Lesson 11
Handling massive terrains

GEO1015.2023

Hugo Ledoux

Raster pyramids == simple & easy & used

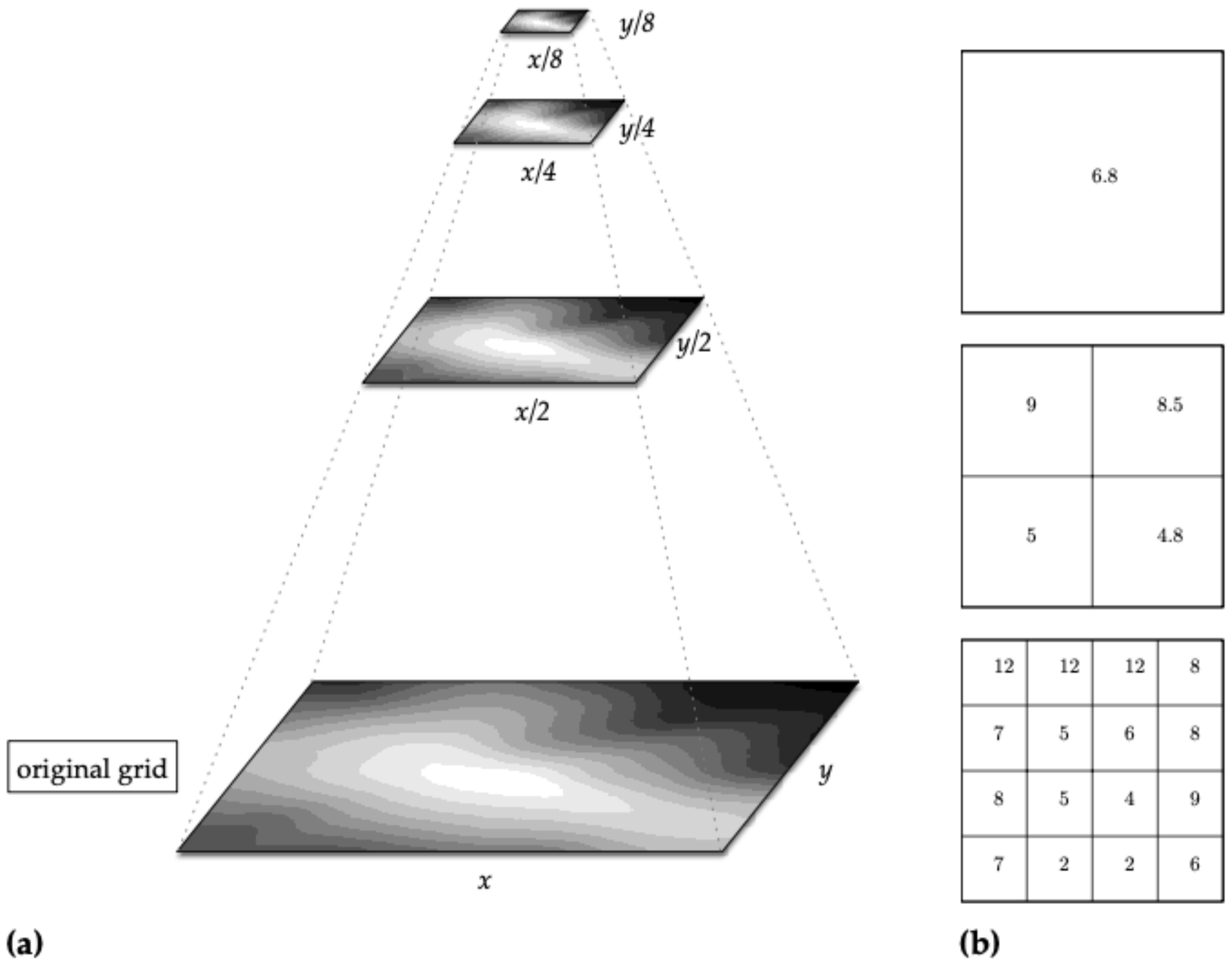


Figure 12.1: (a) The pyramid for a given raster file. **(b)** One 4 × 4 raster downsampled twice with average-method.

Raster pyramids == simple & easy & used

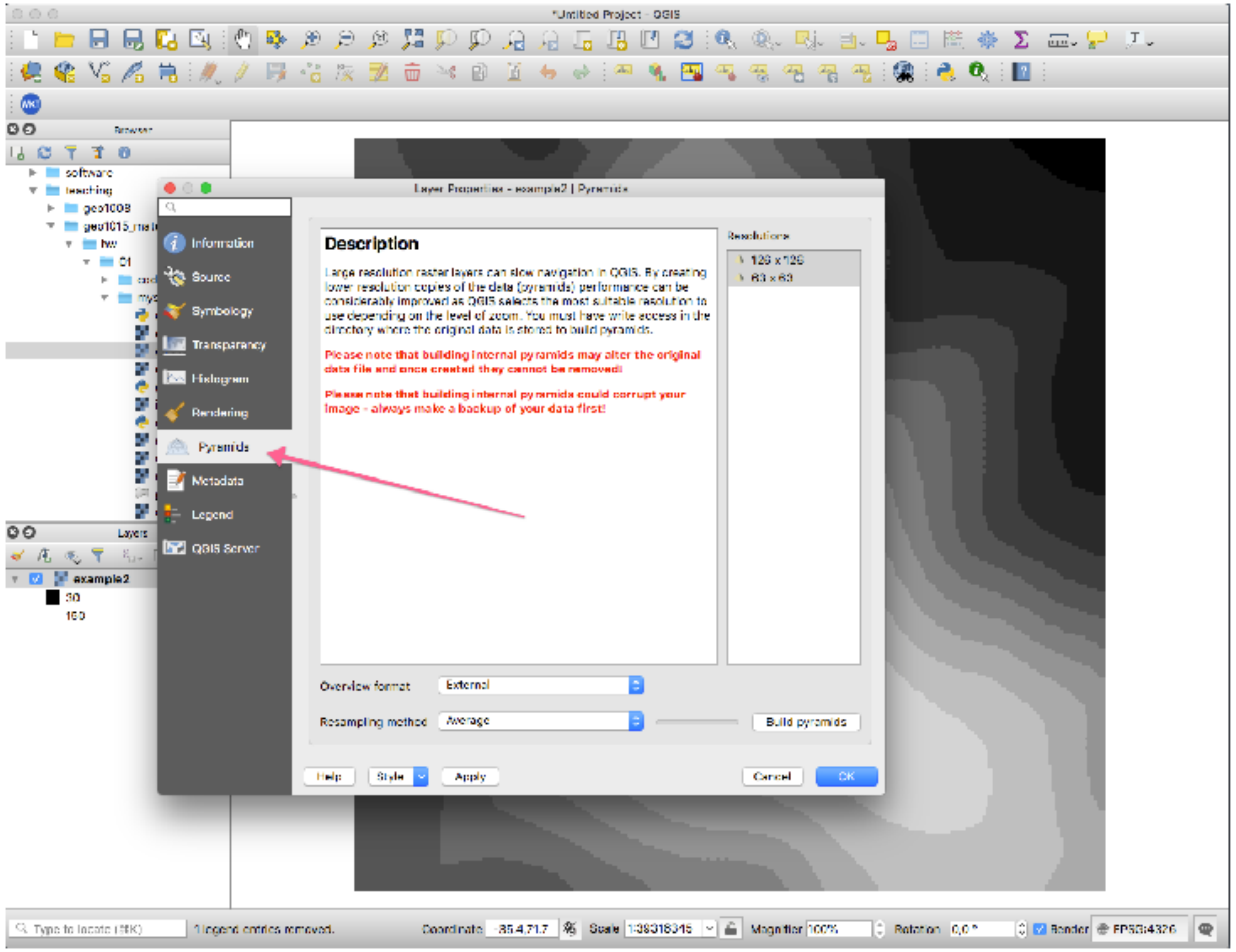


Figure 12.2: QGIS has the option to create the pyramids automatically.

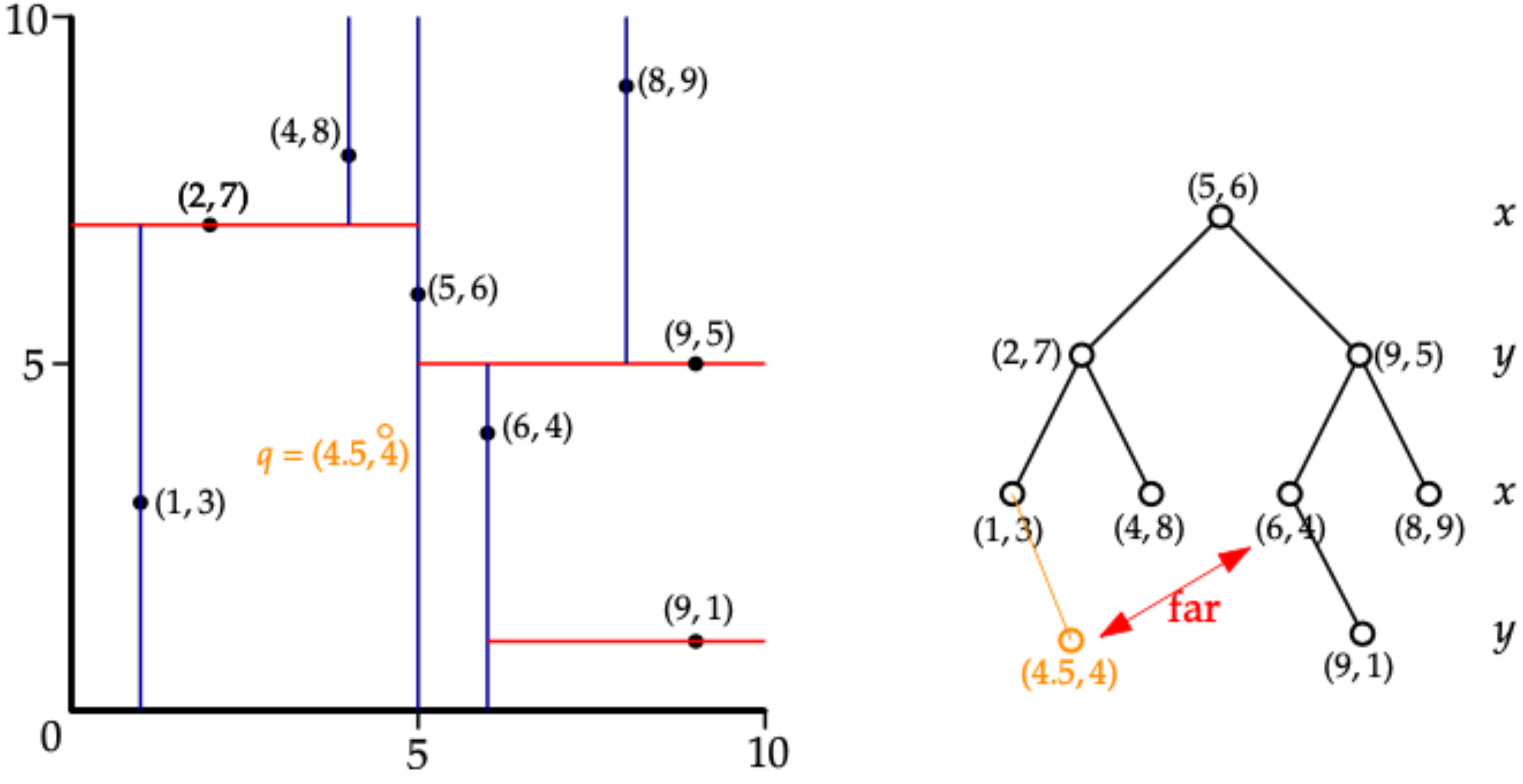
✿ How does it work in practice?

