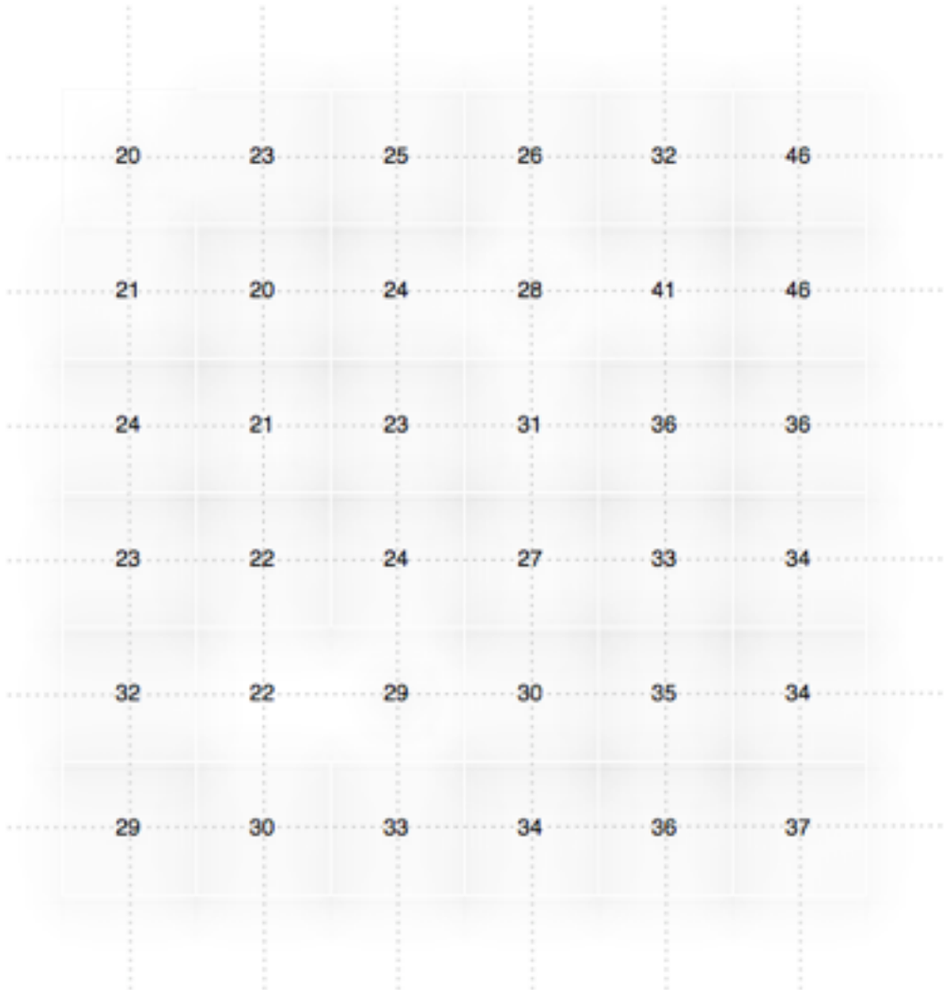
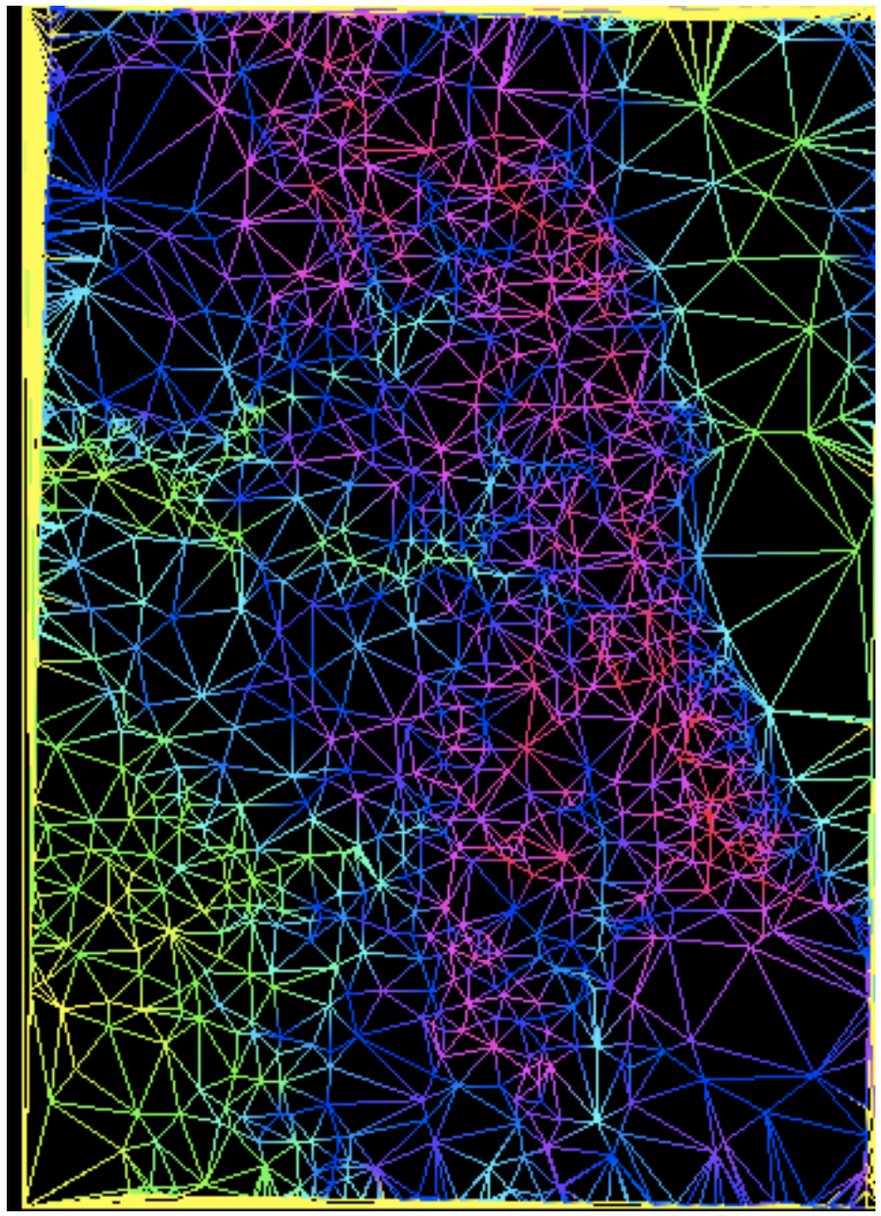
Algorithms for GIS:

