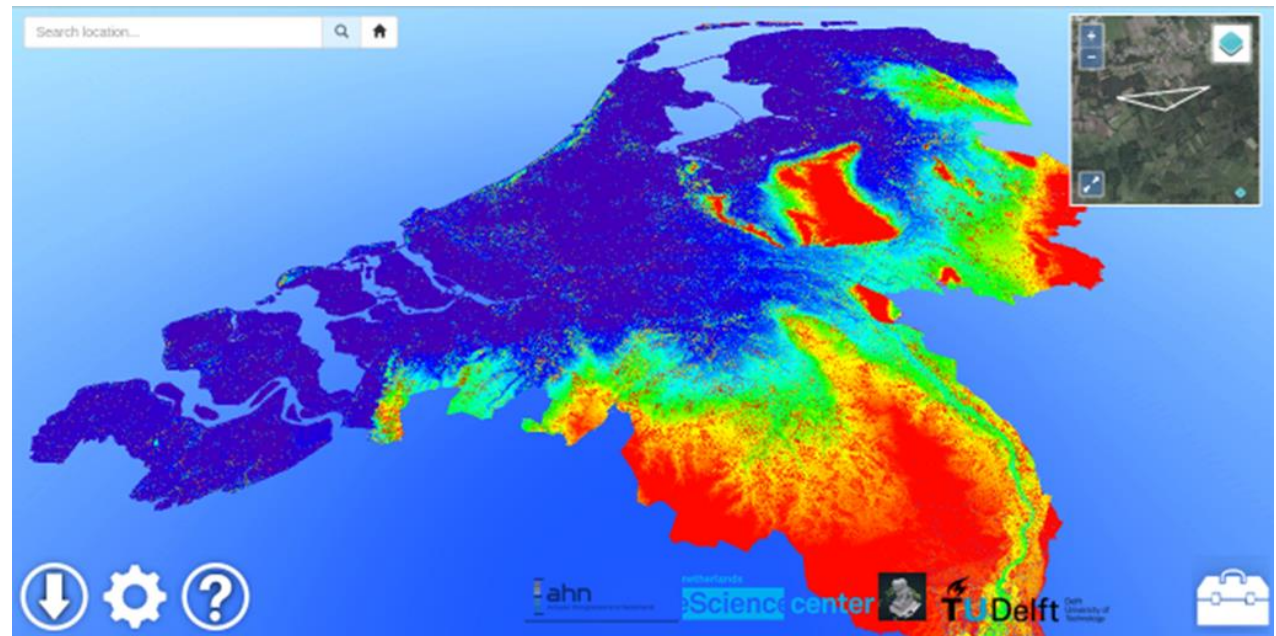
```
CREATE TABLE PC_demo (hm_code BIGINT PRIMARY KEY) ;  
CLUSTER pc_demo ON pc_demo_pkey;
```

# Lessons learnt so far...

- IOT (e.g. as in Oracle) is efficient structure, **table/index together**
- full high-res key allows **omitting storing attributes** (e.g. x or y as they can be decoded in full resolution from the SFC key)
- full high-res and higher dimensional keys results in **large keys** (not fitting in 64 bits, options varchar or raw bytes)
- SFC encoding/decoding, range generation **outside** database results in quite a bit of communication overhead
- **roles** of organizing dimensions (x, y, time, importance) and other attributes (non-organizing) is **context dependent**
- **relative scaling** organizing dimensions needed to compute SFC (e.g. 1 meter = 1 second, influences the actual clustering)
- SFC ranges for **(non-box) query geometry** shapes has big benefits over simple bounding box (more precise approx, less false hits)
- high-res SFC query ranges in nD space may result in **many ranges** esp. when omitting some dimensions -> implies infinite extend

# Overview

1. Motivation – nD PointClouds
2. Space Filling Curves
3. **Operations**
4. DBMS interface
5. Conclusion



# SFC DBMS Interface specification

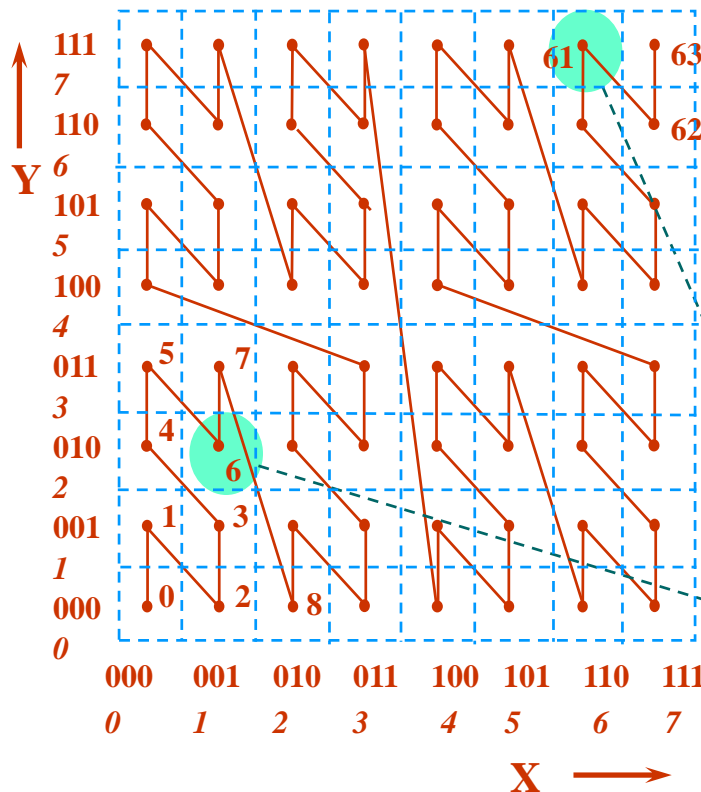
- A. define functions for given square/cubic/... nD domain:
1. **SFC\_ENCODE** (point, domain)  $\rightarrow$  SFCkey; (for storage)
  2. **SFC\_DECODE** (SFCkey, domain)  $\rightarrow$  point; (for use/computation)
  3. **SFC\_RANGES** (query\_geometry, domain)  $\rightarrow$  ranges; (for query)
- B. **SFC\_CREATE** create table, add SFCkey during bulk load
- or even replace point coordinates
  - modify table from default heap to b-tree/IOT on SFCkey
- C. **SFC\_DROP** remove the point cloud and related IOT

SFC code (corresponds to cells of Quadtree in 2D, Octree in 3D, ...)

# SFC\_ENCODE (point, domain)

→ SFCkey

- example Morton\_code / Peano key / Z-order
- bitwise interleaving x-y coordinates
- also works in higher dimensions (nD)



two examples of Morton code:

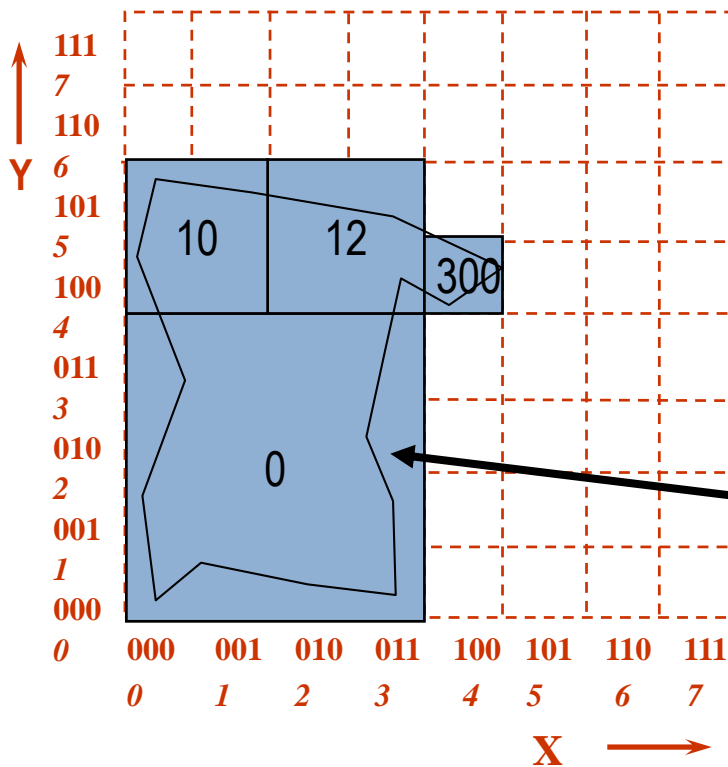
$x = 110, y = 111 \rightarrow xy = 111101$  (decimal 61)

$x = 001, y = 010 \rightarrow xy = 000110$  (decimal 6)

# SFC\_RANGES (query\_geometry, domain)

## → ranges

- based on concepts of Region Quadtree & Quadcodes
- works for any type of query geometry (point, polyline, polygon)
- also works in 3D (Octree) and higher dimensions



Quadcode 0: Morton range 0-15  
Quadcode 10: Morton range 16-19  
Quadcode 12: Morton range 24-27  
Quadcode 300: Morton range 48-48  
(Morton code gaps resp. 0, 4, 20)

query\_geometry, polygon

**Note : SW=0, NW=1, SE=2, NE=3**

# Merge ranges → analyze effects

deeper in recursion → more ranges generated

- Pro: approximates query geometry better (less unwanted points)
- Con: too many range makes the join slower

techniques to avoid too many ranges:

- terminate recursion early,
- merge nearby ranges

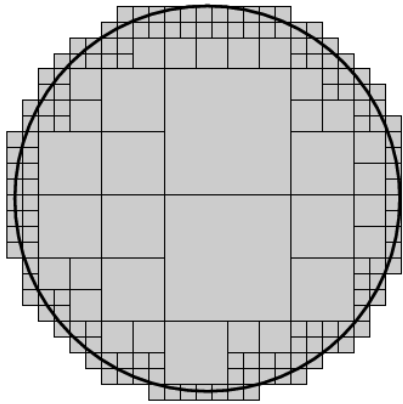
balance to be investigated: how deep, how much merging w.r.t.

