Streaming Construction of Delaunay Triangulations

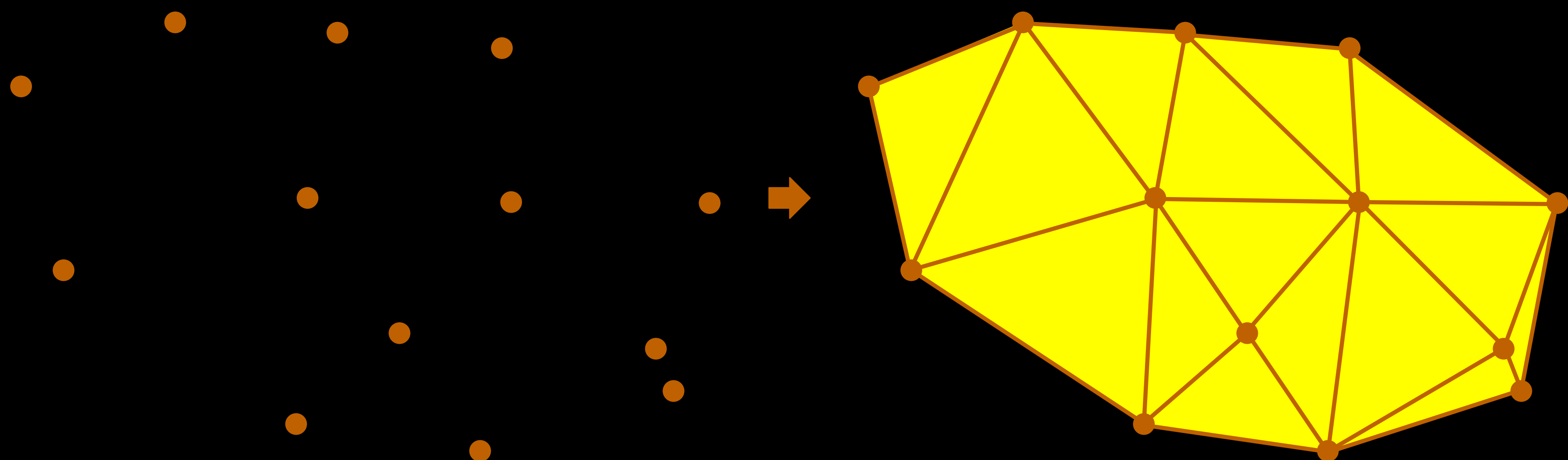
Jonathan Shewchuk
University of California at Berkeley
Joint work with Martin Isenburg, Yuanxin Liu, & Jack Snoeyink

Streaming Construction of Delaunay Triangulations

Or, ‘‘How we compute supercomputer–sized Delaunay triangulations on an ordinary laptop computer.’’

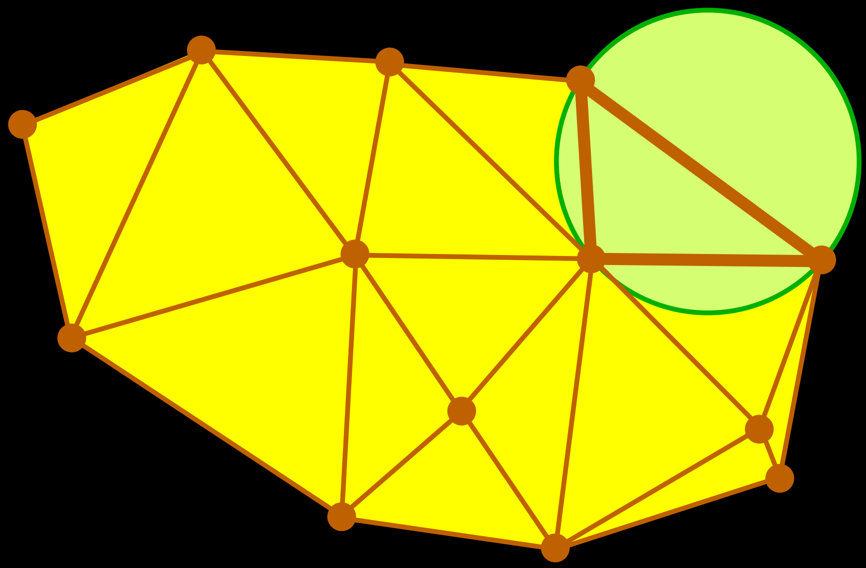
The Delaunay Triangulation

Every point set has a Delaunay triangulation. Think of it as a function that takes a set of points and outputs a triangulation.



The Delaunay Triangulation

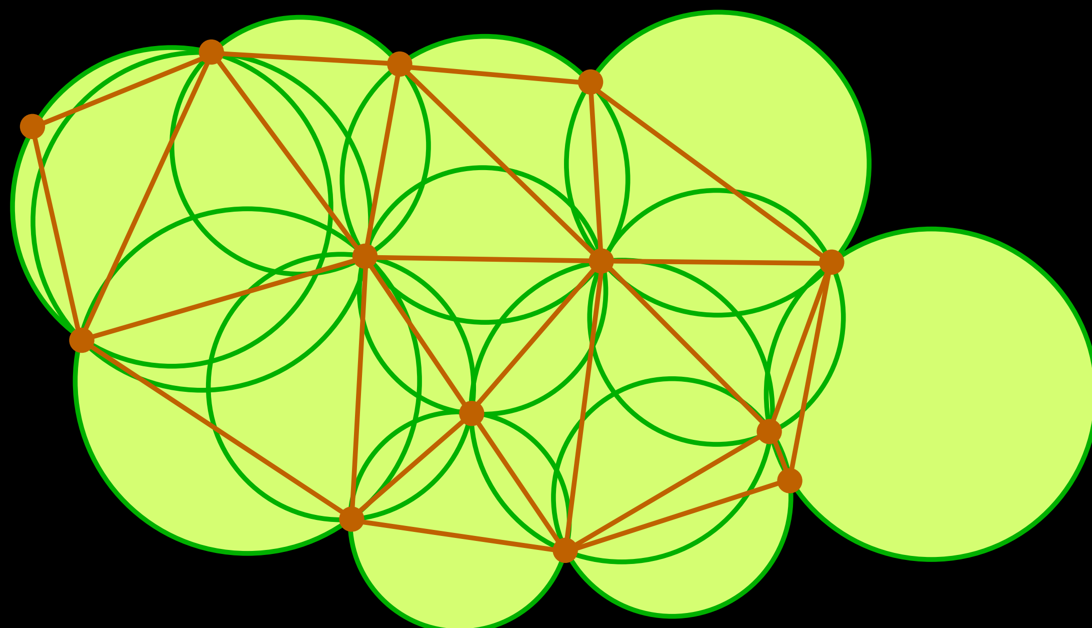
...is a triangulation whose triangles are all *Delaunay.*



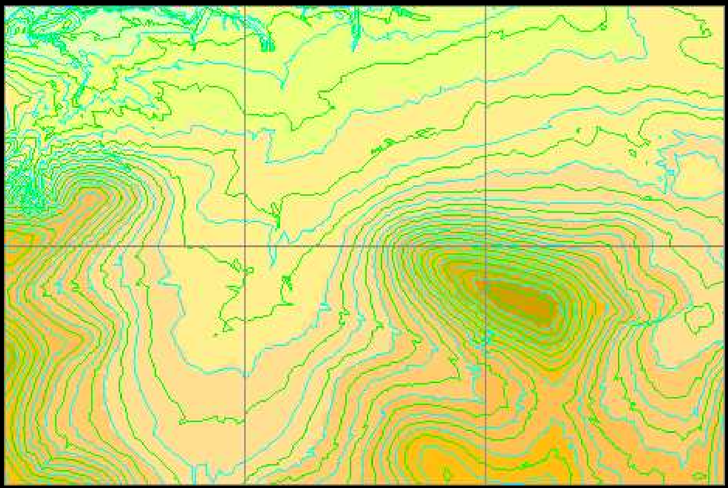
An triangle is Delaunay if it has an empty circumscribing circle – one that encloses no vertex.

The Delaunay Triangulation

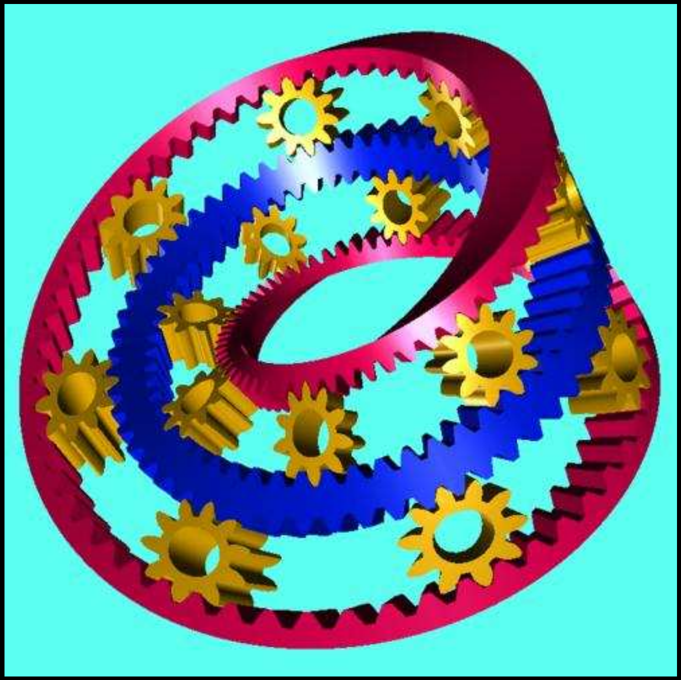
The circumcircle of every Delaunay triangle is empty.



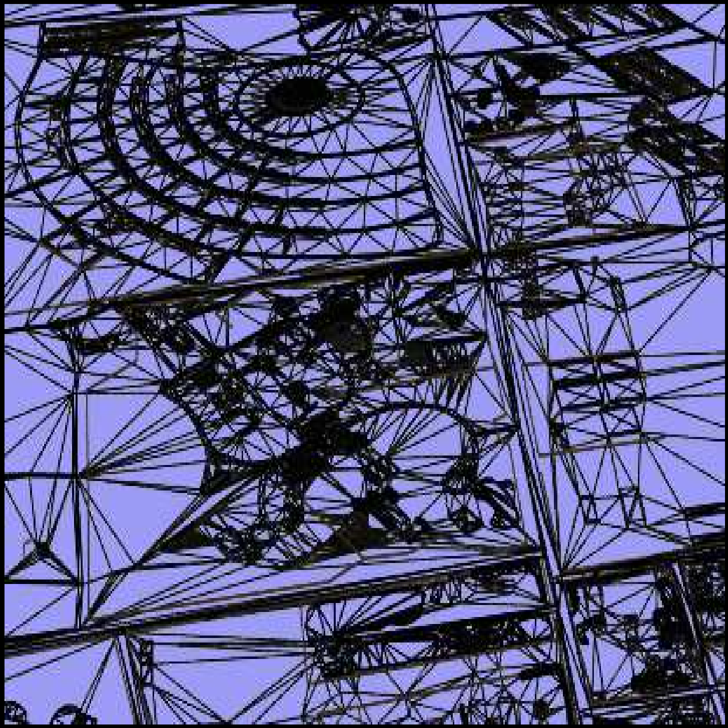
Applications of Triangulations



Contouring

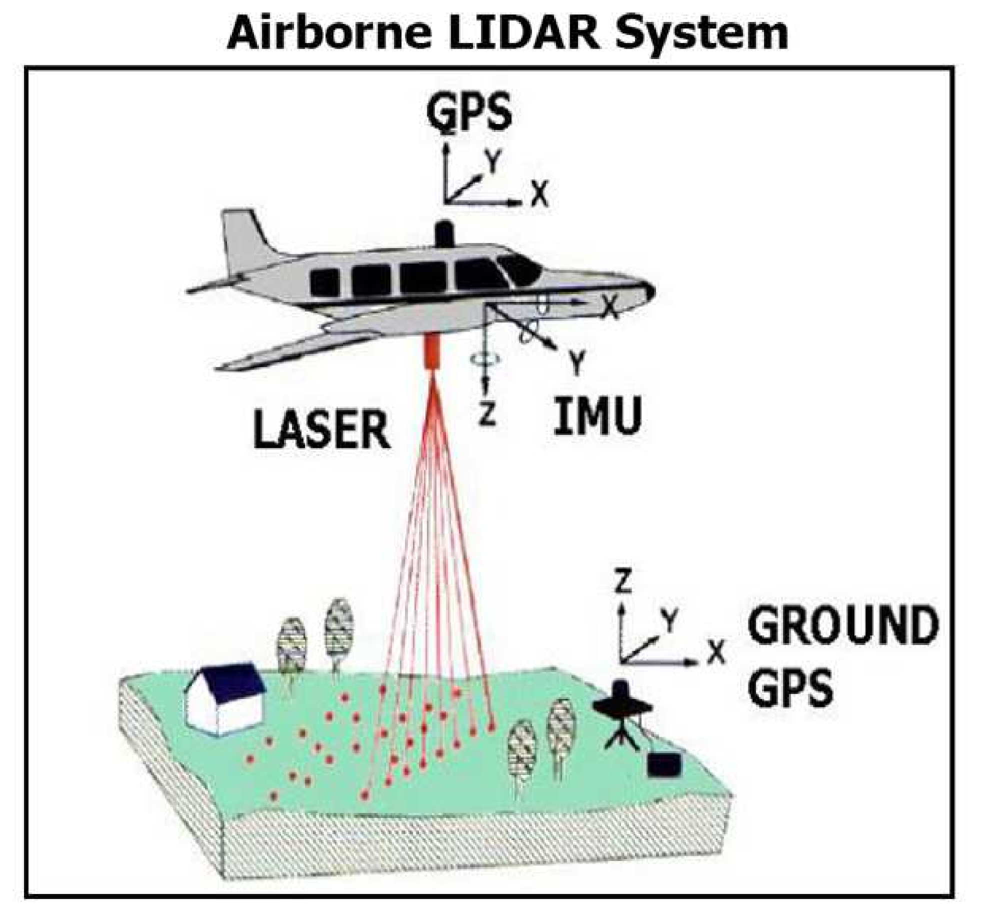


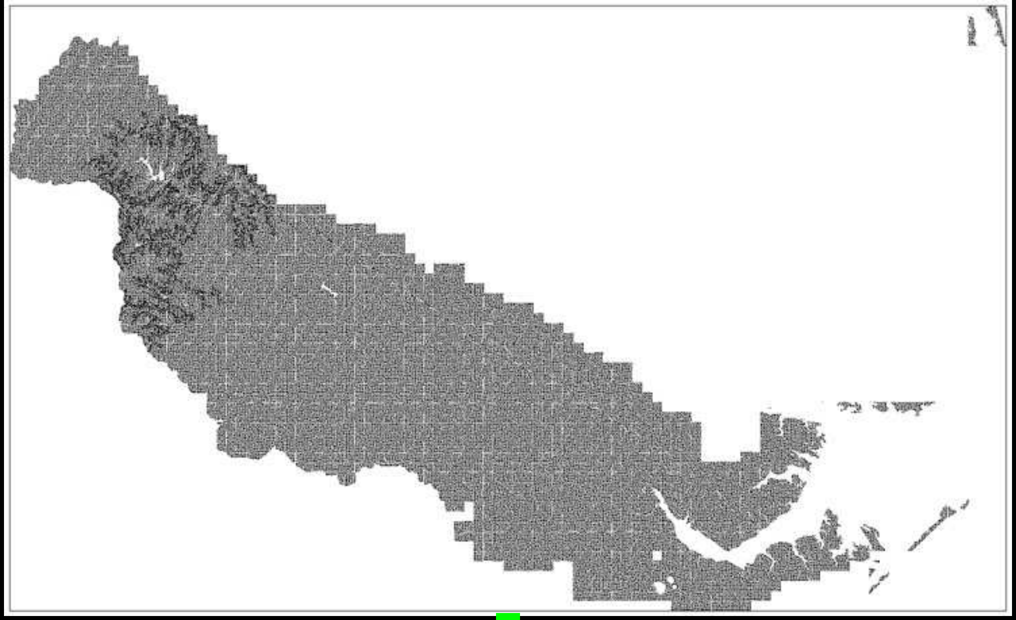
Rendering

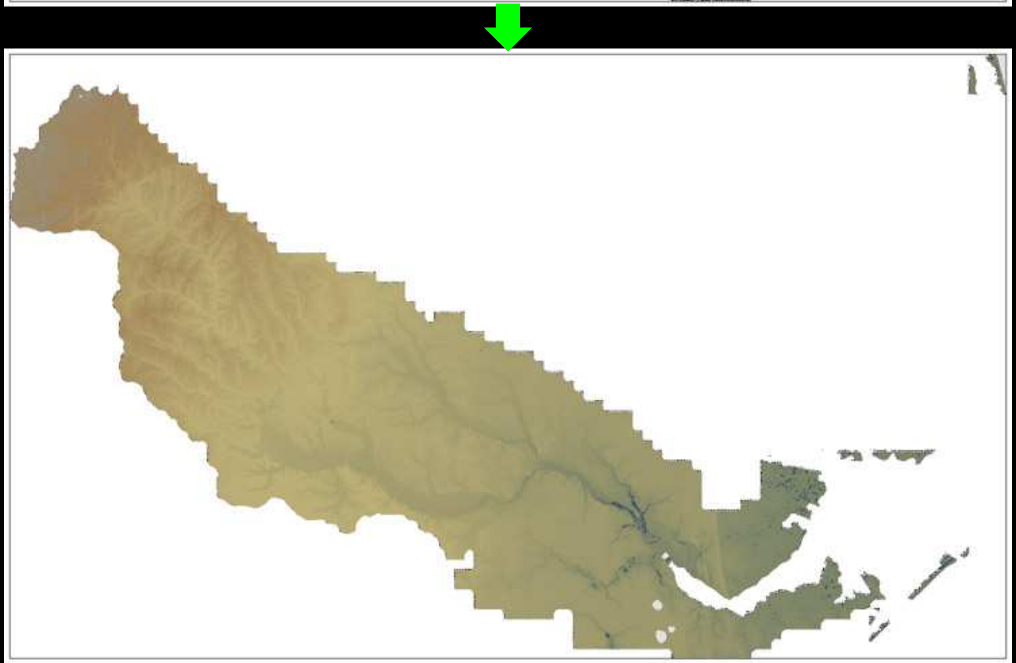


Terrain Databases and Geographical Information Systems

Elevation Collection by LIDAR







Related Work

- Amenta, Choi, & Rote [2003] use *biased randomized insertion orders* to improve virtual memory performance of 3D Delaunay triangulation construction.

Related Work

- Amenta, Choi, & Rote [2003] use *biased randomized insertion orders* to improve virtual memory performance of 3D Delaunay triangulation construction.
- Agarwal, Arge, and Yi [2005] implemented an external memory 2D triangulator based on randomized divide−and−conquer.
  (1 billion triangles on desktop machine.)

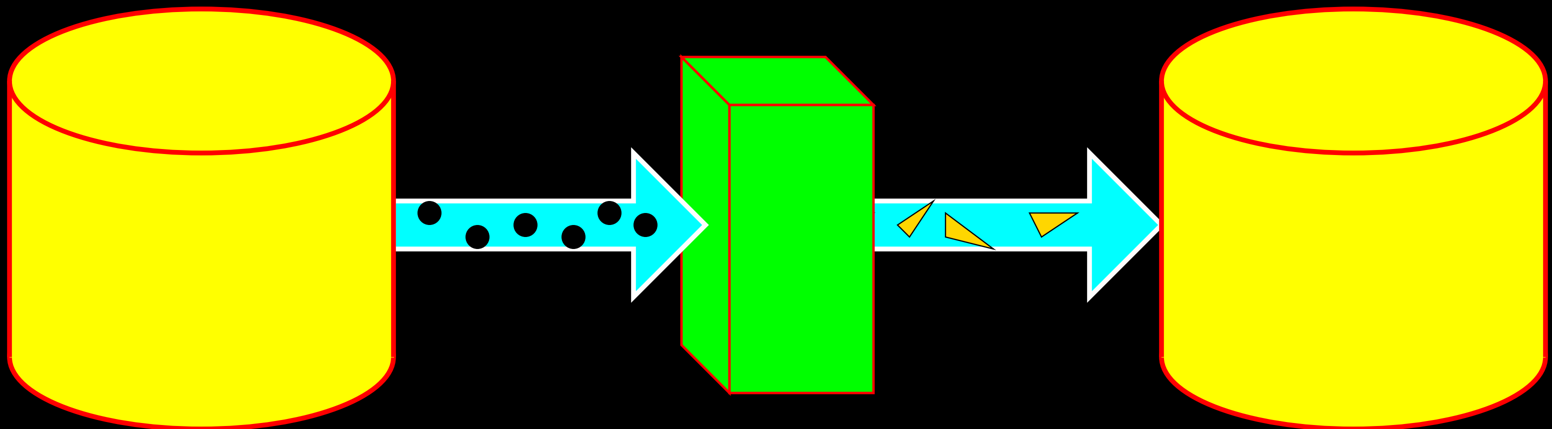
Related Work

- Amenta, Choi, & Rote [2003] use *biased randomized insertion orders* to improve virtual memory performance of 3D Delaunay triangulation construction.
- Agarwal, Arge, and Yi [2005] implemented an external memory 2D triangulator based on randomized divide−and−conquer. (1 billion triangles on desktop machine.)
- Blandford, Blelloch, and Kadow [2006] implemented a 3D parallel triangulator. (10 billion tetrahedra on 64 processors.)

Streaming Delaunay

Streaming Computation

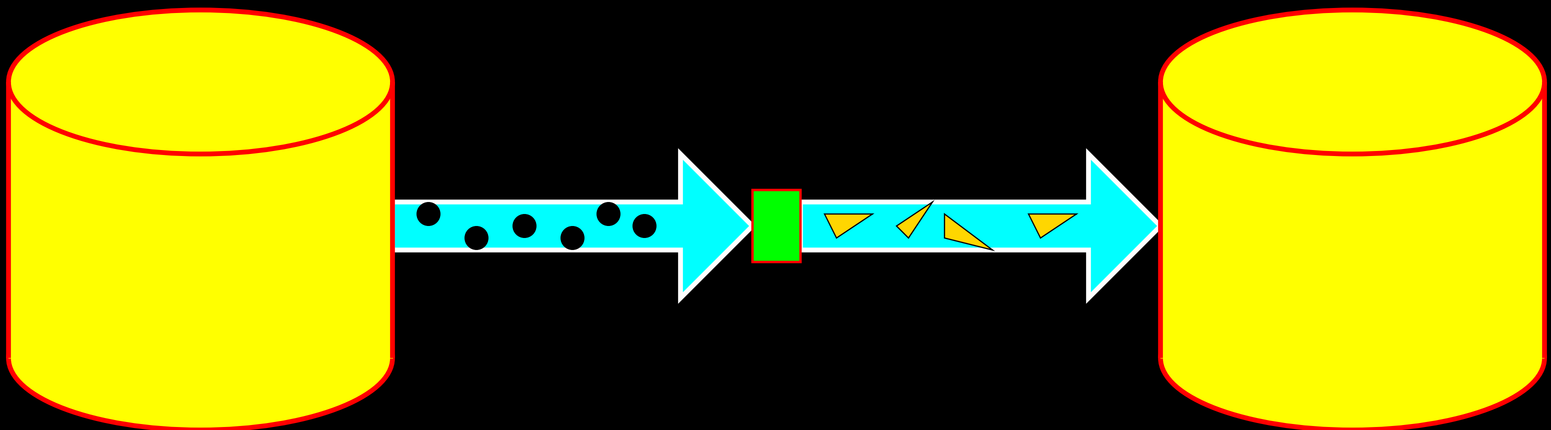
A restricted version of out−of−core computation.



- Algorithm makes one (or a few) pass(es) over input stream, and writes output stream.

Streaming Computation

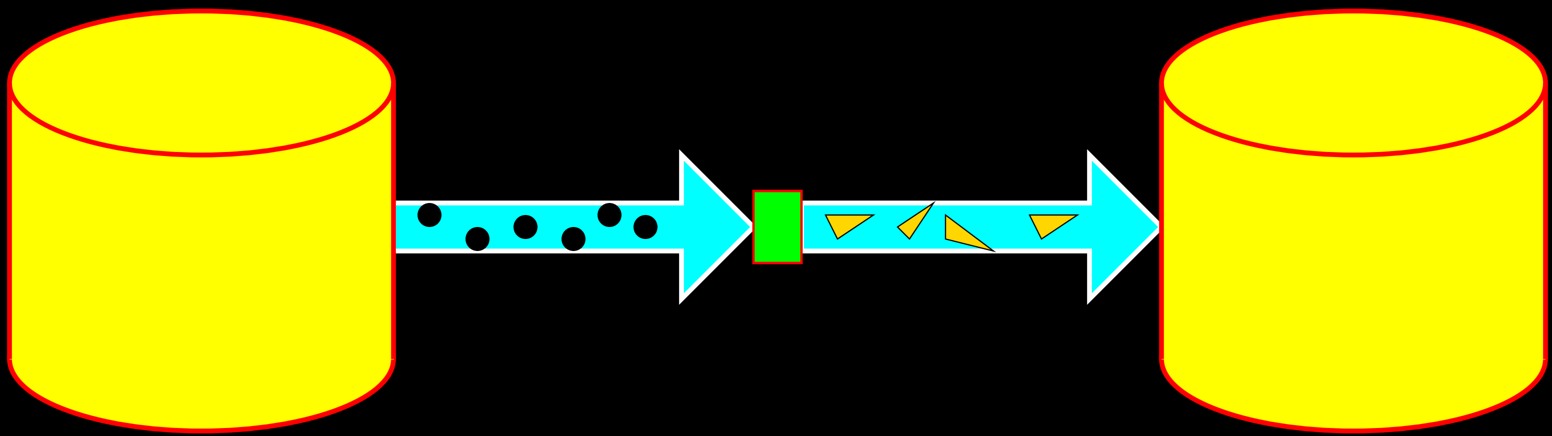
A restricted version of out−of−core computation.



- Algorithm makes one (or a few) pass(es) over input stream, and writes output stream.
- Processes data in memory buffer much smaller than the stream sizes.

Streaming Computation

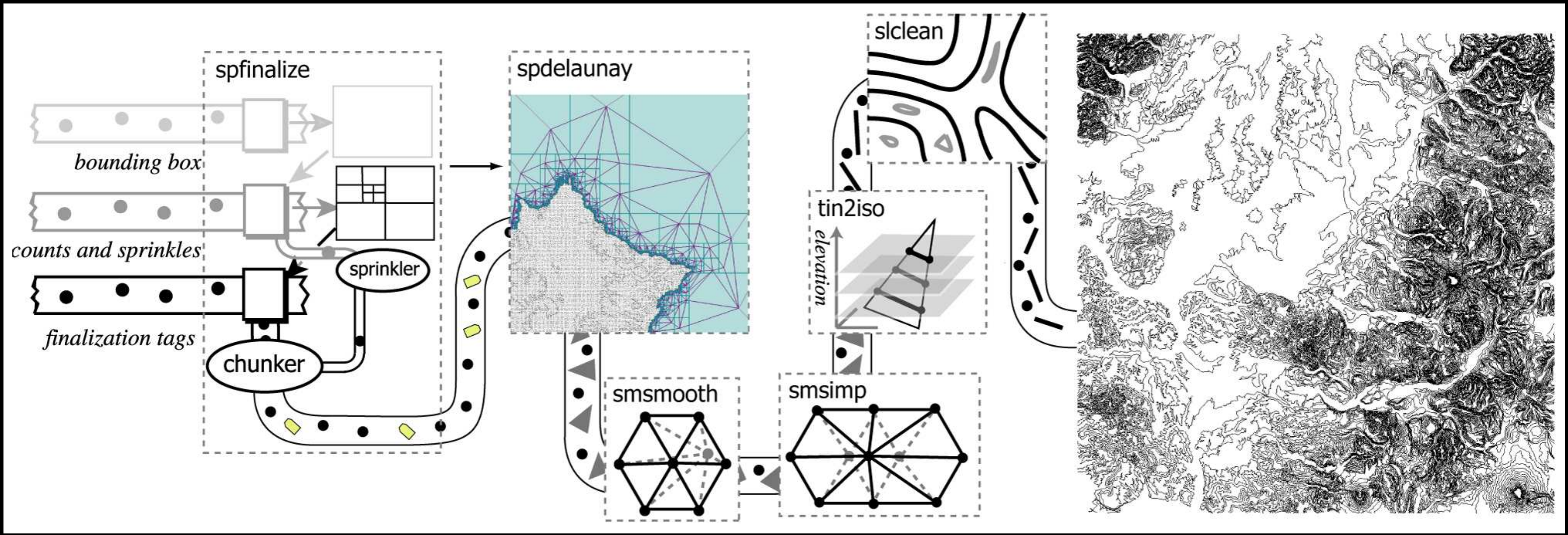
A restricted version of out−of−core computation.



- Algorithm makes one (or a few) pass(es) over input stream, and writes output stream.
- Processes data in memory buffer much smaller than the stream sizes.
- Nothing is stored temporarily to disk.

Advantages of Streaming

- Streaming tools can run concurrently in a pipeline.



- Often much faster than other out−of−core algorithms!

Our Accomplishments

- 9 billion triangles in 6.6 hours on a laptop.

Our Accomplishments

- 9 billion triangles in 6.6 hours on a laptop.
- 12 times faster than Agarwal−Arge−Yi.

Our Accomplishments

- 9 billion triangles in 6.6 hours on a laptop.
- 12 times faster than Agarwal−Arge−Yi.
- 800 million tetrahedra in under 3 hours.

Algorithm Choice

The *incremental insertion* algorithm for constructing Delaunay triangulations.