For certain GIS formats, eg GeoTIFF, the lower-resolutions rasters can be stored directly in the same file as the original raster, and this is standardised.

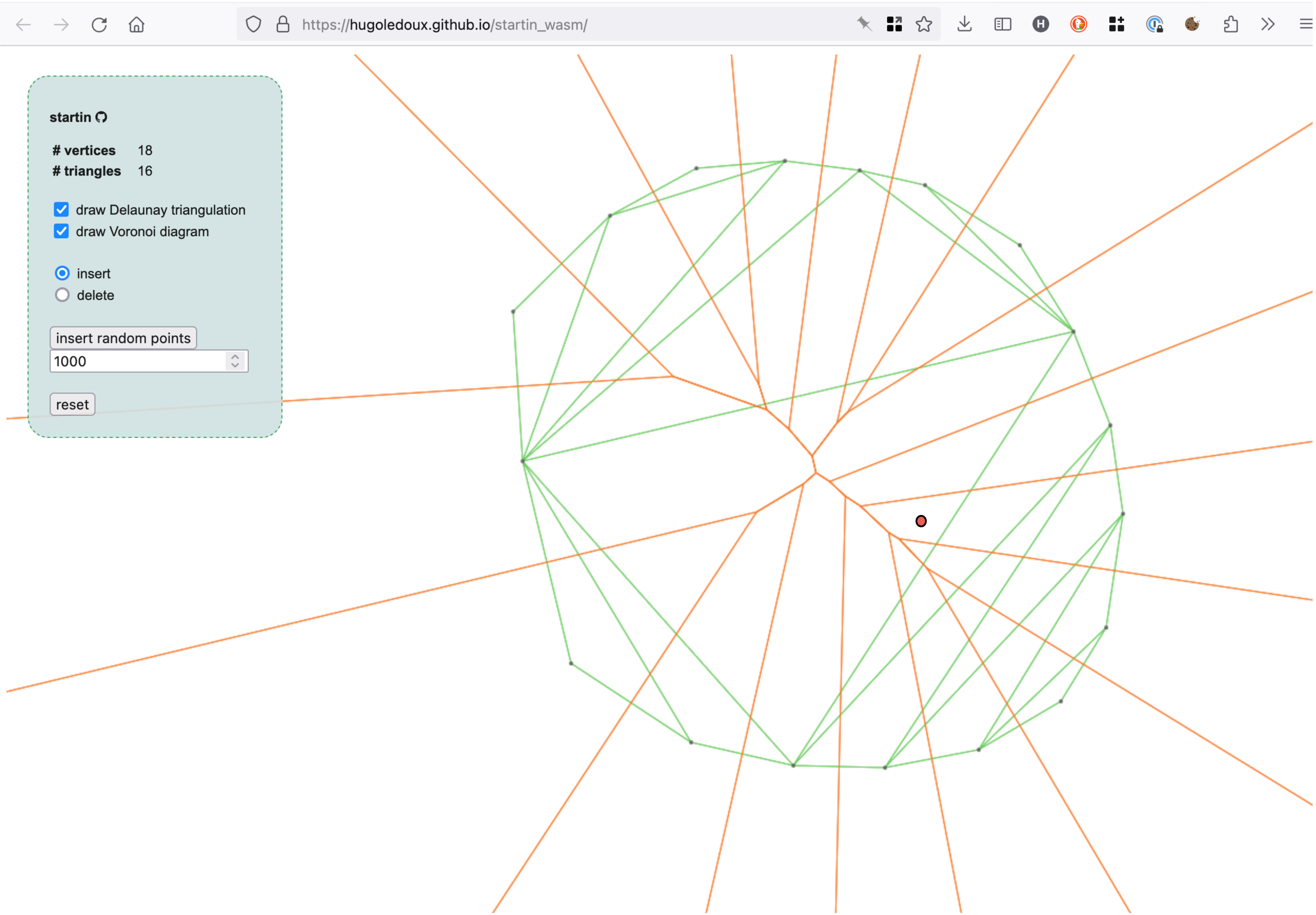
For other formats, if the GDAL library is used (the *de facto* open-source library for GIS images and grids), the pyramids can be stored in an auxiliary file with the extension `.ovr`, which is actually a TIFF format.

The GDAL utility gdaladdo can create automatically the pyramids for a few formats, and the downsampling method can be chosen. In QGIS, one can use `gdaladdo`, or there is also a built-in mechanism, as can be seen in Figure 12.2

Spatially indexing points (aka find nearest neighbourgs)



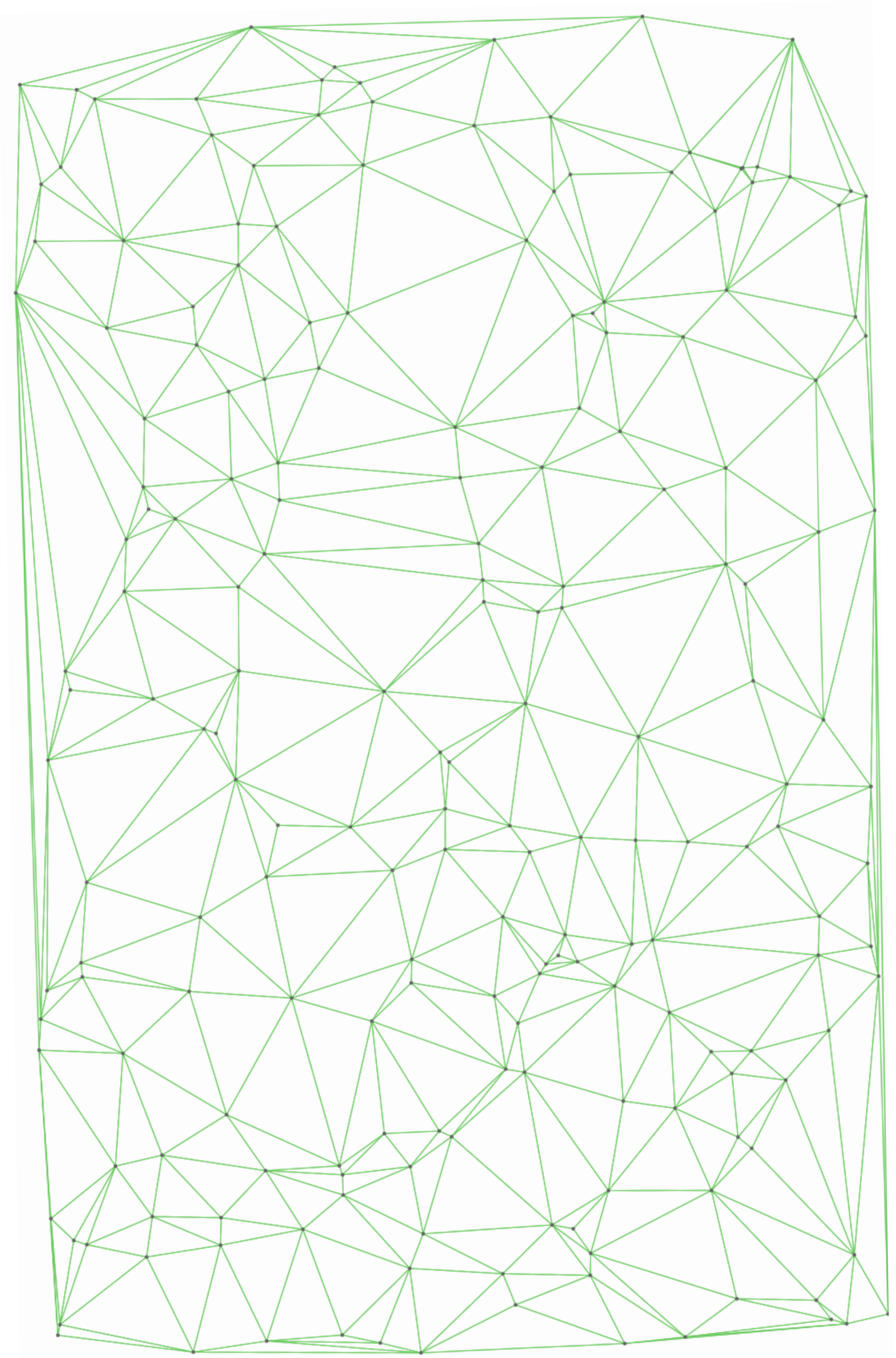
- Question: startinpy has a `closest_point()` function and yet it builds the DT: how does that work?