Terrain simplification

Digital terrain models in GIS



grid (raster)



TIN

Data Sources: digitizing contour maps

Data Sources: satellite imagery

Data Sources: satellite imagery

Data Sources: satellite imagery

Data Sources: satellite imagery

Data Sources: satellite imagery

Data Sources: satellite imagery

Data Sources: satellite imagery

The resolution of satellite images varies depending on the instrument used and the altitude of the satellite's orbit. For example, the Landsat archive offers repeated imagery at 30 meter resolution for the planet, but most of it has not been processed from the raw data. Landsat 7 has an average return period of 16 days. For many smaller areas, images with resolution as high as 41 cm can be available.[5]

Satellite imagery is sometimes supplemented with aerial photography, which has higher resolution, but is more expensive per square meter. Satellite imagery can be combined with vector or raster data in a GIS provided that the imagery has been spatially rectified so that it will properly align with other data sets.

\

Data Sources: satellite imagery

Imaging satellites e.g.

- GeoEye
  - launched September 6, 2008
  - has the highest resolution imaging system and is able to collect images with a ground resolution of 0.41 meters (16 inches) in the black and white mode. It collects multispectral or color imagery at 1.65-meter resolution or about 64 inches.

- ASTER (Advanced Spaceborne Thermal Emission and Reflection Radiometer)
  - on board NASA's satellite Terra, part of NASA's EOS