Algorithm Choice

The *incremental insertion* algorithm for constructing Delaunay triangulations.

Why?

- We have little control over the order of points in the input stream.

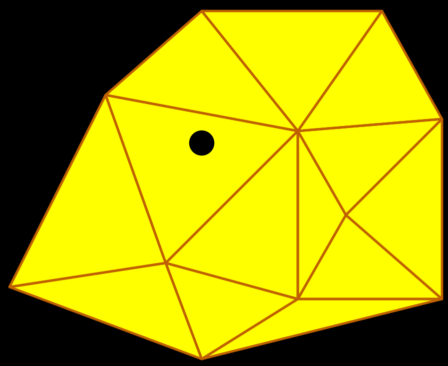
Algorithm Choice

The *incremental insertion* algorithm for constructing Delaunay triangulations.

Why?

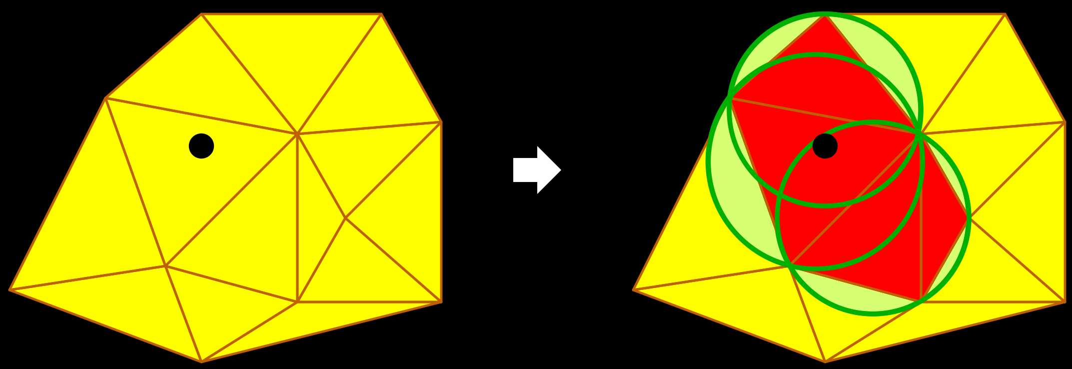
- We have little control over the order of points in the input stream.
- We modified existing, sequential Delaunay triangulation codes (2D & 3D) for streaming.

Bowyer–Watson Algorithm



Insert one vertex at a time.

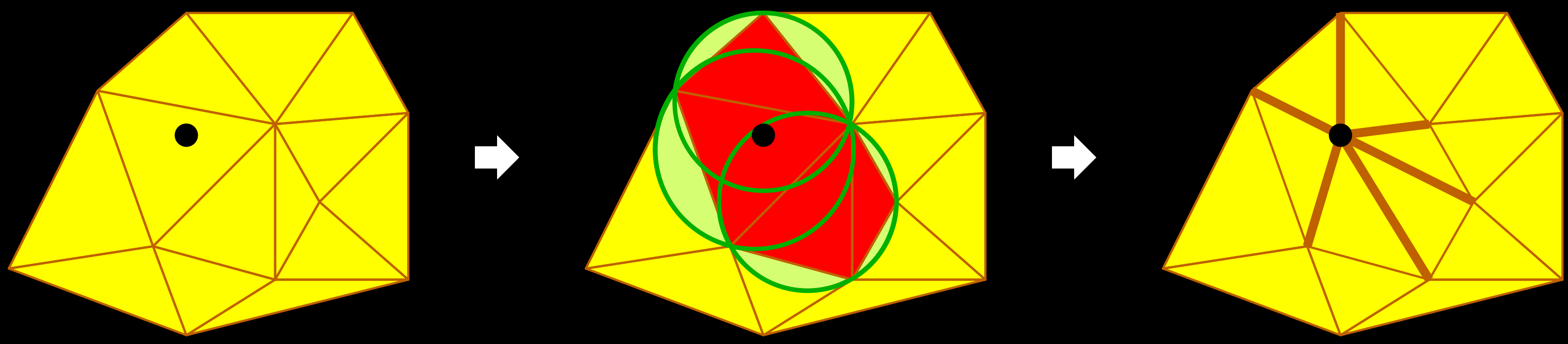
Bowyer–Watson Algorithm



Insert one vertex at a time.

Remove all triangles/tetrahedra that are no longer Delaunay.

Bowyer–Watson Algorithm

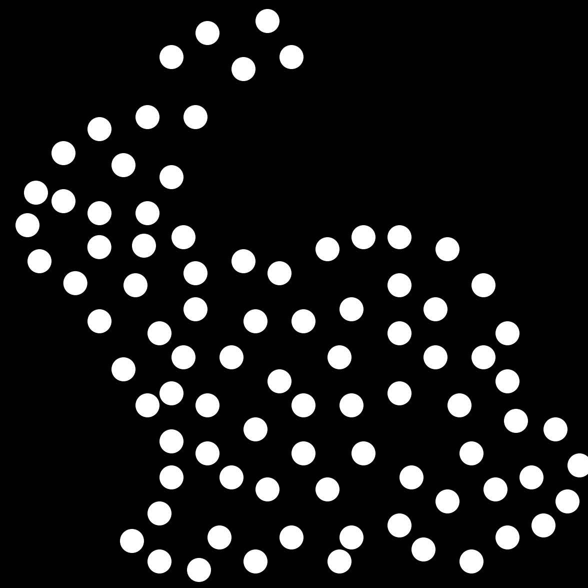


Insert one vertex at a time.

Remove all triangles/tetrahedra that are no longer Delaunay.

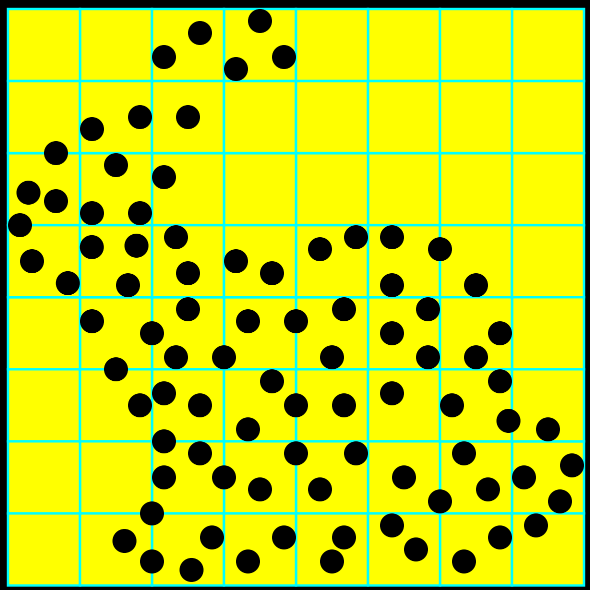
Retriangulate the cavity with a fan around the new vertex.

Spatial Finalization



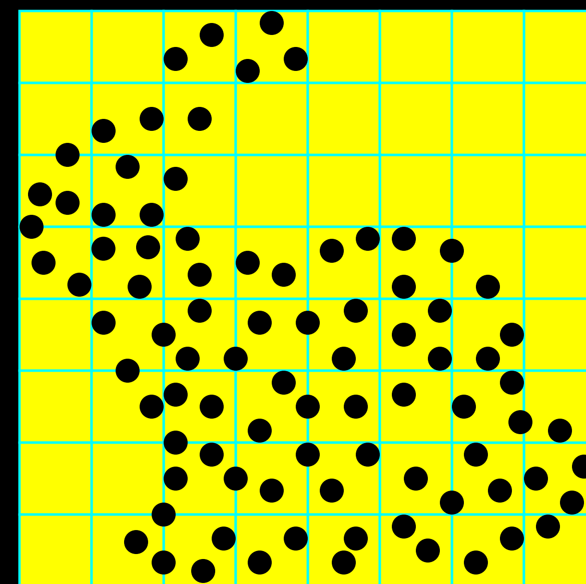
Spatial Finalization

- Subdivide space into *finalization regions.*

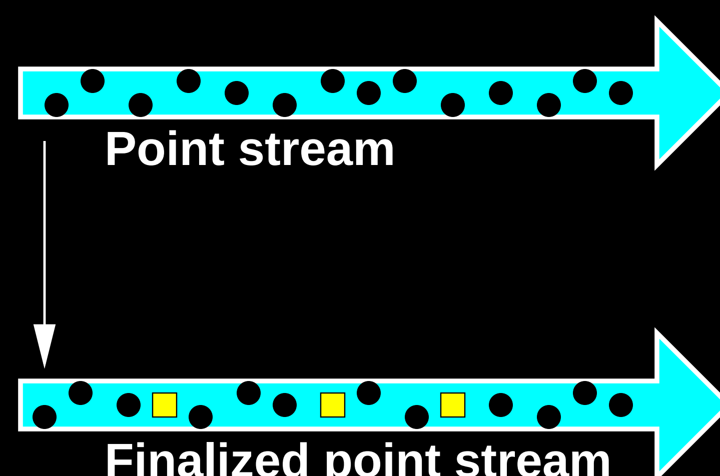


Spatial Finalization

- Subdivide space into *finalization regions.*

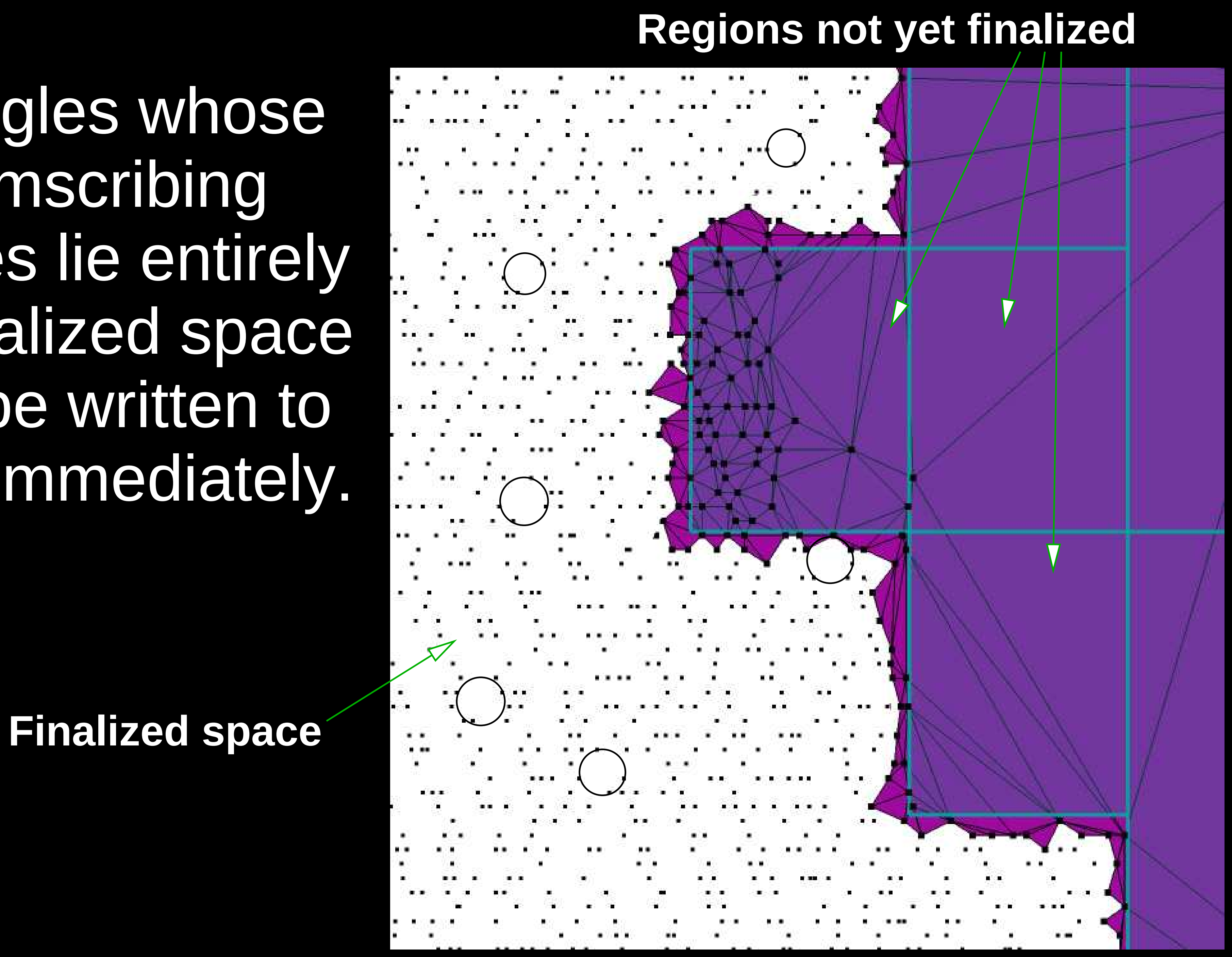


- Inject into the stream *spatial finalization tags* that indicate "there are no more points in this region."

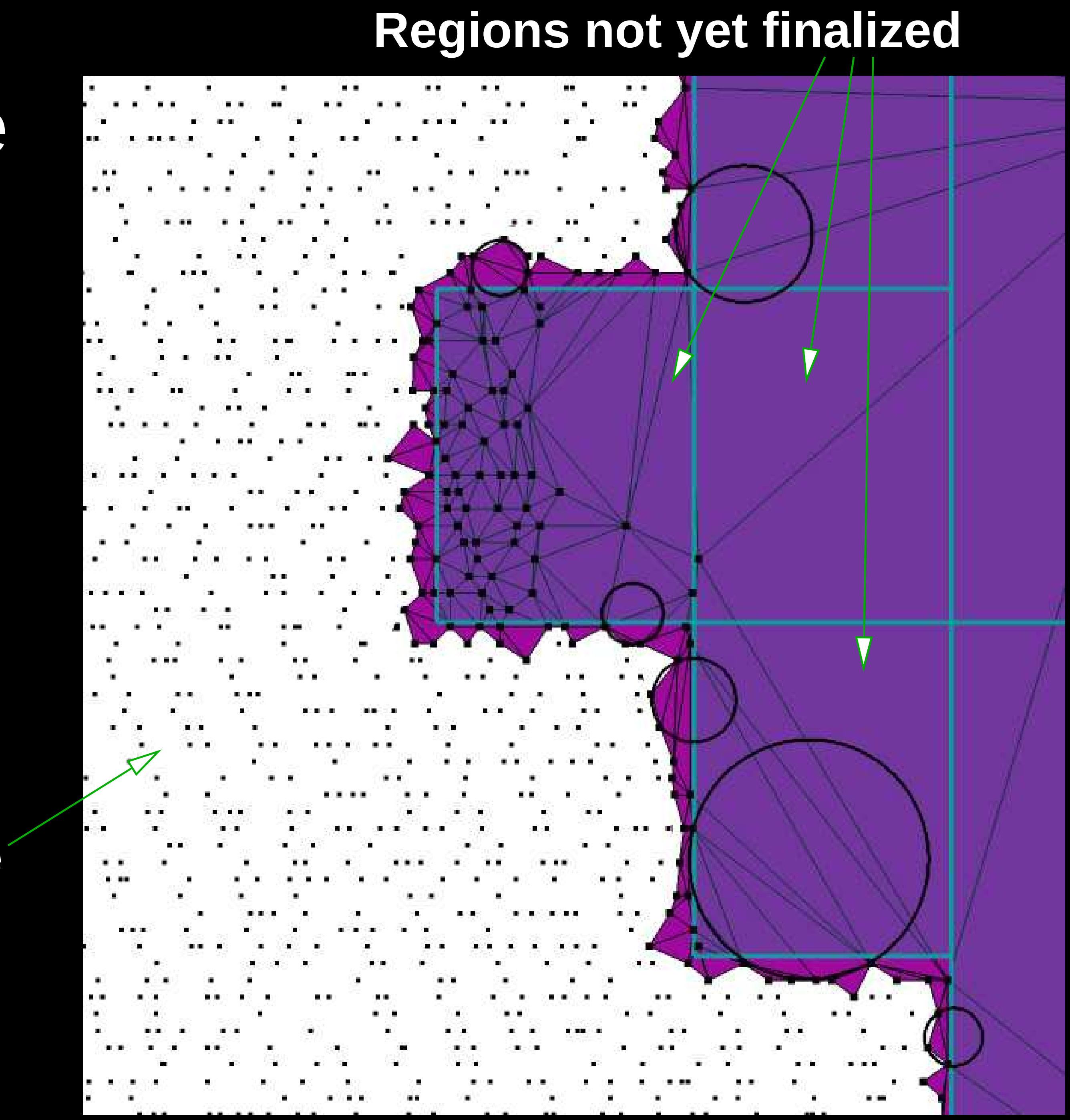


Spatial Finalization: Why?

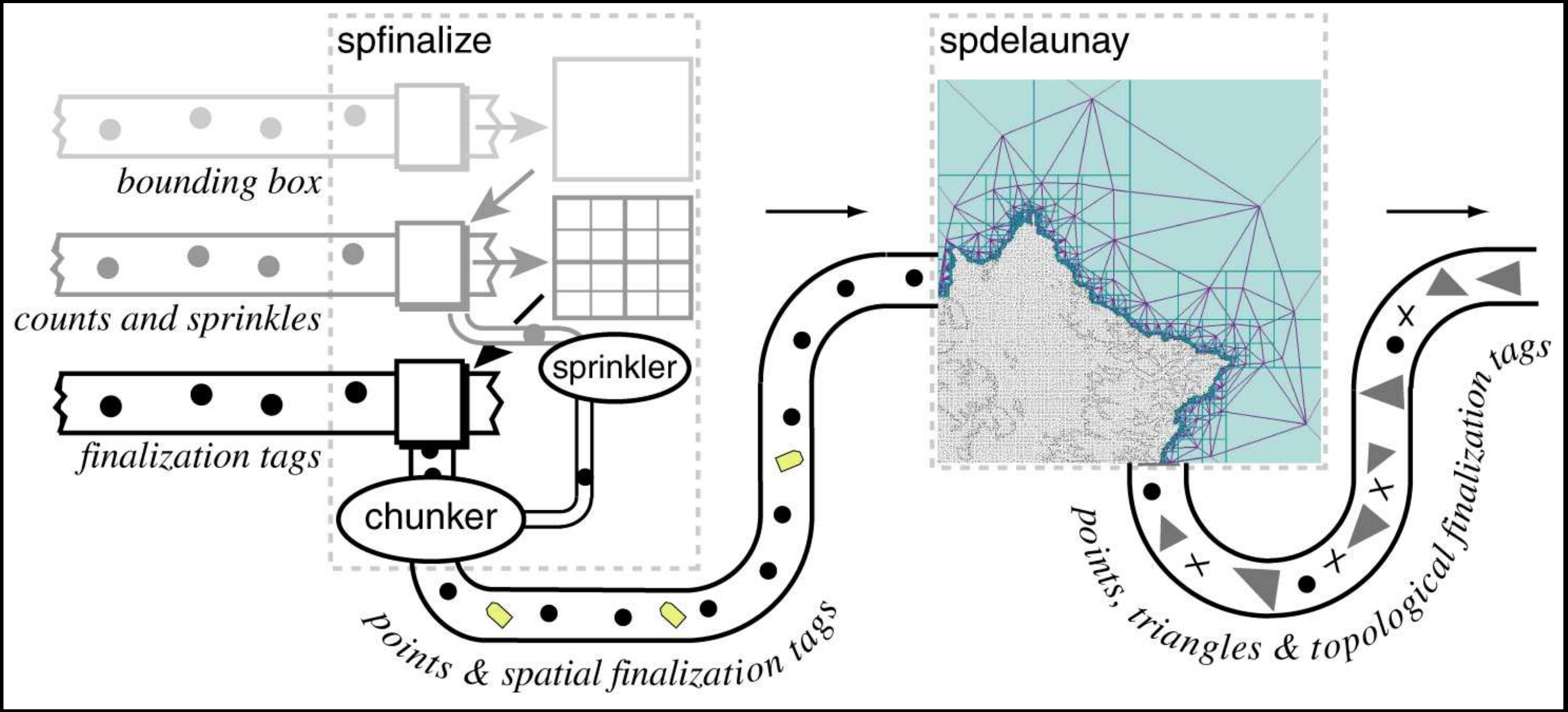
Triangles whose circumscribing circles lie entirely in finalized space can be written to disk immediately.



Triangles whose circumscribing circles intersect an unfinalized region remain in memory.



Streaming Delaunay Pipeline



Finalizer

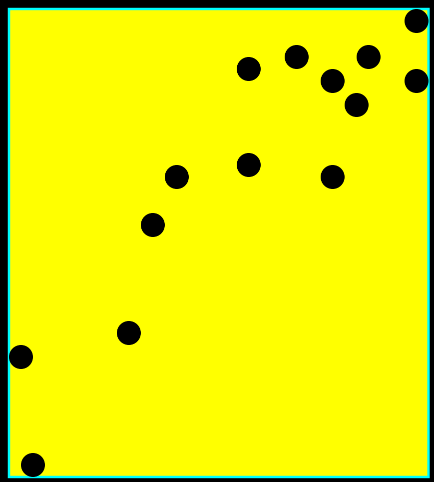
Triangulator

The Finalizer

How to Finalize

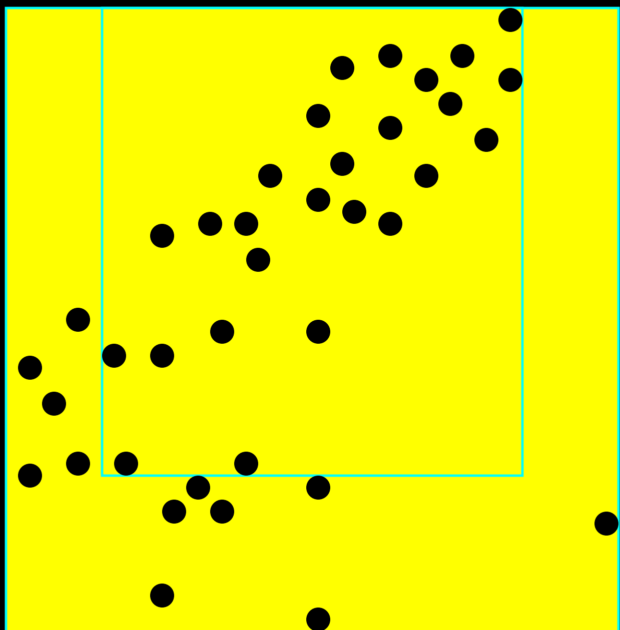
1. Compute bounding box.

How to Finalize



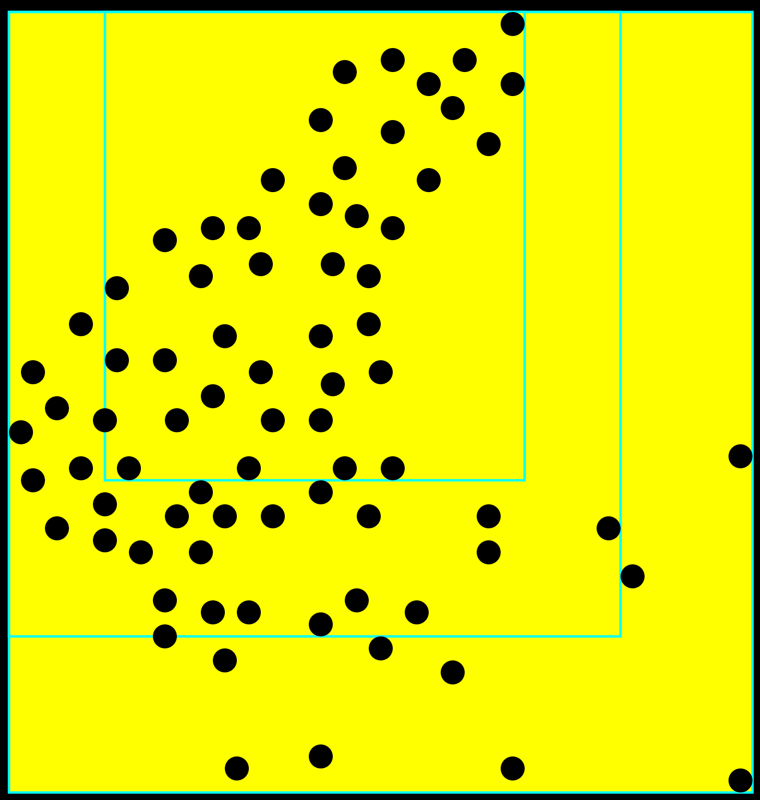
1 Compute bounding box.

How to Finalize



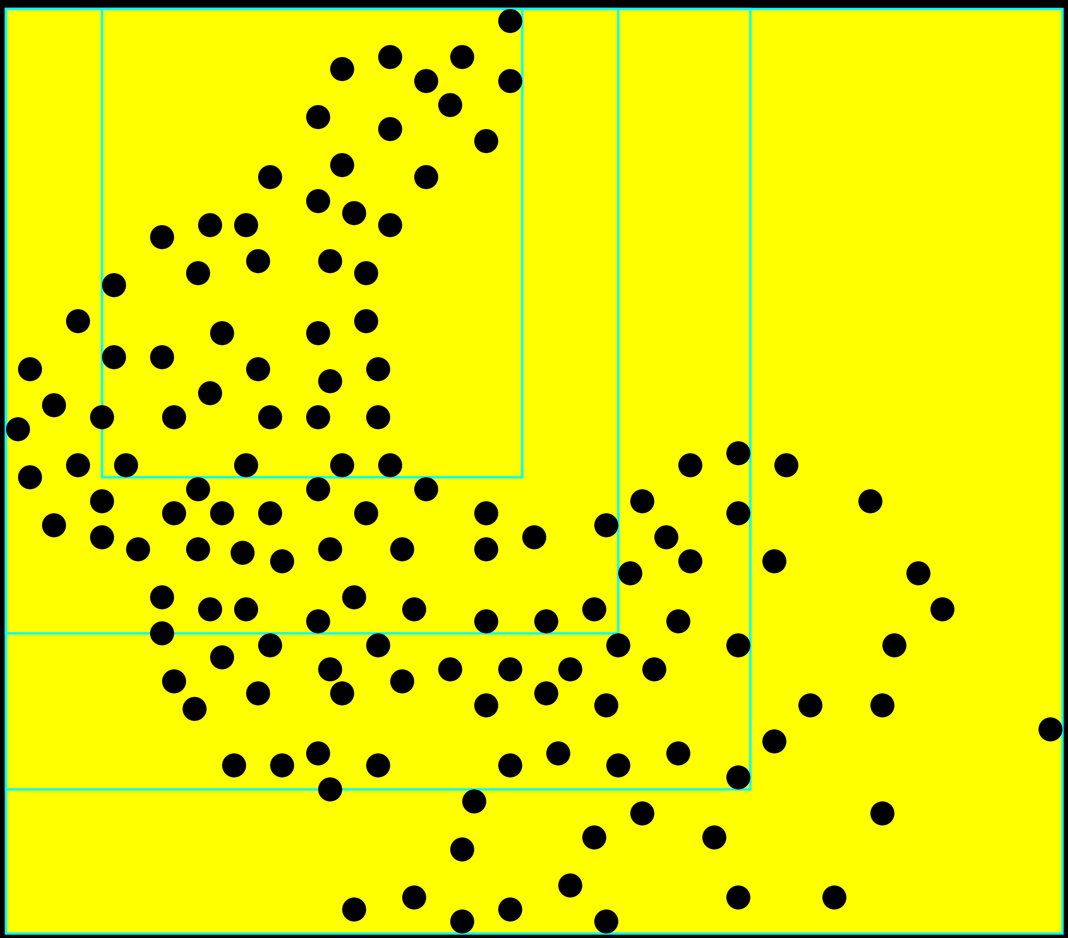
1 Compute bounding box.

How to Finalize



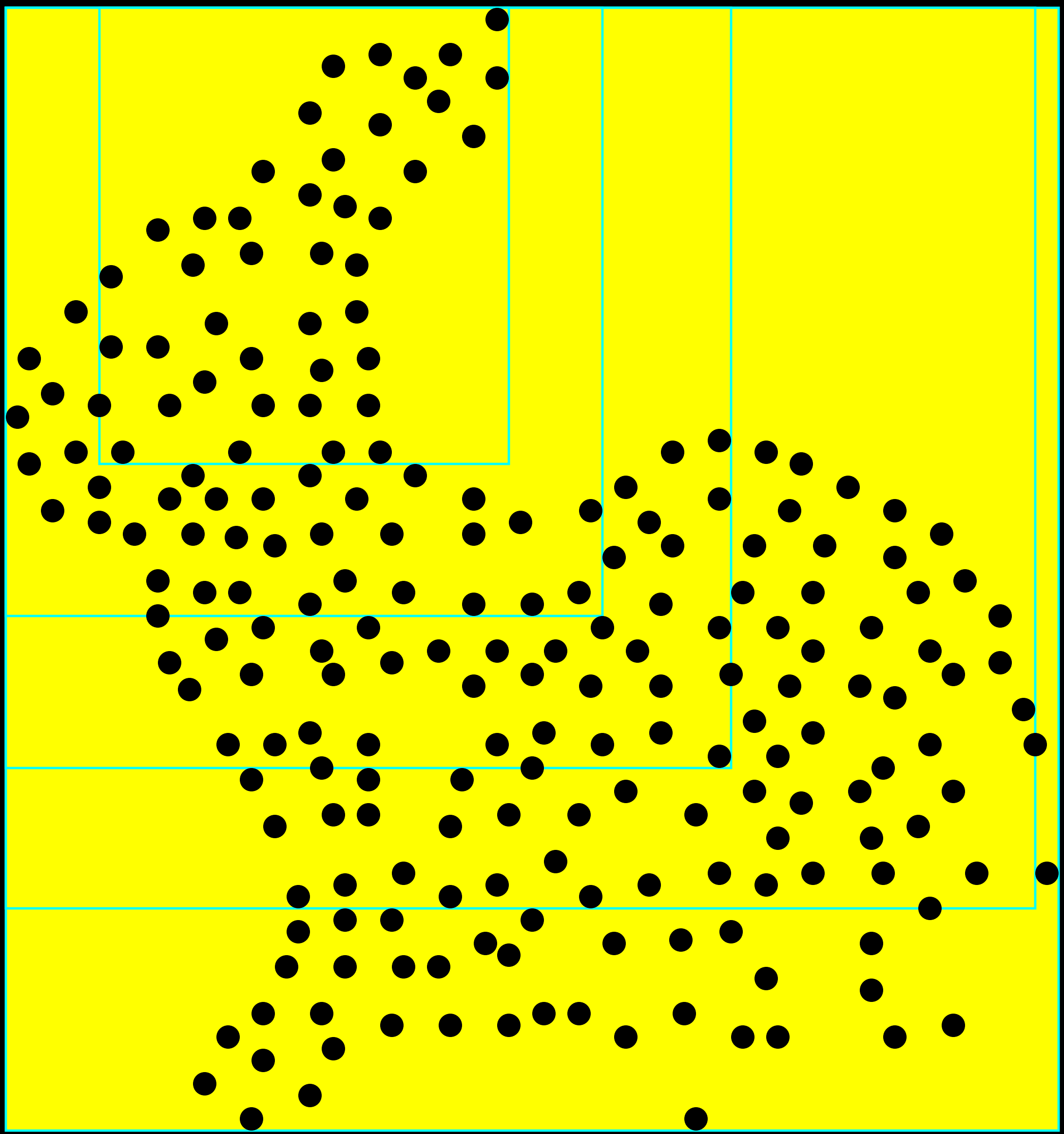
1 Compute bounding box.