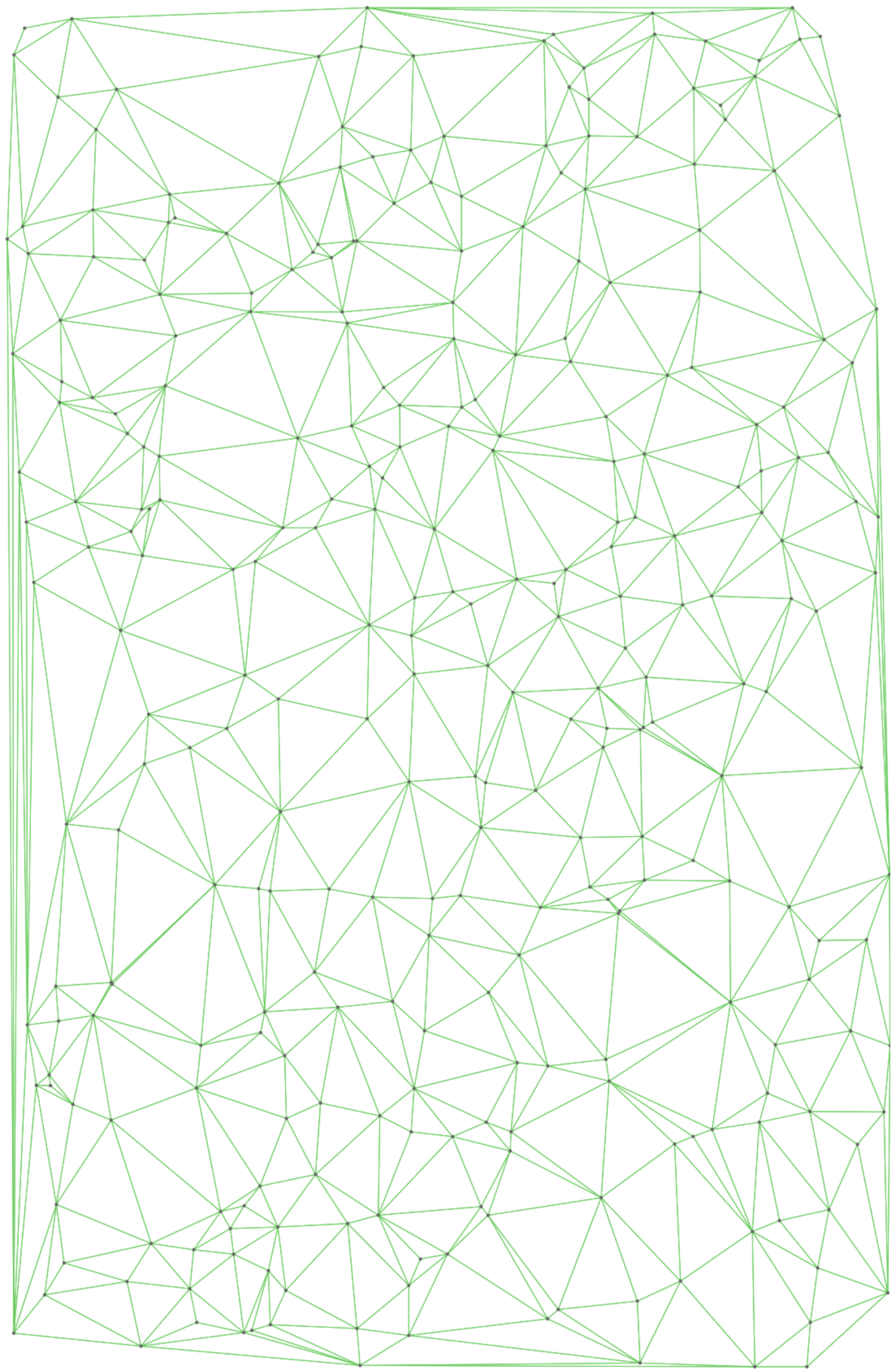

https://hugoledoux.github.io/startin_wasm/
startin
vertices 18
triangles 16
draw Delaunay triangulation
draw Voronoi diagram
insert
delete
insert random points
1000
reset

Triangulation of billions of points? Why should you care?

- (raw points not always the best: too many, noise, doesn't fit in memory, etc)
- We want to extract iso-contours
- We want to create a grid of the area
- Noise project of RIVM: one and only one "base terrain" of NL (to replicate all computations)
- Good simplification of an area necessitate creating the TIN in the first place 🙀

Partitioning the area into sub-tiles and triangulate each? Easy in theory…





What I describe today is basically this paper

Streaming Computation of Delaunay Triangulations

Martin Isenburg, University of California at Berkeley
Yuanxin Liu, University of North Carolina at Chapel Hill
Jonathan Shewchuk, University of California at Berkeley
Jack Snoeyink, University of North Carolina at Chapel Hill

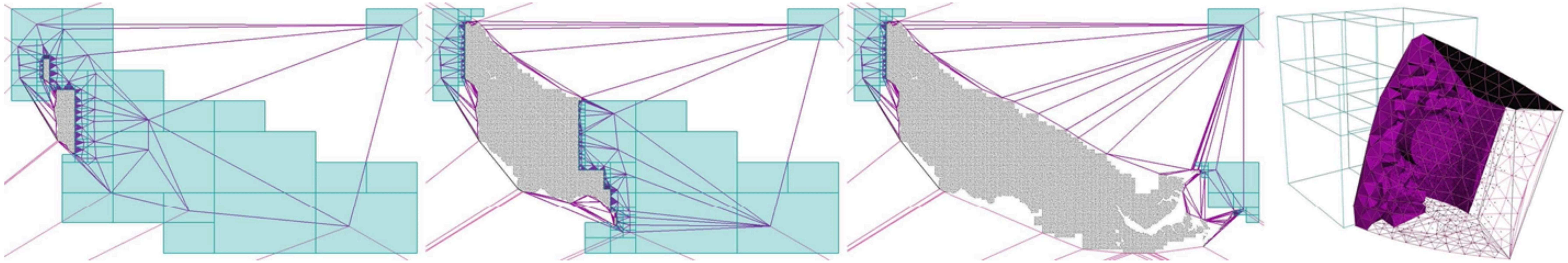


Figure 1: Streaming computation of Delaunay triangulations in 2D (Neuse River) and 3D. Blue quadrants or octants are unfinalized space where future points will arrive. Purple triangles and tetrahedra are in memory. Black points and their triangles and tetrahedra have already been written to disk or piped to the next application.

Abstract

We show how to greatly accelerate algorithms that compute Delaunay triangulations of huge, well-distributed point sets in 2D and 3D by exploiting the natural spatial coherence in a stream of points. We achieve large performance gains by introducing *spatial finalization* into point streams: we partition space into regions, and augment a stream of input points with finalization tags that indicate when a point is the last in its region. By extending an incremental algorithm for Delaunay triangulation to use finalization tags and produce streaming mesh output, we compute a billion-triangle terrain representation for the Neuse River system from 11.2 GB of LIDAR

Delaunay triangulator, by Agarwal, Arge, and Yi [2005]; see Section 6. We also construct a nine-billion-triangle, 152 GB triangulation in under seven hours using 166 MB of main memory.

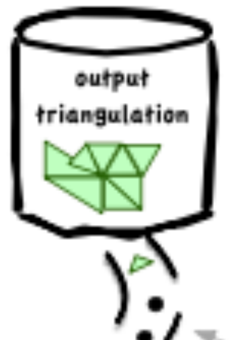
A *streaming* computation makes a small number of sequential passes over a data file (ideally, one pass), and processes the data using a memory buffer whose size is a fraction of the stream length. We have implemented two- and three-dimensional triangulators that read streams of points as input, and produce Delaunay triangulations in streaming mesh formats. The memory footprint of the 2D triangulator is typically less than 0.5% of the output mesh size (sometimes much less). The memory footprint of the 3D triangu-

Unix pipes for streaming

Streaming is realised with Unix pipes

The key to implementing streaming of geometries is to use Unix pipes (also called *pipelines*).
Pipelines were designed by Douglas McIlroy at Bell Labs during the development of Unix, and they allow to chain several processes together. The output of a process becomes the input of the next one, and so on (the data flowing through the process is the *stream*). Given 2 processes, the 2nd one can usually start before the 1st one has finished processing all the data.
In Unix, the pipe operator is the vertical line "`|`", and several commands can be chained with it: "`cmd1 | cmd2 | cmd3`". A simple example would be "`ls -l | grep json | wc -l`" which would:

1. list all the files in the current directory (one file name per line);
2. send this to the operator *grep* which would discard all lines not having the keyword `"json"`;
3. send this to the operator "`wc -l`" which counts the number of line.



Wavefront .obj file

From Wikipedia, the free encyclopedia

For other uses, see Obj (disambiguation).