- query precision (add not too much space → more unwanted points)
- query speed (many ranges, means lot of work)

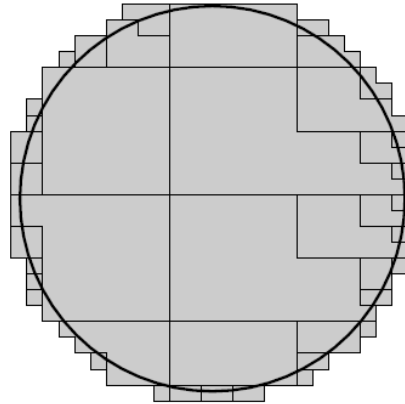
effect of range merging with gaps is adding space (more cells)



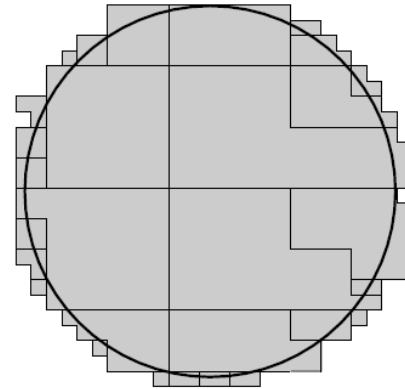
# Merge ranges → add space



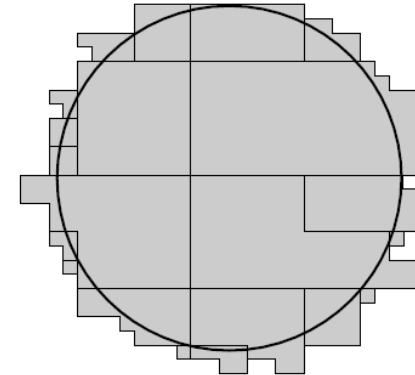
(a) Original 176 tree cells



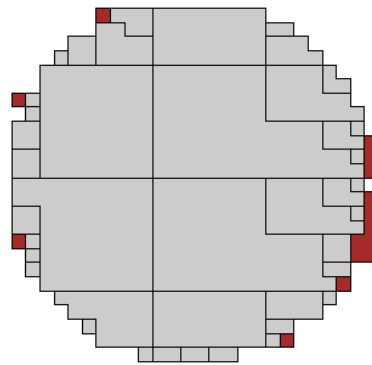
(b) Merging of direct neighbours leading to 42 cells



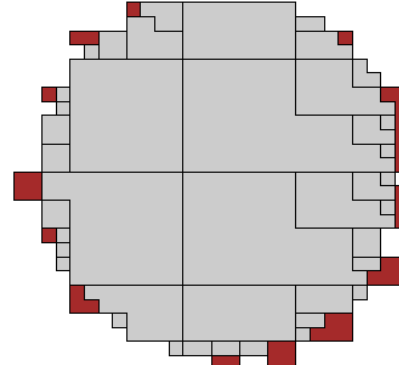
(c) Merged to maximum number of 30



(d) Merged to maximum number of 20



(e) The expansion of the original geometry (case (b)) in red after merging to 30

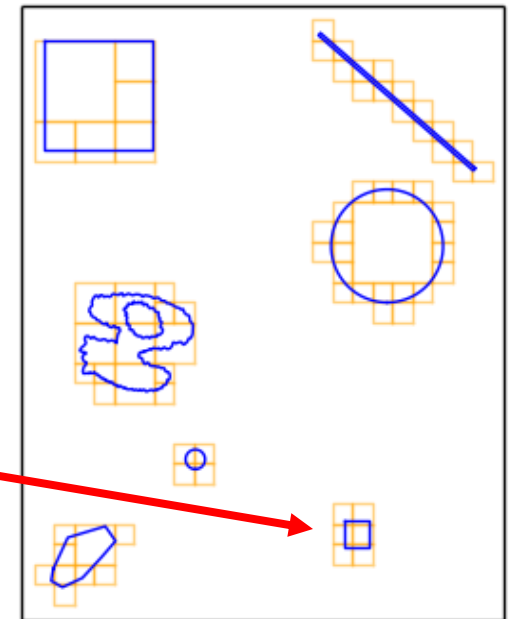


(f) The expansion of the original geometry (case (b)) in red after merging to 20

# Quadcells / ranges and queries

```
CREATE TABLE query_results_1 AS (  
  SELECT * FROM  
    (SELECT x,y,z FROM ahn_flat WHERE  
      (hm_code between 1341720113446912 and 1341720117641215) OR  
      (hm_code between 1341720126029824 and 1341720134418431) OR  
      (hm_code between 1341720310579200 and 1341720314773503) OR  
      (hm_code between 1341720474157056 and 1341720478351359) OR  
      (hm_code between 1341720482545664 and 1341720503517183) OR  
      (hm_code between 1341720671289344 and 1341720675483647) OR  
      (hm_code between 1341720679677952 and 1341720683872255)) a  
  WHERE (x between 85670.0 and 85721.0)  
        and (y between 446416.0 and 446469.0))
```

Query 1 (small rectangle)



# Use of Morton codes, AHN2 data (PostgreSQL flat model example)

data	Q1	Q4	Q7
size	rect	circle	line
20M	0.16	0.85	2.32
210M	0.38	1.80	3.65
2201M	0.93	4.18	7.11
23090M	3.14	14.54	21.44

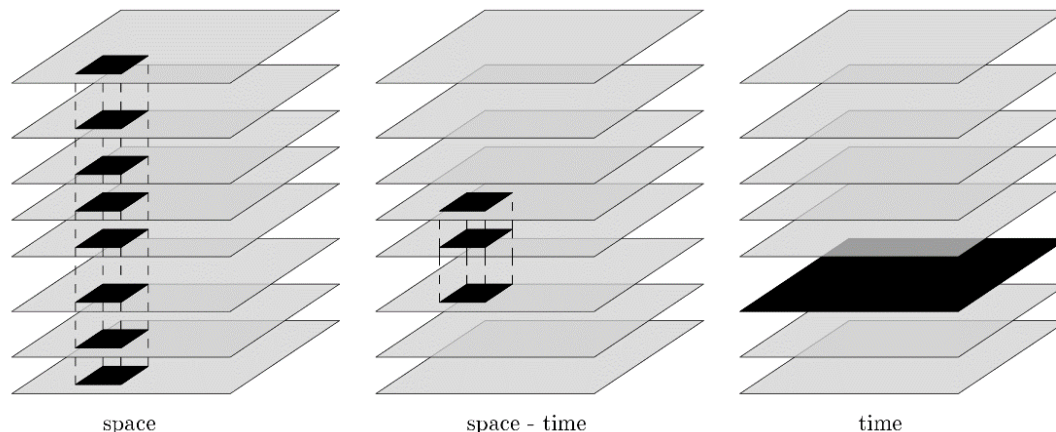
response in seconds  
(of hot/second query  
first query exact  
same pattern, but  
3-10 times slower  
both for normal flat  
model and for Morton  
flat model)

with	Q1	Q4	Q7
Morton	rect	circle	line
20M	0.15	0.56	0.82
210M	0.15	0.56	0.42
2201M	0.13	0.64	0.41
23090M	0.15	0.70	0.60

# Storage model balancing

'best' organization is dependent on data and (frequent) queries; e.g.

- asking for time slice (map of one moment in time)
- performing time needle query (one location trough time)
- selecting data for time interval in limited area



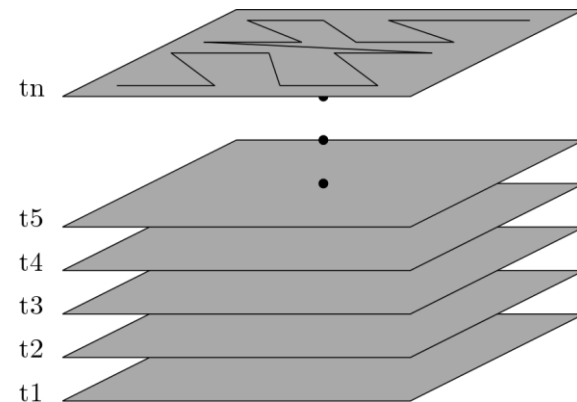
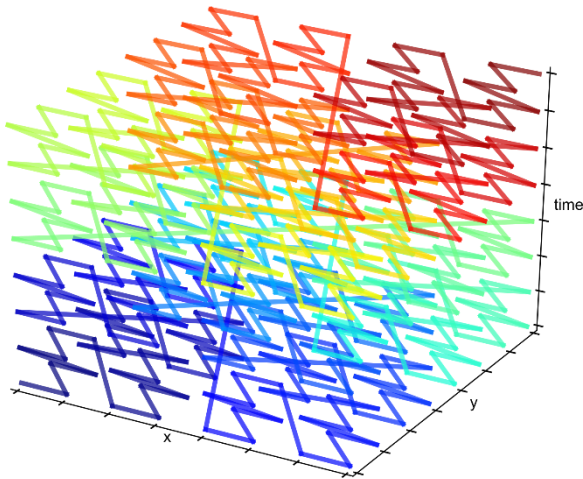
dynamic data optimizing for space/time queries contradicts:

1. points close in space and time should be stored (to some extent) close in memory for fast spatio-temporal retrieval
2. already organized points should not be reorganized when inserting new data to achieve fast loading

# Storage Model

storage of space and time:

1. integrated space and time approach: space and time have an equal role in the SFC code
2. non-integrated space and time approach: time dominates over space (and used first in organization)



second option easier to add data (dynamic scenario), no reorganization

# Overview

1. Motivation – nD PointClouds
2. Space Filling Curves
3. Operations
4. DBMS interface
5. Conclusion

ORACLE®

PostgreSQL



# DBMS SFC (nD-PC) Standardization?



# DBMS SFC requirements

1. Flexibility requirements (generic)
2. Performance requirements (BIG data)
3. Ease-of-use requirements



# DBMS SFC 1. flexibility requirements

- support for multiple dimensions with flexible ordering, scaling
- support for various types of SFC (e.g. Morton, Hilbert)
- support for different representations of SFCkey (e.g. number, character) for high-res large nD values needing  $> 64$  (or 128) bits
- support for partial and full resolution keys
- support for implicit (as part of SFCkey) or explicit (as additional attributes) storage of dimensions
- higher dimensional query geometries of various types (more than just an nD box)

# DBMS SFC 2. performance requirements

- data that will be used together must be stored together
- processing related to storage and query of point cloud data should take place in the environment where the data is stored: in the database
- data movements should be minimised (and data encode/decode conversions with SFC software should be inside database to get access to attribute in full resolution SFCkey)

# DBMS SFC 3. ease-of-use requirements

- functionality must be available via SQL functions
- despite all tuning options, use of functions should be straightforward (balance: one time tuning via the info in metadata, and easy use of encode/ decode/ range functions)
- functionality should fit well with other spatial DBMS functionality (e.g. to perform secondary filter point-in-polygon test)
- specifying a nD query geometry may be done via nD simplicial complexes or nD regular polytopes, but users may be unfamiliar with this, so simpler alternatives needed (e.g. extruded polygon)
- the number of SFC ranges is a crucial performance factor which user should be able to influence (comparable to resolution, depth in recursion, and involve glueing of ranges)

# DBMS SFC Dataset metadata

avoid passing all info in every function call → one place

<b>PC_ID</b>	<b>number</b>	<b>system-generated, unique number (link to functions)</b>
PC_NAME	varchar2(32)	name of point cloud dataset
PC_DESCRIPTION	varchar2(256)	description of point cloud
PC_TABLE	varchar2(32)	name final structured table IOT
STAGING_TABLE	varchar2(32)	name intermediate staging table (needed, may be virtual)
PC_DIMENSIONS	number	number of dimensions
PC_NUMBITS	number	number of bits per dimension with which to encode
SFC_TYPE	varchar2(32)	'Morton' or 'Hilbert'
SFC_ENCODING	varchar2(32)	'Number' or 'Character' or 'Base32' or 'Base64' or 'Raw'...
KEY_TYPE	varchar2(32)	'Partial' or 'Full'
<b>DIM_PROPERTIES</b>	<b>varray(PC_DIMENSIONS) of DIM_SPEC</b>	
SRID	number	spatial reference of points in point cloud (in 2D or 3D)
TIME_EPOCH	datetime	start encoded time value (UTC)
TIME_UNIT	varchar2	unit time: year, week, day, hour
TIME_UCT_OFFSET	number	UTC offset

# DBMS SFC Dimension specific metadata

for every organizing dimension the following metadata is specified:

<code>DIM.NAME</code>	<code>varchar2(32)</code>	name of dimension
<code>DIM.DESCRPTION</code>	<code>varchar2(256)</code>	description of dimension
<code>DIM.OFFSET</code>	number	value to be added for normalized dimension value
<code>DIM.SCALE</code>	number	scale factor for normalized dimension value
<code>DIM.MINVAL</code>	number	minimal dim value in dataset
<code>DIM.MAXVAL</code>	number	maximum dim value in dataset
<code>DIM.TOL</code>	number	tolerance associated with dim

# DBMS SFC encode and decode

- encode function

**SFC\_ENCODE** (**PC\_ID**, VAL\_D1, VAL\_D2, ... , VAL\_Dn)

**Return:** SFCkey

note: The actual encoding depends on and must be in sync with the metadata (Morton/Hilbert, resolution, etc.)

- decode function

**SFC\_DECODE** (**PC\_ID**, SFCkey)

**Return:** VAL\_D1, VAL\_D2, ... , VAL\_Dn

# DBMS SFC encode and decode (bulk)

- function calls are expensive, so for bulk of M points in N dimension space pass array's and process all points
- bulk encode function

```
SFC_ENCODE (PC_ID, M, VAL[M*N])  
Return: SFCkey[M]
```

- bulk decode function

```
SFC_DECODE (PC_ID, M, SFCkey[M])  
Return: VAL[M*N]
```

# DBMS SFC query geometry

```
SFC_RANGES (PC_ID, QUERY_GEOMETRY, [recursion depth,]  
                                                    [max_nr_ranges,] [type] ...)
```

Return: table of SFCkey\_LOW, SFCkey\_HIGH values [,range\_type]

Notes:

1. returned **SFC\_RANGES** are sorted
2. **SFC\_RANGES** are used for the primary filter
3. **QUERY\_GEOMETRY** options: a. nD-hyperbox, b. nD-hypersphere, c. 2D/3D geometry+extrusion min/max other dimensions, d. intersection nD-halfspaces (regular polytope), e. view frustum  
important special case perspective view (trans-dimension query)
4. **recursion depth** → actual resolution/cell size (be careful)
5. ranges with smallest gap merges until at **max\_nr\_ranges**
6. SFC ranges **type**: complete **in** query geometry, or **on** boundary  
(only for point in the on boundary ranges secondary filter needed)



# DBMS SFC examples, loading stage

- filling metadata:

```
insert into user_pc_sfc_metadata values (1, 'my test',  
'a test point cloud', 'my_pc', 'test_pc_staging', 3, 21,  
'Hilbert', 'Number', 'Full', 28992, -1, 'na', 0)
```

- create and start loading data into a nD-PC SFC table:

```
select sfc_create(pc_id);
```

(during sfc\_create execution function sfc\_encode used)

# DBMS SFC query examples (1 / 2)

- from query geometry to SFC ranges:

```
select * from sfc_ranges(pc_id, query_geom);  
  min | max  
-----+-----  
  502 | 503  
  507 | 508  
  608 | 609  
(3 rows)
```

- combined with the nD-PC data table in a join:

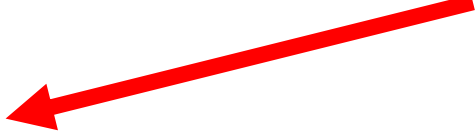
```
select sfc_decode(pc_id, key)  
from my_pc pc, sfc_ranges(pc_id, query_geom) range  
where pc.key between range.min and range.max;
```

# DBMS SFC query examples (2/2)

add **secondary filter** for ranges on boundary:

```
select sfc_decode(my_pc, key)
from pc_table pc,
      sfc_ranges(my_pc, query_geom, 'added') range
where (range.type = 'in' and
       pc.key between range.min and range.max)
or (range.type = 'on' and
    pc.key between range.min and range.max and
    overlaps(query_geom, geometry(sfc_decode(my_pc, key))));
```

range type requested

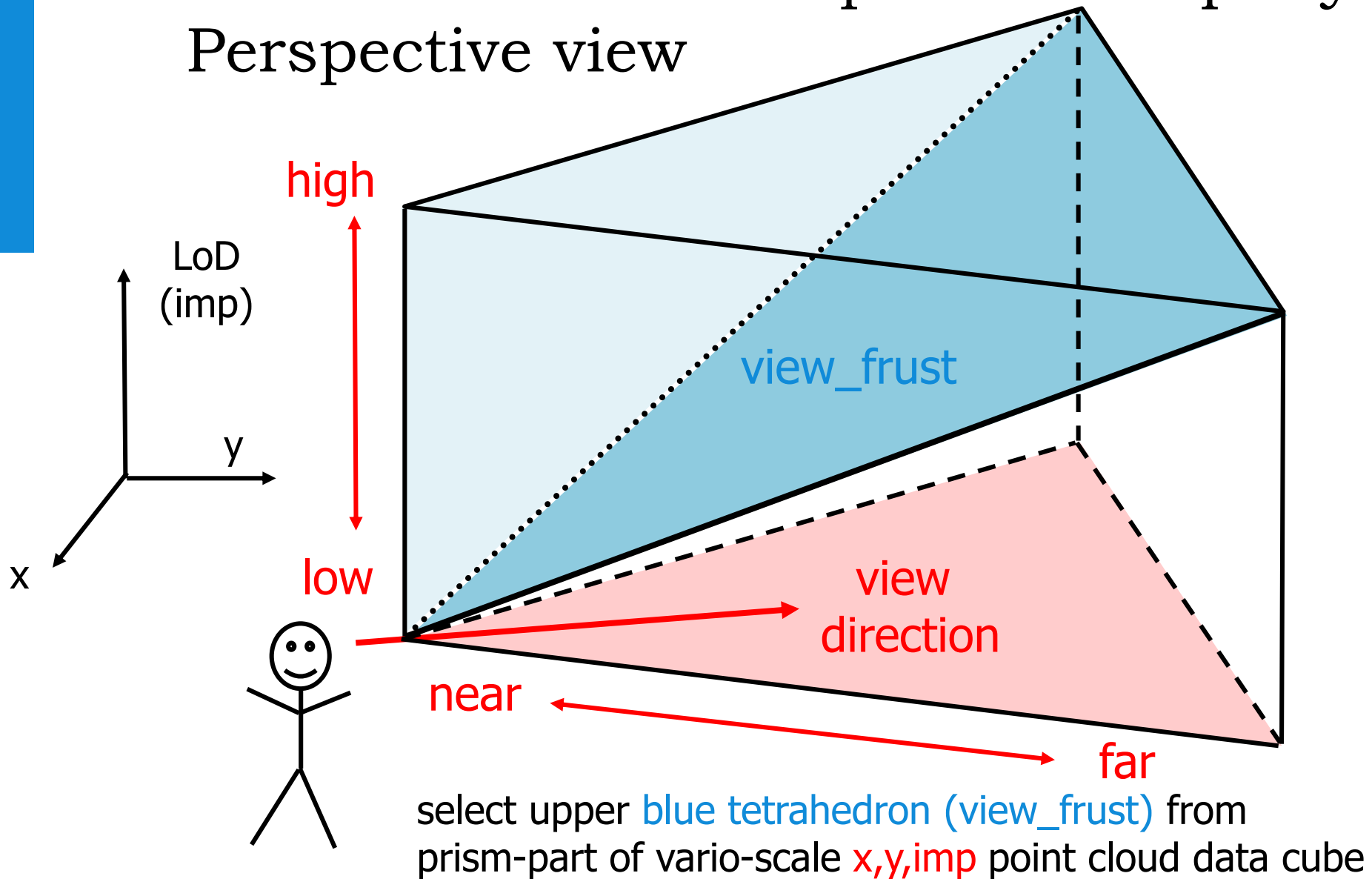


Drop a point cloud:

```
select drop_sfc_pc(pc_id);
```

# nD-PC **trans-dimension** space-scale query

## Perspective view



# Normal view frustum selection and streaming based on importance

- view frustum selection (pseudo code)

```
select point
from point_cloud
where overlaps (point, view_frust)
```