How to Finalize



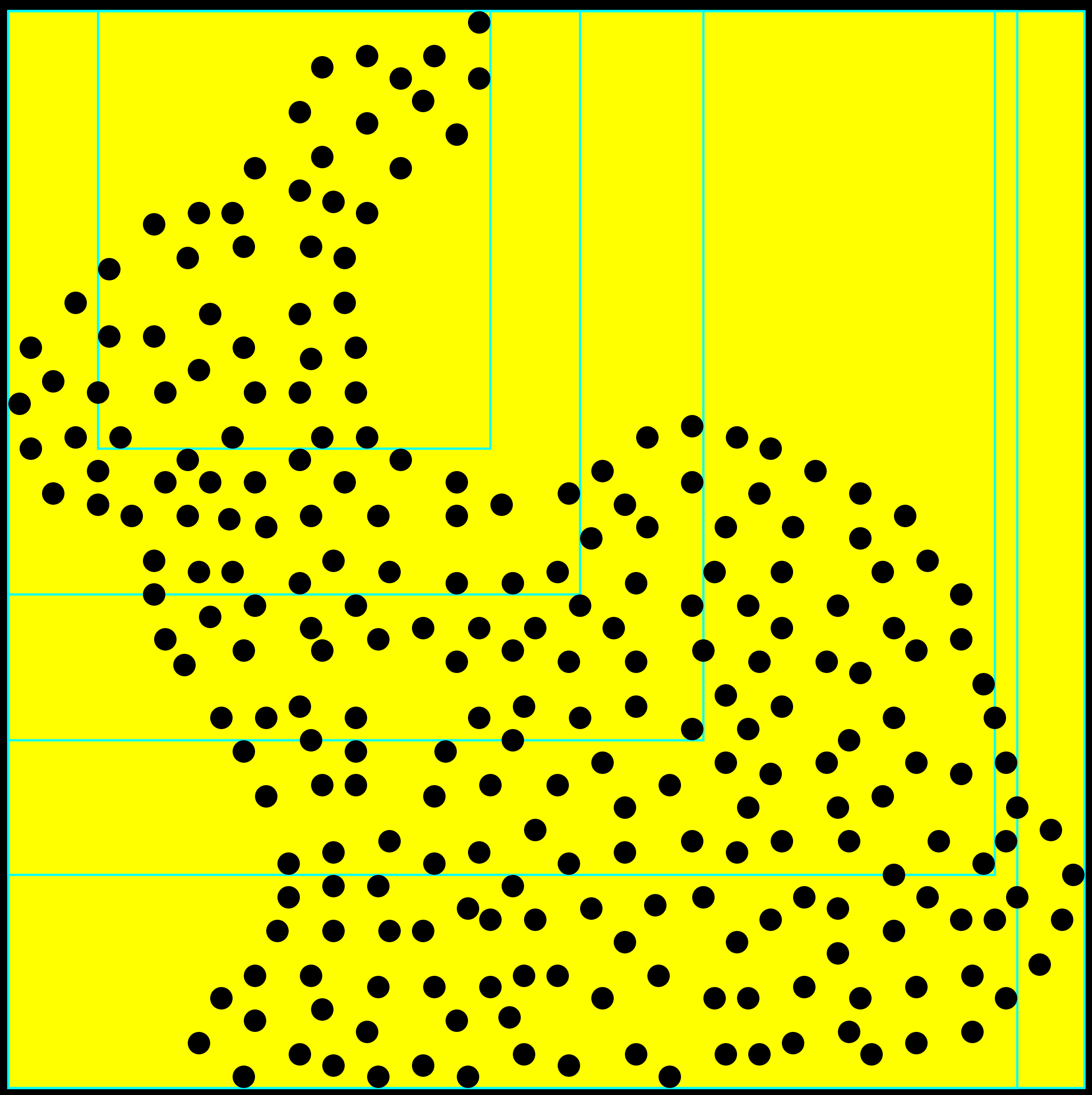
1 Compute bounding box.

How to Finalize



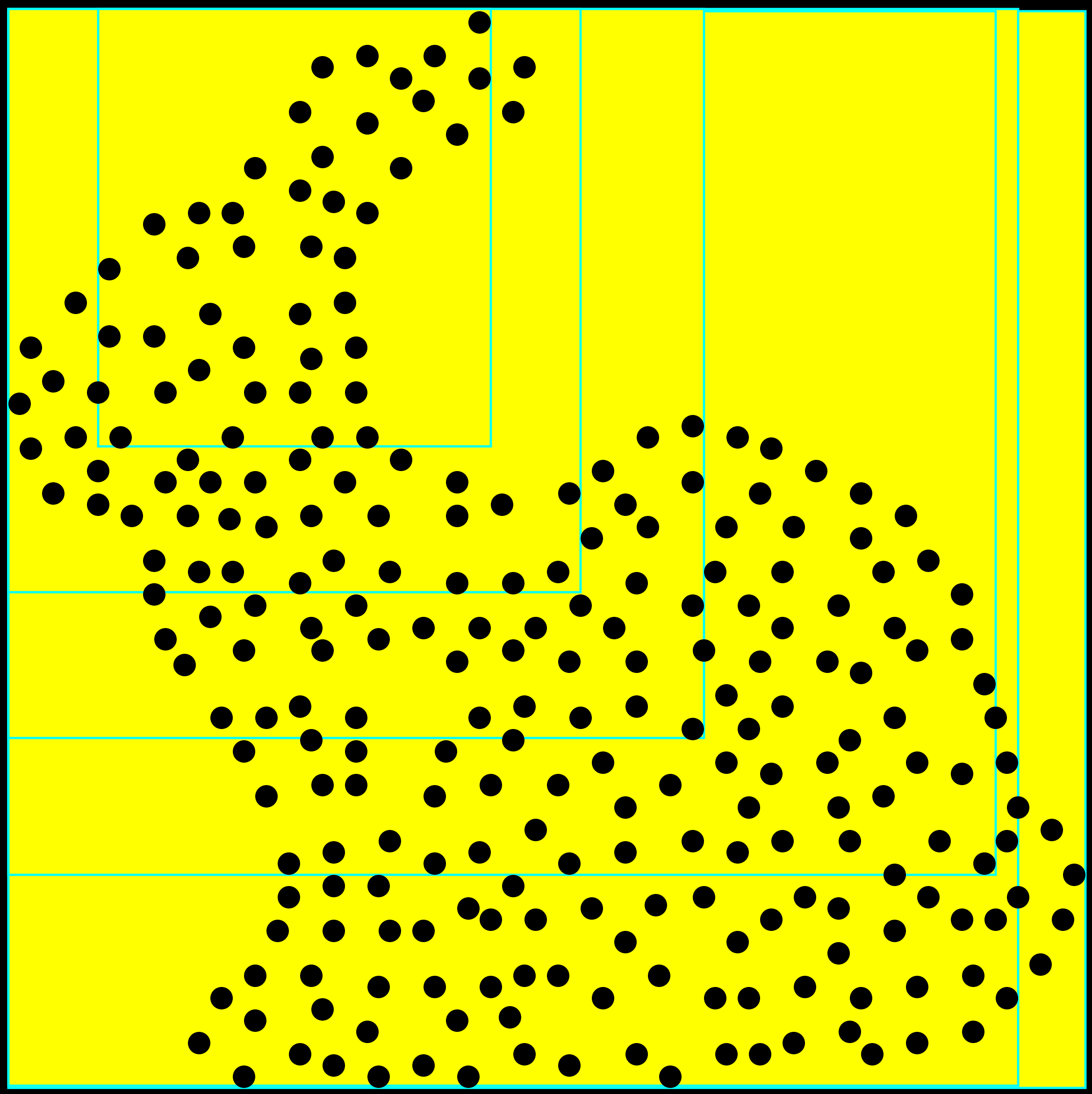
1 Compute bounding box.

How to Finalize



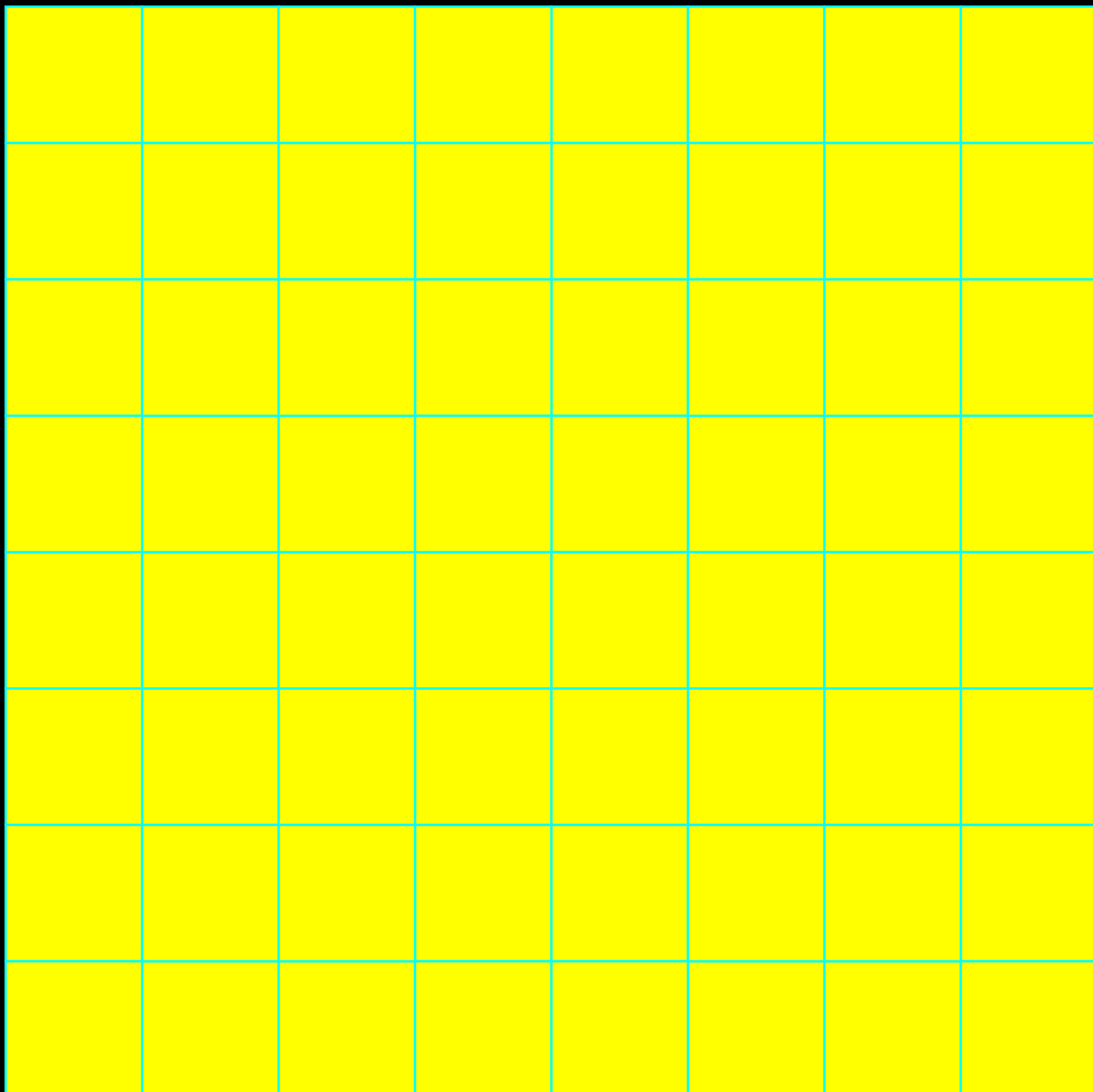
1 Compute bounding box.

How to Finalize



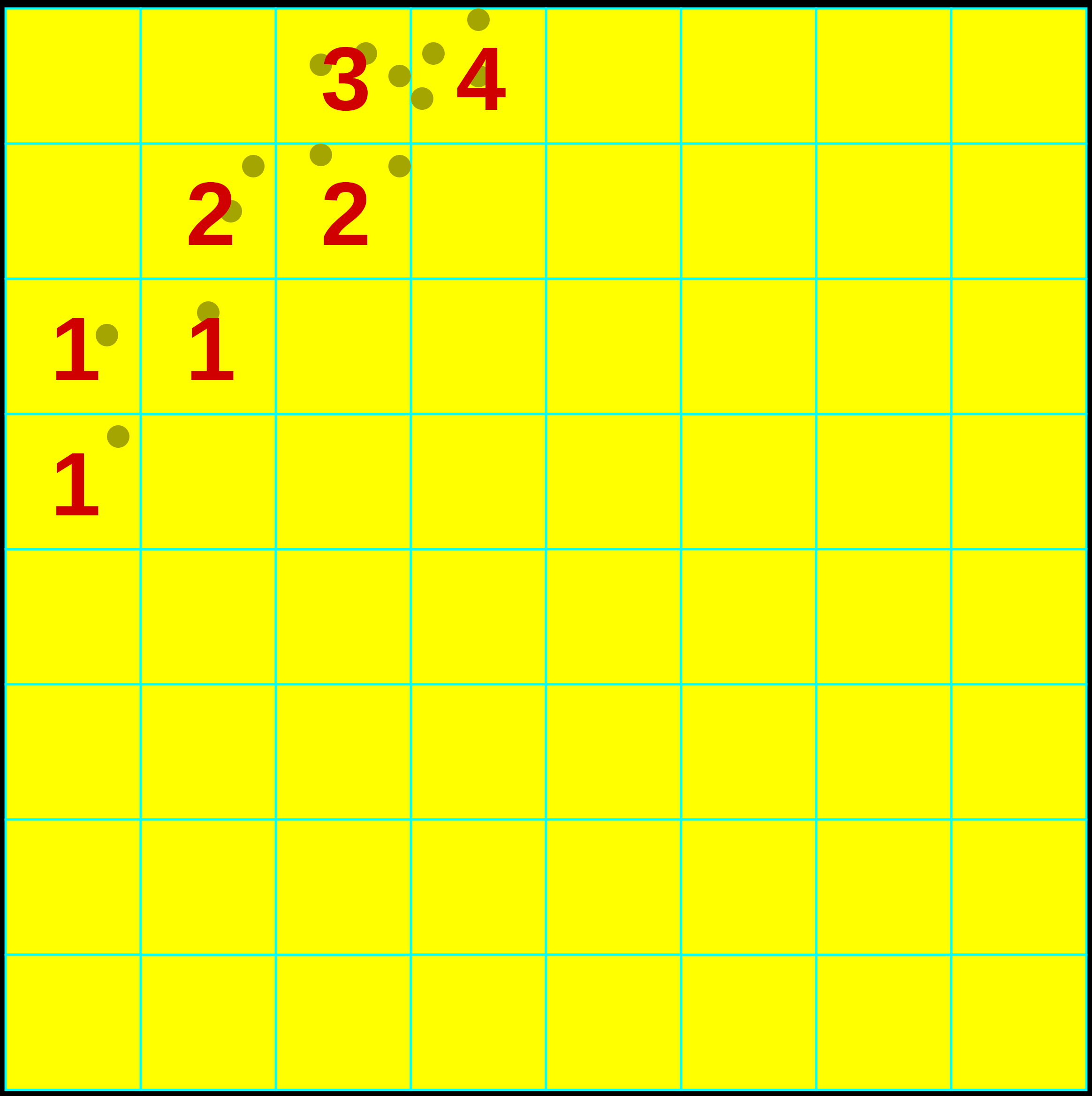
1 Compute bounding box.

How to Finalize



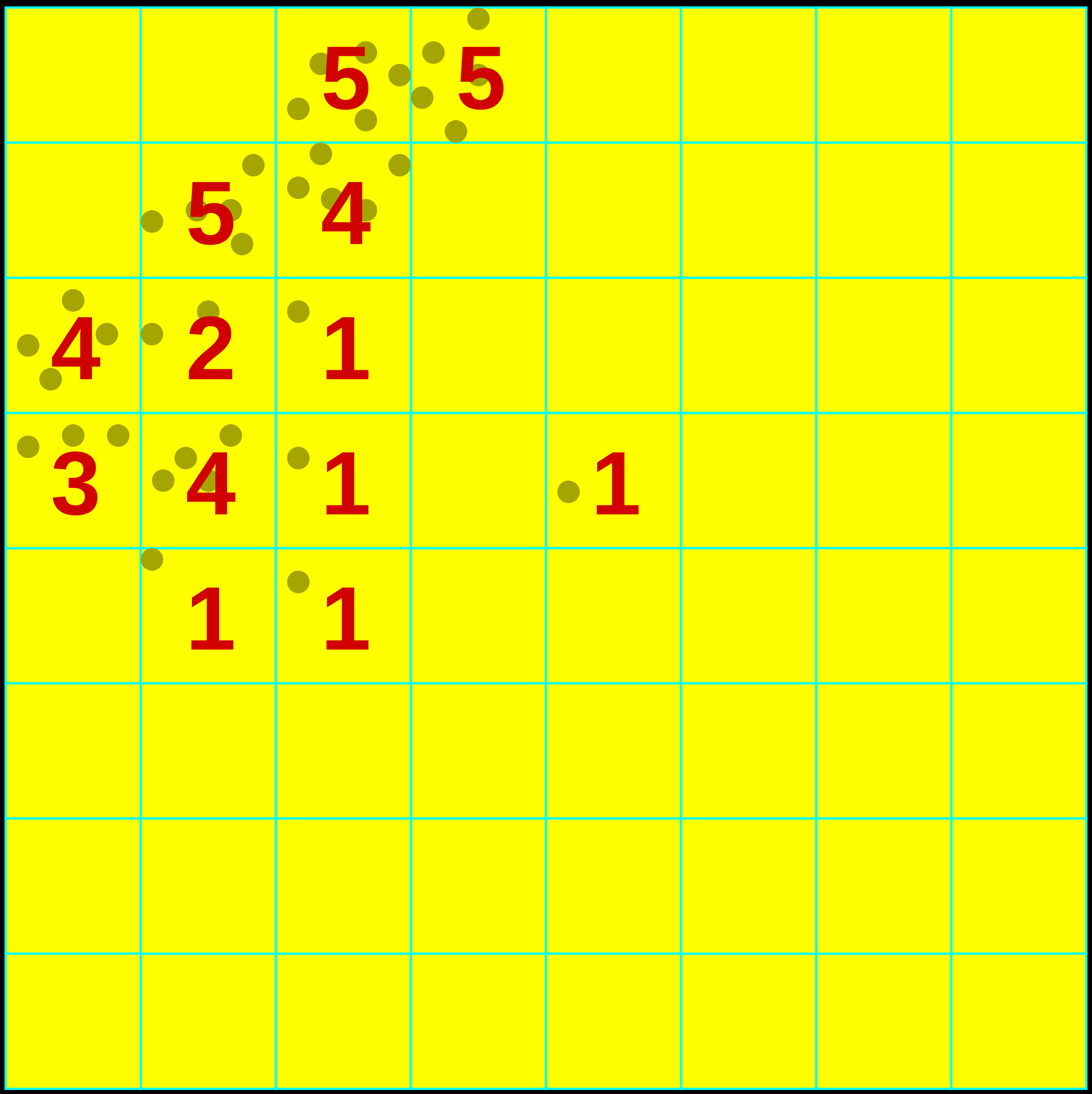
1. Compute bounding box.
2. Finalization grid.

How to Finalize



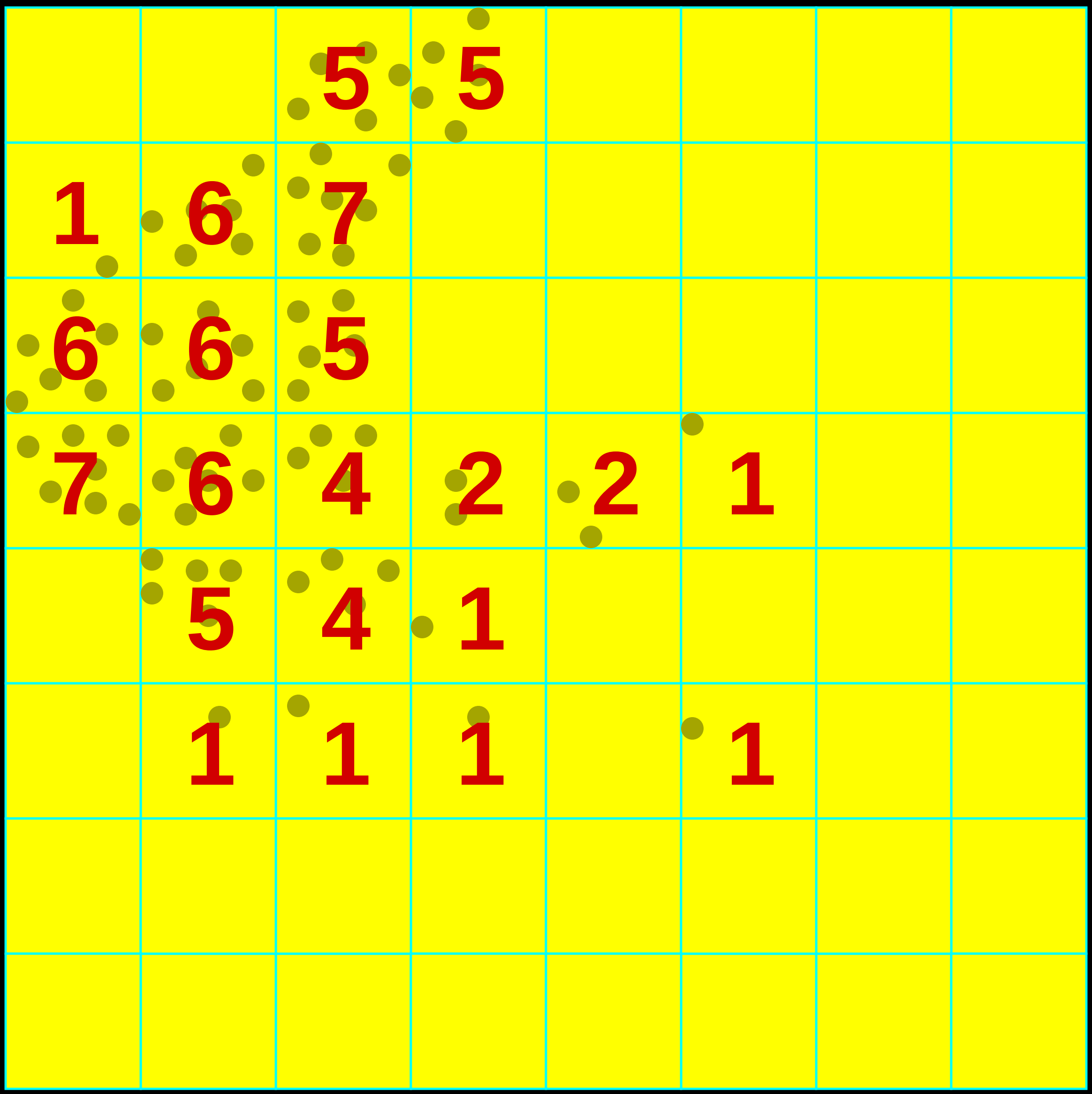
1. Compute bounding box.
2. Finalization grid.
   - Count points.

How to Finalize



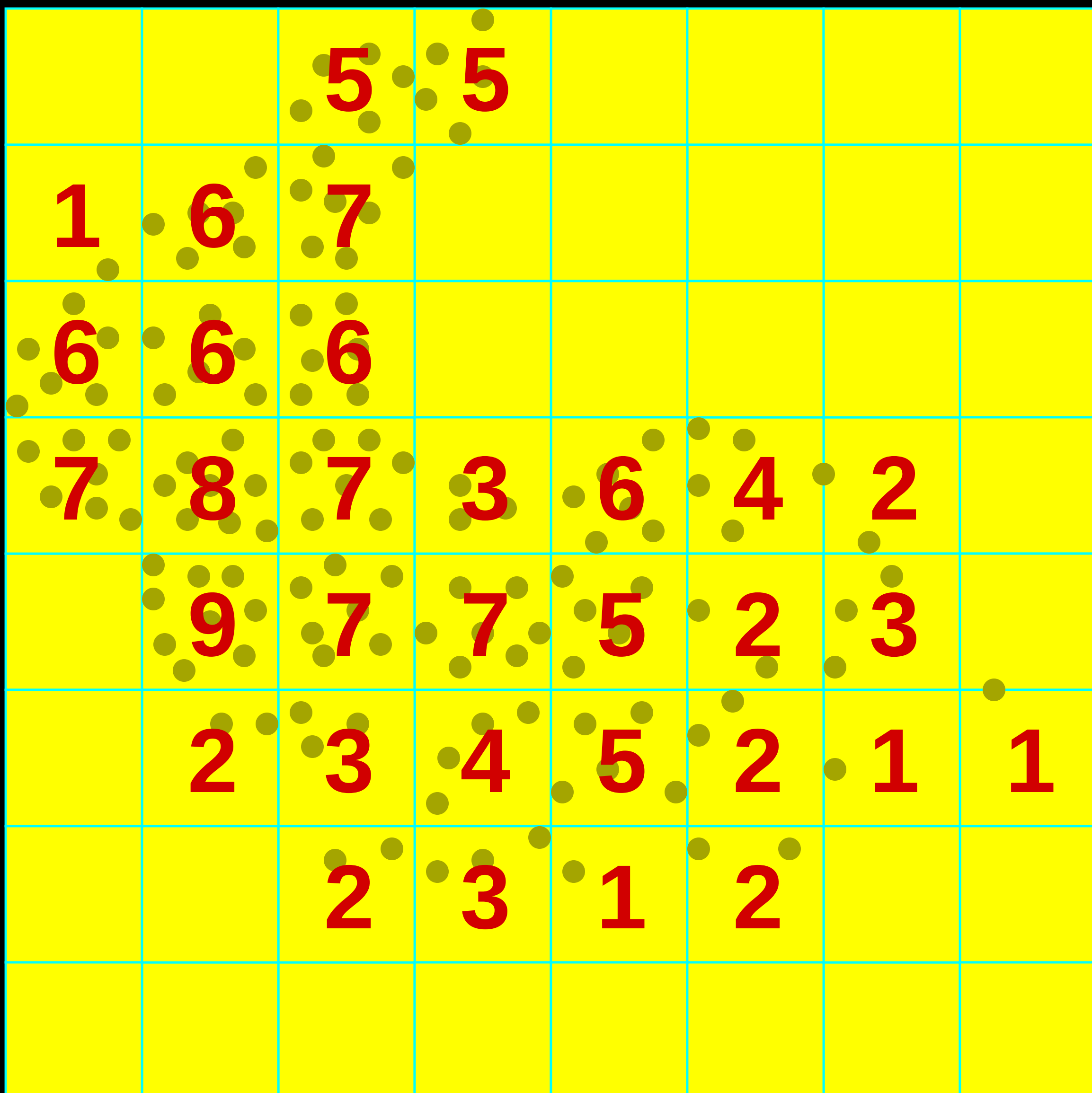
1. Compute bounding box.
2. Finalization grid.
   - Count points.