OBJ geometry format

Filename extension	`.obj`
Internet media type	`model/obj` [1]
Developed by	Wavefront Technologies
Type of format	3D model format

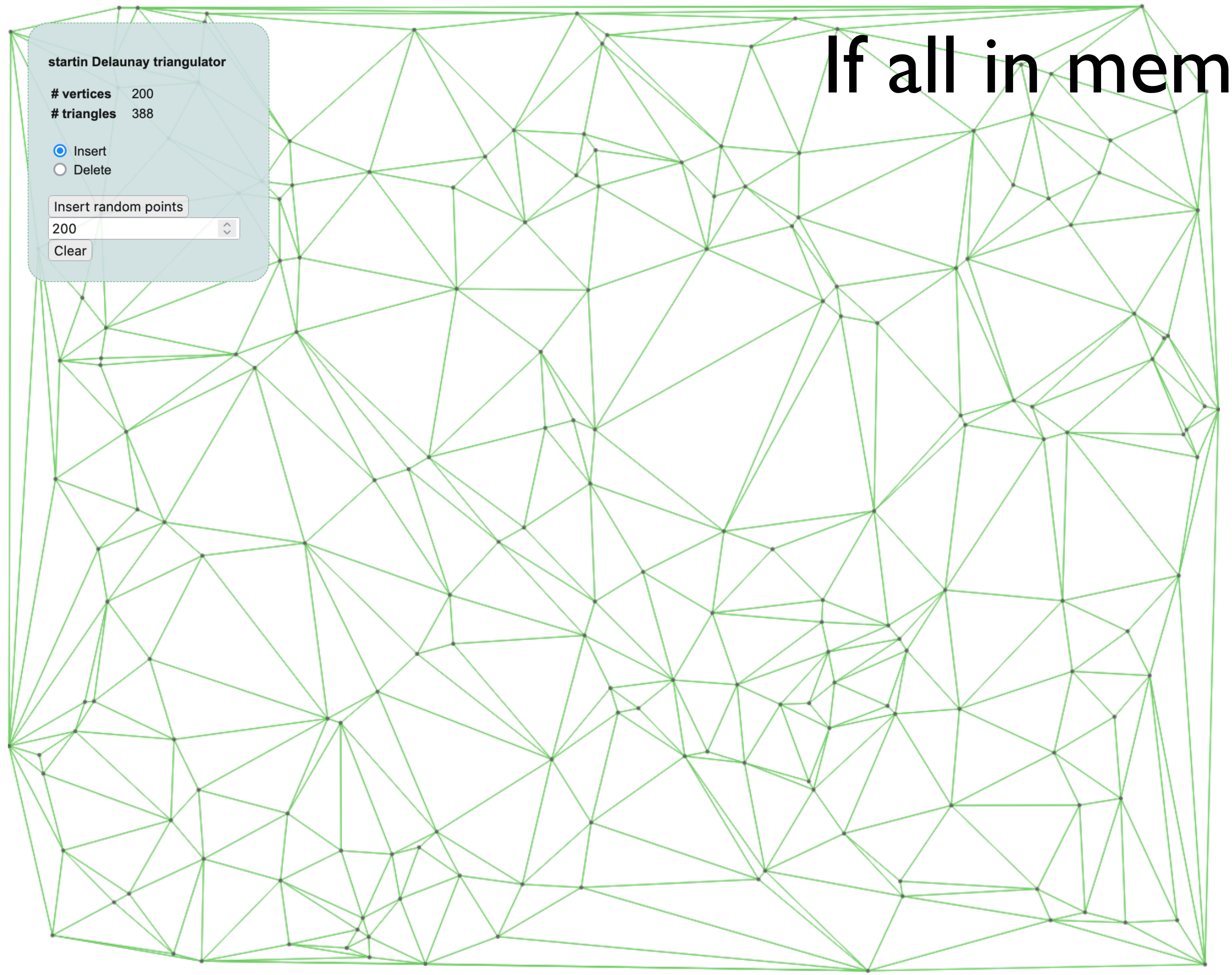
OBJ (or .OBJ) is a geometry definition file format first developed by Wavefront Technologies for its Advanced Visualizer animation package. The file format is open and has been adopted by other 3D graphics application vendors.

The OBJ file format is a simple data-format that represents 3D geometry alone — namely, the position of each vertex, the UV position of each texture coordinate vertex, vertex normals, and the faces that make each polygon defined as a list of vertices, and texture vertices. Vertices are stored in a counter-clockwise order by default, making explicit declaration of face normals unnecessary. OBJ coordinates have no units, but OBJ files can contain scale information in a human readable comment line.

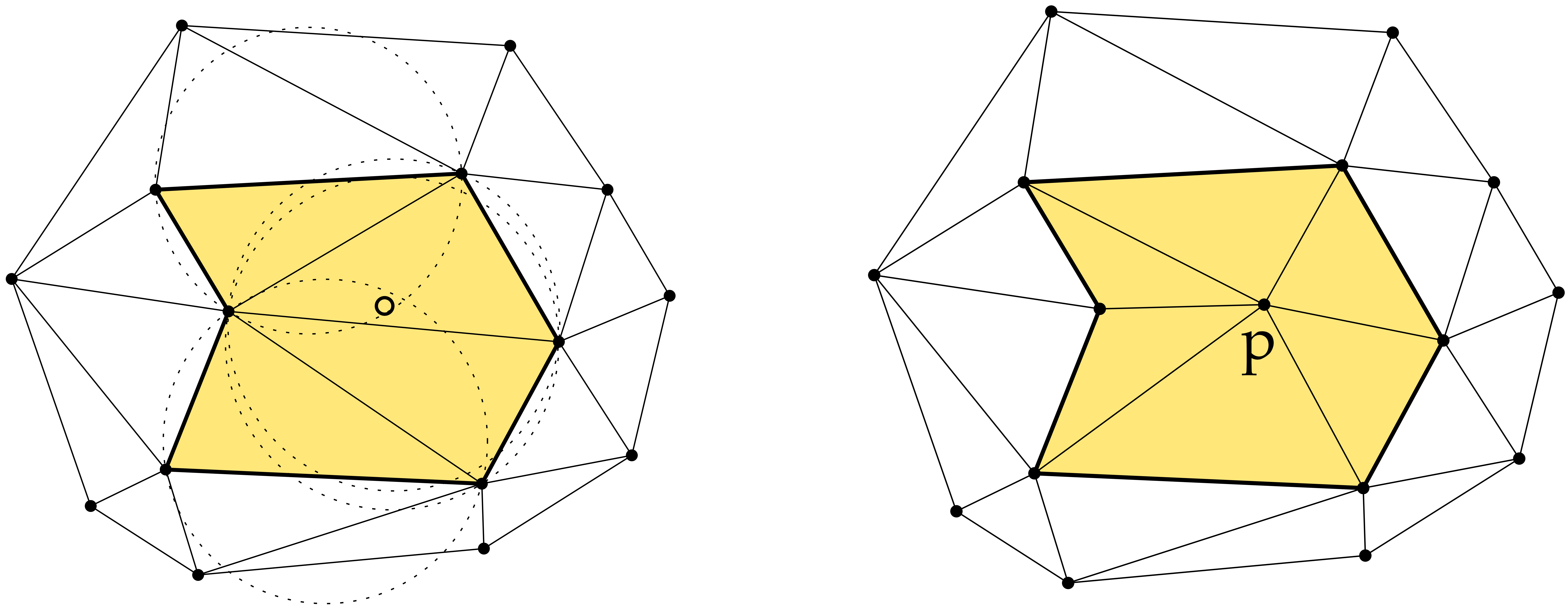
Contents [hide]

1 File format
1.1 Geometric vertex
1.2 Parameter space vertices
1.3 Face elements
1.3.1 Vertex indices
1.3.2 Vertex texture coordinate indices
1.3.3 Vertex normal indices
1.3.4 Vertex normal indices without texture coordinate indices
1.3.5 Line elements
1.4 Other geometry formats
1.5 Reference materials
1.6 Relative and absolute indices
2 Material template library
2.1 Basic materials
2.2 Texture maps
2.3 Texture options
2.4 Vendor specific alterations
2.4.1 Physically-based Rendering
3 See also

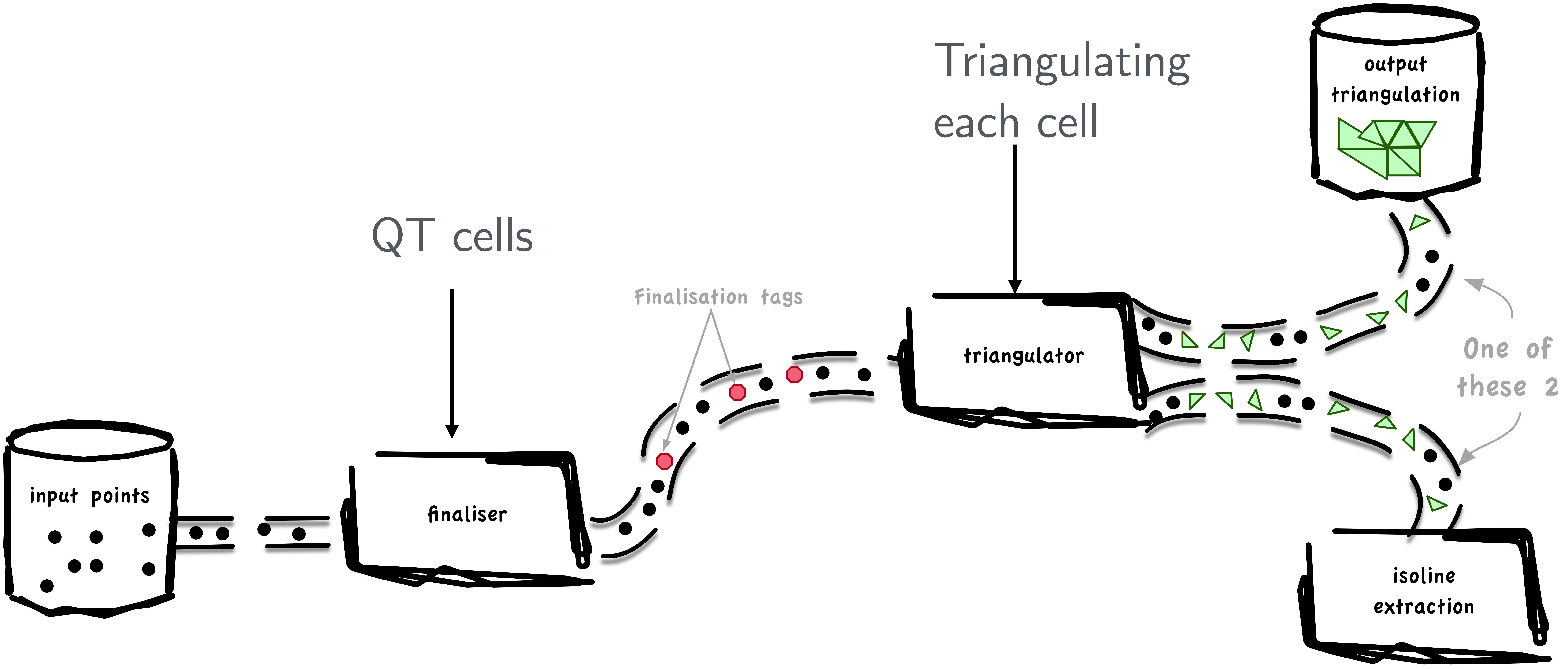
If all in memory it's "easy"