becomes with SLC ranges

```
select sfc_decode(pc_id, key) --> x y z imp
from my_pc pc, sfc_ranges(pc_id, view_frust) range
where pc.key between range.min and range.max;
```

- ordered on importance for streaming add **order by imp desc;**  
(or distance from tilted plane)

# Delta queries for moving and zoom in/out (in VR/AR environments)

- select and send new points (pseudo SQL):  
`point in new_frust and point not in old_frust`
- find and drop old points:  
`point in old_frust and not in new_frust`
- note this works form both
  1. changing view position x,y(,z)
  2. zooming in or out ('view from above', imp-dimension)
- optional to work at point or block granularity  
(in selection and server-client communication)

# Drawback of high dimensional SFC?

- nD SFC keys have benefits: space-time-scale (and perhaps even other attributes) in compact organization
- may select on multiple/**trans** dimensions at same time efficiently
- possible drawbacks of high dimensional point cloud:
  1. need big SFC code (128 bits number or other encoding, like varchar)
  2. if just limited number of dimensions are specified for selection → other dimensions than range form min-to-max: '**infinite tall prisms**' many (empty?) cells, what are the query performance consequences
- needs further exploration  
(as the relative scaling of dimensions need attention → basis for defining trans-dimension distance → actual grouping/ clustering)

# Avoiding the curse of range explosion

- using **recursion\_depth** and **max\_nr\_ranges** may result in non-optimal ranges as query geometry is translated into ranges without considering point cloud data distribution
- first refinement: avoid generation ranges in spaces where there is no data (outside actual domain), simple min-max test.  
Esp. when dimension in key, but not specified (infinite column)
- second refinement: create (recursive) nD-histogram and use to generate more ranges in parts of space with high data density
- needs further exploration



# Overview

1. Motivation – nD PointClouds
2. Space Filling Curves
3. Operations
4. DBMS interface
5. **Conclusion**



# Conclusion

- nD-PointClouds as 3<sup>rd</sup> representation: direct use (storage, analysis, visualization) or conversion to vector or raster
- develop functionality inside the database: encoding and decoding SFC, SFC ranges generation
- investigate different space-time-scale relative dimension representations in hypercube (for surface PC data, but also for more dynamic data: moving object trajectories)
- investigate other SFCs (Morton/Hilbert, less ranges)
- generation of blocks using the same integrations of space, time and scale (more efficient: less rows, block compression, ...)
- standardize streaming, progressive nD-PointCloud web-services

# Implementation / code

- Python code Dynamic Point Cloud available at:  
<https://github.com/stpsomad/DynamicPCDMS>
- C++ code for Morton/Hilbert encode/decode/range generation  
<https://github.com/kwan2004/SFCLib>
- Python and Rust code for running inside PostgreSQL  
<https://bitbucket.org/bmmeijers/sfc-rs>
- eScience Massive Point Cloud code (database/ viewer) & docu  
<http://pointclouds.nl>
- Oracle Database 12c  
(Enterprise Edition Release 12.1.0.1.2 – 64 bit)
  - Use of Index Organized Table (IOT)
  - NUMBER data type for 128 bit Morton/Hilbert keys

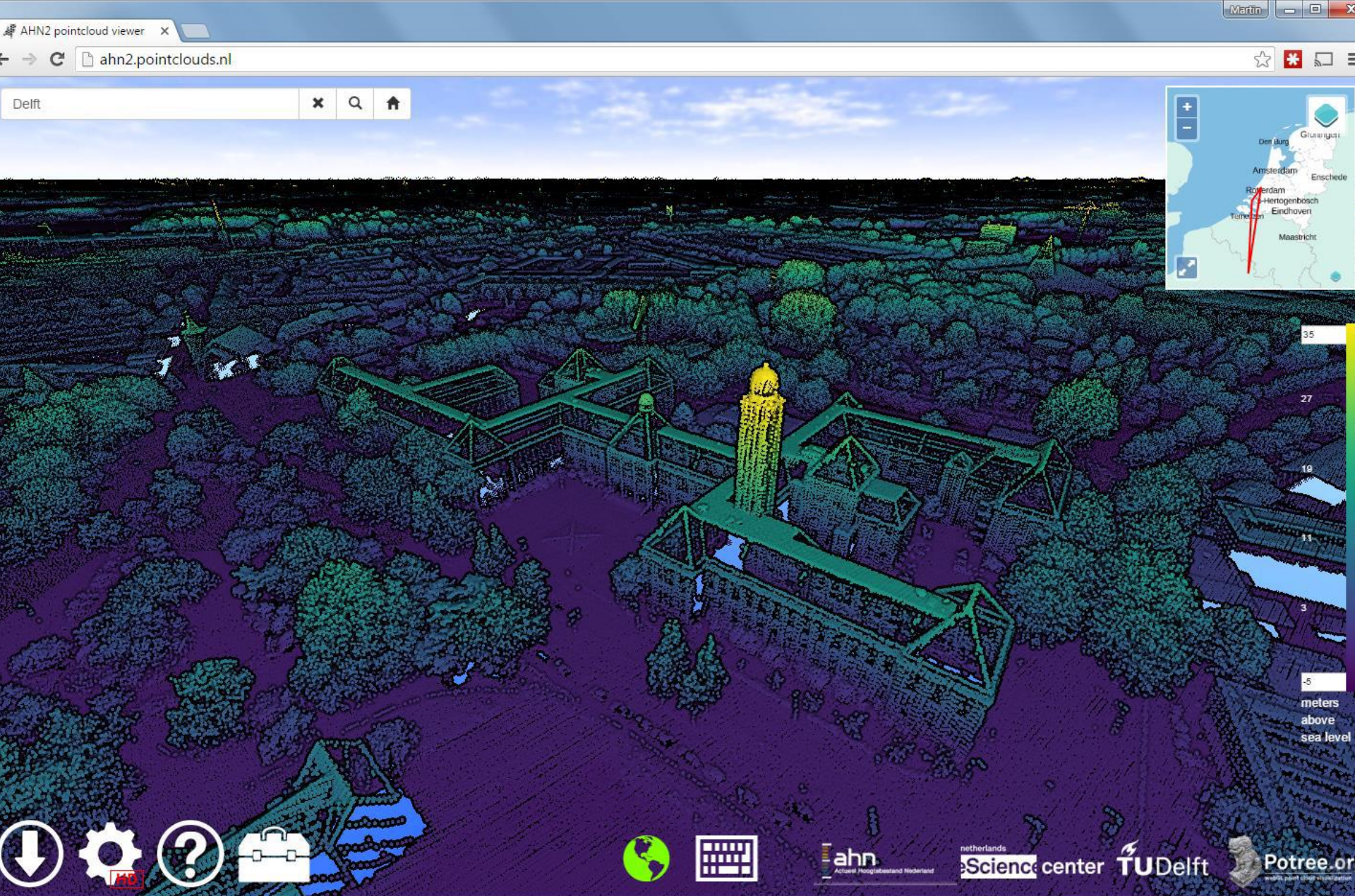
# Related projects and PhD theses

- Massive Point Clouds (NL): NL eScience Center, Oracle, RWS, Fugro, CWI/MonetDB, TUD <http://www.pointclouds.nl/>
- Harvest4D (EU): Uni Wien, TUD computer graphics <https://harvest4d.org/>
- IQumulus (EU): UCL, TUD, many more <http://iqmulus.eu/>
  
- Ahn Vu Vo: **Spatial Data Storage and processing Strategies for Urban Laser Scanning**, PhD thesis, University College Dublin, October 2016.
- Remi Cura: **Inverse Procedural Street Modelling from interactive to automatic reconstruction**, PhD thesis, University Paris Est (IGN/Thales), September 2016.
- Haicheng Liu: **Towards  $10^{15}$  points management – an nD PointCloud approach**, on-going PhD research, TU Delft

# Some Related Publications

- Xuefeng Guan, Peter van Oosterom, Bo Cheng, A Parallel N-dimensional Space-Filling-Curve Library and Application in Massive Point Cloud Management, to be submitted to ISPRS Int. J. Geo-Inf., 2018.
- Haicheng Liu, Peter van Oosterom, Martijn Meijers, Edward Verbree. Towards  $10^{15}$ -level point clouds management - a nD PointCloud structure, In: Proceedings of the 21th AGILE Conference on Geographic Information Science, Lund, pp. 7, 2018.
- Martijn Meijers, Wilko Quak, Peter van Oosterom, Archiving AIS messages in a Geo-DBMS, In: Proceedings of the 20th AGILE Conference on Geographic Information Science, Wageningen University & Research, pp. 3, 2017.
- Stella Psomadaki, Peter van Oosterom, Theo Tijssen, Fedor Baart, Using a Space Filling Curve Approach for the Management of Dynamic Point Clouds, Chapter in: ISPRS Annals Volume IV-2/W1, 11th 3D Geoinfo Conference, Athens, pp. 107-118, 2016.
- Stella Psomadaki, Using a Space Filling Curve for the Management of Dynamic Point Cloud Data in a Relational DBMS, Master's thesis, Delft University of Technology, pp. 158, 2016.
- Irene de Vreede, Managing Historic Automatic Identification System data by using a proper Database Management System structure, Master's thesis, Delft University of Technology, pp. 90, 2016.
- Peter van Oosterom, Oscar Martinez-Rubi, Milena Ivanova, Mike Horhammer, Daniel Geringer, Siva Ravada, Theo Tijssen, Martin Kodde, Romulo Gonçalves, Massive point cloud data management: Design, implementation and execution of a point cloud benchmark, In: Computers & Graphics, 49, pp. 92-125, 2015.