satellite imagery

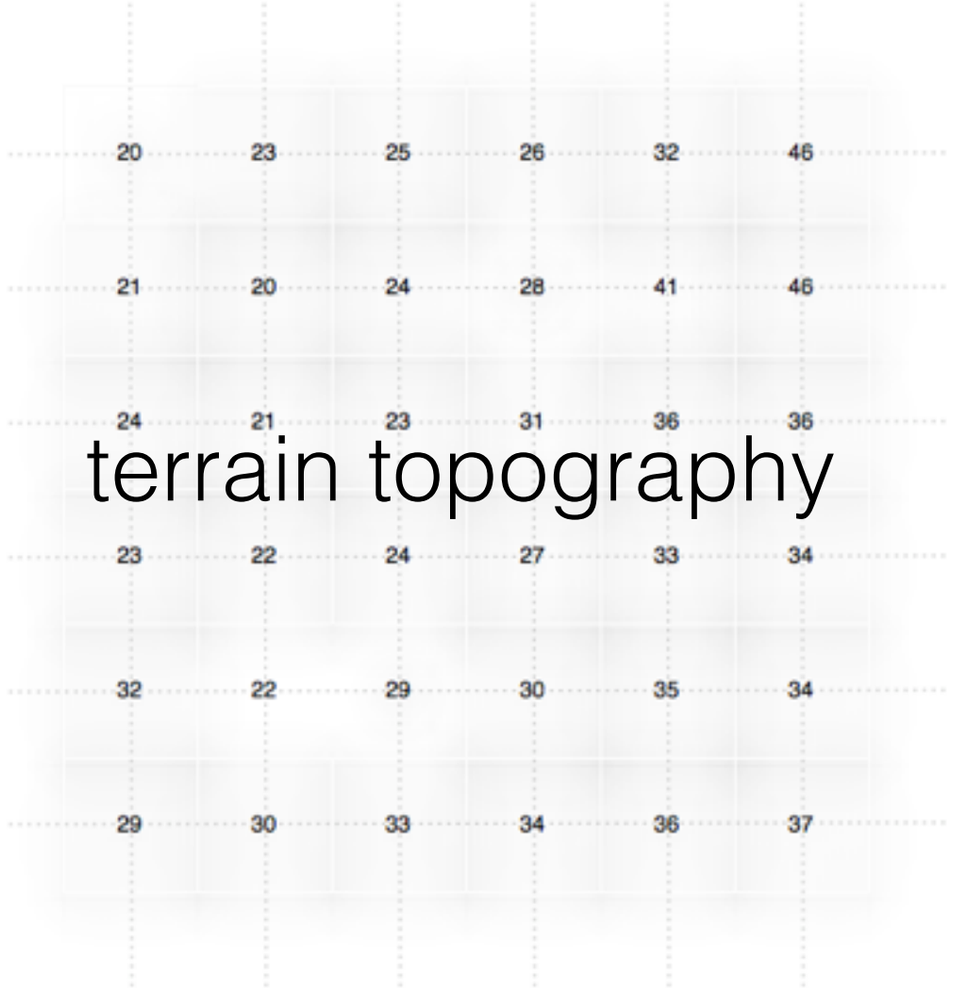
terrain topography

photogrammetry
altimetry
SAR interpherometry
…

e.g. http://www.earsel.org/tutorials/Jac_03DEMGhent_red.pdf

satellite imagery

photogrammetry
altimetry
SAR interpherometry
…

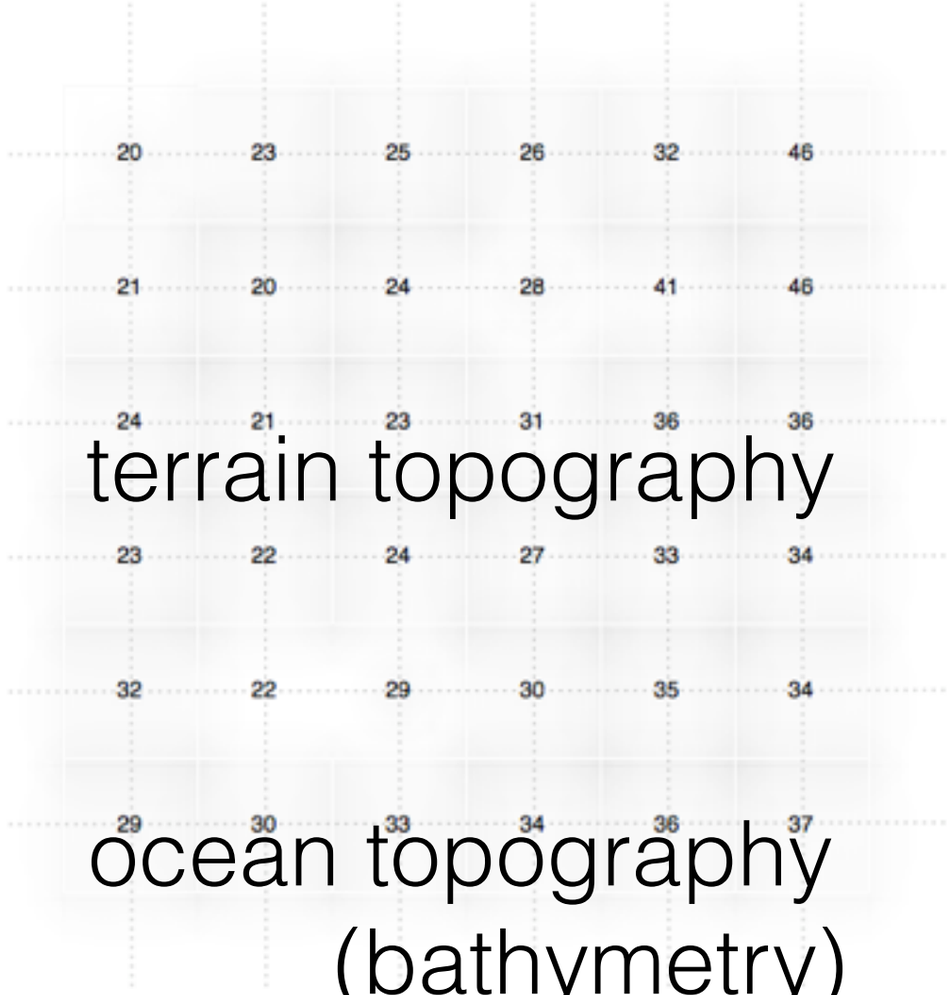


terrain topography

satellite imagery

photogrammetry
altimetry
SAR interpherometry
…

terrain topography



ocean topography
(bathymetry)

Data Sources: satellite imagery

BY SAMANTHA MURPHY KELLY
FEB 28, 2013