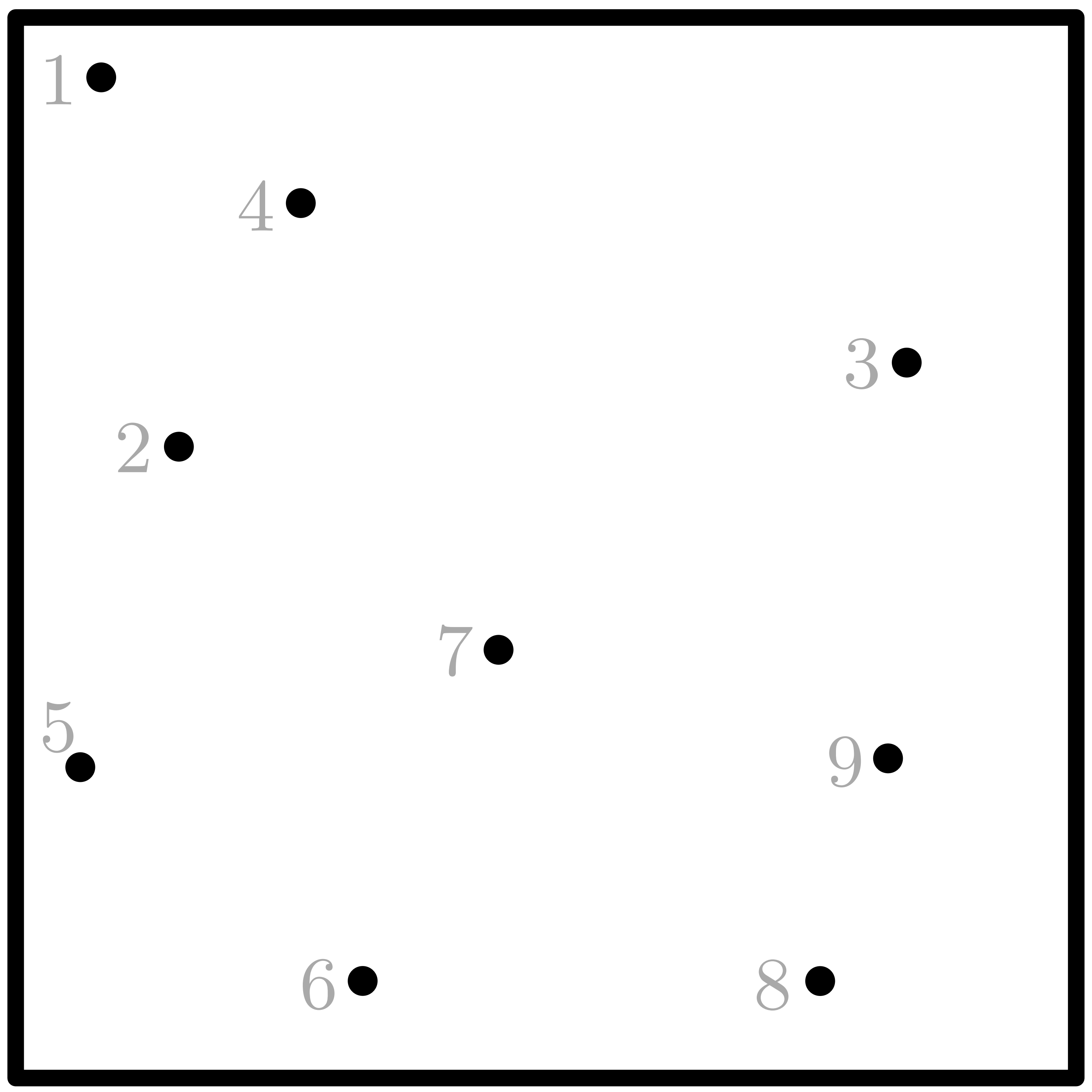
Insertion deletes all the triangles whose circumcircle contains p