# Steaming Point Cloud webservices

- Web services protocol (request/selection, response)
  - Data format
  - Streaming, ordering, compression
  - Caching
  - Progressive refinement
  - Support LoD's
  - Visualization
- 
- Fitting in existing WxS (WCS, WFS) or new service needed (WPCS)?
  - Earlier work of OS Geo pointdown
    - <https://lists.osgeo.org/mailman/listinfo/pointdown>
    - <https://github.com/pointdown/protocol>

# Webservices

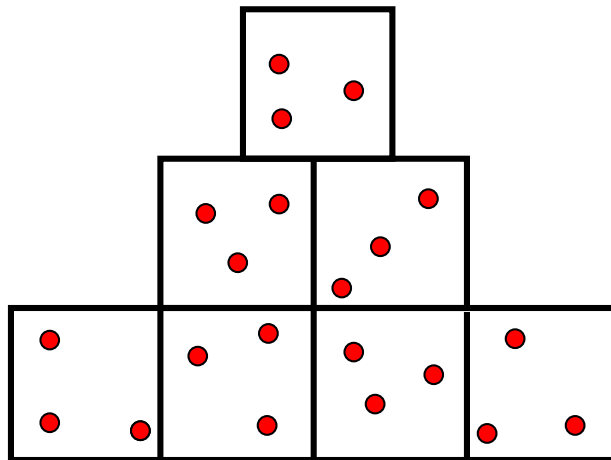
- better not try to standardize point clouds at database level (not much support/ partners expected), but rather focus on webservices level (more support/ partners expected)
- there is overlap between WMS, WFS and WCS...
- OGC point cloud DWG should explore if WCS is good start for point cloud services:
  - If so, then analyse if it needs extension
  - If not good starting point, consider a specific WPCS, web point cloud service standards (and perhaps further increase the overlapping family of WMS, WFS, WCS,... )



# Point cloud data pyramid

- overview queries just want top-subset
- detailed queries part of bottom-subset
- organize in data pyramid

2D schematic view, data blocks....

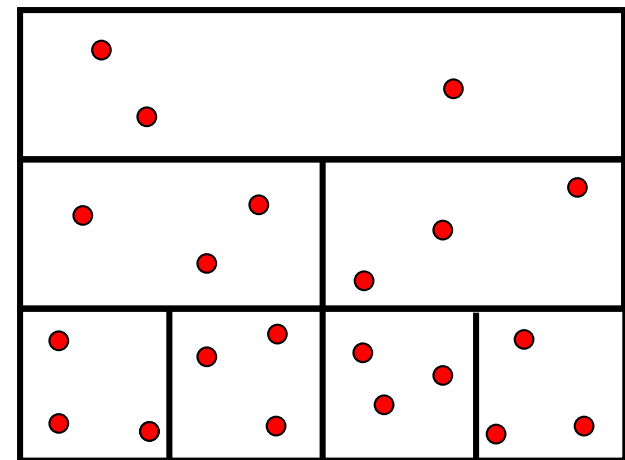


LoD 2

LoD 1

LoD 0

stretched over domain



density  
low



high

every next higher level, density  $2^k$  times less (2D  $\rightarrow 4$ , 3D  $\rightarrow 8$ )

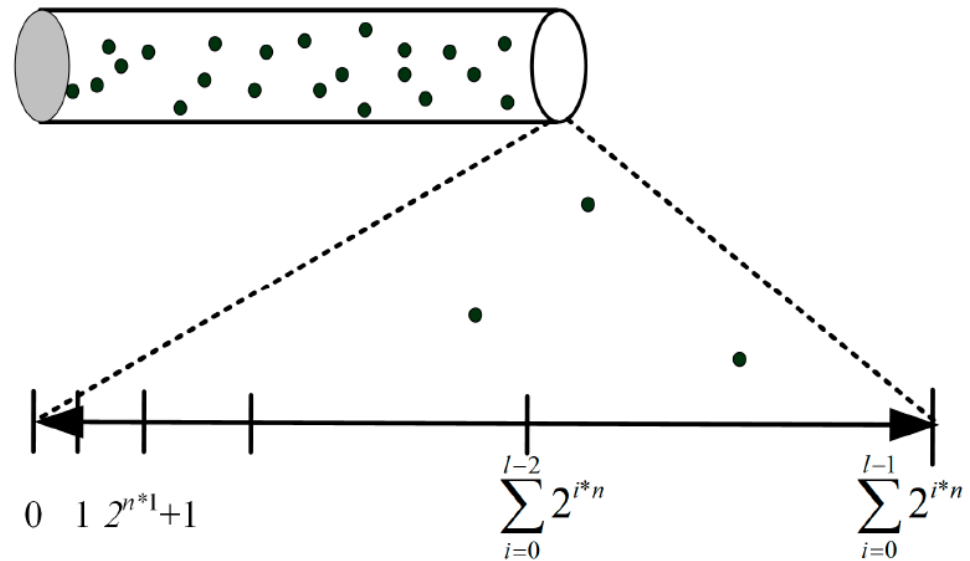
# Random LOD

## ideal distribution among levels

- $N(l)$  is number of points in  $nD$  space on level  $l$  ( $l=0$ : top level)
- $N(0) = 2^{n*0}$ ,  $N(1) = 2^{n*1}$ , .....,  $N(l) = 2^{n*l}$
- The  $l+1$  ranges will be defined:

$$[0, 1], [1, 2^n + 1], [2^n + 1, 2^{n*2} + 2^n + 1], \dots, \left[ \sum_{i=0}^{l-2} 2^{i*n}, \sum_{i=0}^{l-1} 2^{i*n} \right]$$

- Uniform random sampling used to distribute point over the level ( $l+1$  ranges):
- Implemented in SFCLib (fast C++ parallel)



# Getting rid of discrete level postprocessing (1/2)

## Real-Time Continuous Level of Detail Rendering of Point Clouds

Markus Schütz\*  
TU Wien

Katharina Krösl†  
TU Wien,  
VRVis Forschungs-GmbH

Michael Wimmer‡  
TU Wien

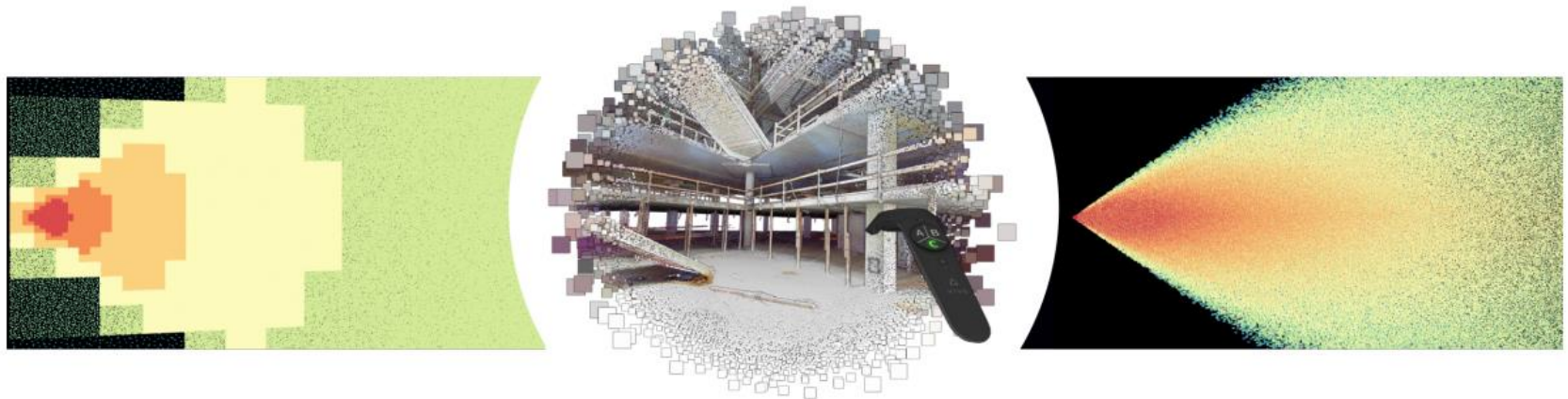
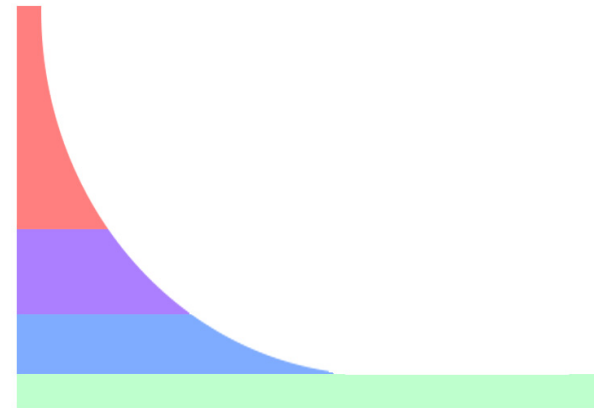
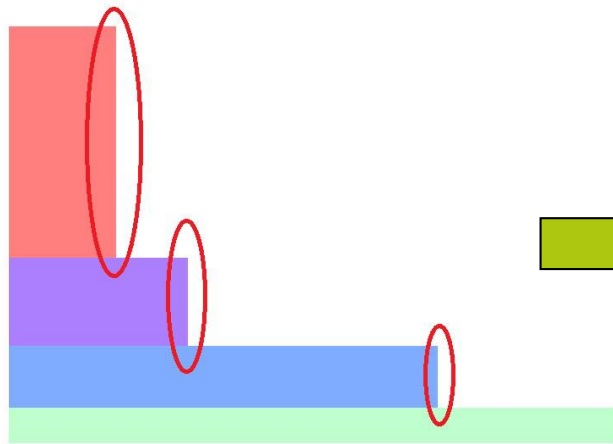