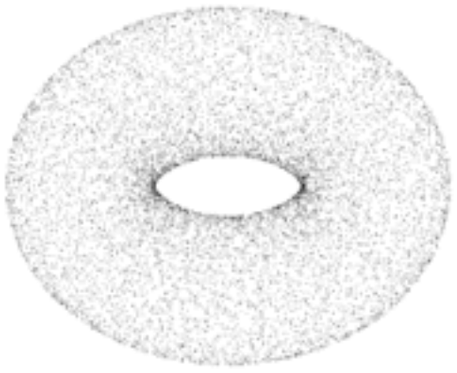
Want to take a look at the depths of the ocean without leaving your chair?

As a part of an effort to better compete with Google Maps, Bing just rolled out a significant update to its Maps platform, adding 13 million square kilometers of updated satellite imagery to its database.

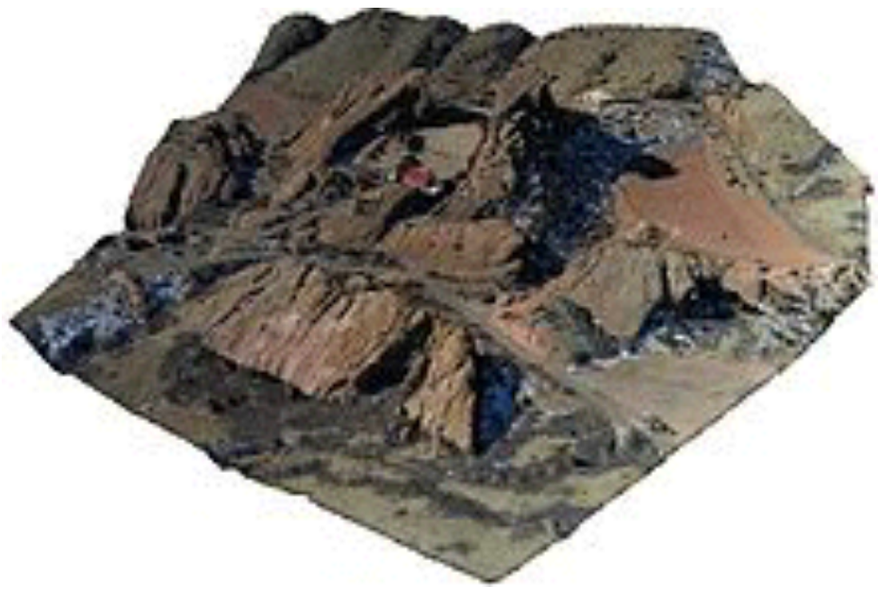
Thanks to satellite imagery provided by TerraColor, the new images' resolution is 15 meters per pixel, providing coverage of the entire world.