Streaming DT architecture

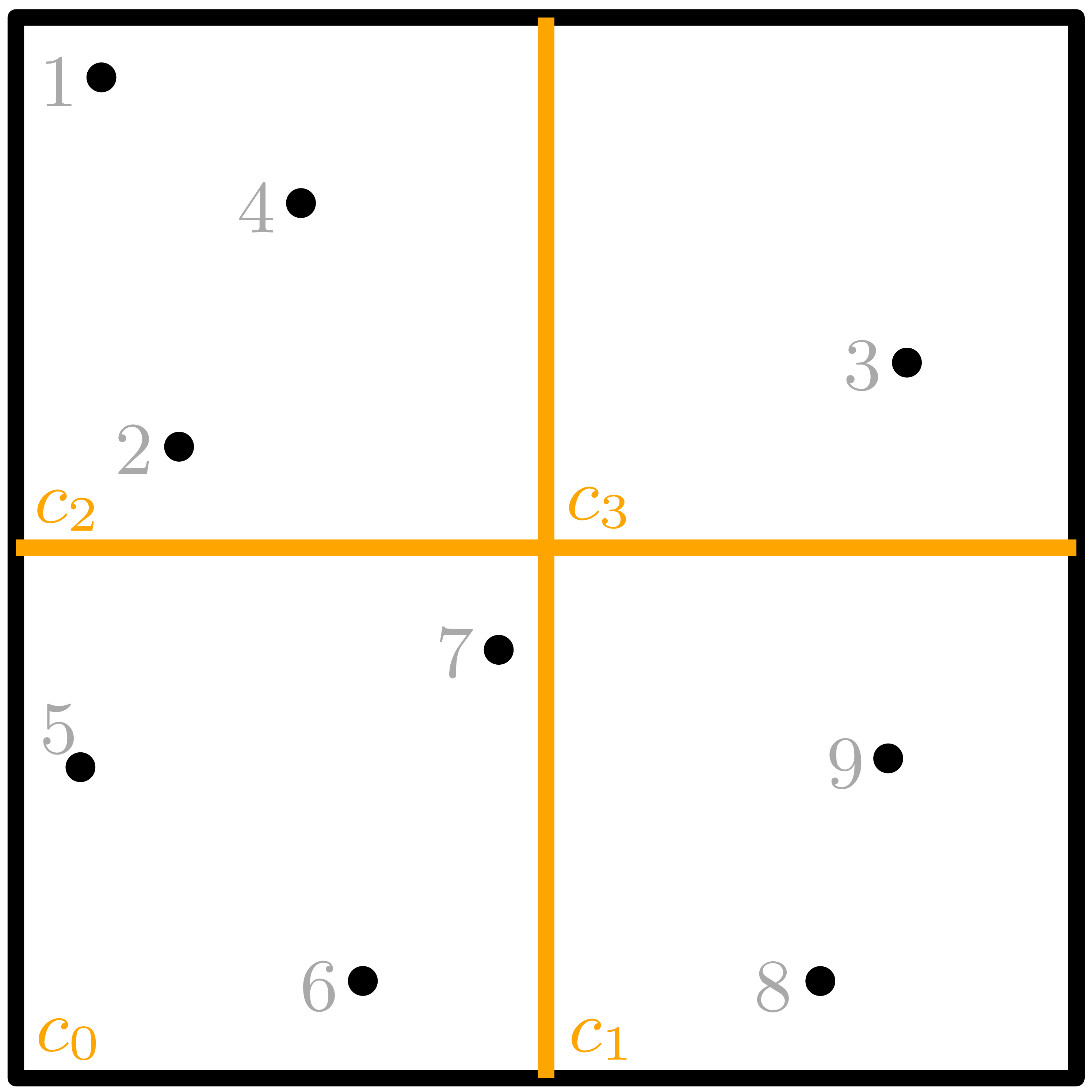


Finaliser based on quadtree



1. x y z
2. x y z
3. x y z
4. x y z
5. x y z
6. x y z
7. x y z
8. x y z
9. x y z

9 points and a ”normal” stream



1. x y z
2. x y z
3. x y z

finalise c_3

4. x y z

finalise c_2

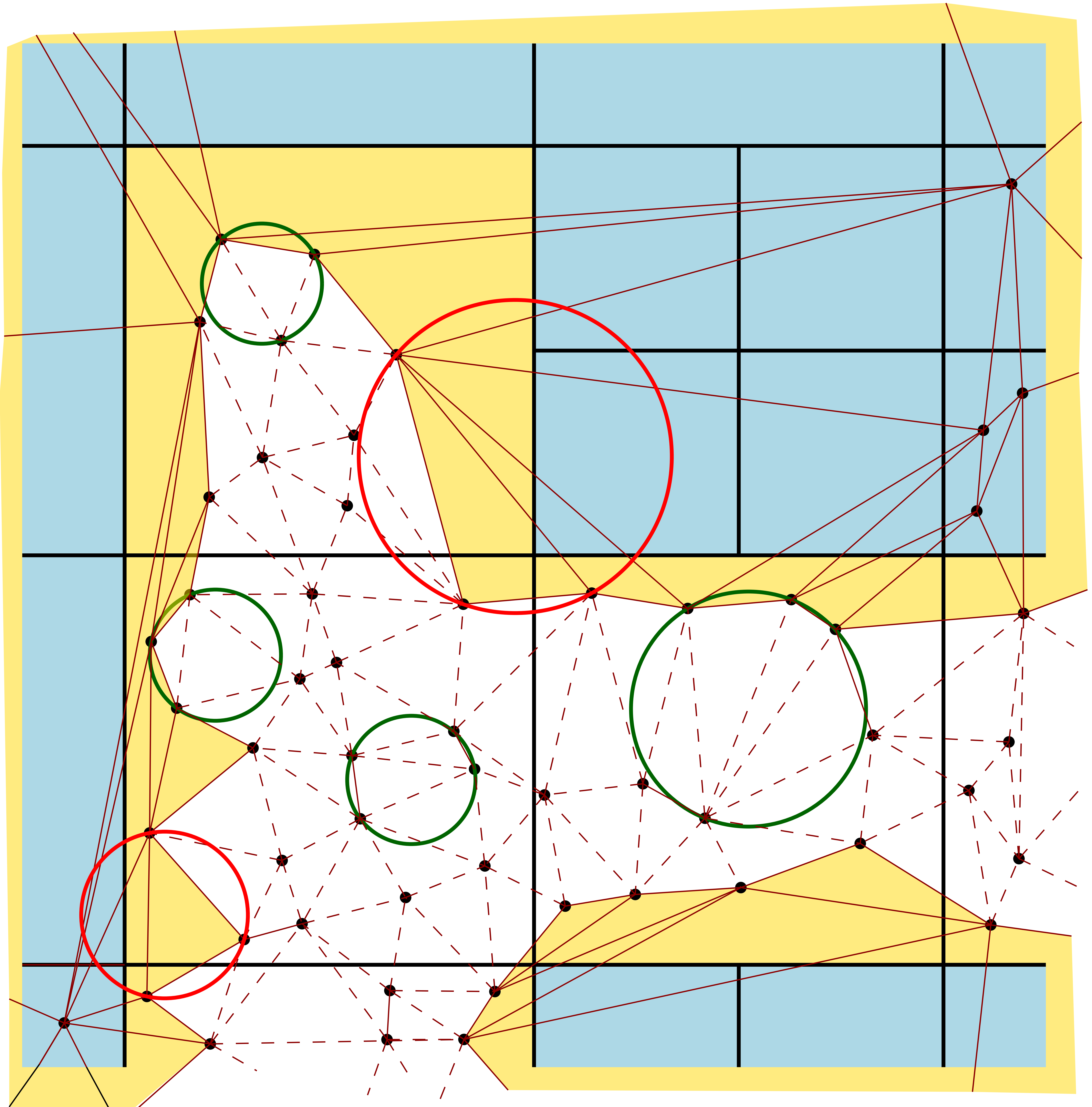
5. x y z
6. x y z
7. x y z

finalise c_0

8. x y z
9. x y z

finalise c_1