Figure 1: Left: Discrete LOD structure with sudden jumps in density and popping artifacts under motion. Middle: Screenshot of our method in virtual reality. No visible seams across different levels of detail due to a continuous reduction in density, and also continuously reduced density towards the periphery to reduce geometric complexity where less is needed. Endeavor point cloud courtesy of NVIDIA [4]. Right: Continuous transition from one LOD to another.

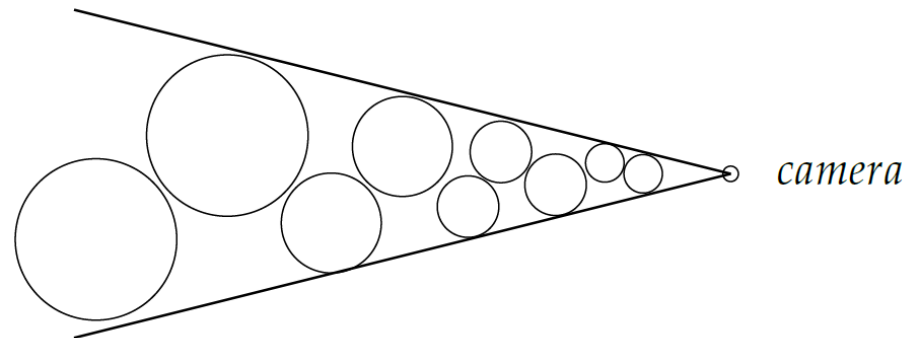
- Paper at IEEE VR 2019 (postprocessing at GPU, from 86.000.000 to 5.000.000 points every 5 to 6 frames → 90 fps left+right = 180 fps)

# Getting rid of discrete level postprocessing (2/2)

- MSc Geomatics thesis 'Vario-scale visualization of the AHN2 point cloud' by Jippe van der Maaden (TU Delft, April 2019).

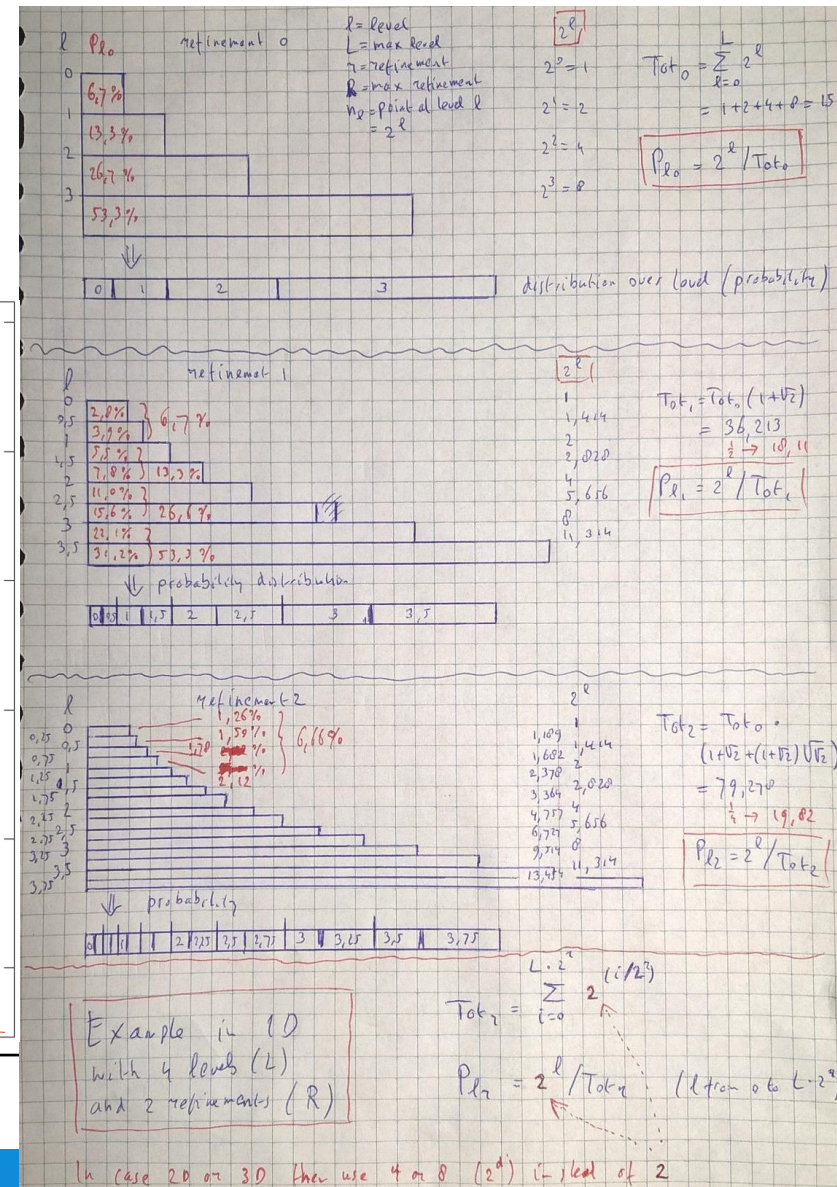
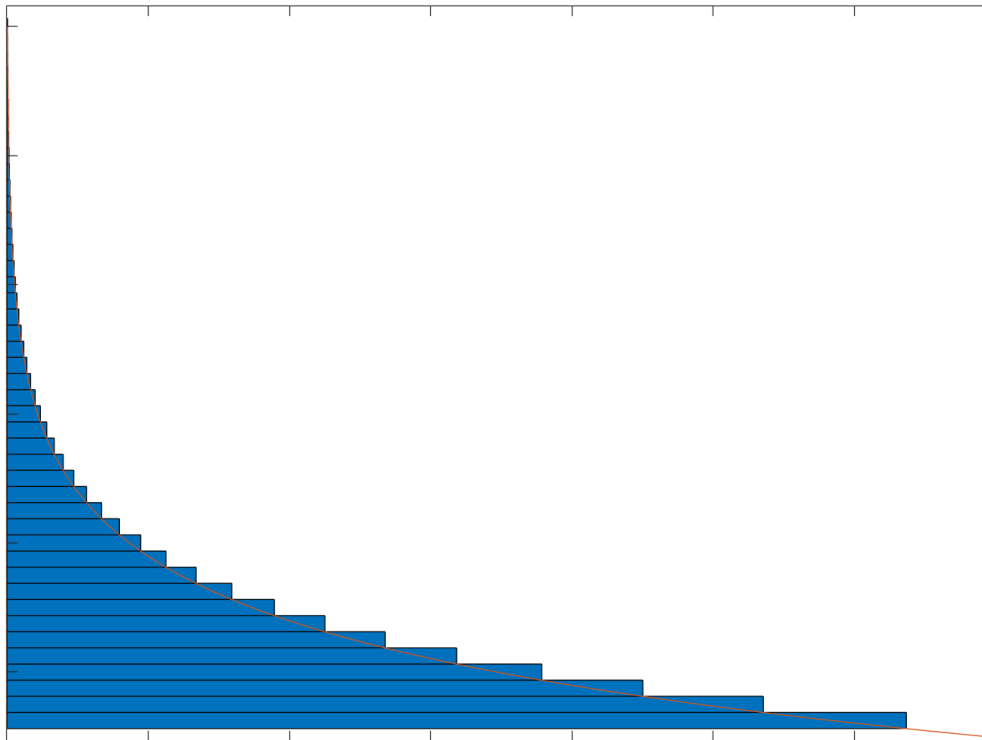


- Assign radius to points based on distance to camera
- Select points with empty circle/spheres