Data Sources: (LIDAR) point clouds

- A point cloud is a set of data points in some coordinate system.
  - in 3D: { (x,y,z)}
- Created by 3D scanners or LIDAR/etc
- Many uses:
  - modeling of surfaces and objects
  - visualization, animation
  - medical imaging/computer tomography



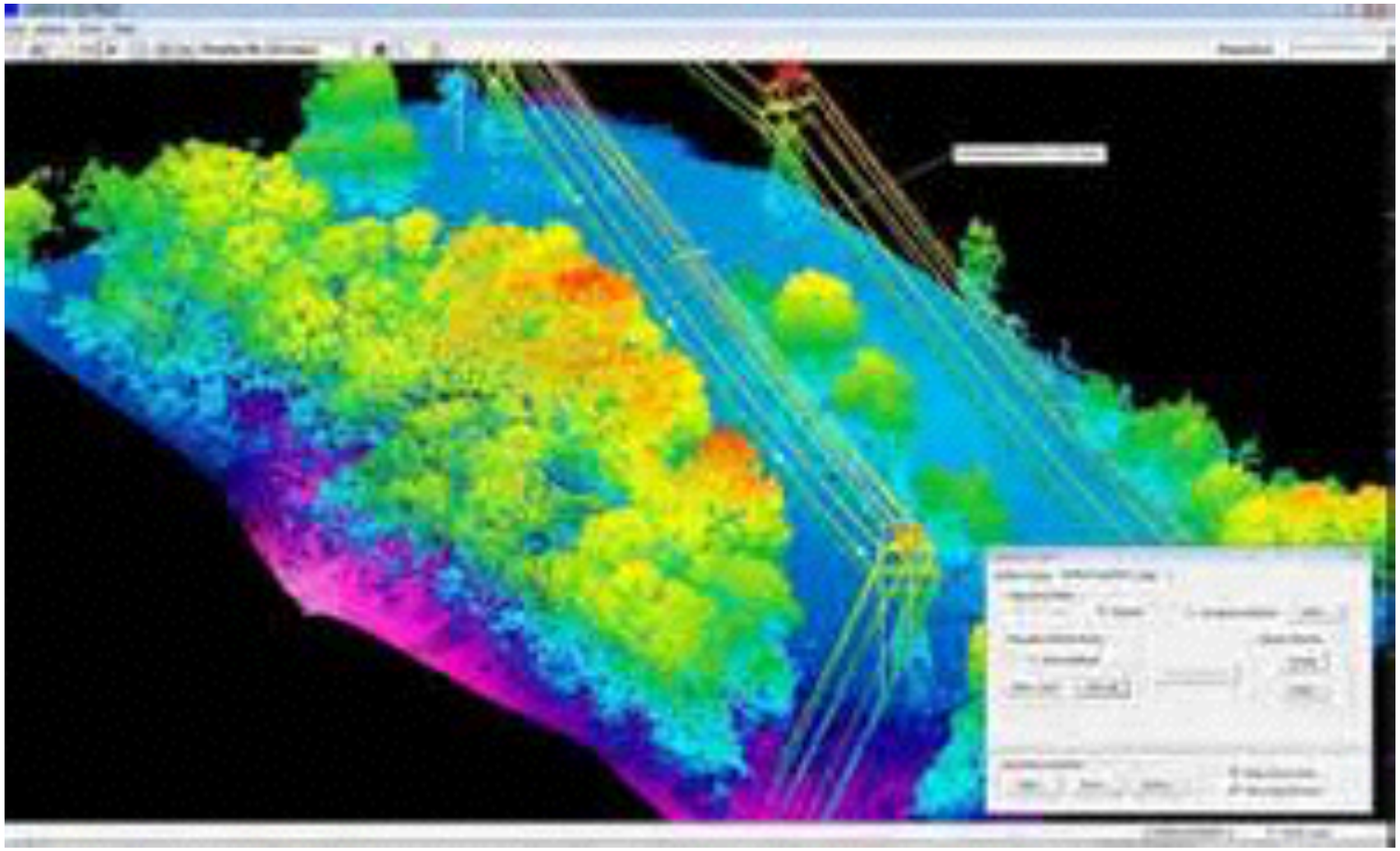
A point cloud image of a torus



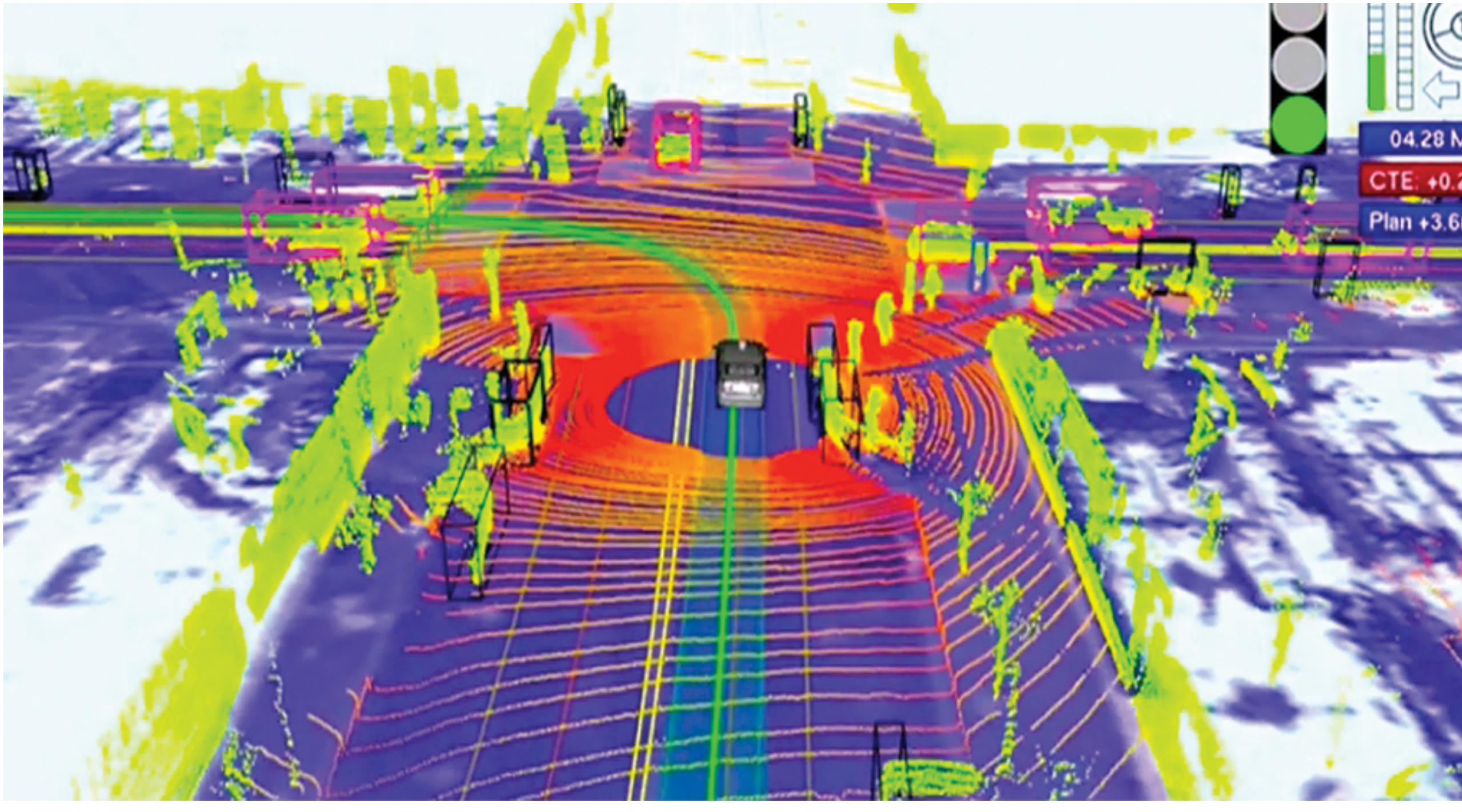
Geo-referenced point cloud of Red Rocks, Colorado (by DroneMapper)