How to Finalize



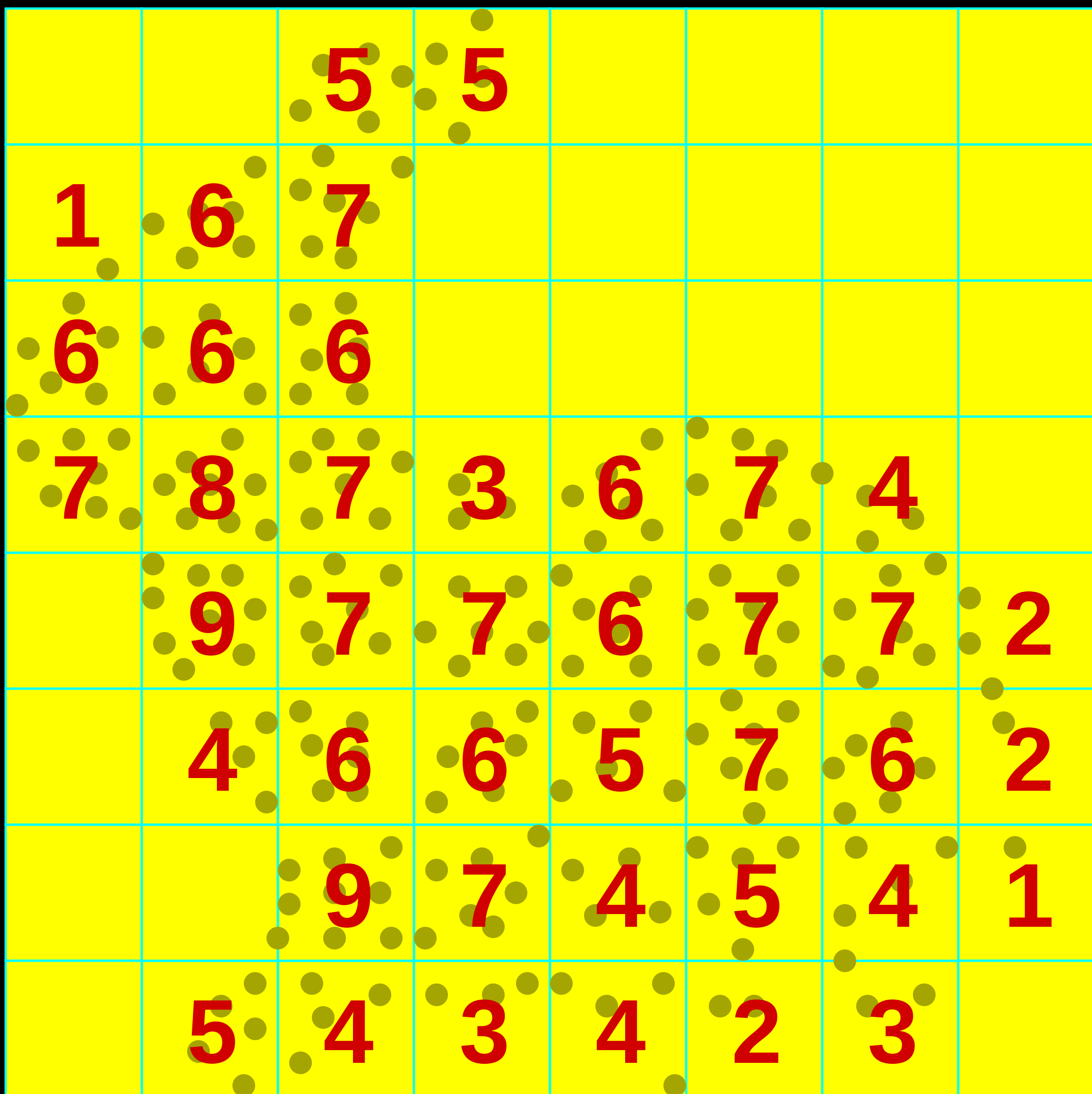
1. Compute bounding box.
2. Finalization grid.
   - Count points.

How to Finalize



1. Compute bounding box.
2. Finalization grid.
   - Count points.

How to Finalize

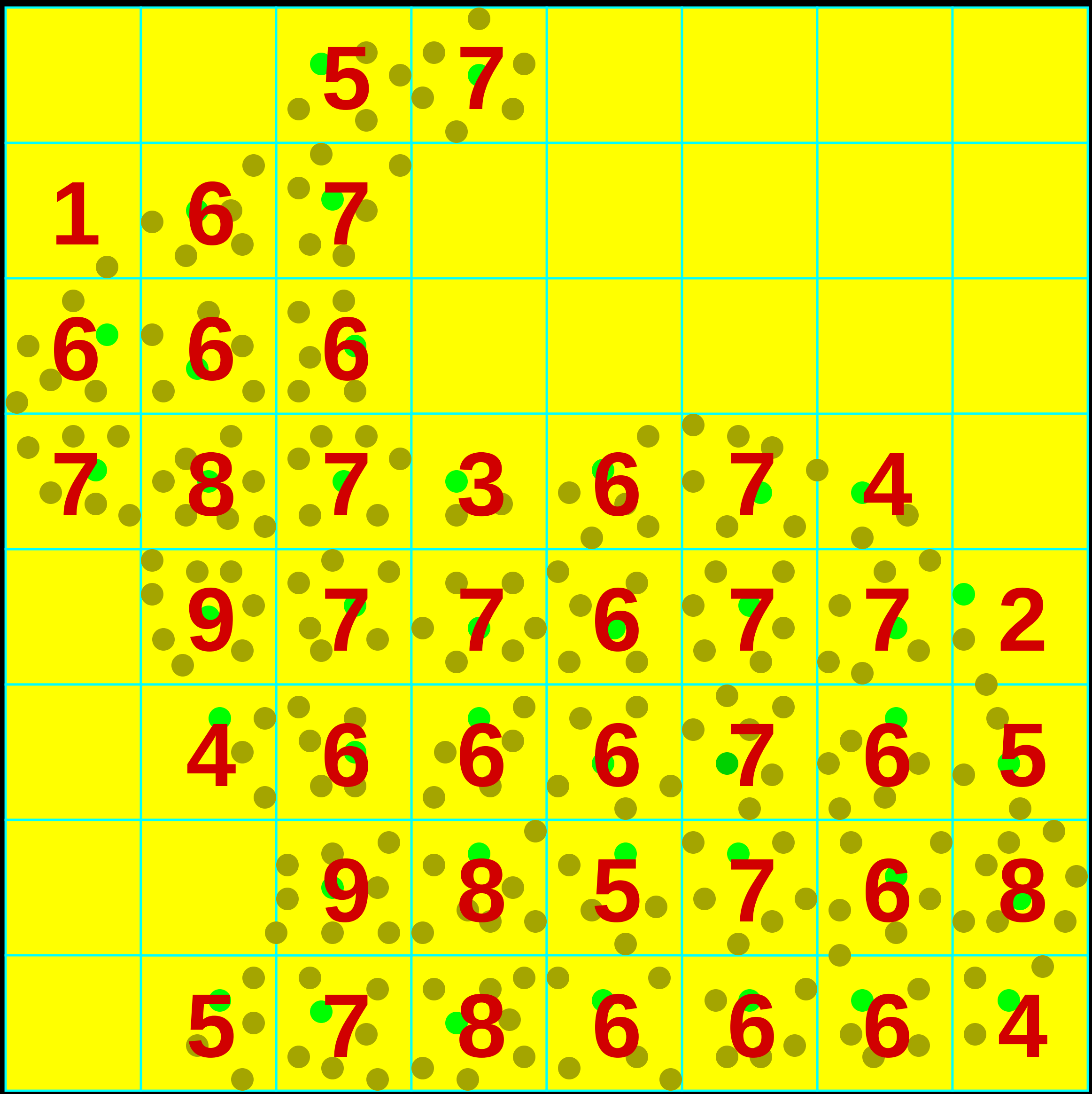


1. Compute bounding box.
2. Finalization grid.
   - Count points.

How to Finalize

1. Compute bounding box.
2. Finalization grid.
   - Count points.

How to Finalize



1. Compute bounding box.
2. Finalization grid.
   - Count points.
   - Store “sprinkle points.”

How to Finalize

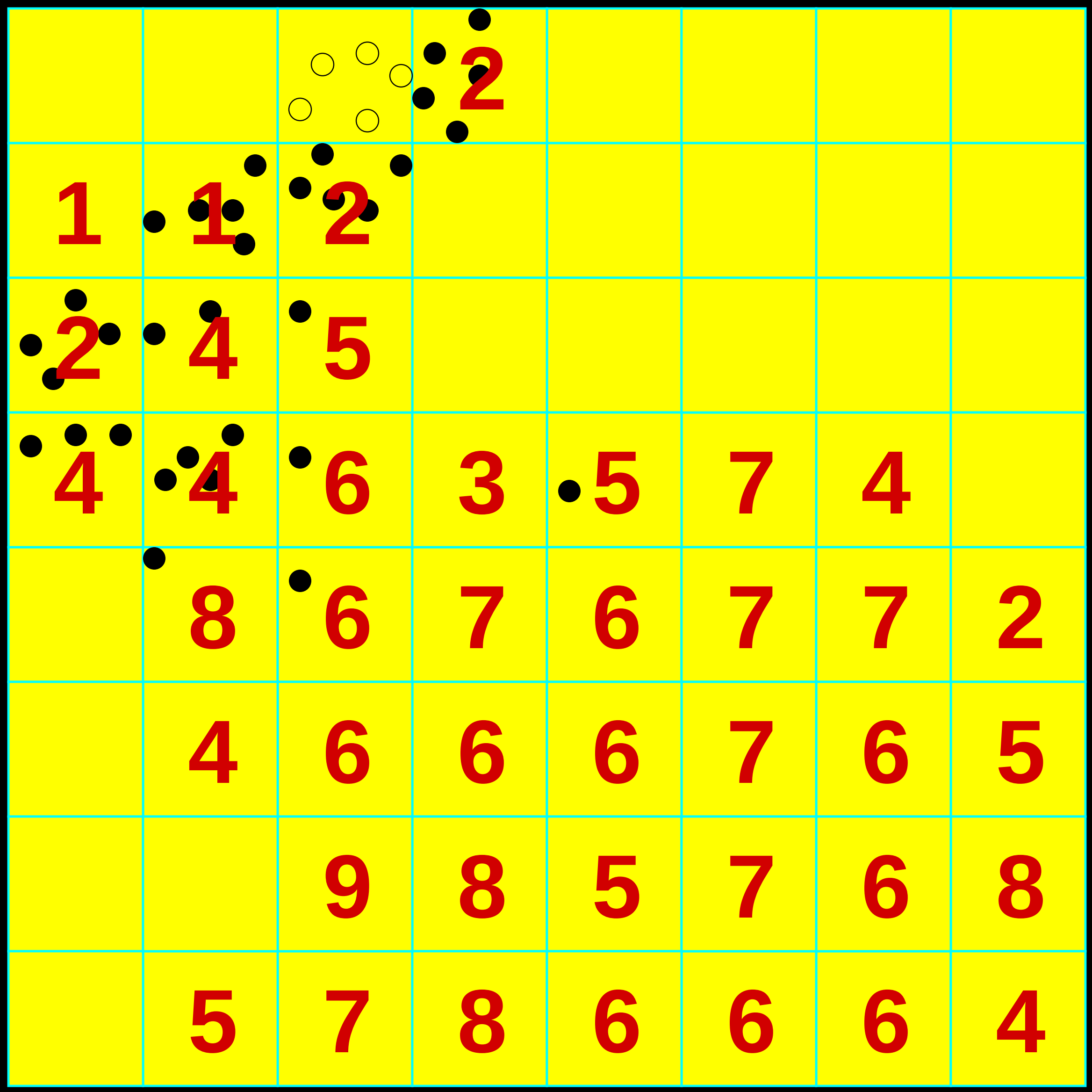
		5	7				
1	6	7					
6	6	6					
7	7	7	3	6	7	4	
	9	7	7	6	7	7	2
	4	6	6	6	7	6	5
		9	8	5	7	6	8
	5	7	8	6	6	6	4

1. Compute bounding box.
2. Finalization grid.
   - Count points.
   - Store “sprinkle points.”
3. Output finalized points.
   - Release chunks.
   - Inject finalization tags.

How to Finalize

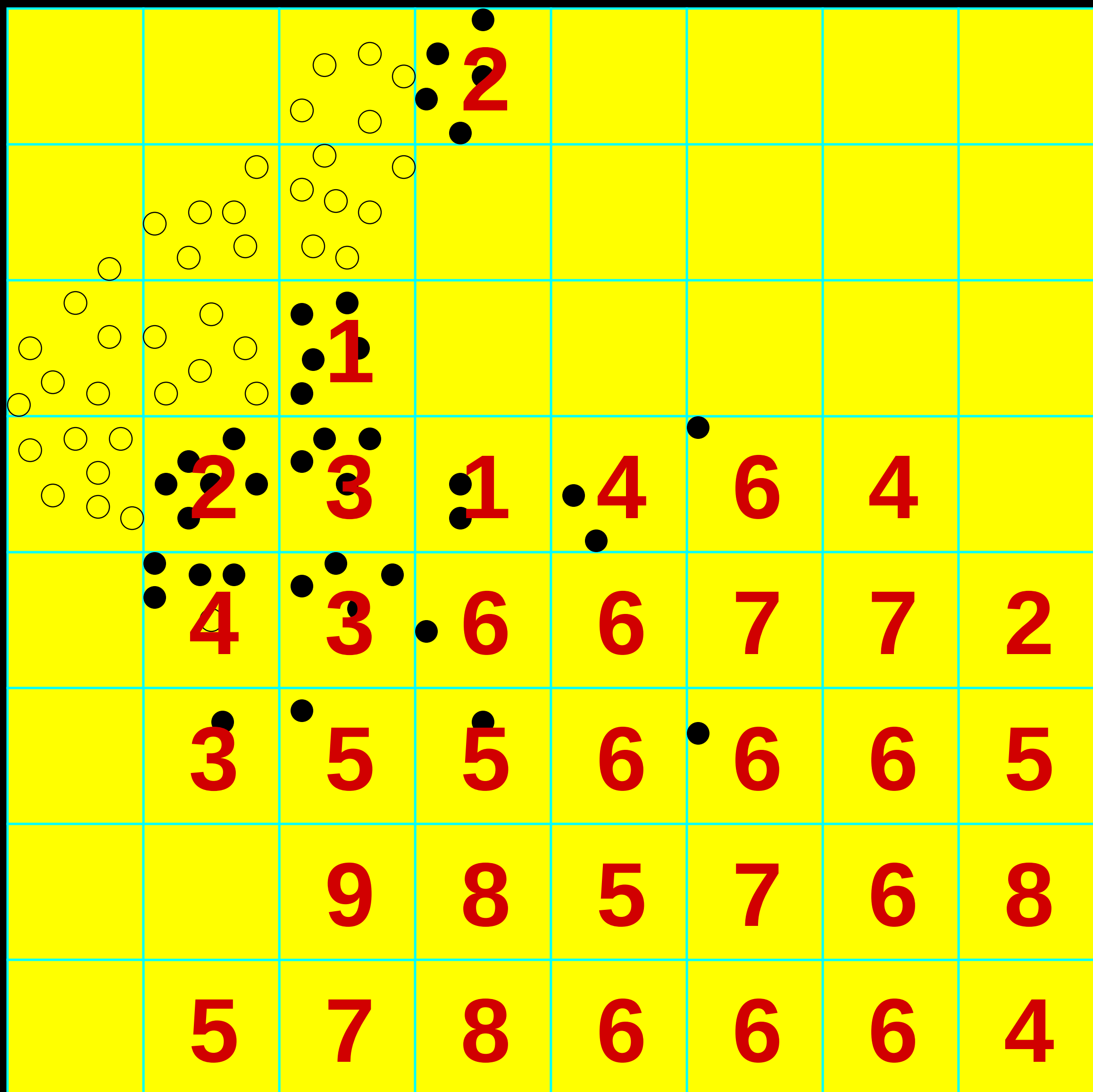
1. Compute bounding box.
2. Finalization grid.
   - Count points.
   - Store “sprinkle points.”
3. Output finalized points.
   - Release chunks.
   - Inject finalization tags.

How to Finalize



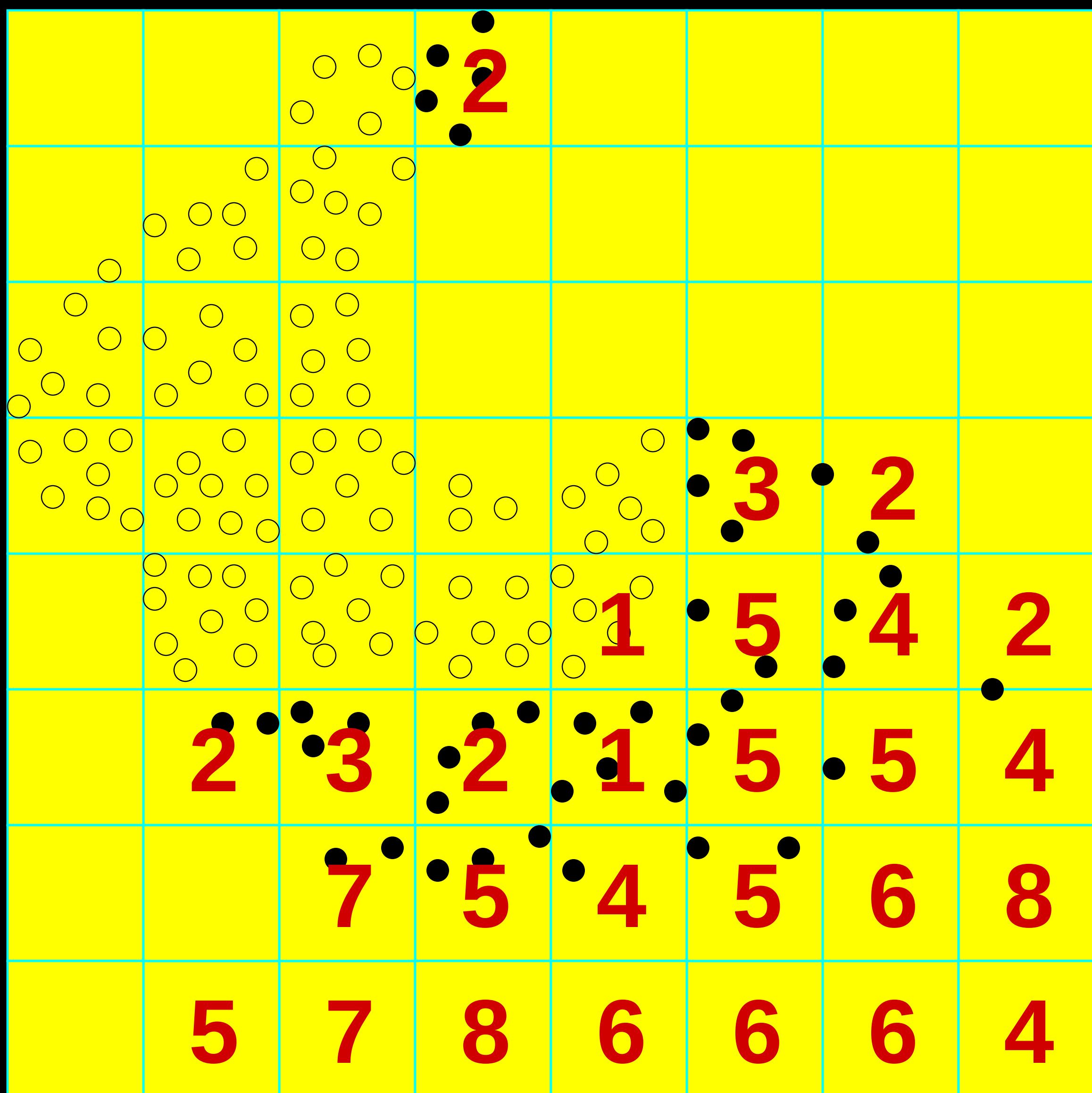
1. Compute bounding box.
2. Finalization grid.
   - Count points.
   - Store "sprinkle points."
3. Output finalized points.
   - Release chunks.
   - Inject finalization tags.

How to Finalize



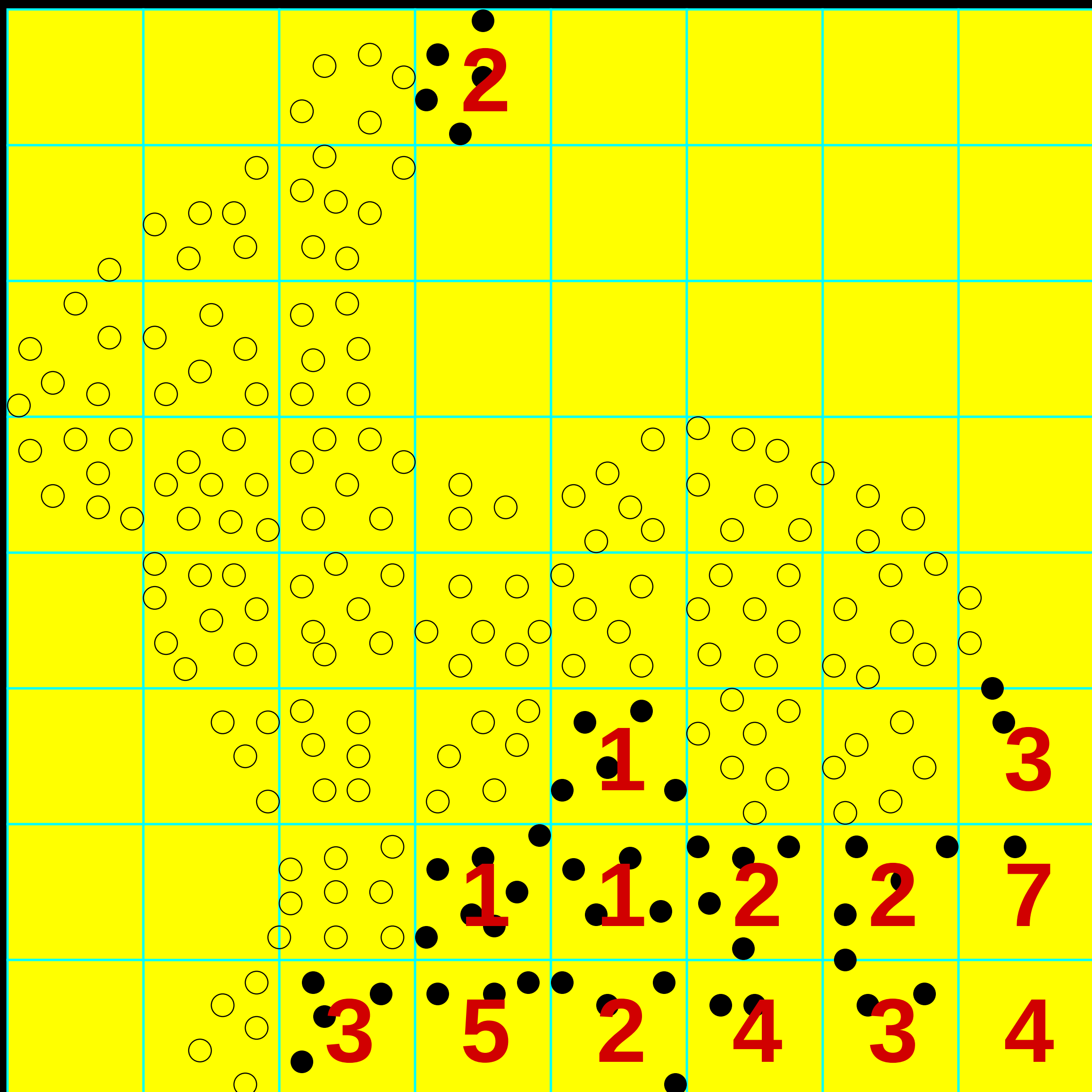
1. Compute bounding box.
2. Finalization grid.
   - Count points.
   - Store “sprinkle points.”
3. Output finalized points.
   - Release chunks.
   - Inject finalization tags.

How to Finalize



1. Compute bounding box.
2. Finalization grid.
   - Count points.
   - Store “sprinkle points.”
3. Output finalized points.
   - Release chunks.
   - Inject finalization tags.

How to Finalize

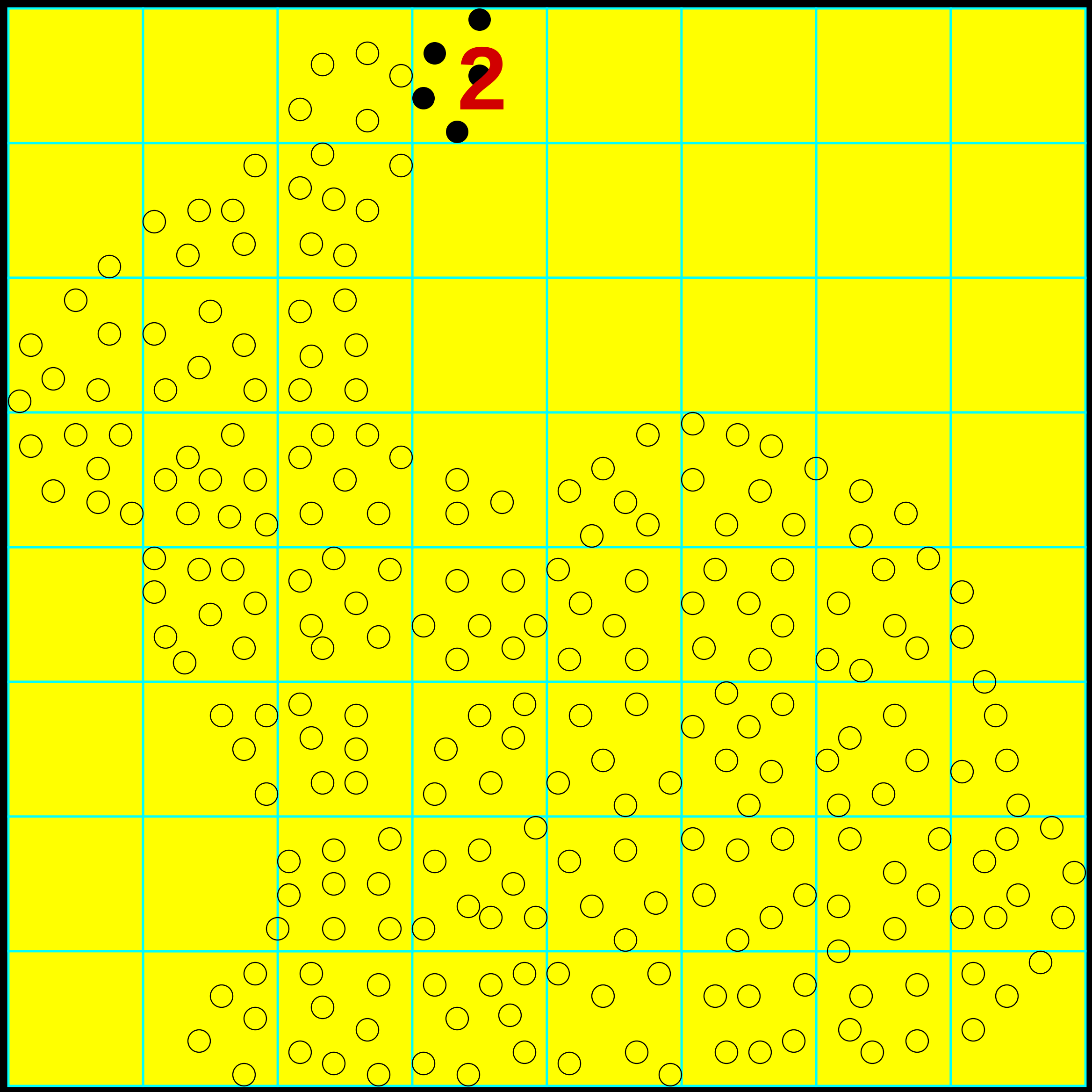


1 Compute bounding box.

2 Finalization grid.
- Count points.
- Store "sprinkle points."

3 Output finalized points.
- Release chunks.
- Inject finalization tags.

How to Finalize

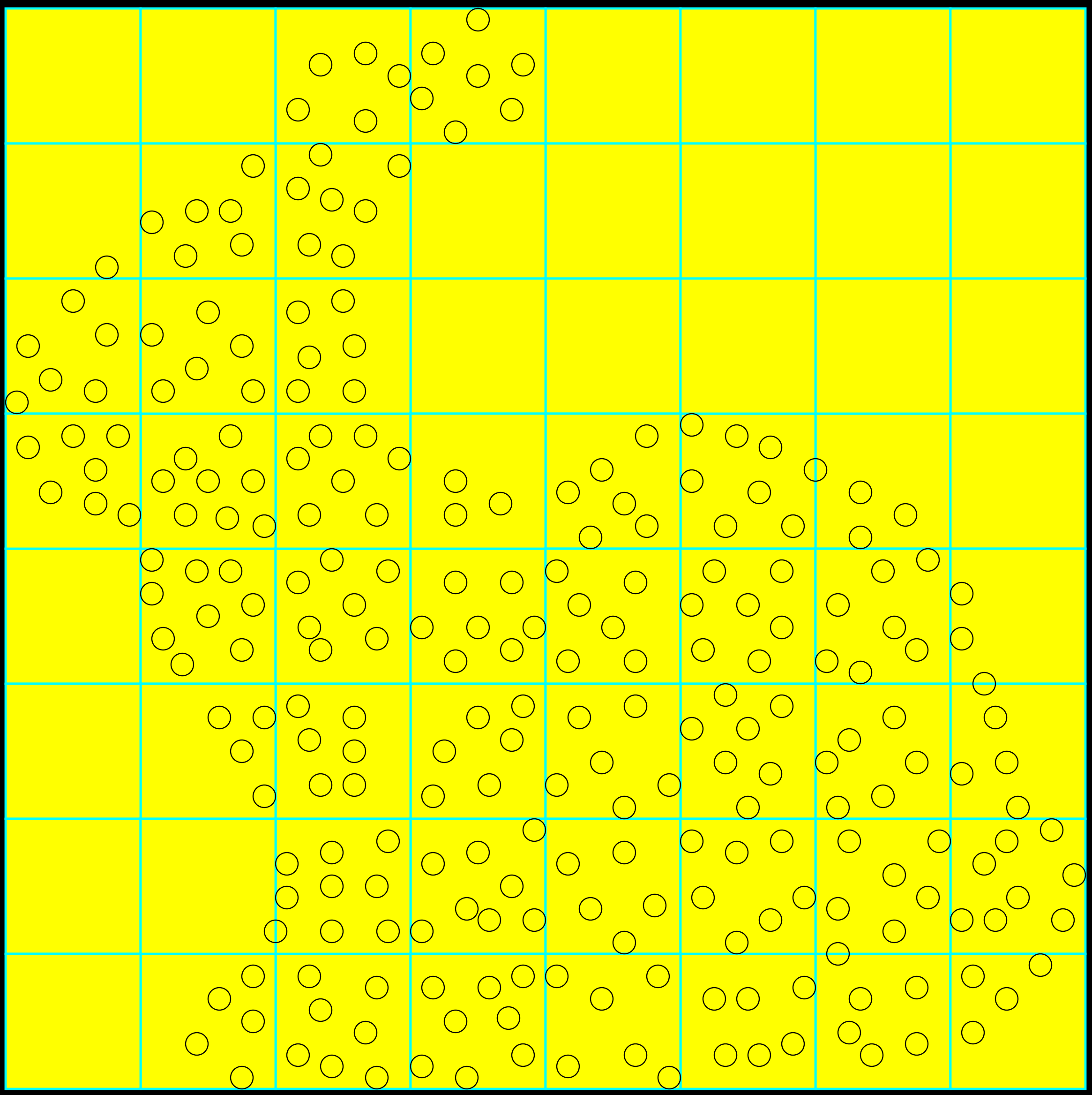


1 Compute bounding box.

2 Finalization grid.
- Count points.
- Store “sprinkle points.”

3 Output finalized points.
- Release chunks.
- Inject finalization tags.

How to Finalize



1 Compute bounding box.

2 Finalization grid.

- Count points.
- Store ‘‘sprinkle points.’’