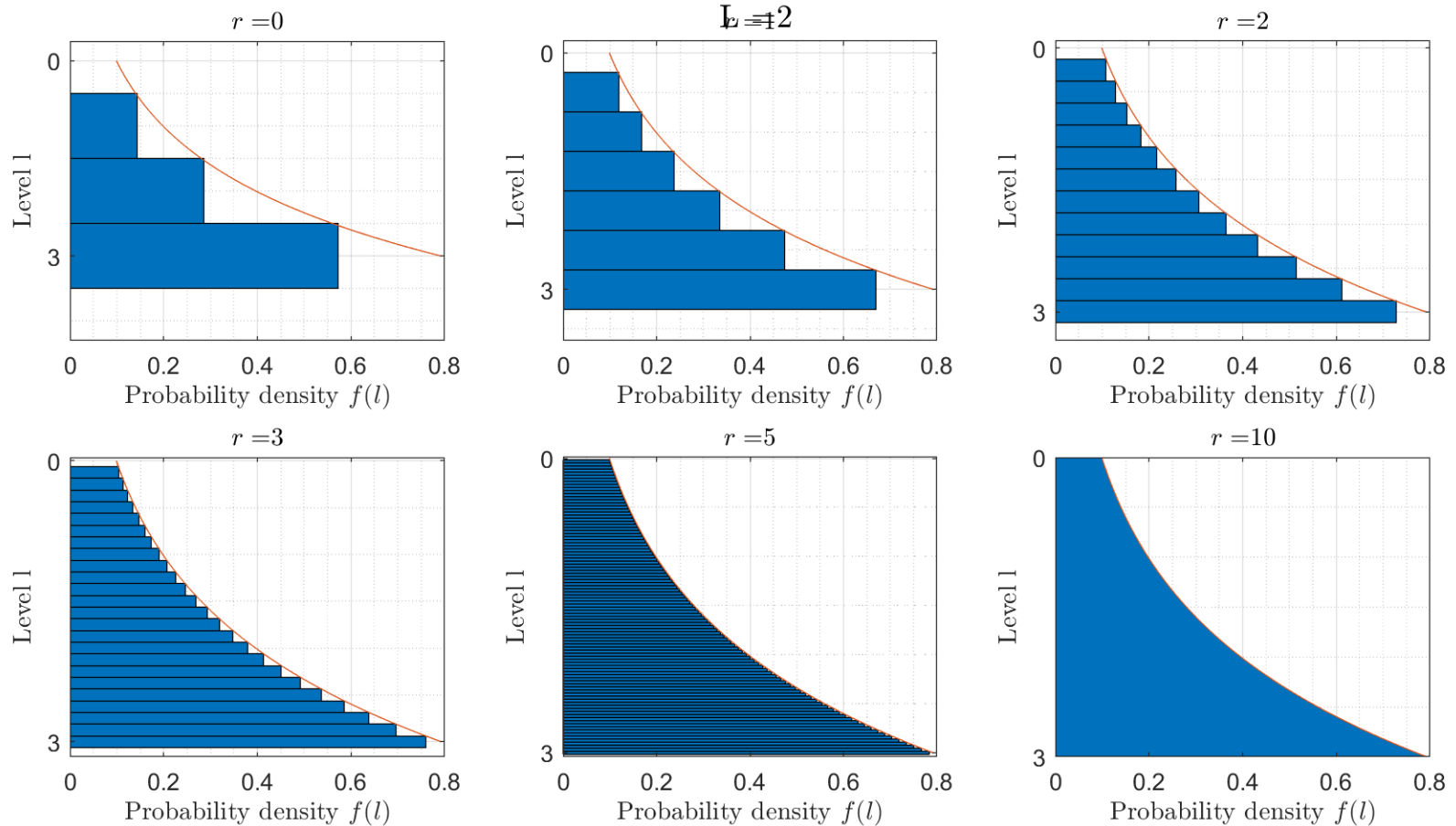


# Getting rid of discrete level refined discrete levels

- $L=10$  (11 levels: 0..10)
- $R=2$  refinements  $2^2=4$  sublevels:
- Ideal probability function (Simon van Oosterom, Matlab)