from wiki page: https://en.wikipedia.org/wiki/Point_cloud

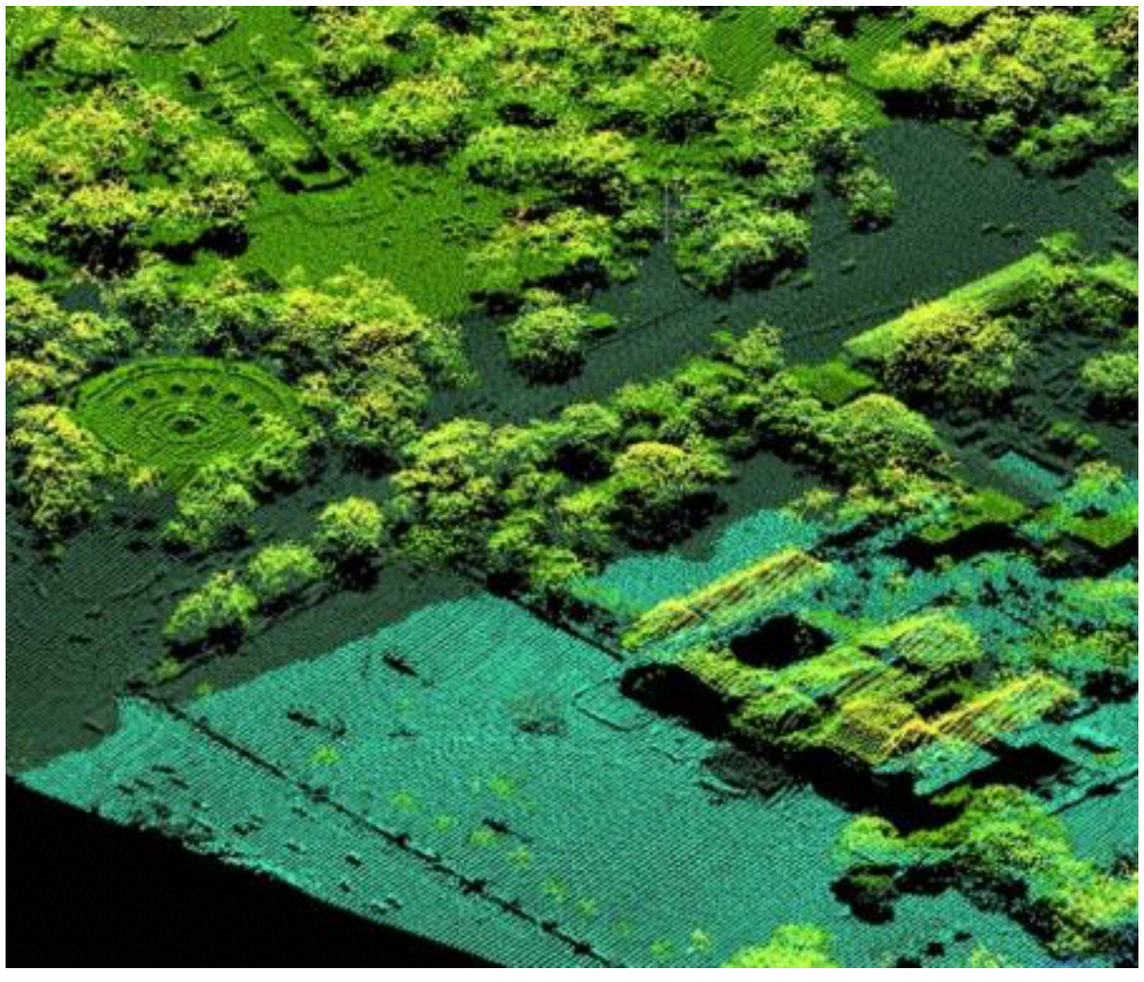
Data Sources: LIDAR point clouds