3 Output finalized points.

- Release chunks.
- Inject finalization tags.

Streaming Point Format

```
# npoints 22514
# bb_min_f 0 0 192.25
# bb_max_f 122.5 158.5 215.7
# datatype SP_FLOAT
v 0 3 193.1
v 0.4375 3.5 193.11
v 0.59375 4 193.09
v 0.6875 4.5 193.11
v 0.90625 2.5 193.13
v 0.90625 4.5 193.07
```

chunk of points

```
x cell 2732
```

finalization tag

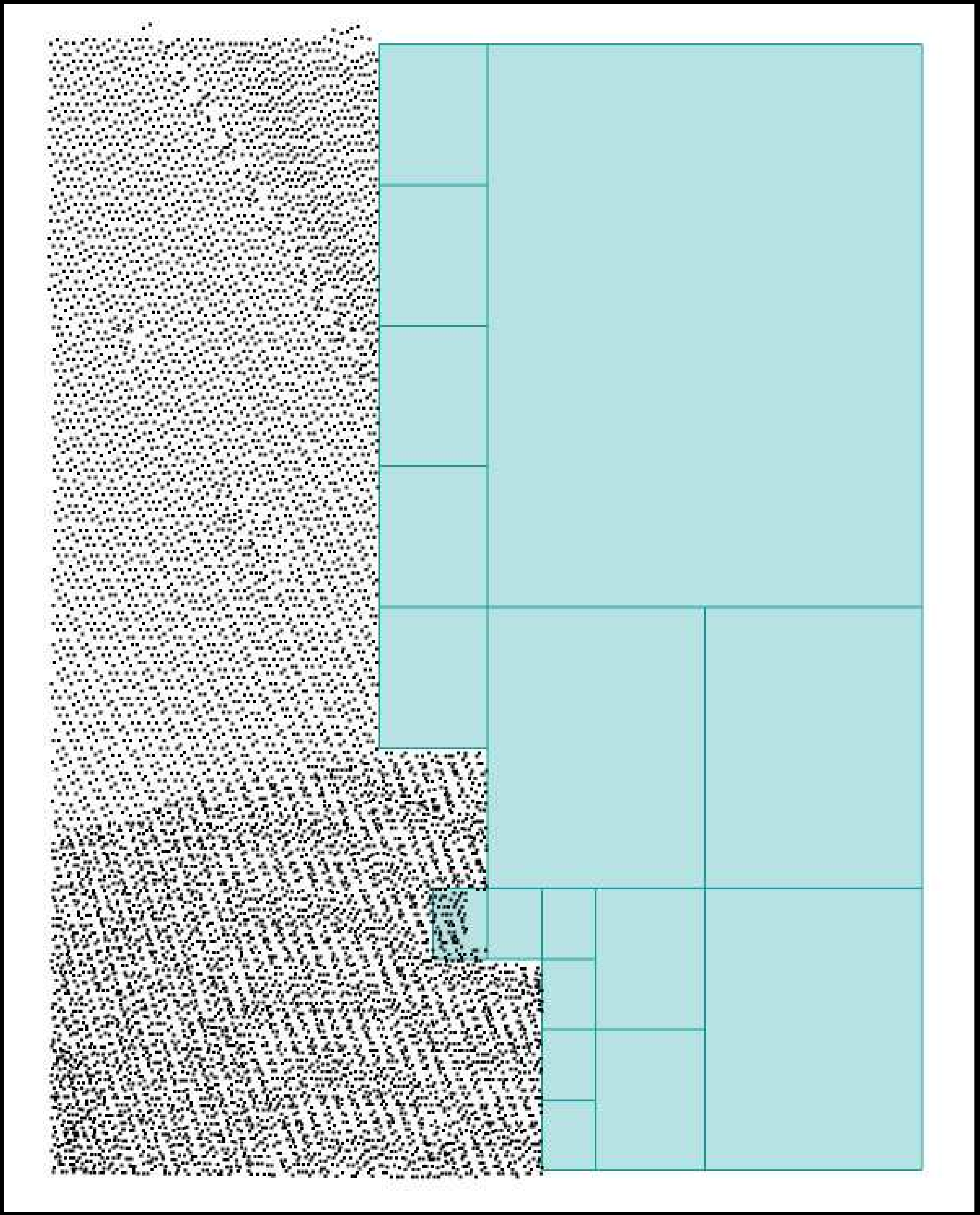
```
v 0 7.5 192.91
v 0.21875 9 192.93
v 0.53125 9 192.99
v 0.8125 7.5 192.96
v 0.9375 9 193.04
v 1.03125 9.5 193.06
v 1.65625 8.5 193.04
```

chunk of points

```
x cell 2740
```

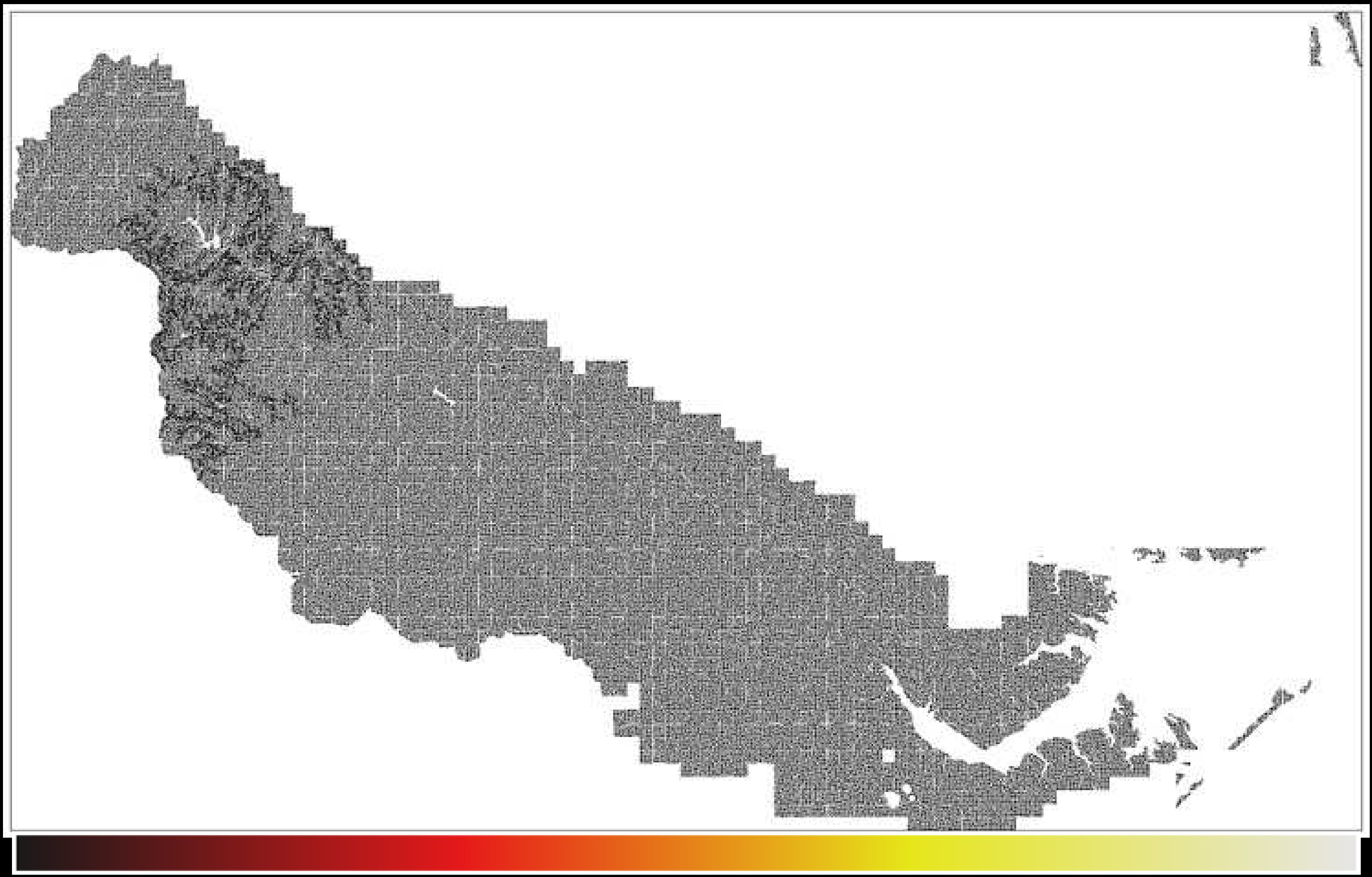
finalization tag

```
v 0.21875 0 193.06
```



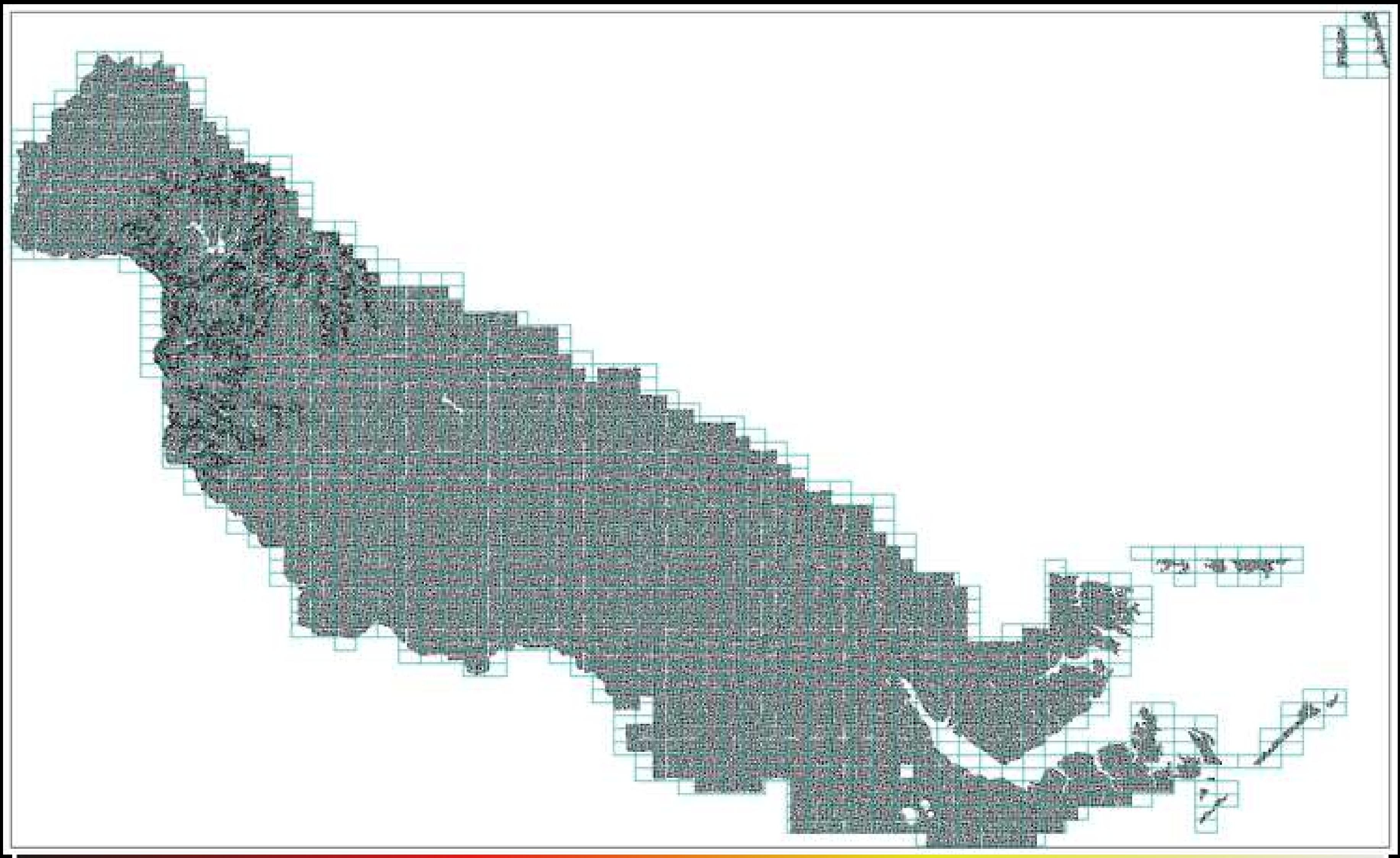
Spatial Coherence

The correlation between the proximity of points in space and their proximity in the stream.



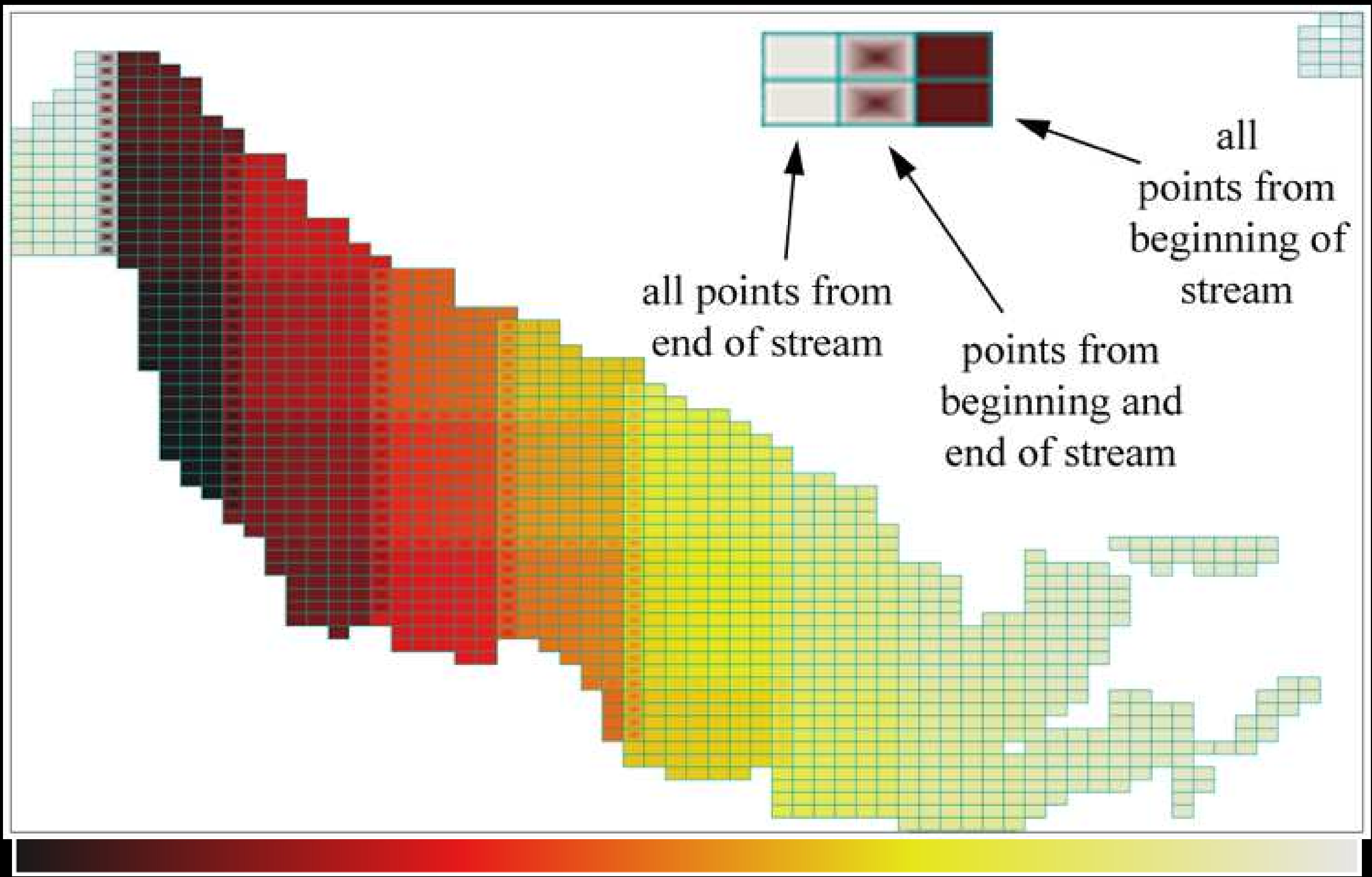
Spatial Coherence

The correlation between the proximity of points in space and their proximity in the stream.



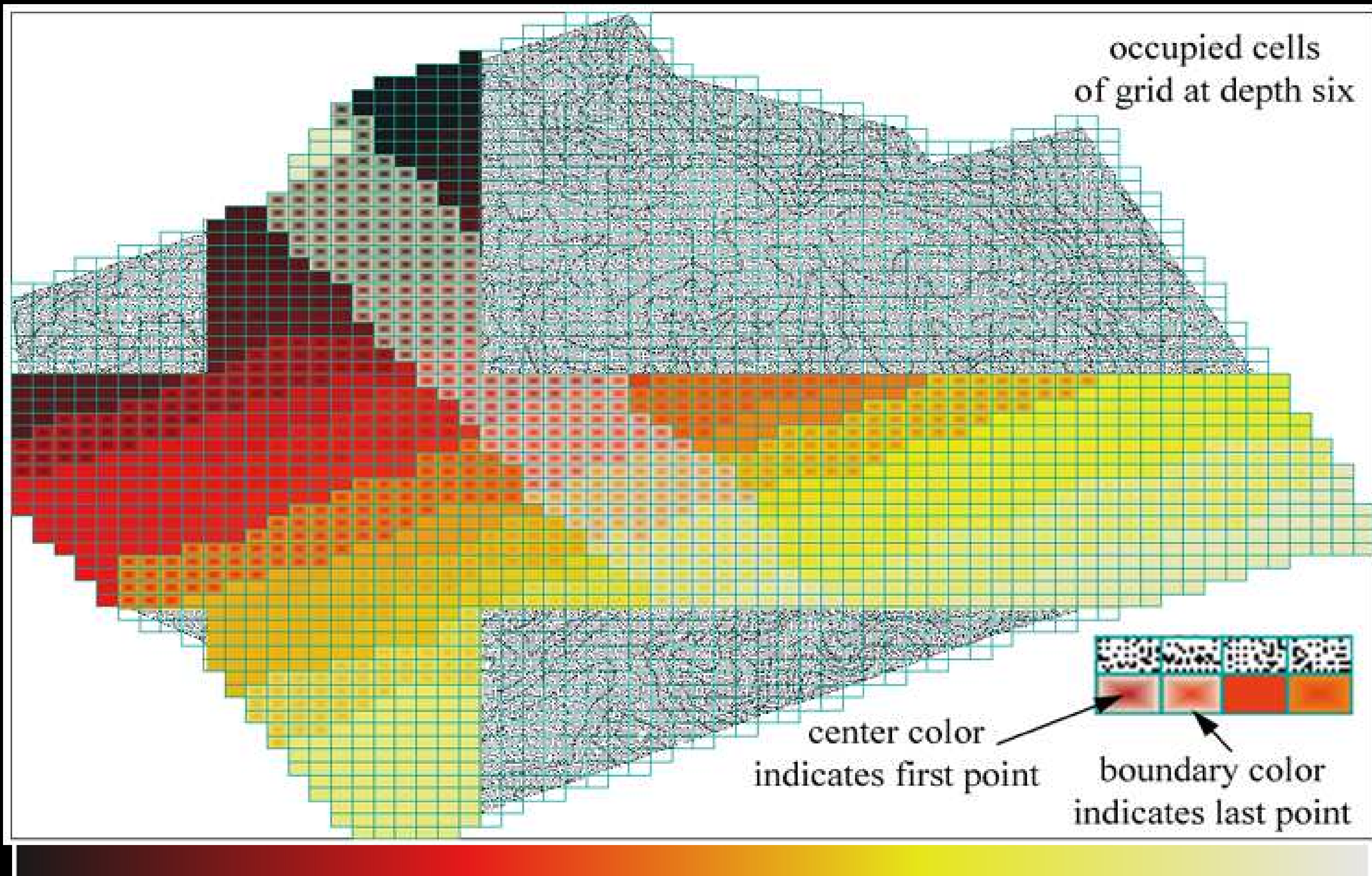
Spatial Coherence

The correlation between the proximity of points in space and their proximity in the stream.

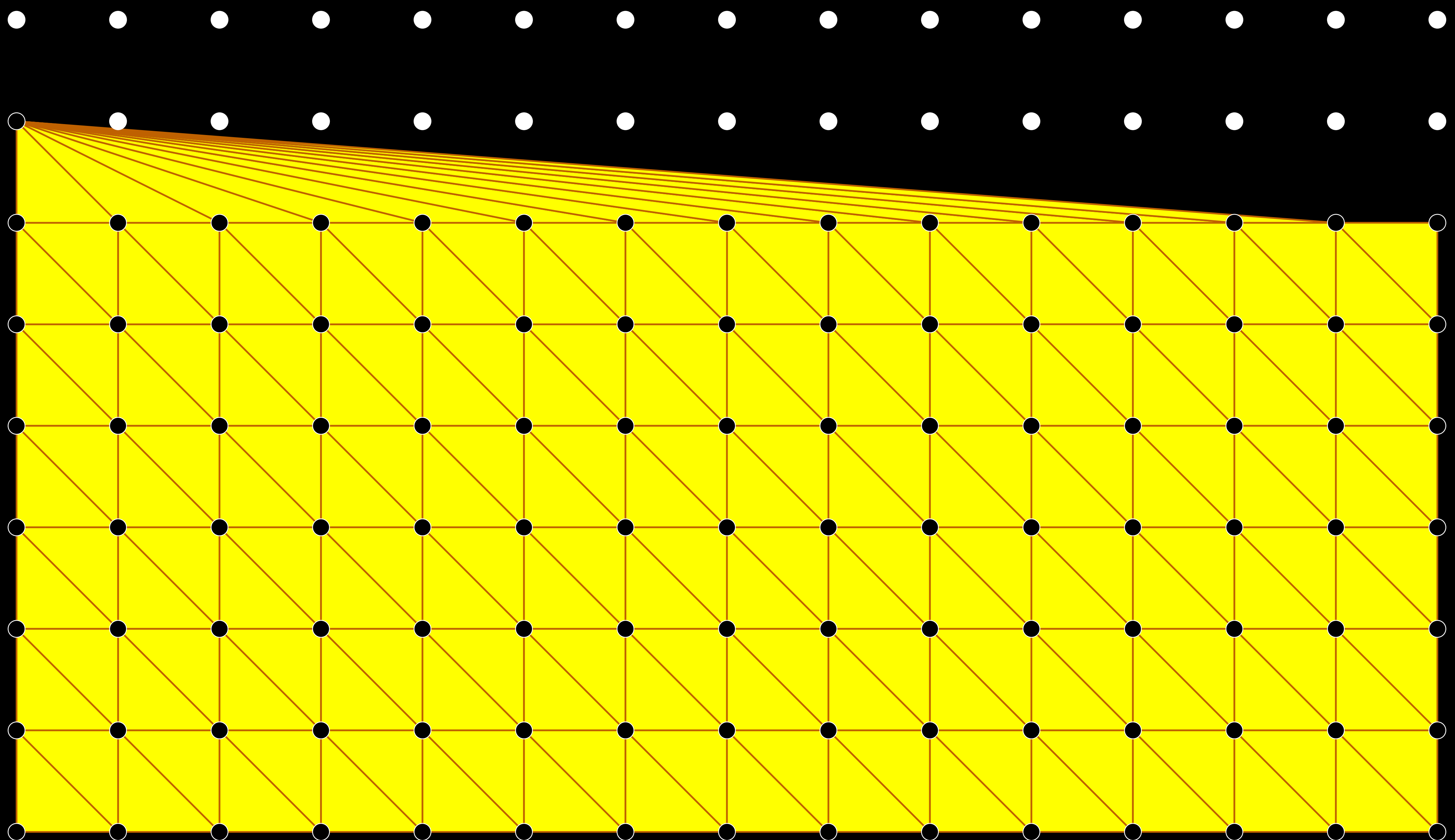


Spatial Coherence

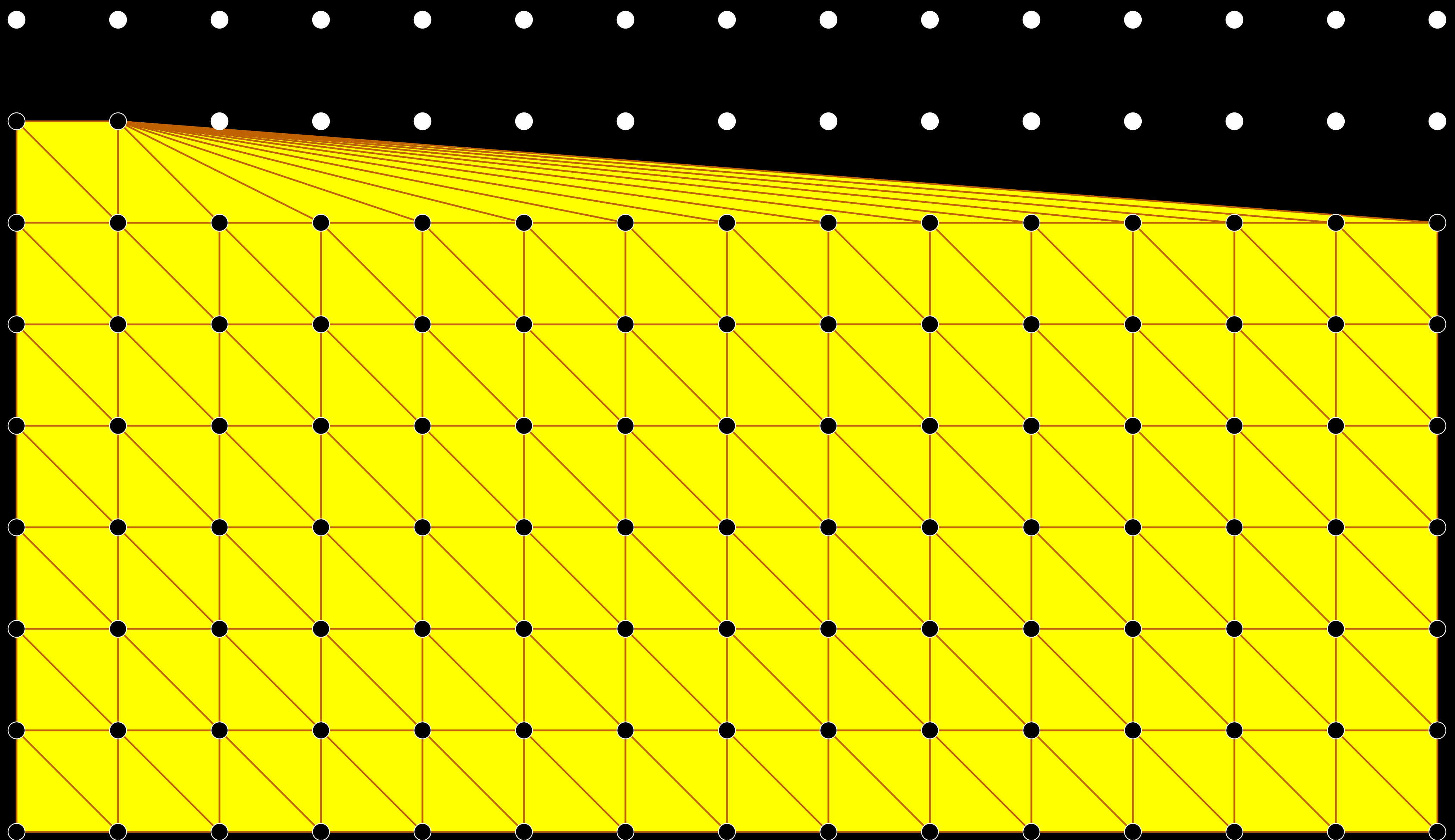
The correlation between the proximity of points in space and their proximity in the stream.