9 points and a stream with finalisation tags



blue cell = not finalised cell

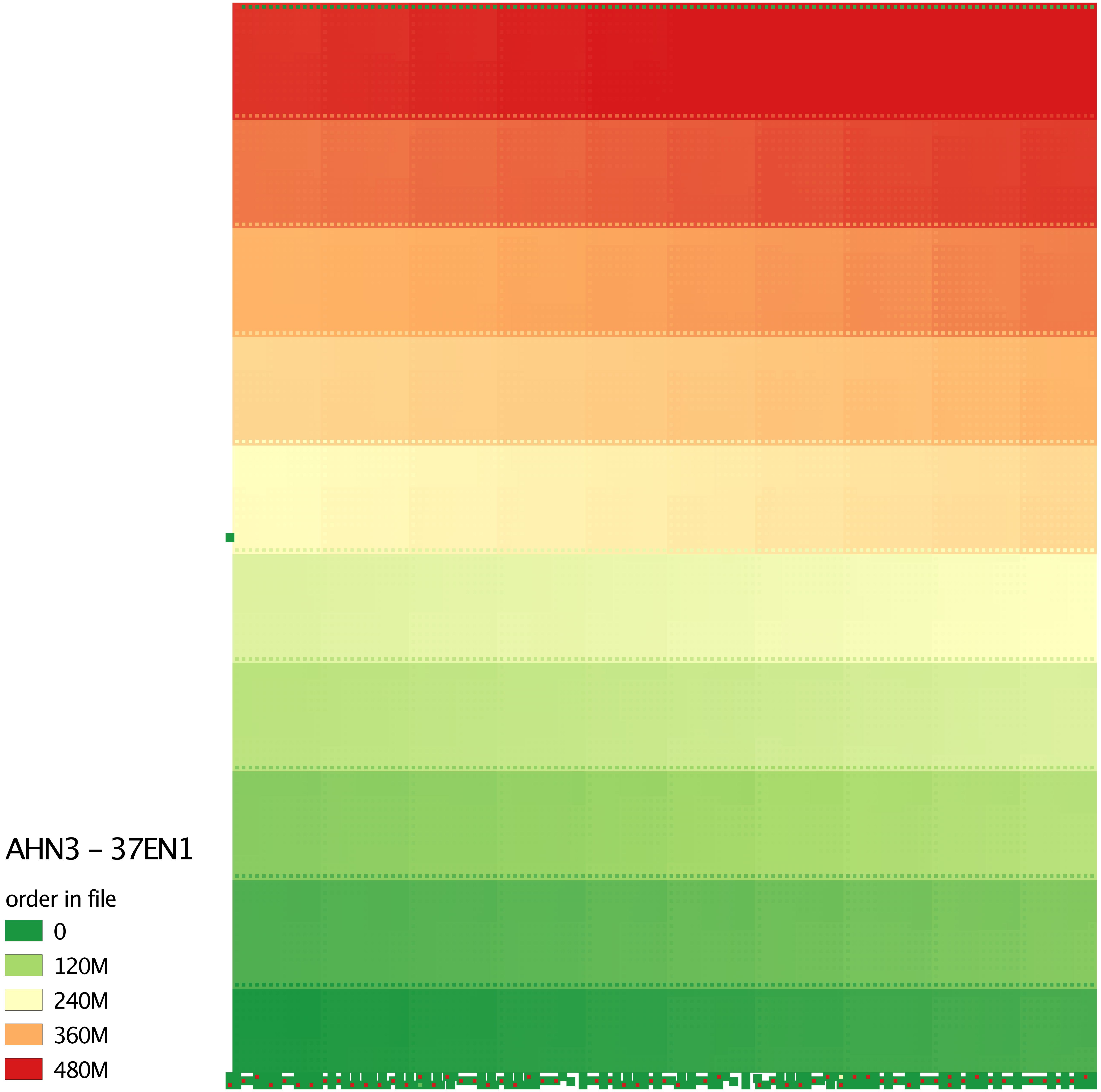
Yellow tr = in memory

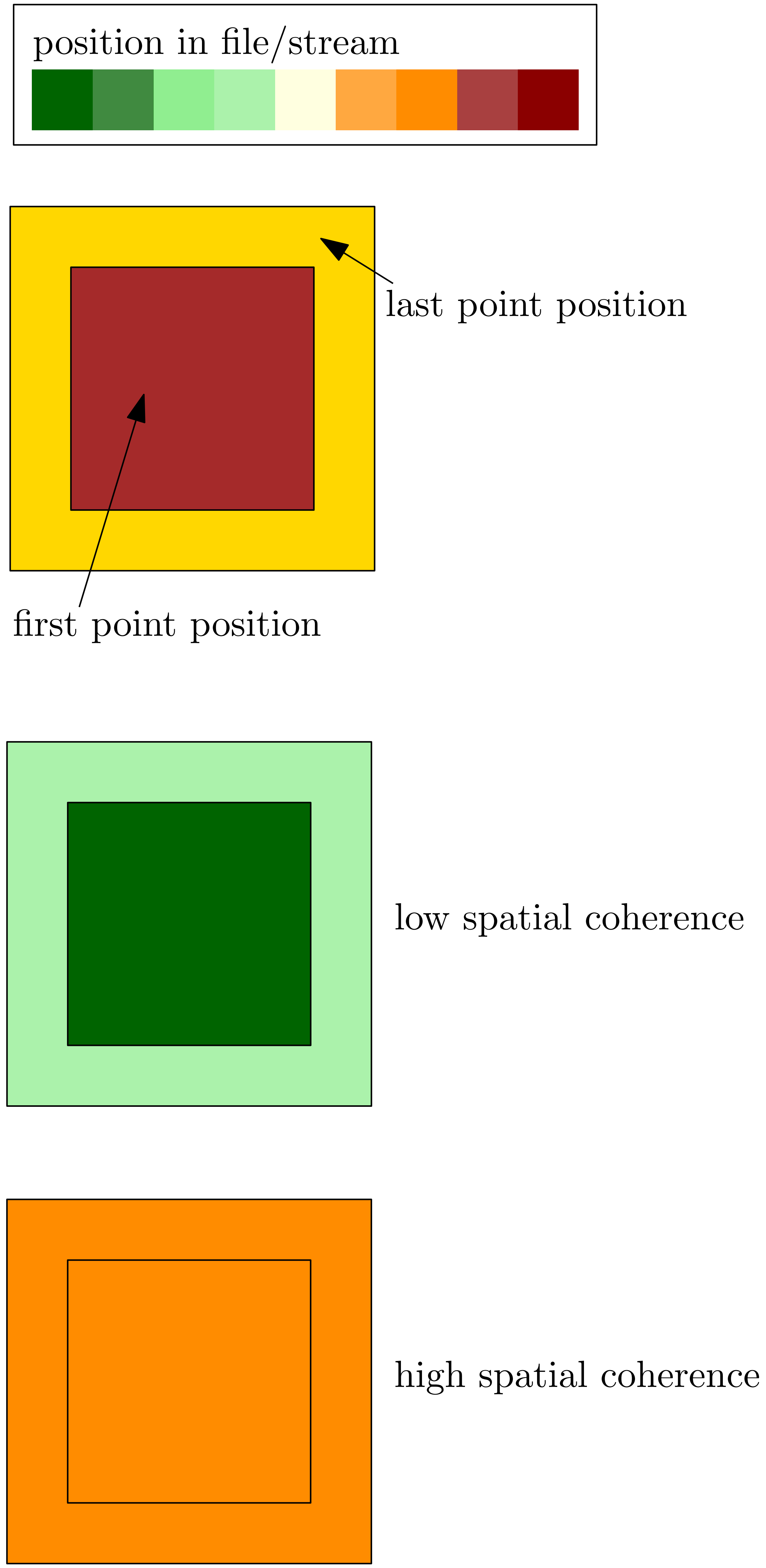
White tr = written to stream

Spatial coherence

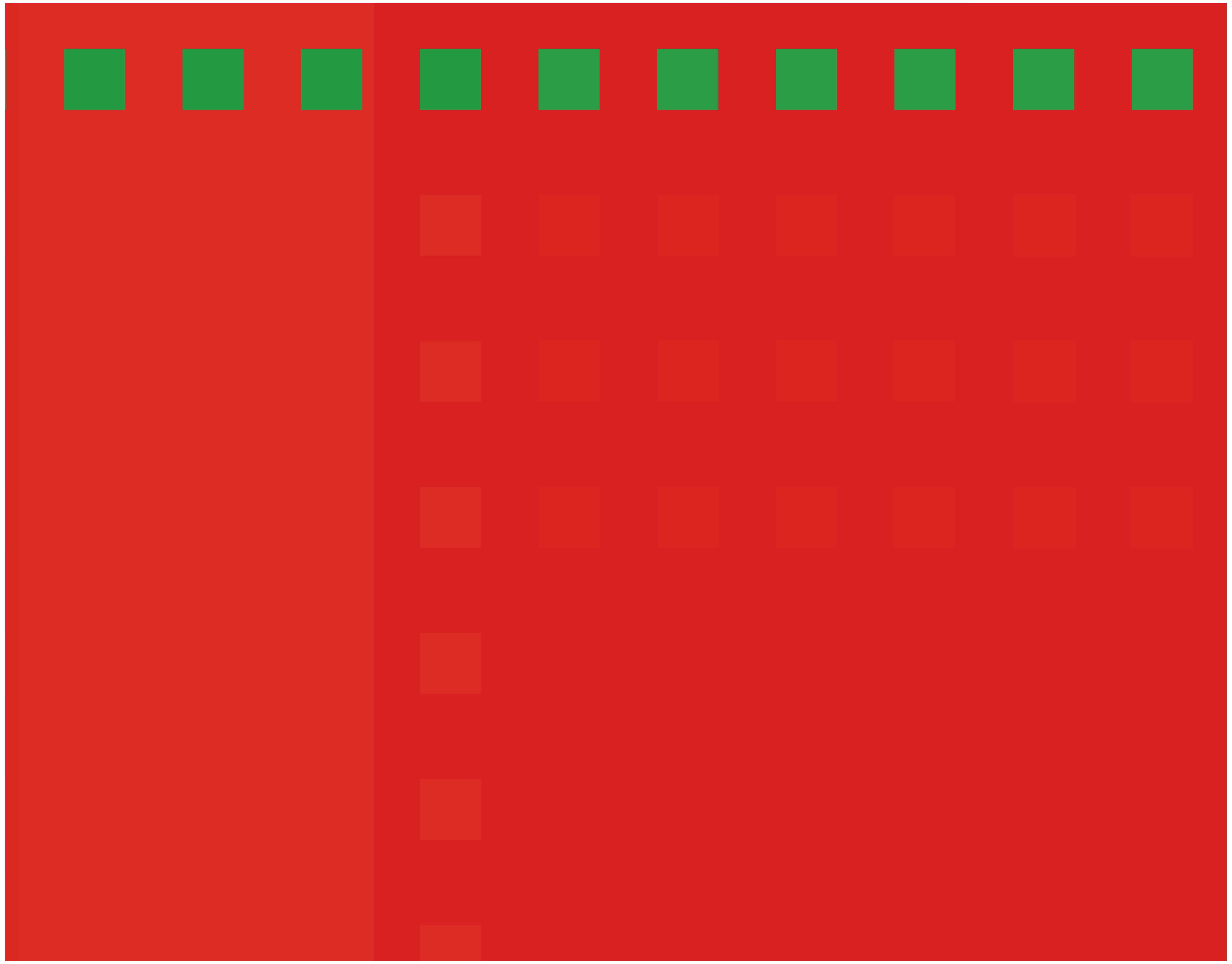
“a correlation between the proximity in space of geometric entities and the proximity of their representations in [the file]”