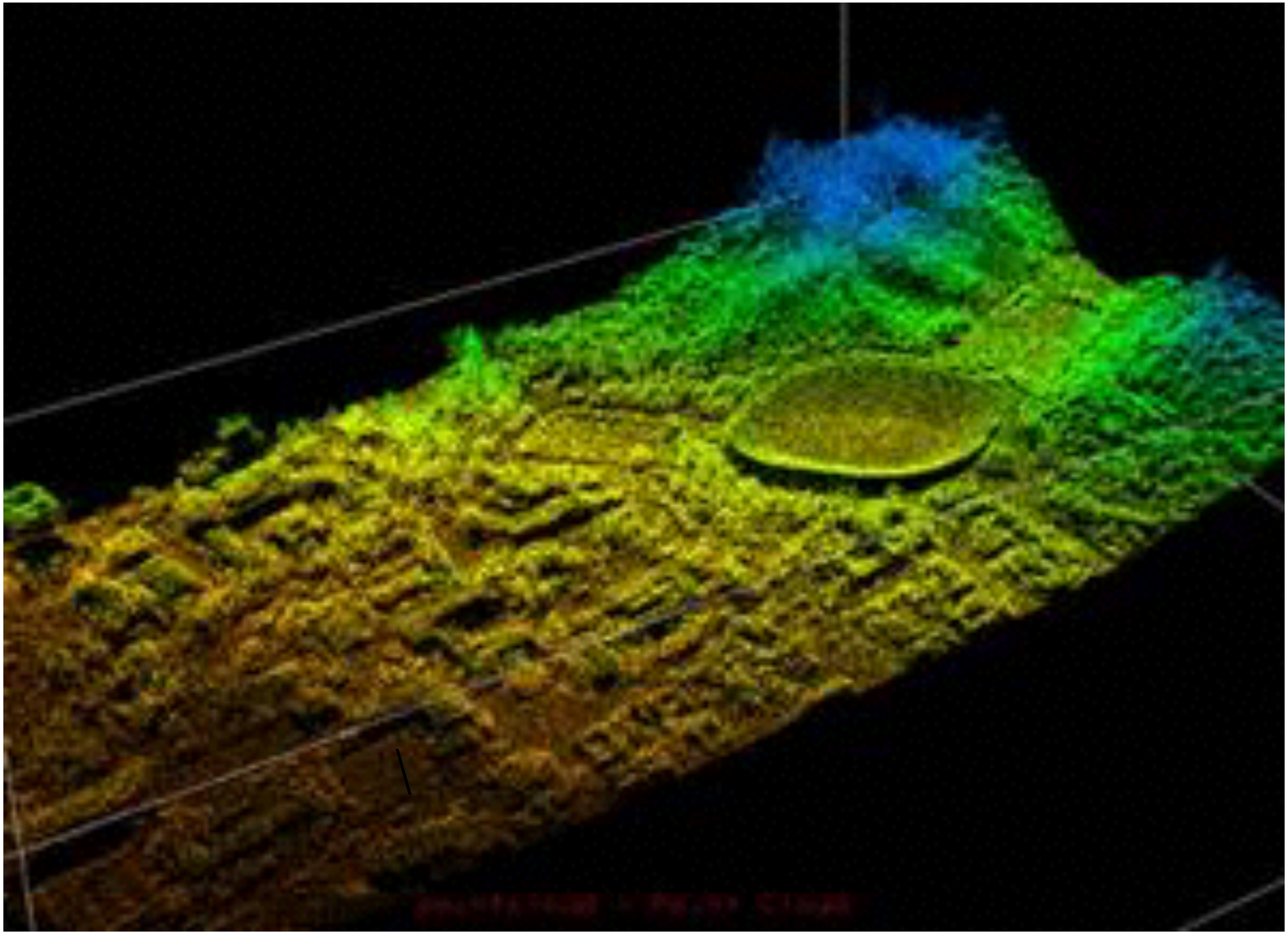
Data Sources: LIDAR point clouds



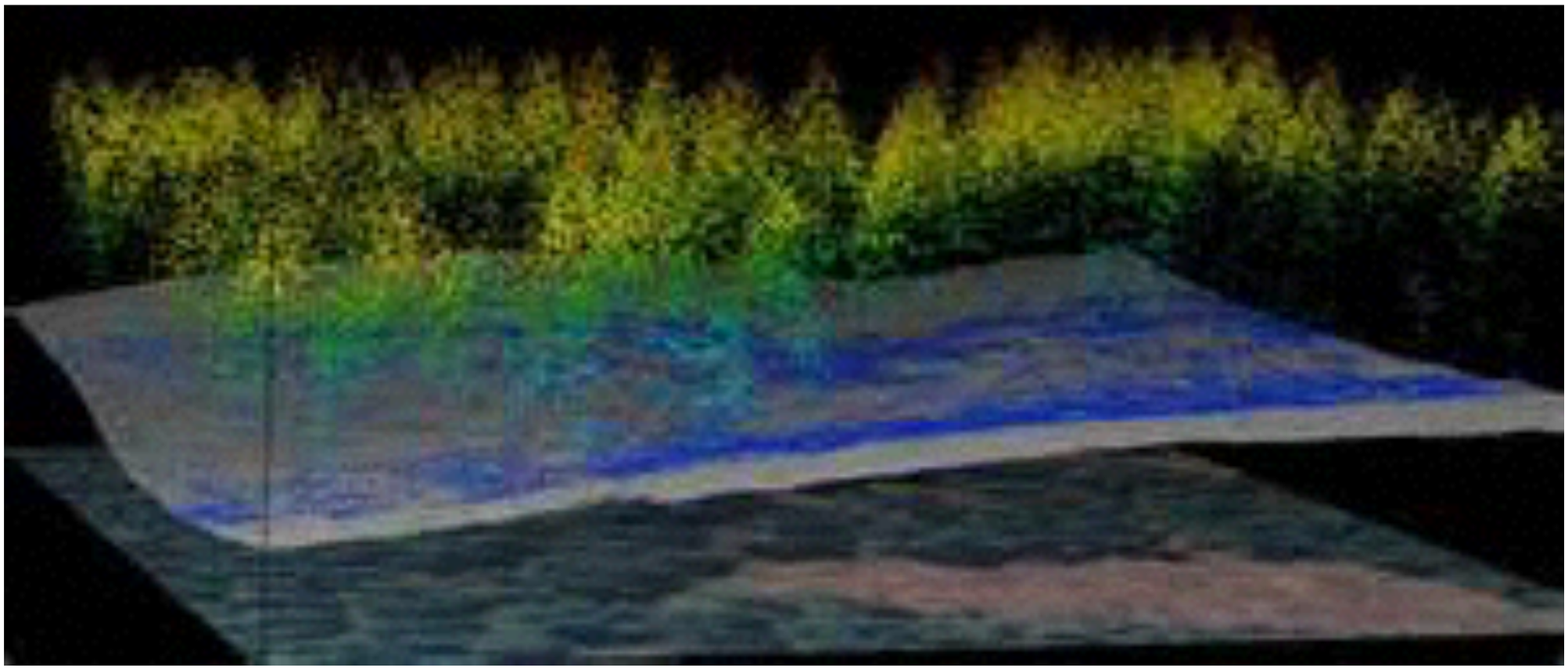
Data Sources: LIDAR point clouds