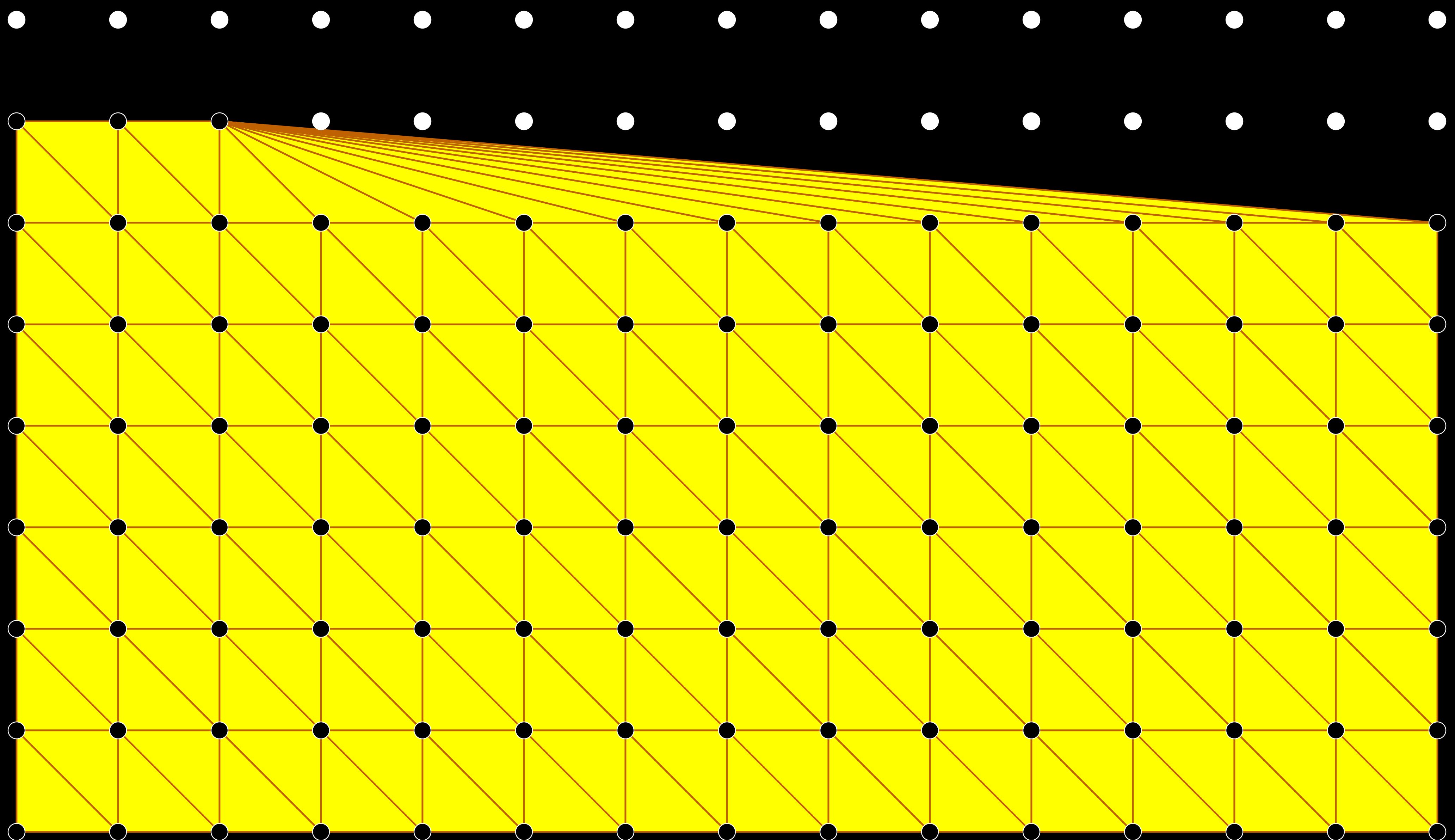
Danger: Potential Quadratic Time



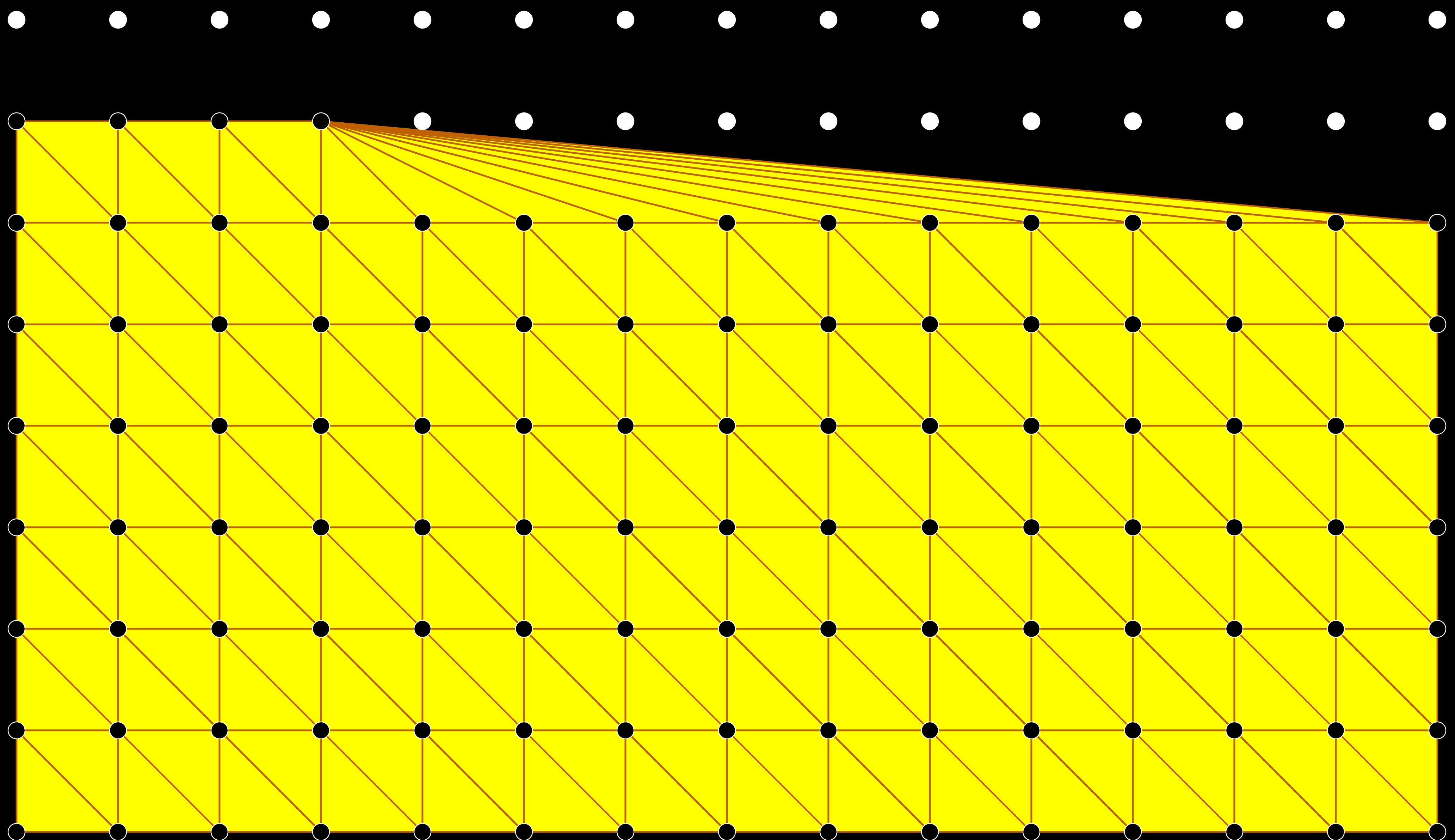
Danger: Potential Quadratic Time



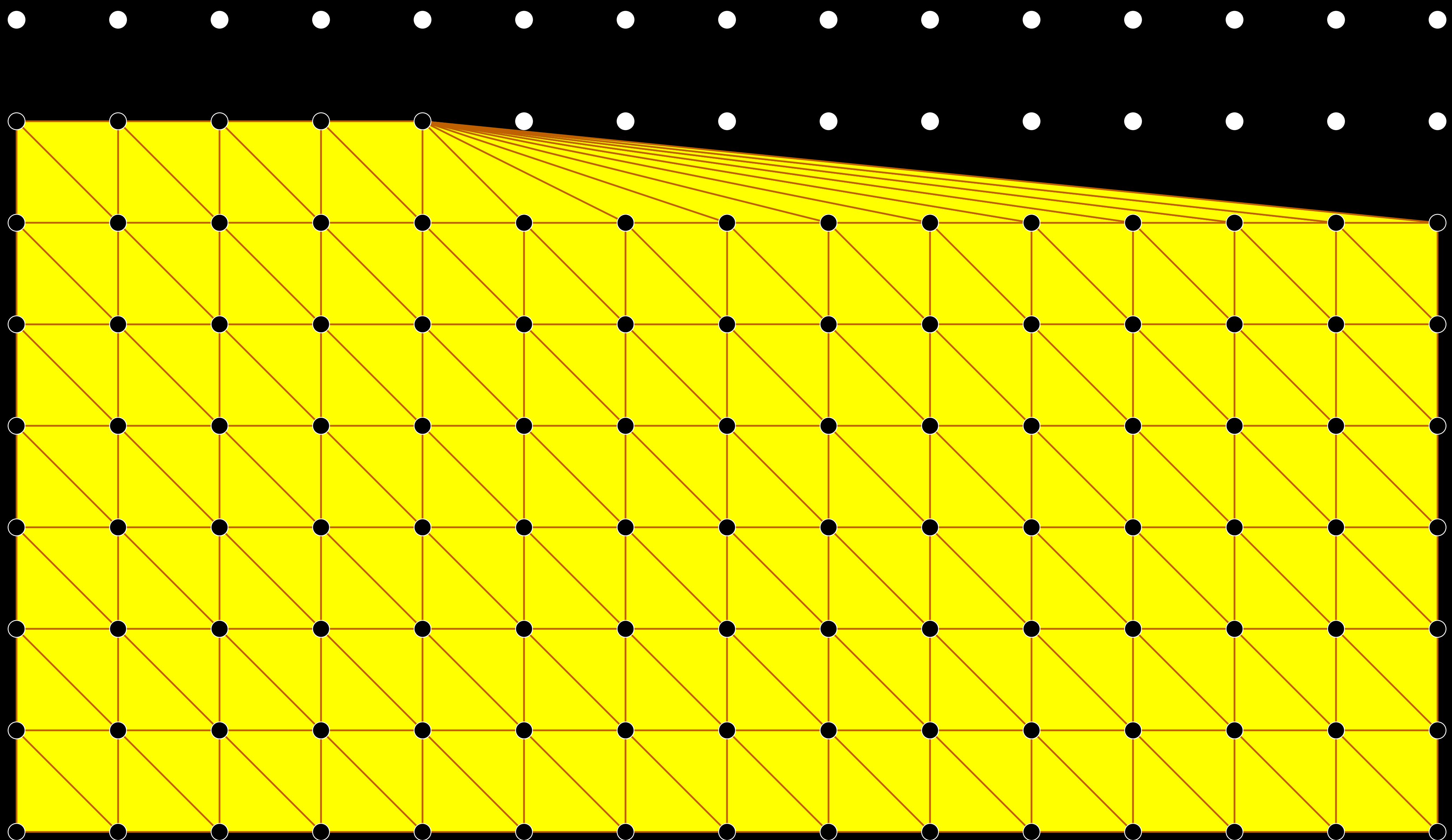
Danger: Potential Quadratic Time



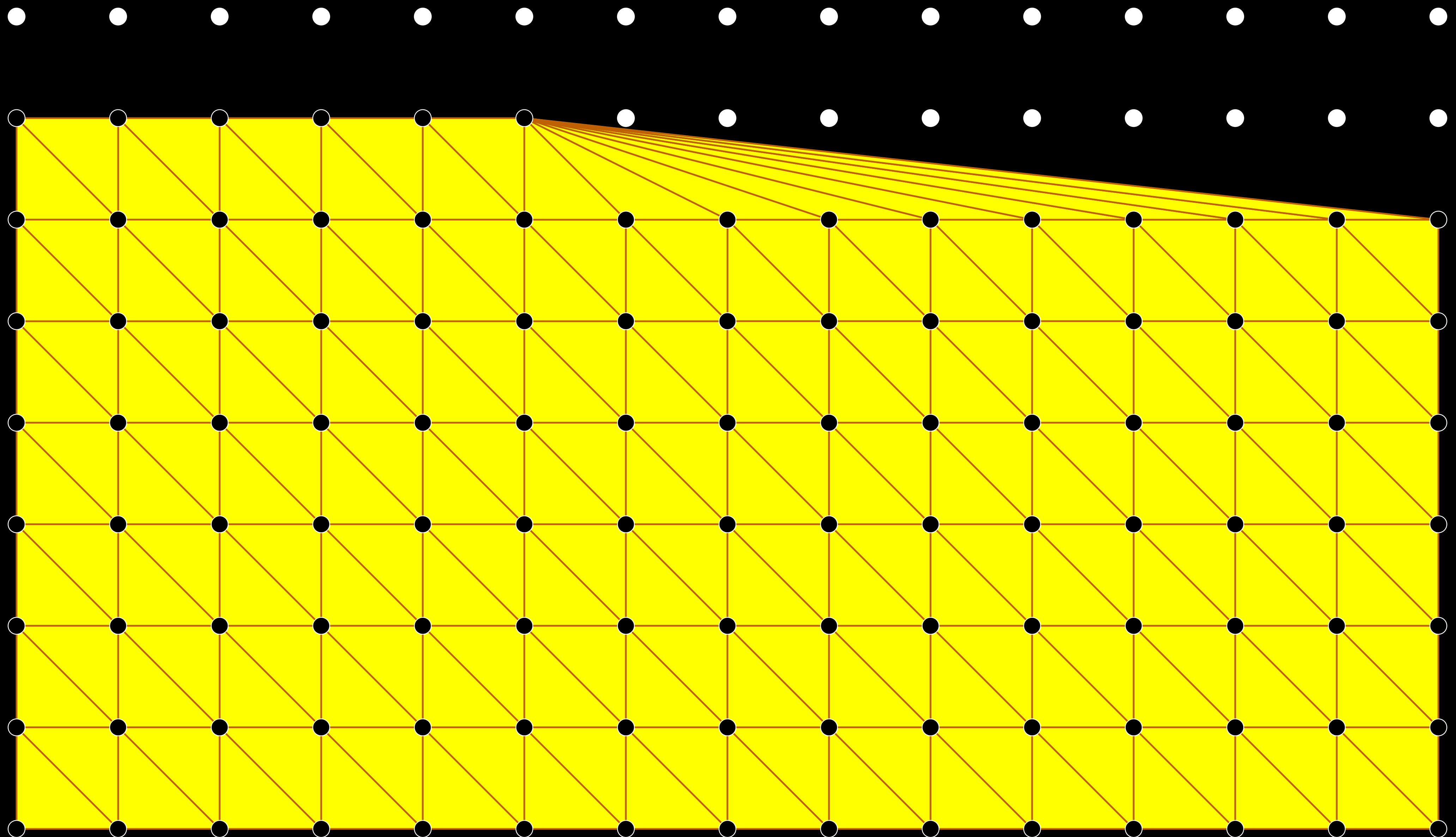
Danger: Potential Quadratic Time



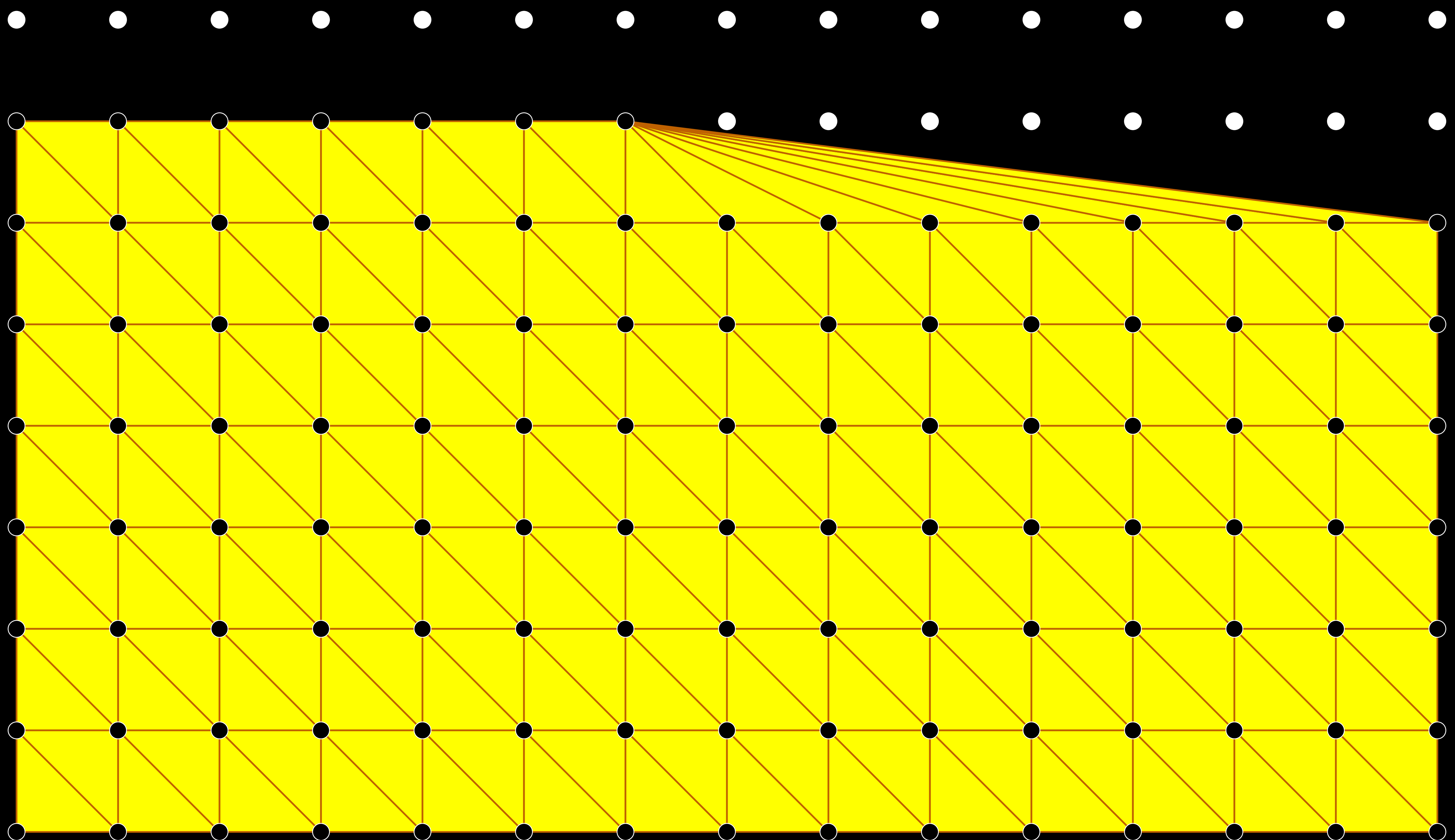
Danger: Potential Quadratic Time



Danger: Potential Quadratic Time



Danger: Potential Quadratic Time

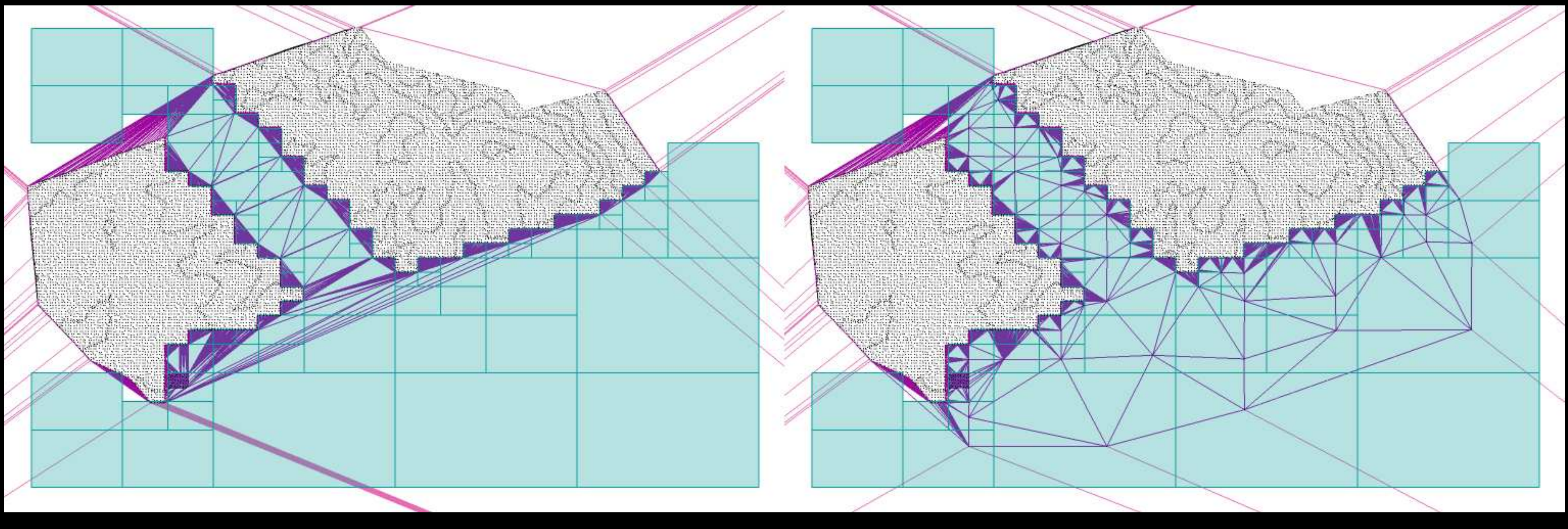


Point Reordering by Finalizer

- *Chunking:* Buffering points in a region until all points arrive, then releasing them together.
  - Points in a chunk are randomly reordered.

Point Reordering by Finalizer

- *Chunking:* Buffering points in a region until all points arrive, then releasing them together.
  - Points in a chunk are randomly reordered.
- *Sprinkling:* Promoting a small random sample of “sprinkle points” to earlier in the stream.
  - Circumvents danger of quadratic behavior.

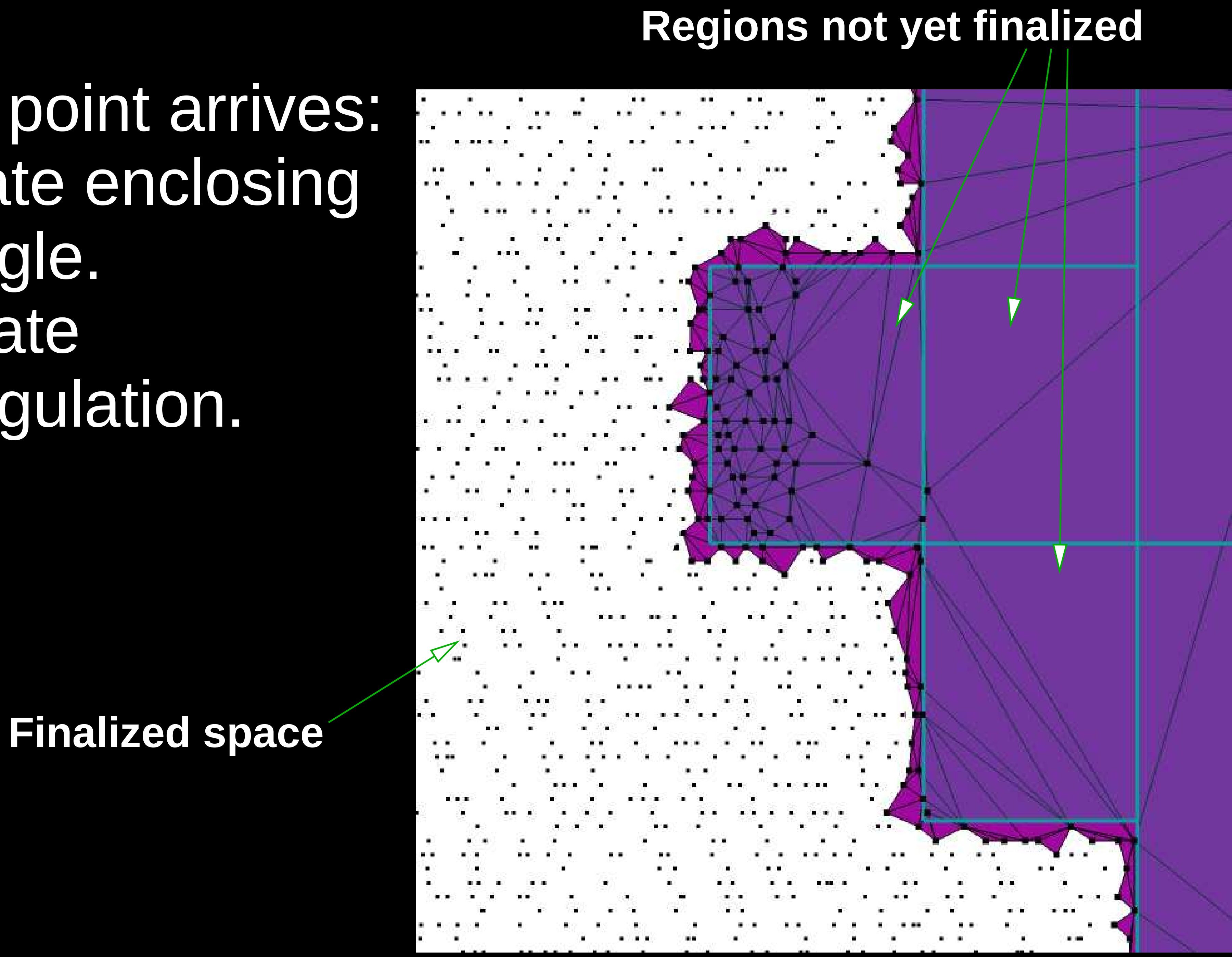


The Triangulator

Streaming Delaunay Triangulation

When point arrives:

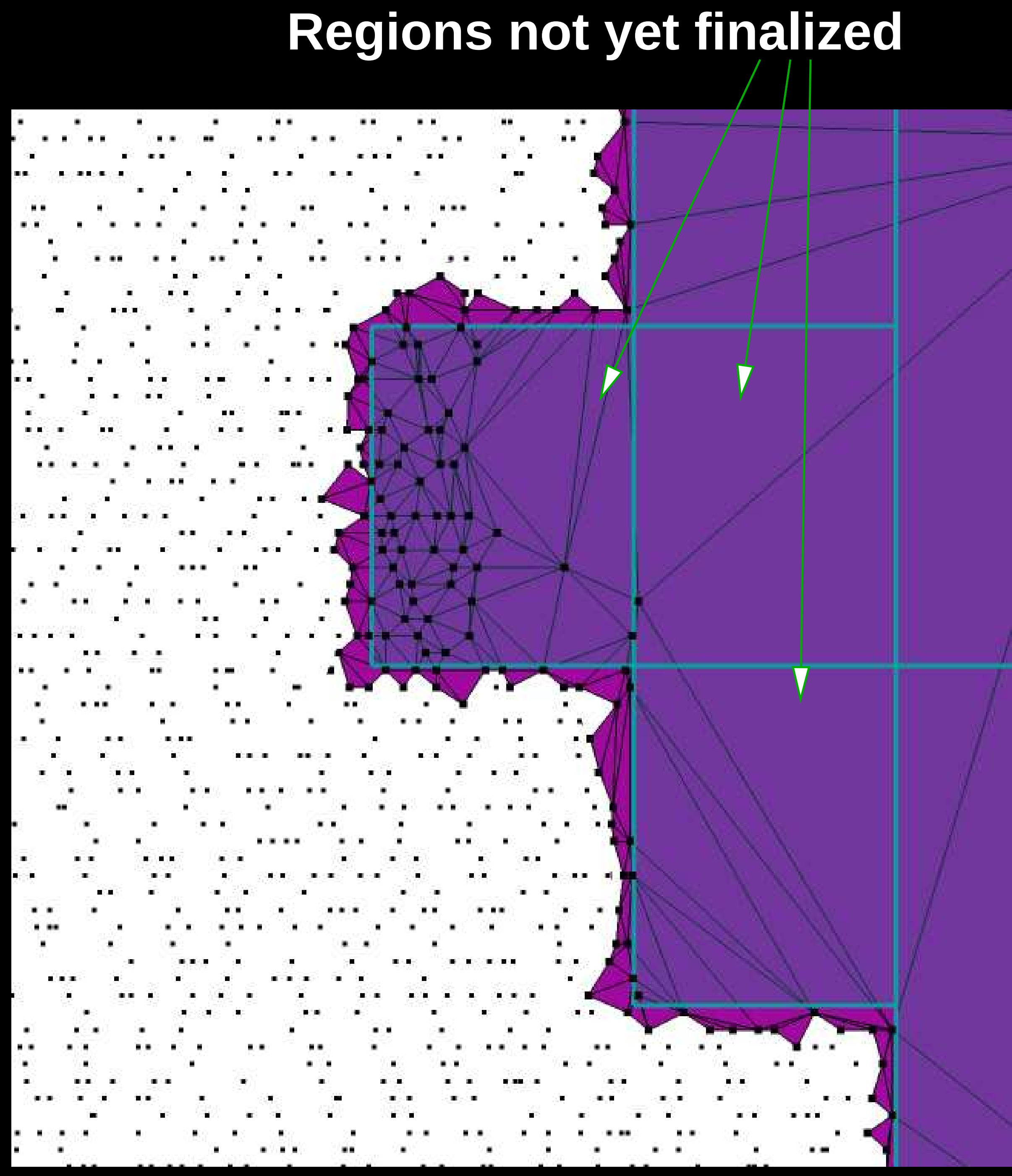
- Locate enclosing triangle.
- Update triangulation.



Streaming Delaunay Triangulation

When point arrives:
- Locate enclosing triangle.
- Update triangulation.