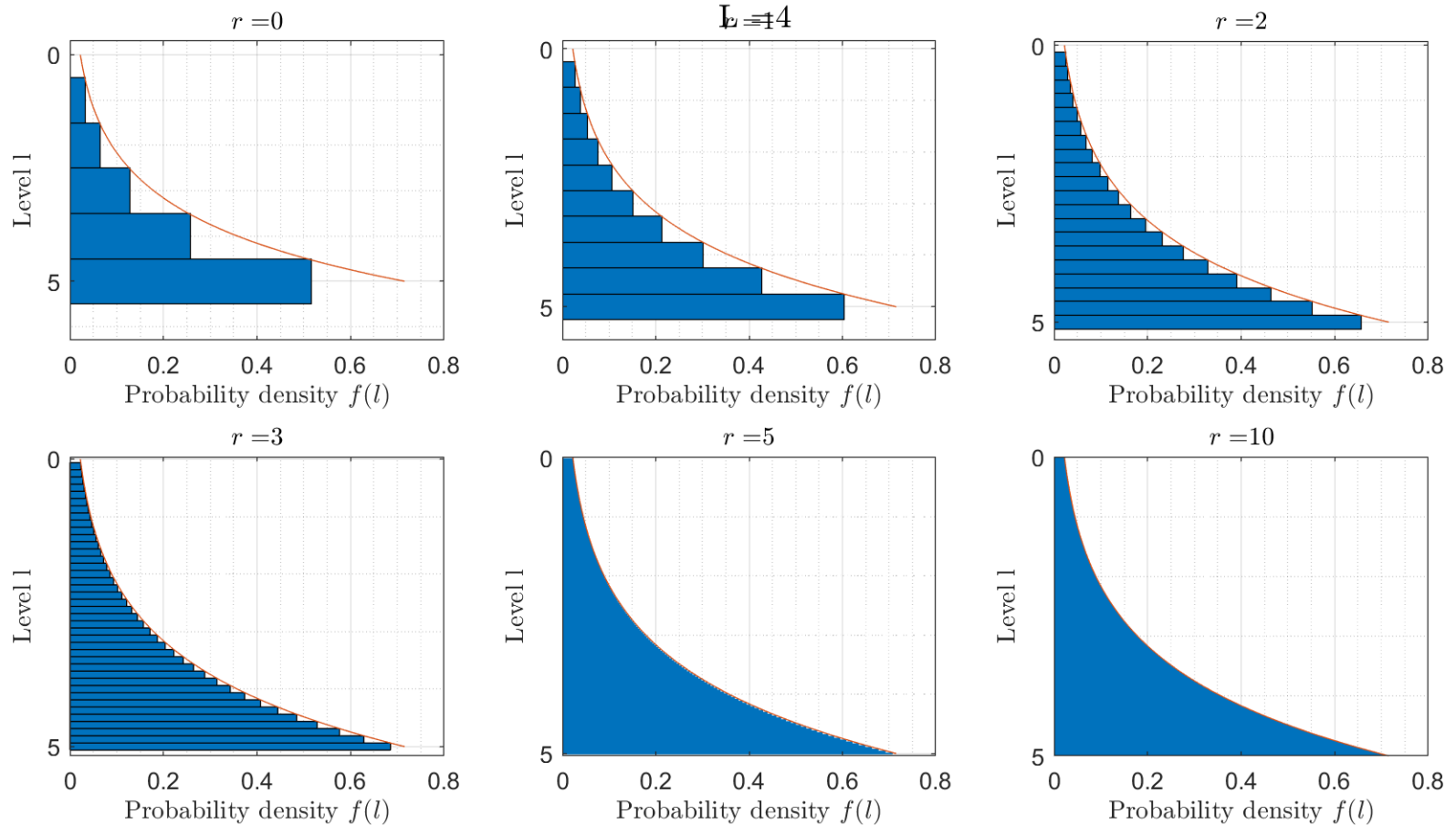


# Refined discrete levels ( $L=2: 0, 1, 2$ )

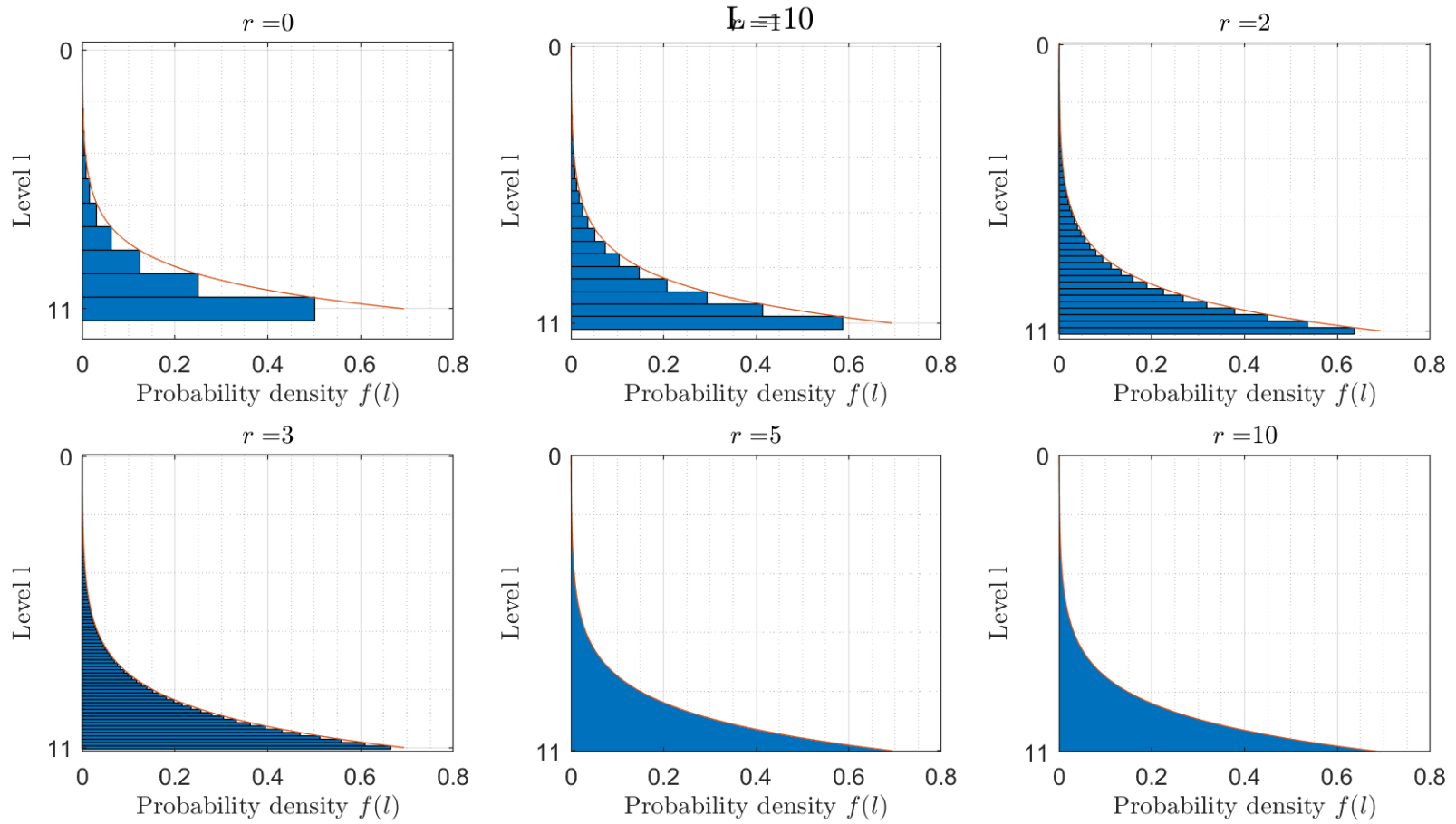




# Refined discrete levels ( $L=4: 0, 1, \dots, 4$ )

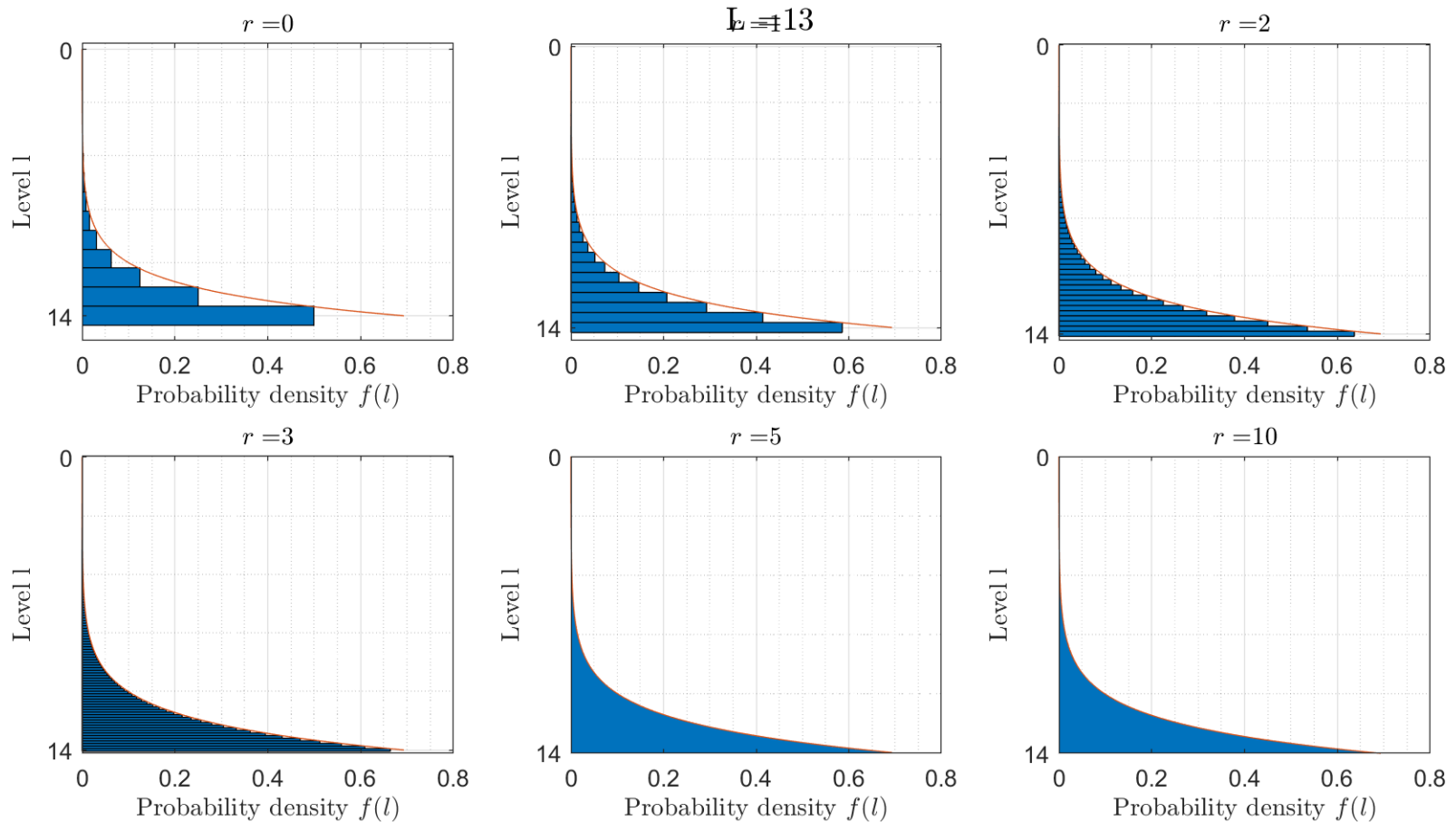


# Refined discrete levels ( $L=10$ : 0, 1, ..10)

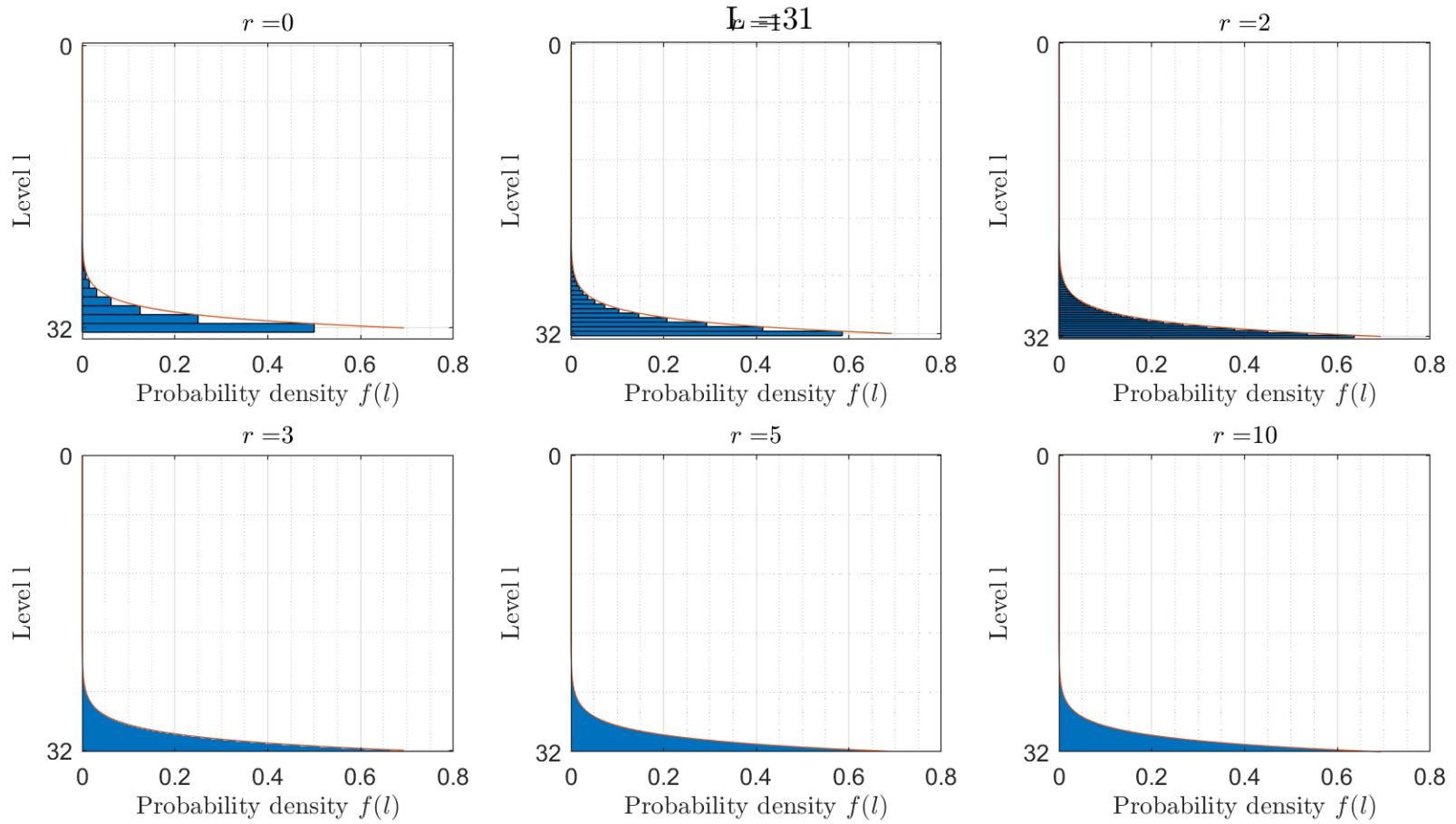




# Refined discrete levels ( $L=13: 0, 1, \dots, 13$ ) = AHN2 levels



# Refined discrete levels ( $L=31: 0, 1, \dots, 31$ )



# Getting rid of discrete level real continuous levels

- Ideal continuous probability function (1D case):

$$P(l) = \ln((2^{(L+1)} - 1) * U + 1) / \ln(2) \quad \text{with } l \text{ between } 0 \text{ and } L+1$$

- And for the nD case:

$$P(l) = \ln((2^{(n*(L+1))} - 1) * U + 1) / (n * \ln(2))$$

- Even easier to compute than discrete levels (no intervals needed)
- Can be rounded to discrete intervals with ideal distribution (if wanted)