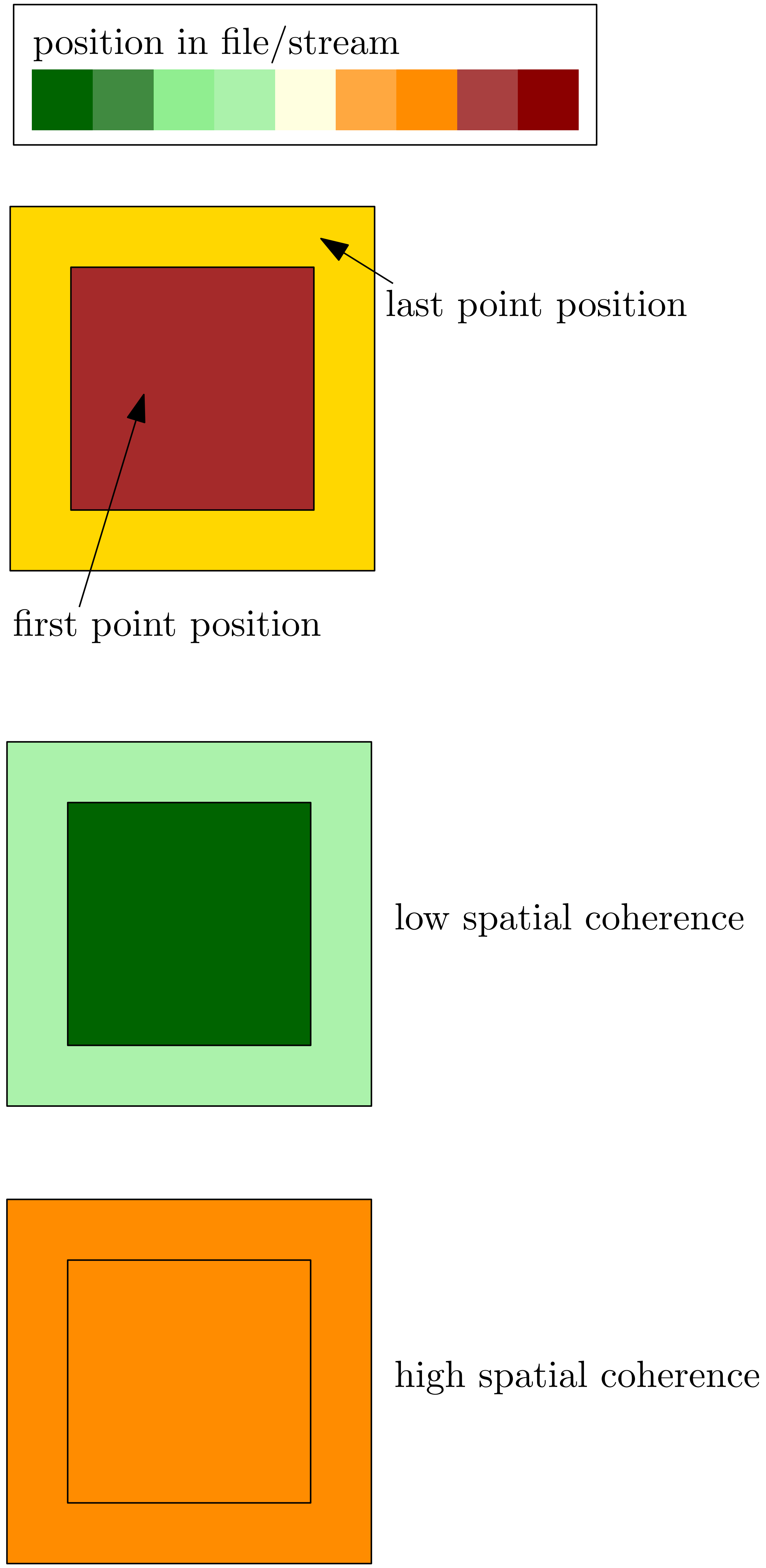
Spatial coherence



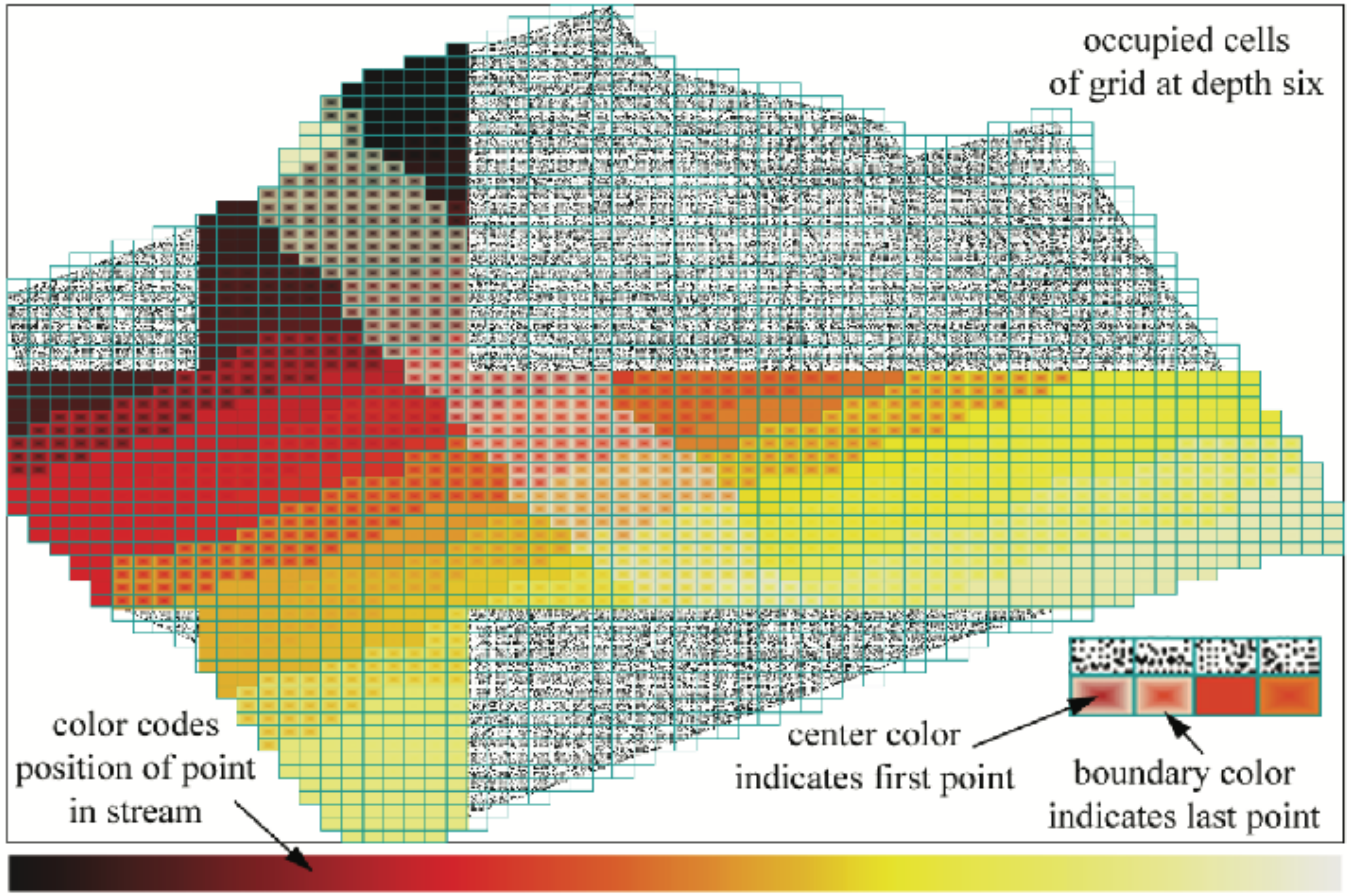


Spatial coherence

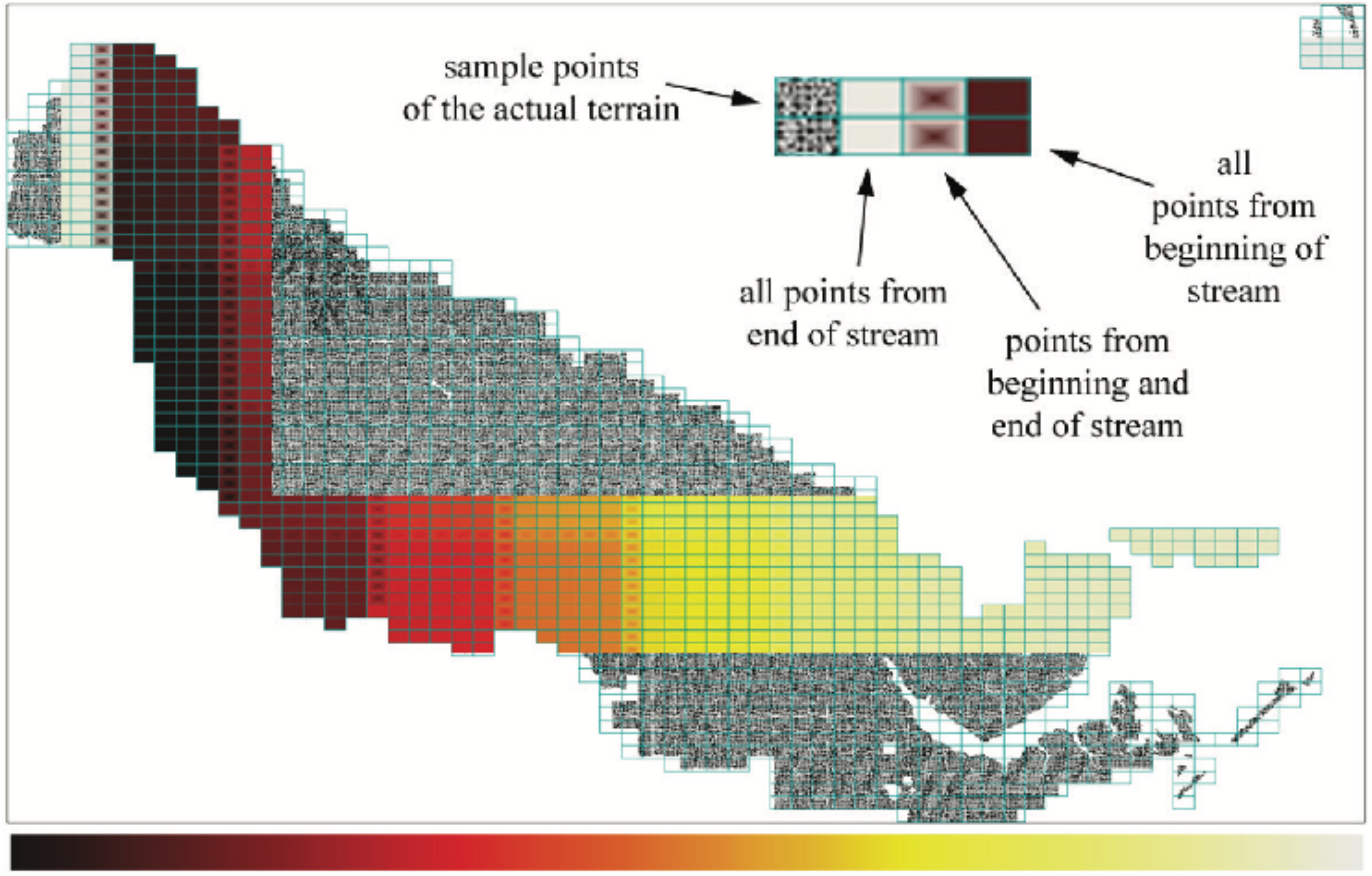




Spatial coherence



(a) 6 million point Baisman Run dataset in Broadmoor, Maryland



(b) 500 million point Neuse river basin dataset in North Carolina

Figure 2.4: Spatial coherence as inherent property of real-world datasets. Adapted from Isenburg et al. [2006b, p. 3].

```
(geo1015) ~/projects/sst git:(develop) (5m 1.94s)
./target/release/sstfin -vv /Users/hugo/data/ahn3/c_37en1_big_delft.laz 20 | ./target/release/sstdt2
 -vv > /dev/null
 2023-01-10T16:24:16Z WARN  sstdt2::triangulator  walk.3
 2023-01-10T16:24:16Z WARN  sstdt2::triangulator  walk.4
 2023-01-10T16:24:16Z WARN  sstdt2::triangulator  walk.3
 2023-01-10T16:24:16Z INFO  sstdt2::triangulator  Cell 43--74 finalised (3775 vertices)
 2023-01-10T16:24:16Z WARN  sstdt2::triangulator  walk.3
 2023-01-10T16:24:16Z WARN  sstdt2::triangulator  walk.4
 2023-01-10T16:24:16Z INFO  sstdt2::triangulator  Cell 43--75 finalised (4637 vertices)
 2023-01-10T16:24:16Z INFO  sstdt2::triangulator  Cell qtc[1, 2, 0, 3, 0, 3] finalised
 2023-01-10T16:24:16Z INFO  sstdt2::triangulator  Cell qtc[1, 2, 0, 3, 0] finalised
 2023-01-10T16:24:16Z WARN  sstdt2::triangulator  walk.3
 2023-01-10T16:24:16Z WARN  sstdt2::triangulator  walk.4
 2023-01-10T16:24:16Z INFO  sstdt2::triangulator  Cell 43--76 finalised (5503 vertices)
 2023-01-10T16:24:16Z WARN  sstdt2::triangulator  walk.3
 2023-01-10T16:24:16Z INFO  sstdt2::triangulator  Cell 43--77 finalised (10477 vertices)
 2023-01-10T16:24:16Z INFO  sstdt2::triangulator  Cell qtc[1, 2, 0, 3, 1, 2] finalised
 2023-01-10T16:24:16Z WARN  sstdt2::triangulator  walk.3
 2023-01-10T16:24:16Z WARN  sstdt2::triangulator  walk.3
 2023-01-10T16:24:16Z WARN  sstdt2::triangulator  walk.3
 2023-01-10T16:24:16Z WARN  sstdt2::triangulator  walk.3
 2023-01-10T16:24:16Z INFO  sstfin  Third pass 🏁
 2023-01-10T16:24:16Z INFO  sstfin  ✅
 2023-01-10T16:24:16Z INFO  sstdt2::triangulator  Cell 43--78 finalised (7309 vertices)
 2023-01-10T16:24:16Z WARN  sstdt2::triangulator  walk.3
 2023-01-10T16:24:16Z WARN  sstdt2::triangulator  walk.4
 2023-01-10T16:24:16Z INFO  sstdt2::triangulator  Cell 43--79 finalised (1294 vertices)
 2023-01-10T16:24:16Z INFO  sstdt2::triangulator  Cell qtc[1, 2, 0, 3, 1, 3] finalised
 2023-01-10T16:24:16Z INFO  sstdt2::triangulator  Cell qtc[1, 2, 0, 3, 1] finalised
 2023-01-10T16:24:16Z INFO  sstdt2::triangulator  Cell qtc[1, 2, 0, 3] finalised
 2023-01-10T16:24:16Z INFO  sstdt2::triangulator  Cell qtc[1, 2, 0] finalised
 2023-01-10T16:24:16Z INFO  sstdt2::triangulator  Cell qtc[1, 2] finalised
 2023-01-10T16:24:16Z INFO  sstdt2::triangulator  Cell qtc[1] finalised
 2023-01-10T16:24:16Z INFO  sstdt2::triangulator  Cell qtc[] finalised
 2023-01-10T16:24:17Z INFO  sstdt2::triangulator  Finalise the quadtree root cell
 2023-01-10T16:24:17Z INFO  sstdt2  dt.number_of_vertices() = 0
 2023-01-10T16:24:17Z INFO  sstdt2  max # points in DT during process: 242733
 2023-01-10T16:24:17Z INFO  sstdt2  max # triangles in DT during process: 300567
 2023-01-10T16:24:17Z INFO  sstdt2  ✅
```

Maarten de Jong MSc Geomatics thesis on the topic

Some questions

- How to create a 50cm grid with TIN linear interpolation for 1B points?
- If we want to use Laplace interpolation, is it possible?