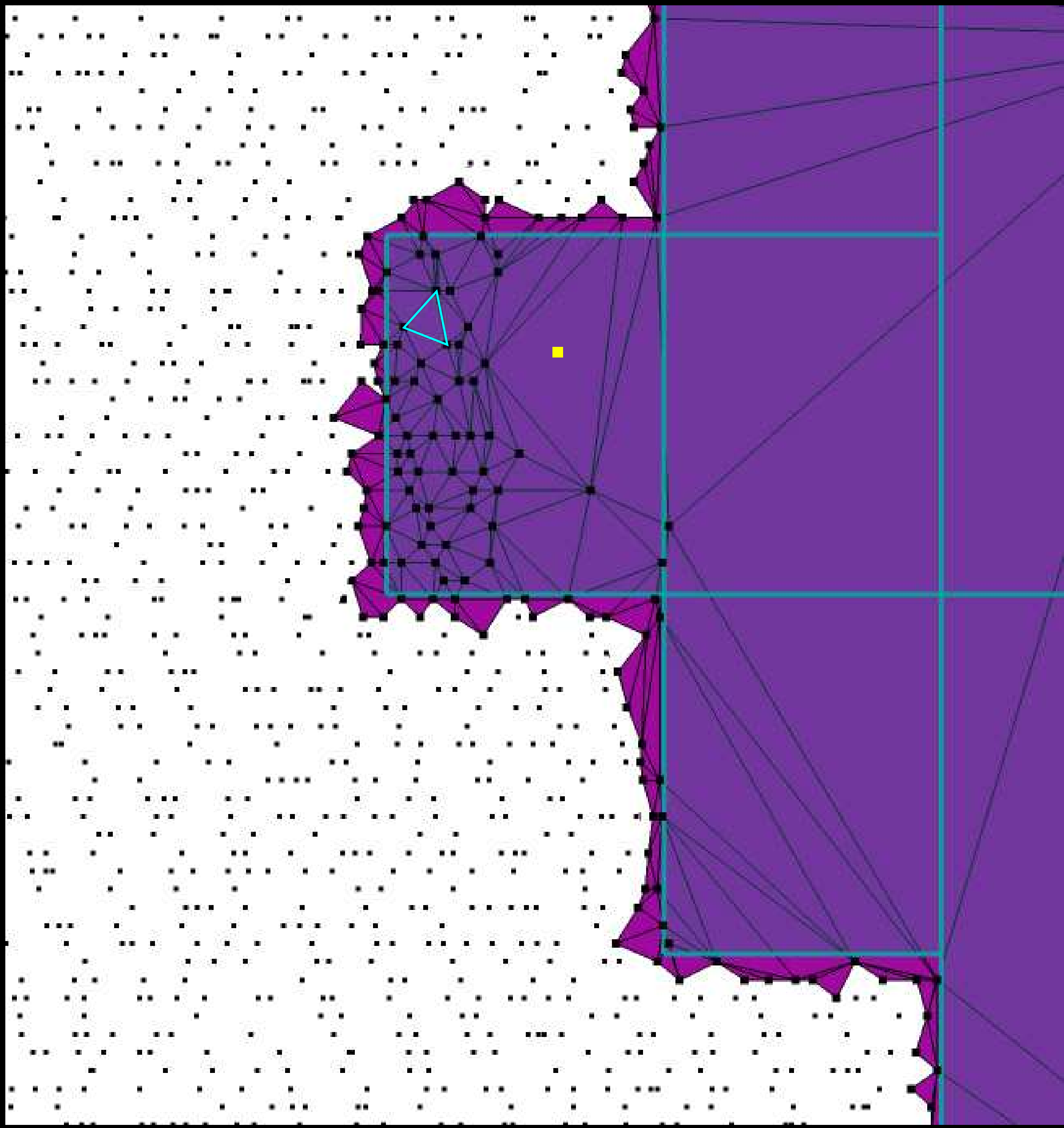
When tag arrives:
- Identify final triangles.
- Write them out & free their memory.



Streaming Delaunay Triangulation

When point arrives:

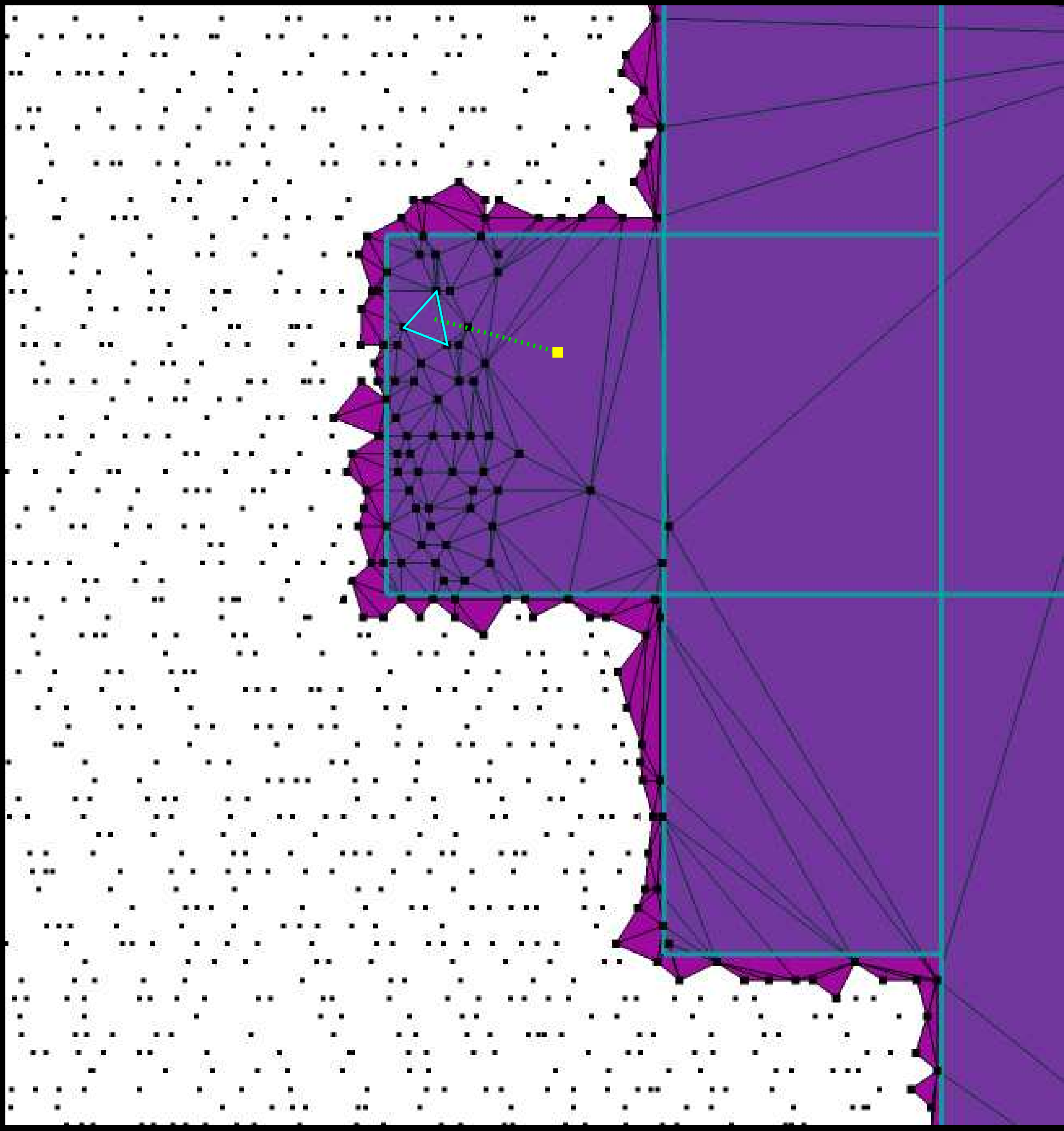
- Locate enclosing triangle.
  - Start at newest triangle.



Streaming Delaunay Triangulation

When point arrives:

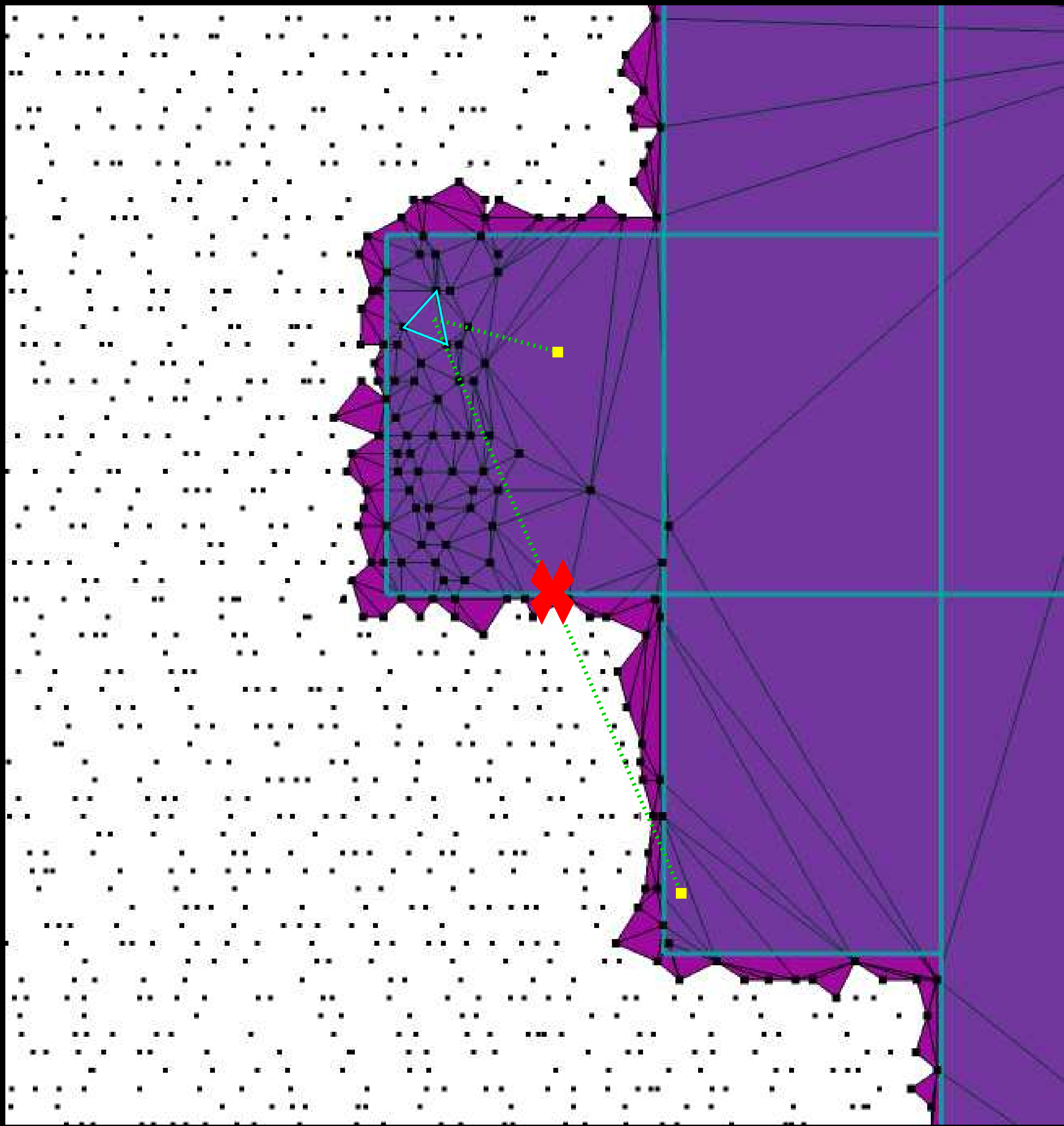
- Locate enclosing triangle.
  - Start at newest triangle.
  - Walk to point.



Streaming Delaunay Triangulation

When point arrives:

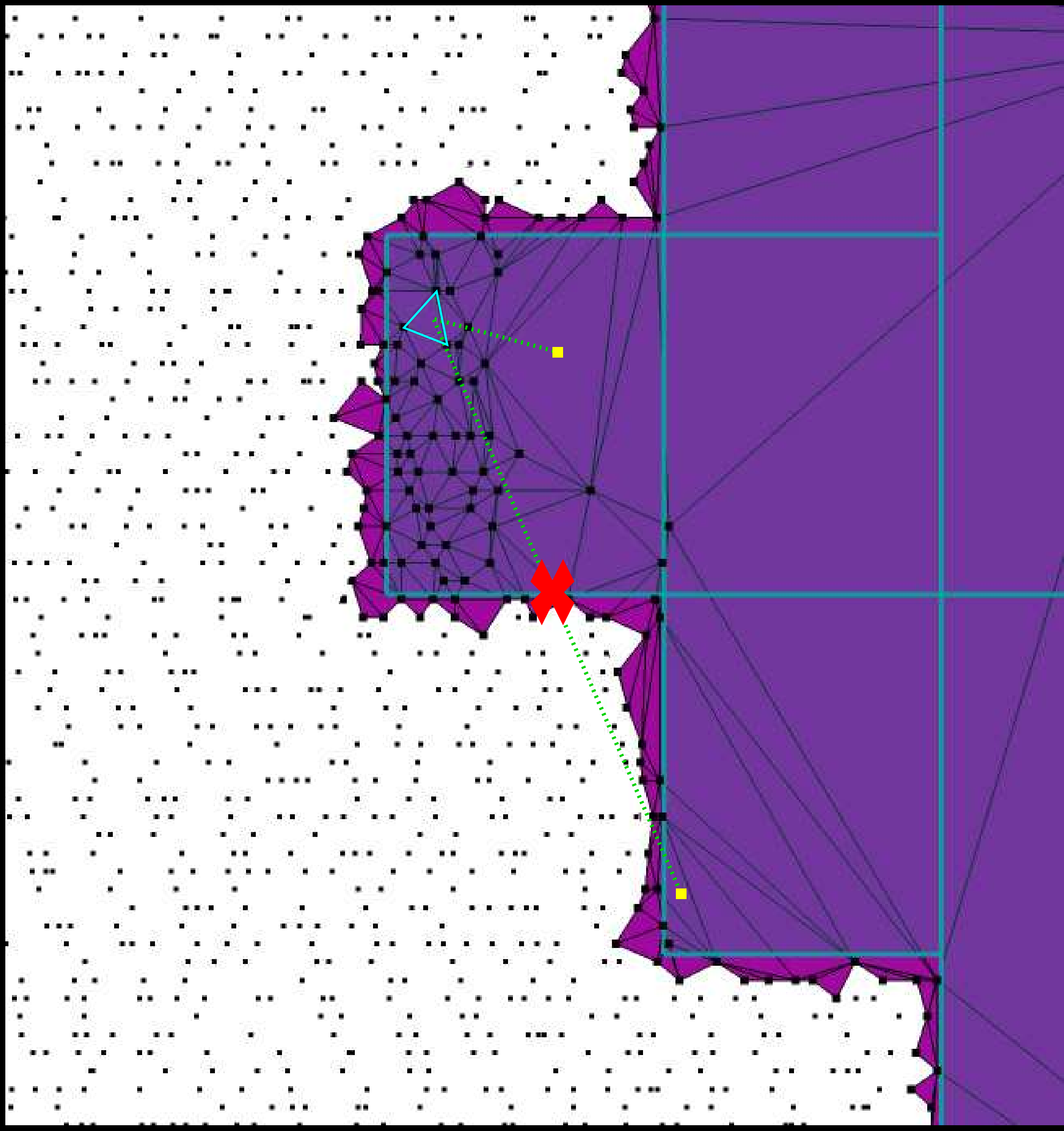
- Locate enclosing triangle.
  - Start at newest triangle.
  - Walk to point.



Streaming Delaunay Triangulation

When point arrives:

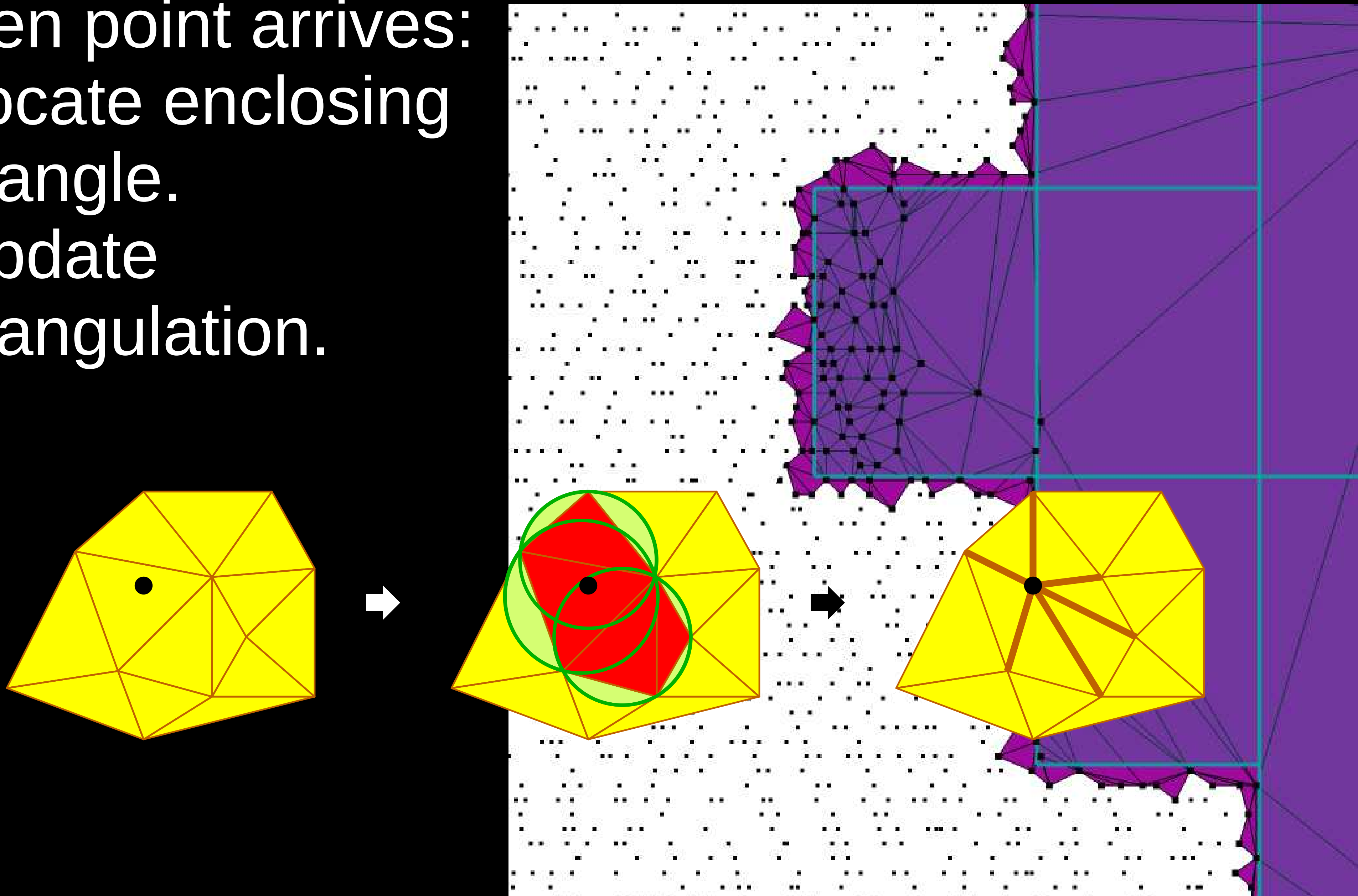
- Locate enclosing triangle.
  - Start at newest triangle.
  - Walk to point.
  - If that fails, restart from cell triangle.
  - If that fails, do exhaustive search.



Streaming Delaunay Triangulation

When point arrives:

- Locate enclosing triangle.
- Update triangulation.



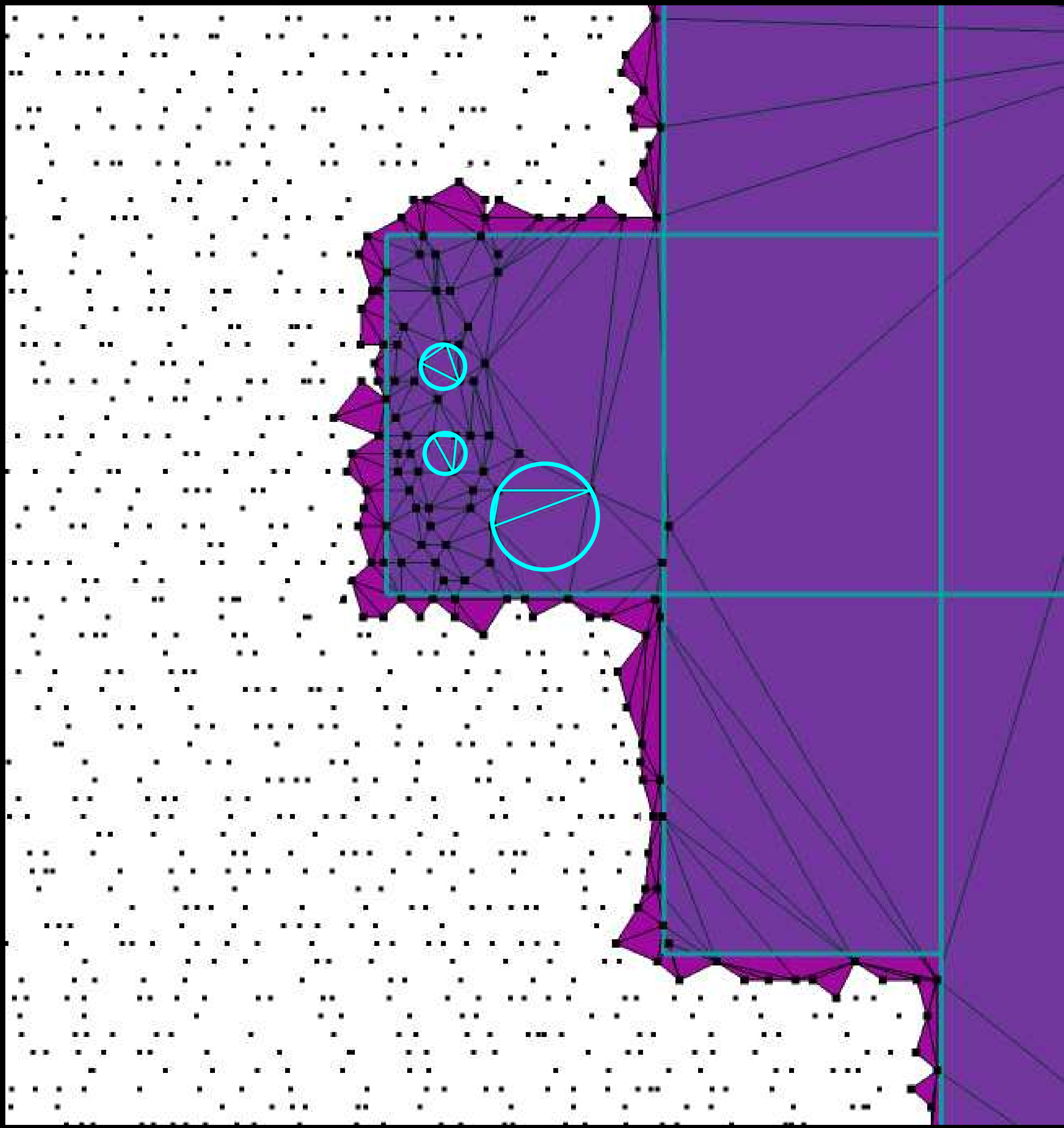
Streaming Delaunay Triangulation

When tag arrives:

- Identify final triangles.

Only these triangles need to be checked:

- New triangles (since last tag).
- Triangles waiting for this tag.

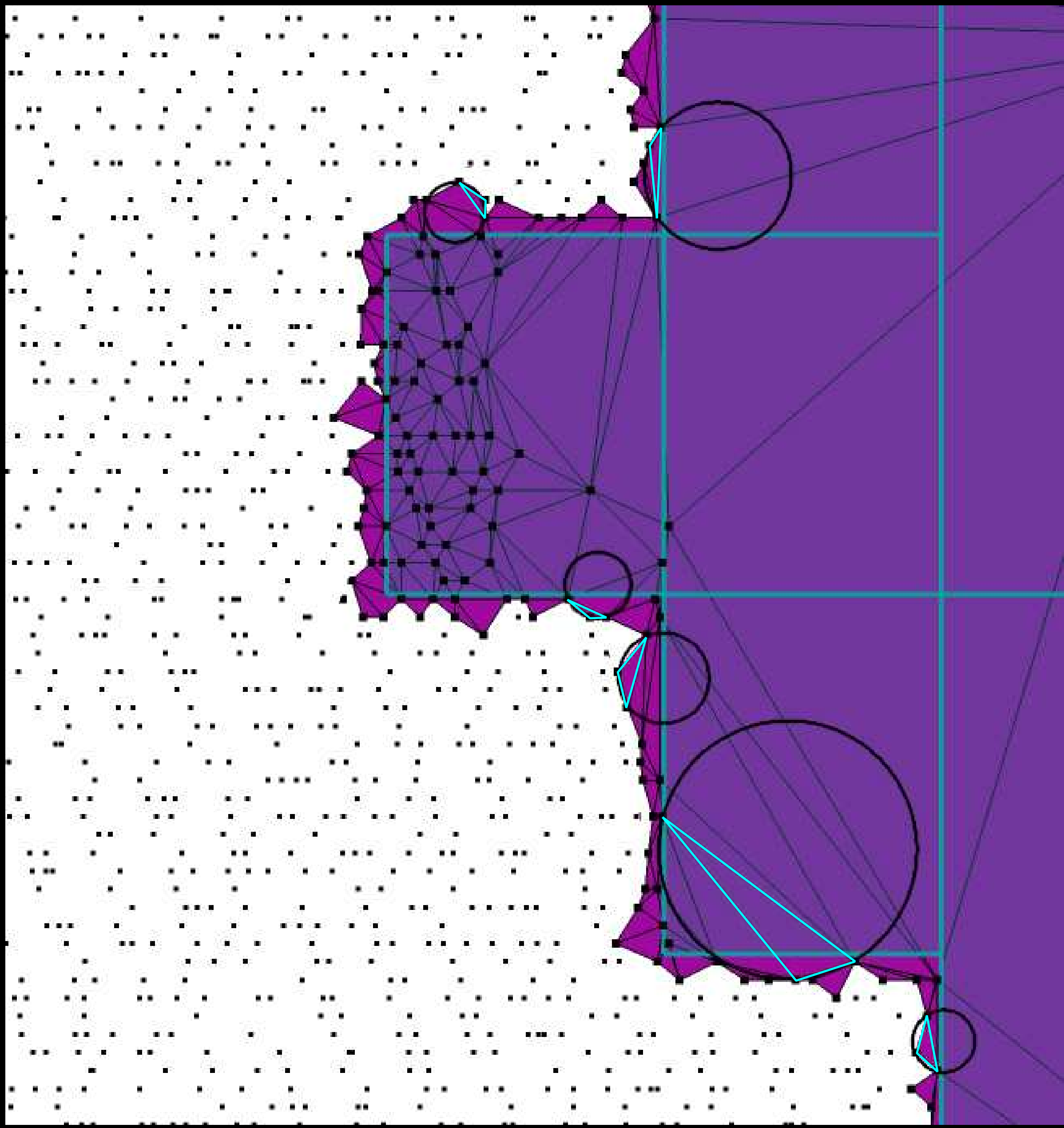


Streaming Delaunay Triangulation

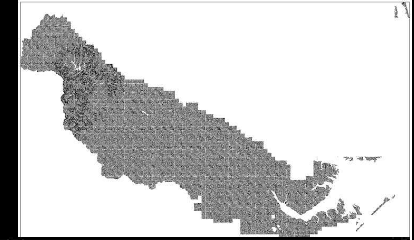
When tag arrives:

- Identify final triangles.

Checked triangles that aren't final are assigned a tag to wait for.

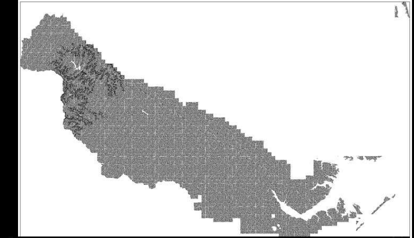


Results



finalized input points			spdelaunay2d				output mesh
name	# of points file size	options	max active triangles	h:mm:ss disk	h:mm:ss pipe	MB	# of triangles file size
neuse	500,141,313	l_8e_4	148,198	39:24	40:53	20	(single)
		l_9e_4	75,705	37:14	38:37	10	1,000,282,528
(double)	11.2 GB	$l_{10}e_4$	60,026	35:15	35:18	11	16.9 GB

Results



finalized input points			spdelaunay2d				output mesh
name	# of points file size	options	max active triangles	h:mm:ss disk	h:mm:ss pipe	MB	# of triangles file size
neuse	500,141,313	l_8e_4	148,198	39:24	40:53	20	(single)
(double)	11.2 GB	l_9e_4	75,705	37:14	38:37	10	1,000,282,528
		$l_{10}e_4$	60,026	35:15	35:18	11	16.9 GB

input points	depth (k)	time/pass, m:ss 1	2	3	MB	points buffered	occu'd cells	points per cell avg	max
neuse	8	5:56	5:56	8:32	94	3,843,621	19,892	25,142	66,171
500M pts	9	5:56	5:56	7:55	61	2,255,569	77,721	6,435	20,208
11.2 GB	10	5:56	5:56	6:49	61	1,518,675	306,334	1,632	6,544

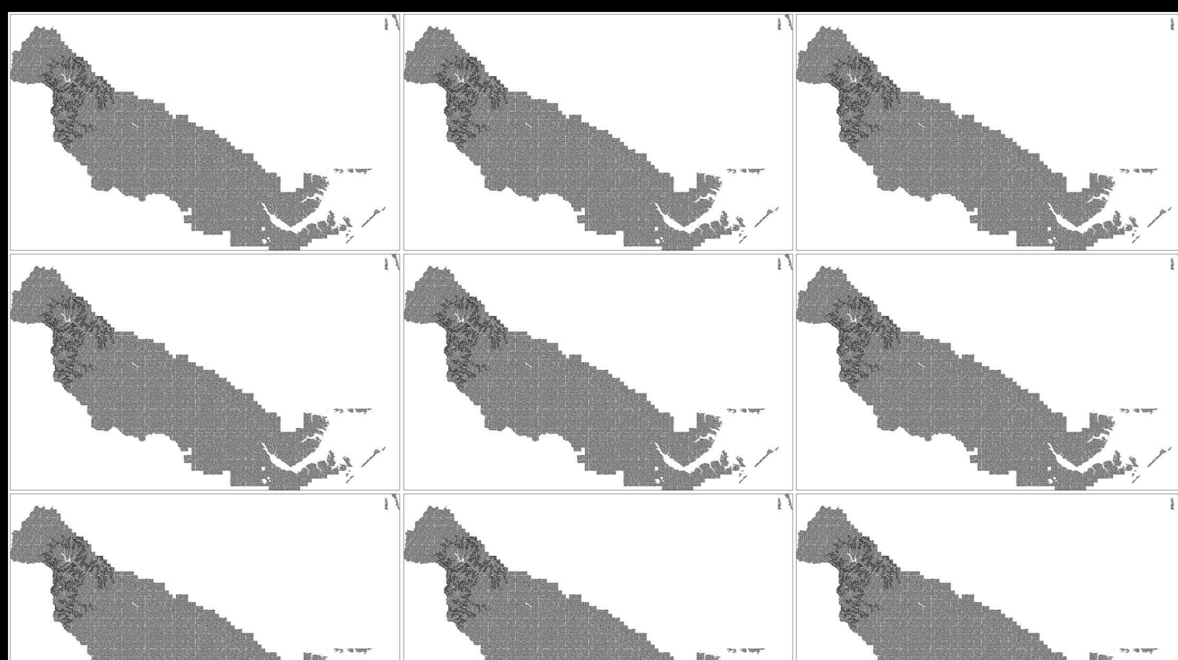
- Us: 12 min finalize + 36 min triang. = 48 min.
- Agarwal−Arge−Yi:
  3 hr sort + 7.5 hr triang. = 10.5 hr.

Results

finalized input points			spdelaunay2d				output mesh
name	# of points file size	op- tions	max active triangles	h:mm:ss disk	h:mm:ss pipe	MB	# of triangles file size
neuse	4,501,271,817	$l_{10}e_5$	114,894	– : –	5:29:02	15	(single)
3 × 3		$l_{11}e_5$	72,732	– : –	4:58:16	11	9,002,543,628
(double)	101 GB	$l_{12}e_5$	78,475	– : –	4:37:21	11	152 GB

107 min finalize + 278 min triang. = 6 hr 25 min.

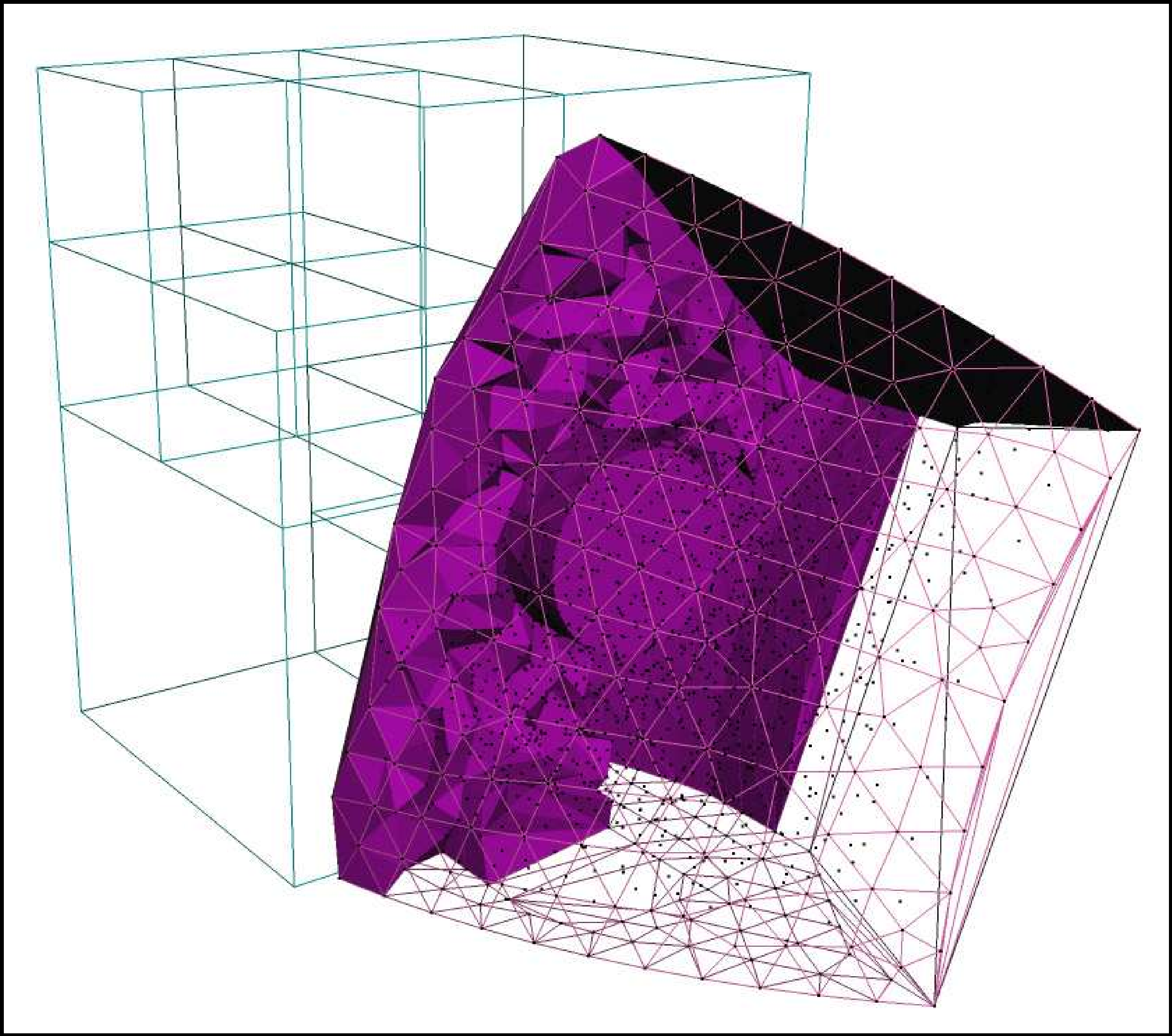
425 MB finalize + 11 MB triang. = 436 MB.



3D

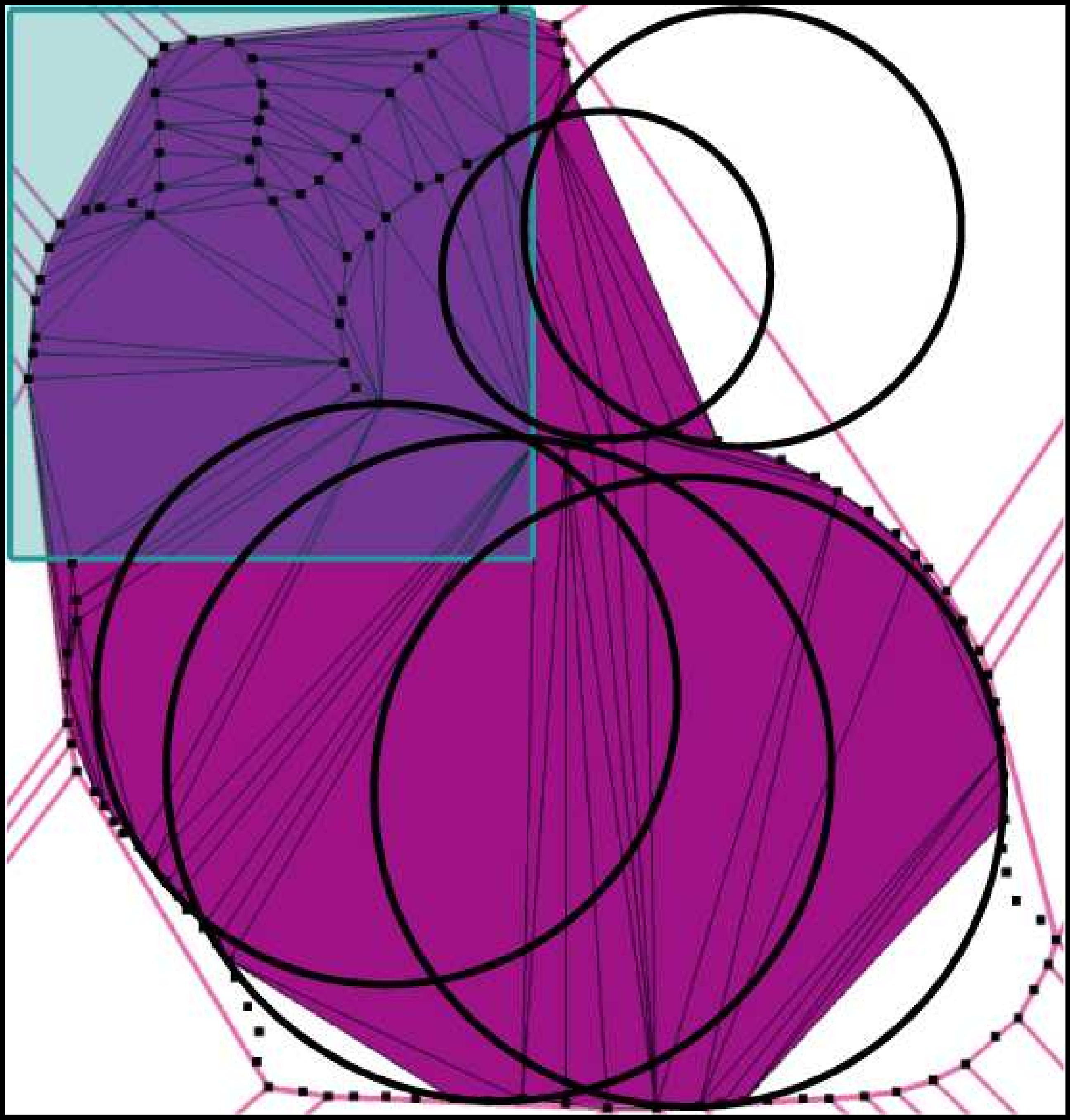
3D Streaming Delaunay

815 million tetrahedra in 2:41 hrs & 795 MB. (Pre−finalized points.)



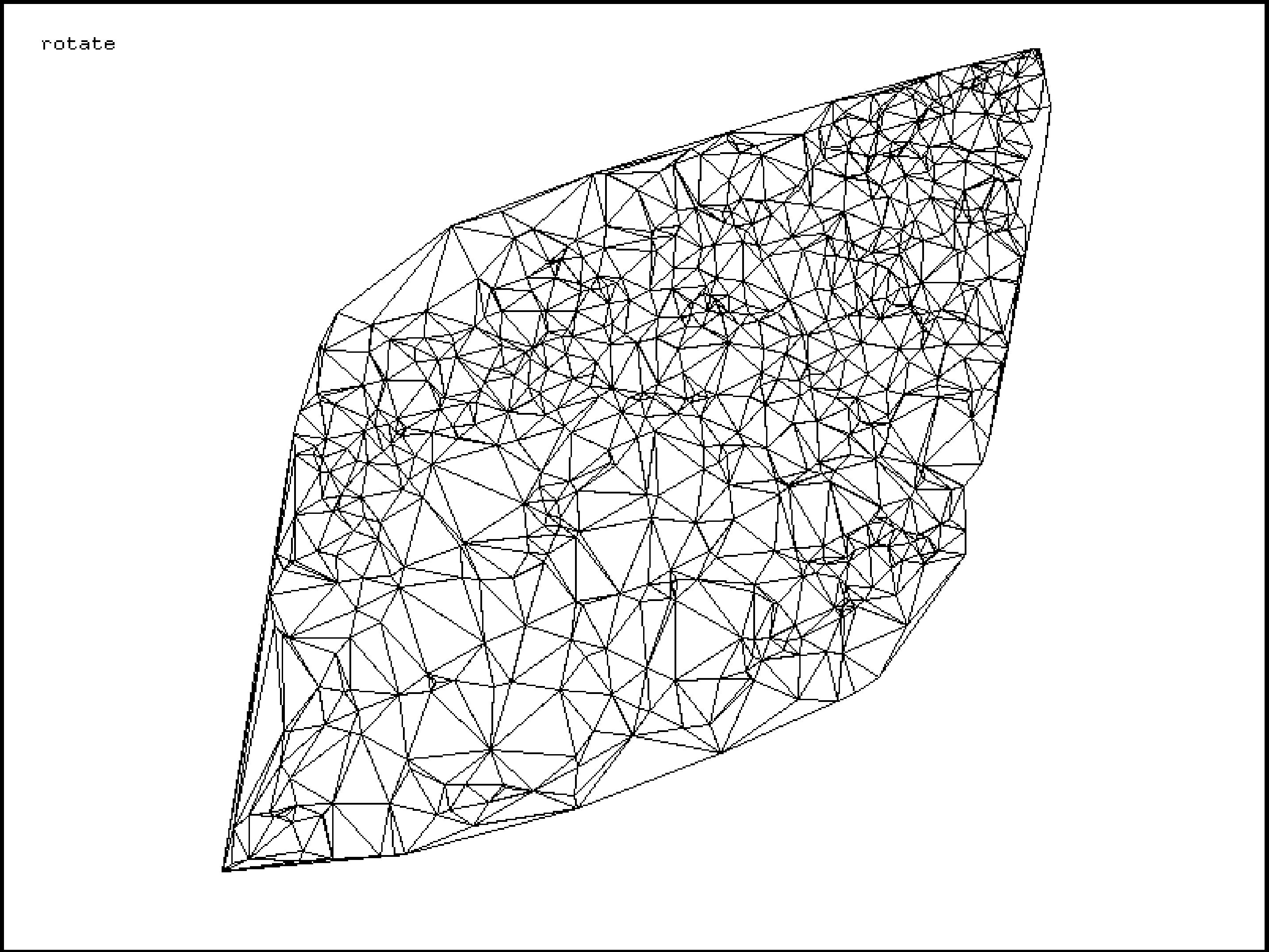
3D Streaming Delaunay

Does not work for surface scans. Circumspheres are too big.



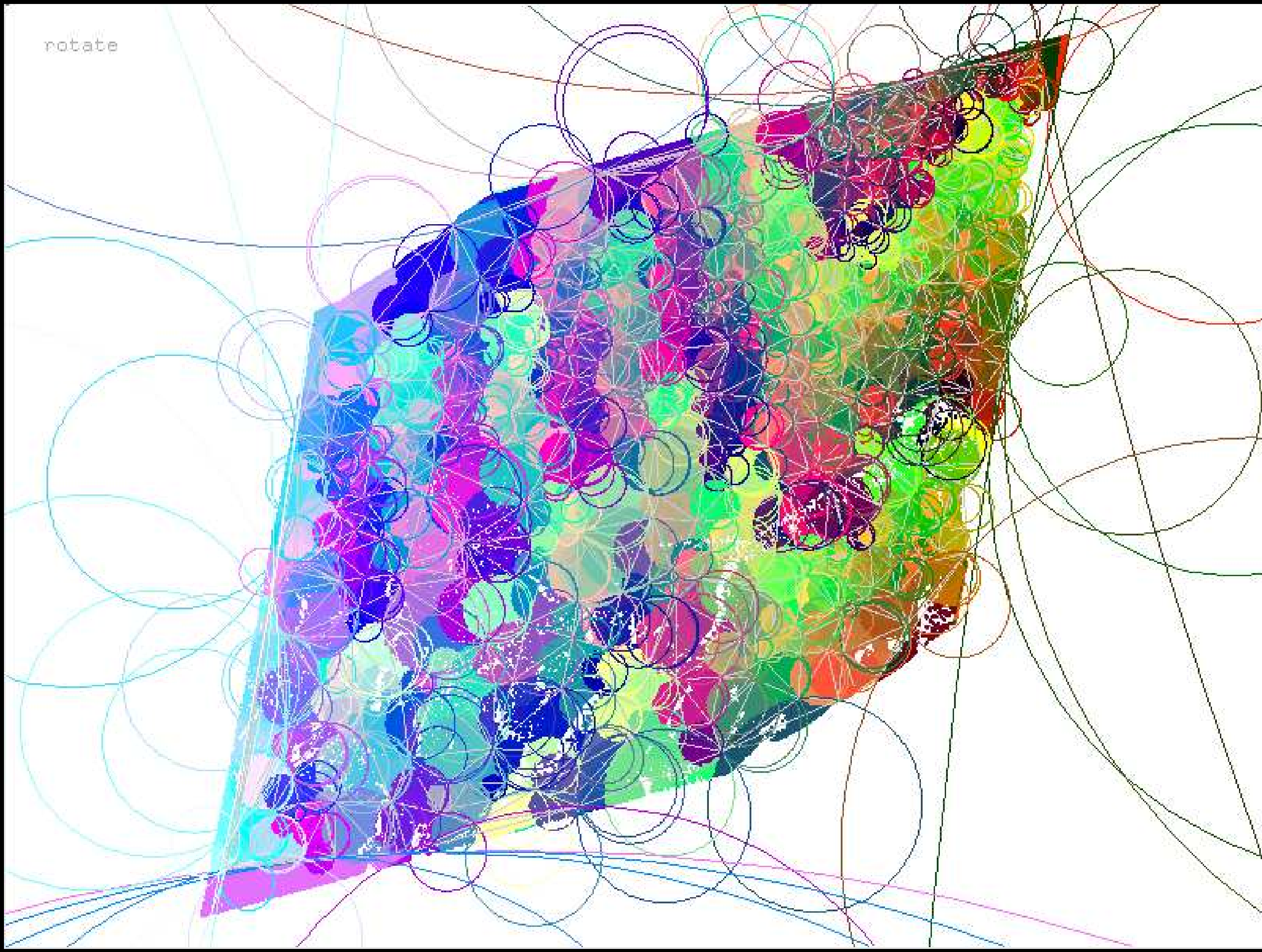
3D Streaming Delaunay

Future work: spherical finalization regions from Delaunay triangulation of a random sample.



3D Streaming Delaunay

Future work: spherical finalization regions from Delaunay triangulation of a random sample.



Conclusions

Conclusions

- Most real−world points sets have lots of spatial coherence → sorting doesn’t help.

Conclusions

- Most real−world points sets have lots of spatial coherence → sorting doesn't help.
- Like aikido: Use your opponent's spatial coherence against him.

Conclusions

- Most real−world points sets have lots of spatial coherence → sorting doesn't help.
- Like aikido: Use your opponent's spatial coherence against him.
- Not an external memory algorithm!
  No temporary storage to disk.

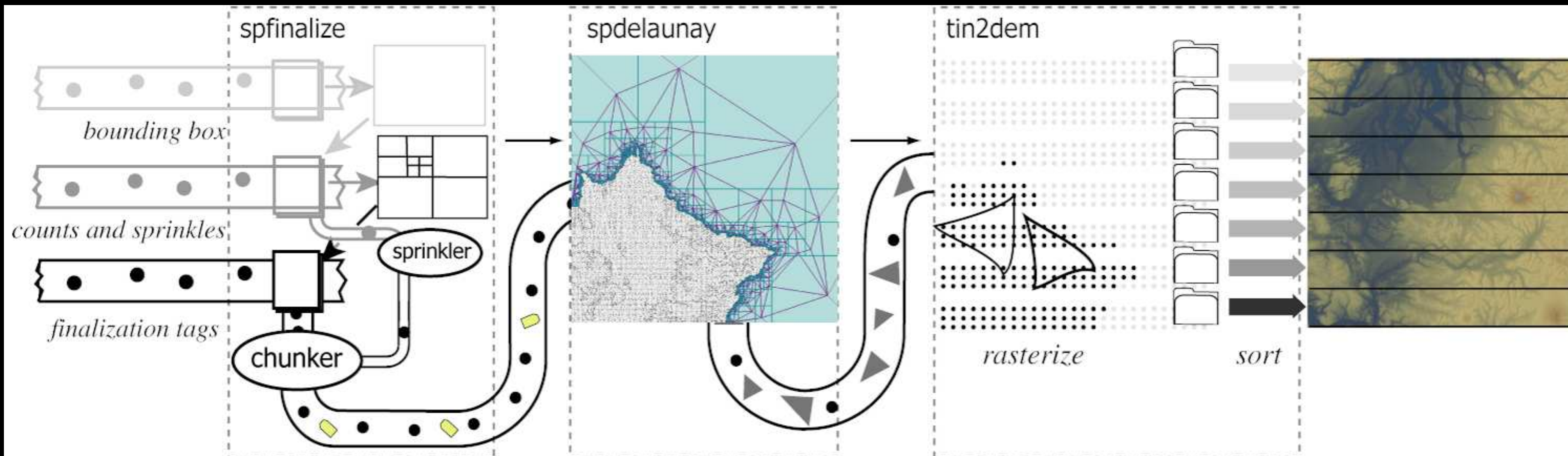
Conclusions

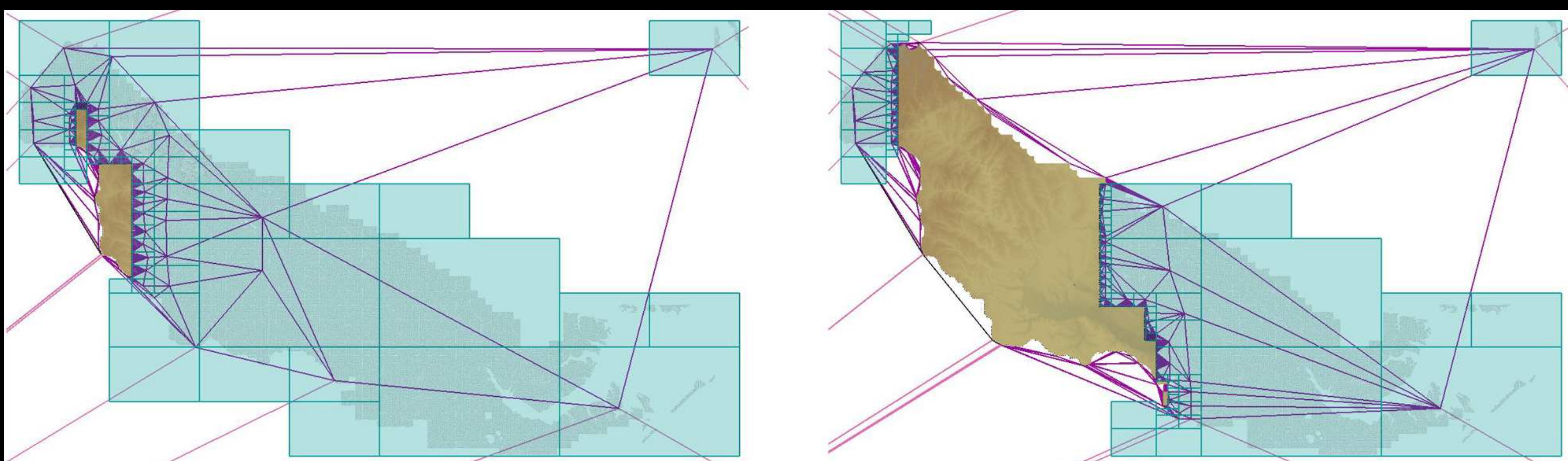
- Most real−world points sets have lots of spatial coherence → sorting doesn't help.
- Like aikido: Use your opponent's spatial coherence against him.
- Not an external memory algorithm! No temporary storage to disk.
- 12 times faster than best previous 2D approach.

Conclusions

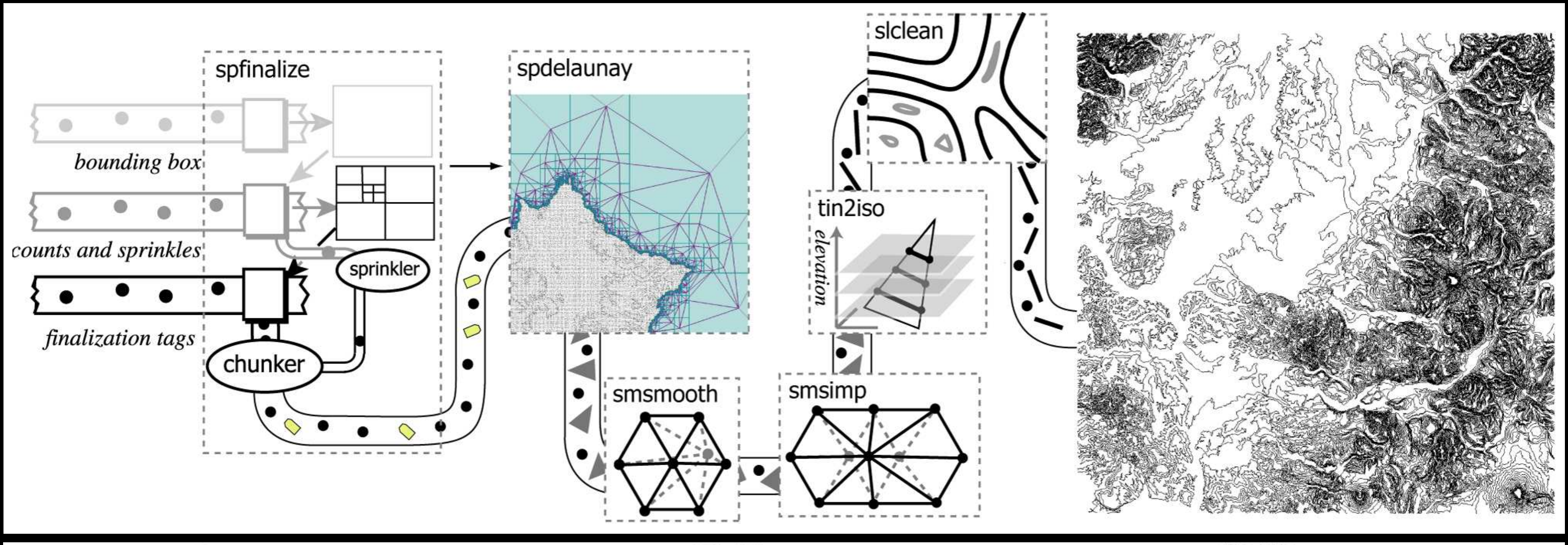
- Most real−world points sets have lots of spatial coherence → sorting doesn’t help.
- Like aikido: Use your opponent’s spatial coherence against him.
- Not an external memory algorithm!
  No temporary storage to disk.
- 12 times faster than best previous 2D approach.
- Streaming triangulation can be piped directly to another streaming application.

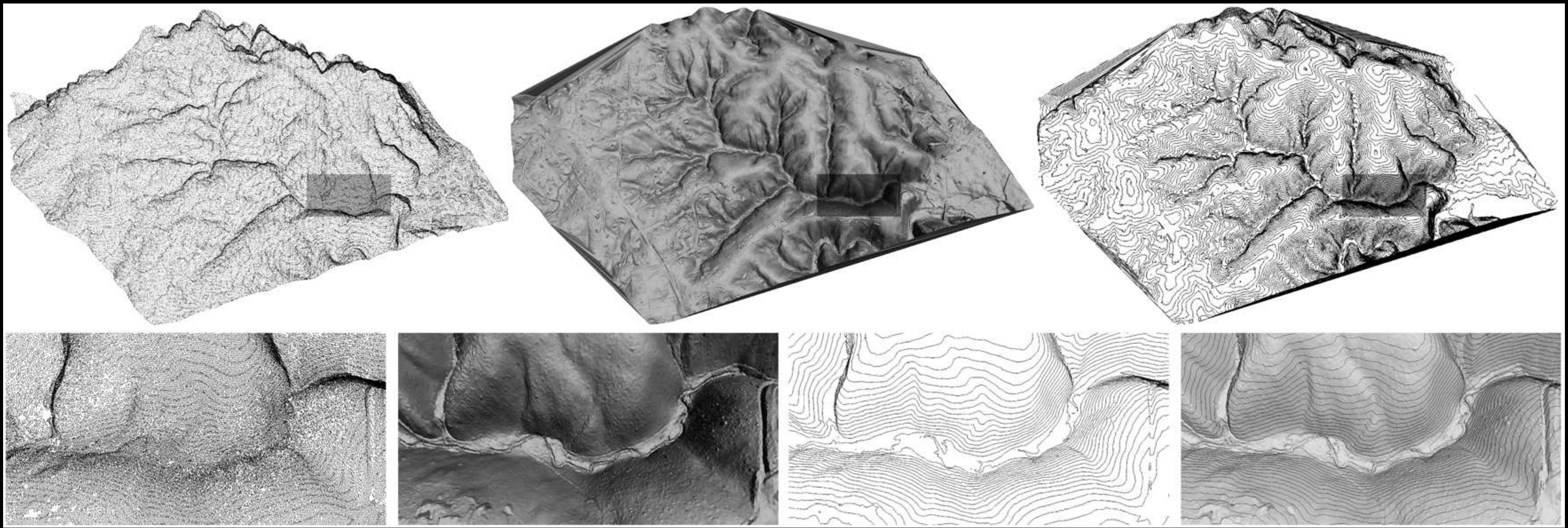
Streaming Digital Elevation Maps





Streaming Contour Extraction





Thanks

- Kevin Yi at Duke supplied the Neuse River Basin data.
- Martin Isenburg & Yuanxin “Leo” Liu did most of the programming.