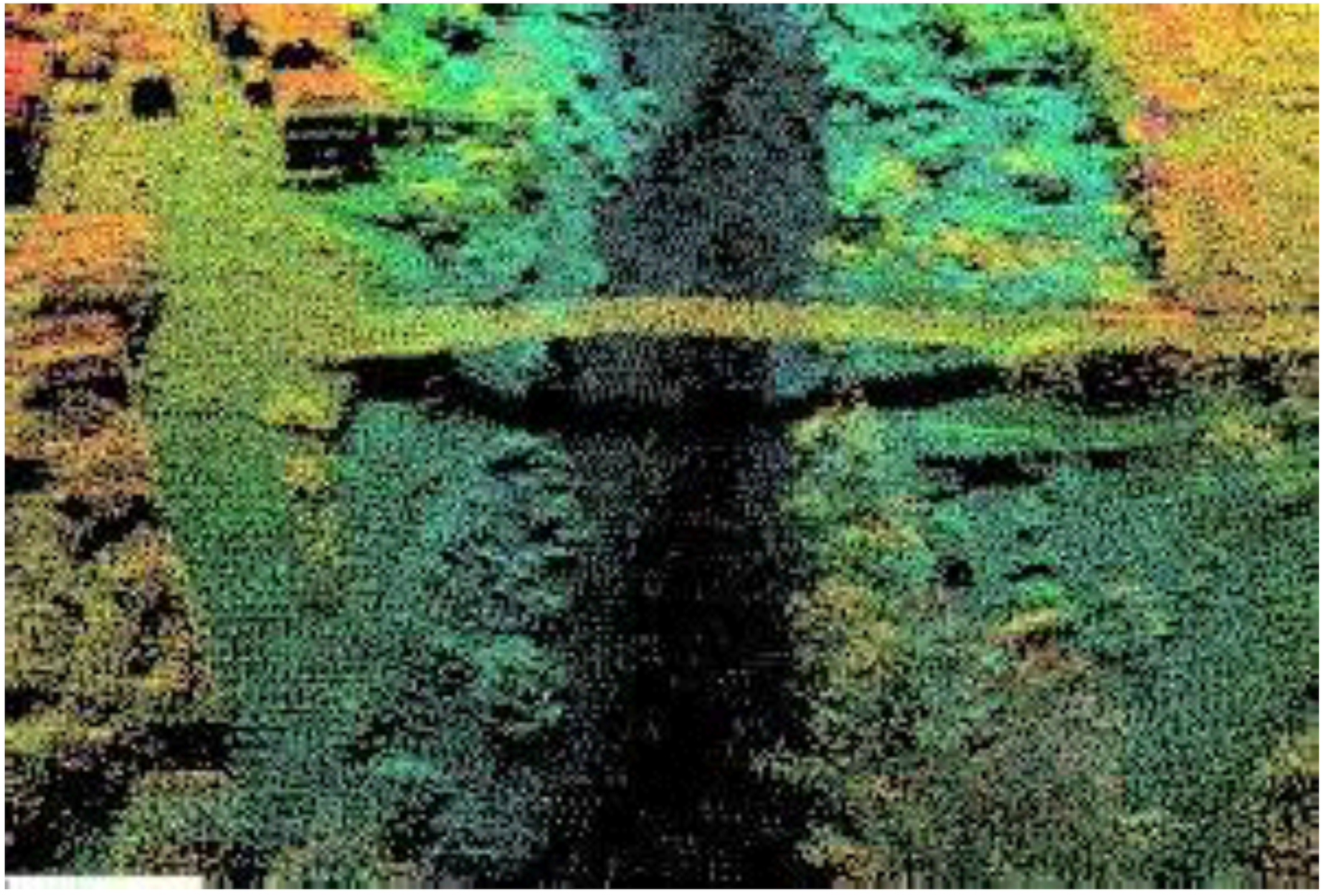
Data Sources: LIDAR point clouds



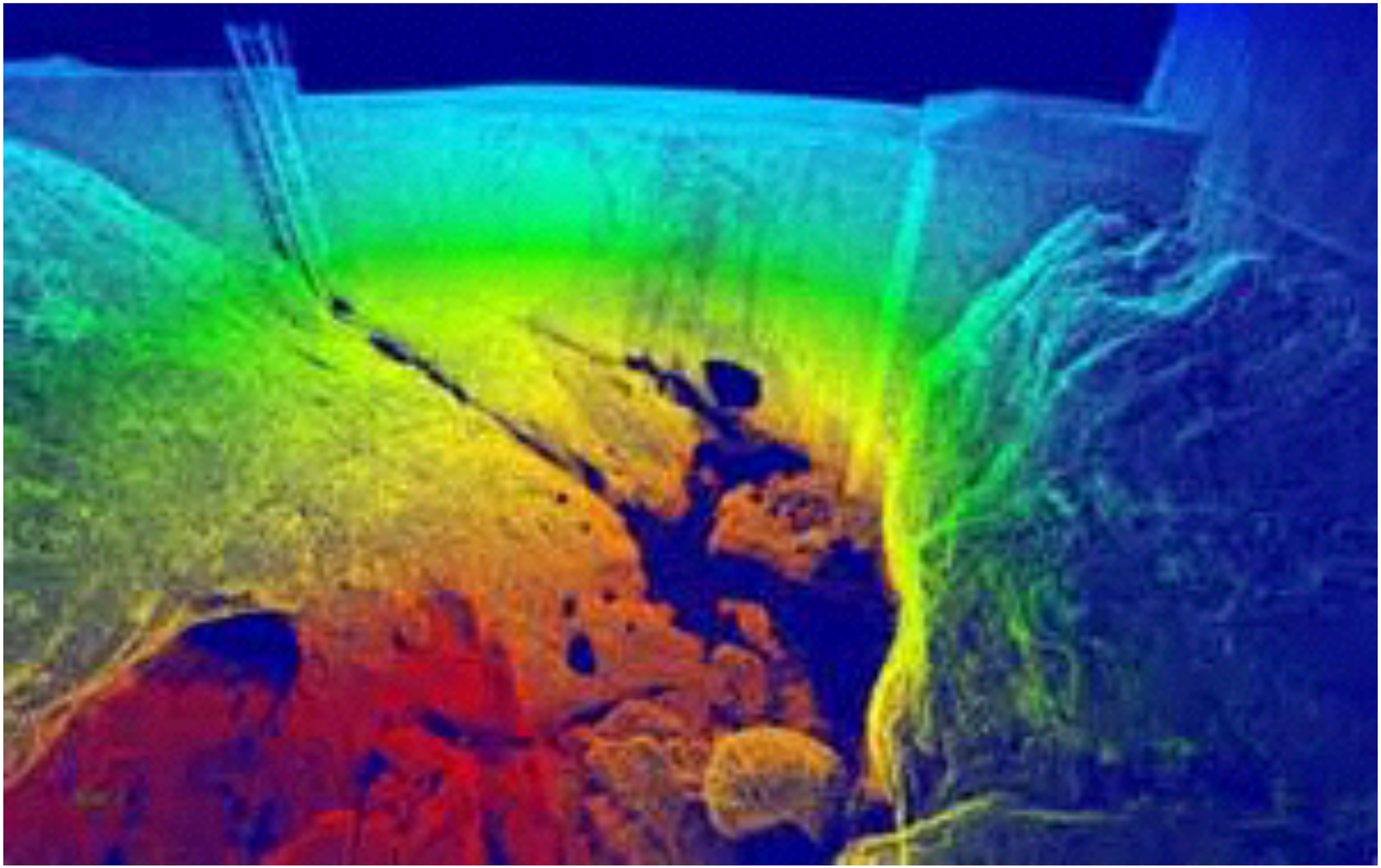
Data Sources: LIDAR point clouds



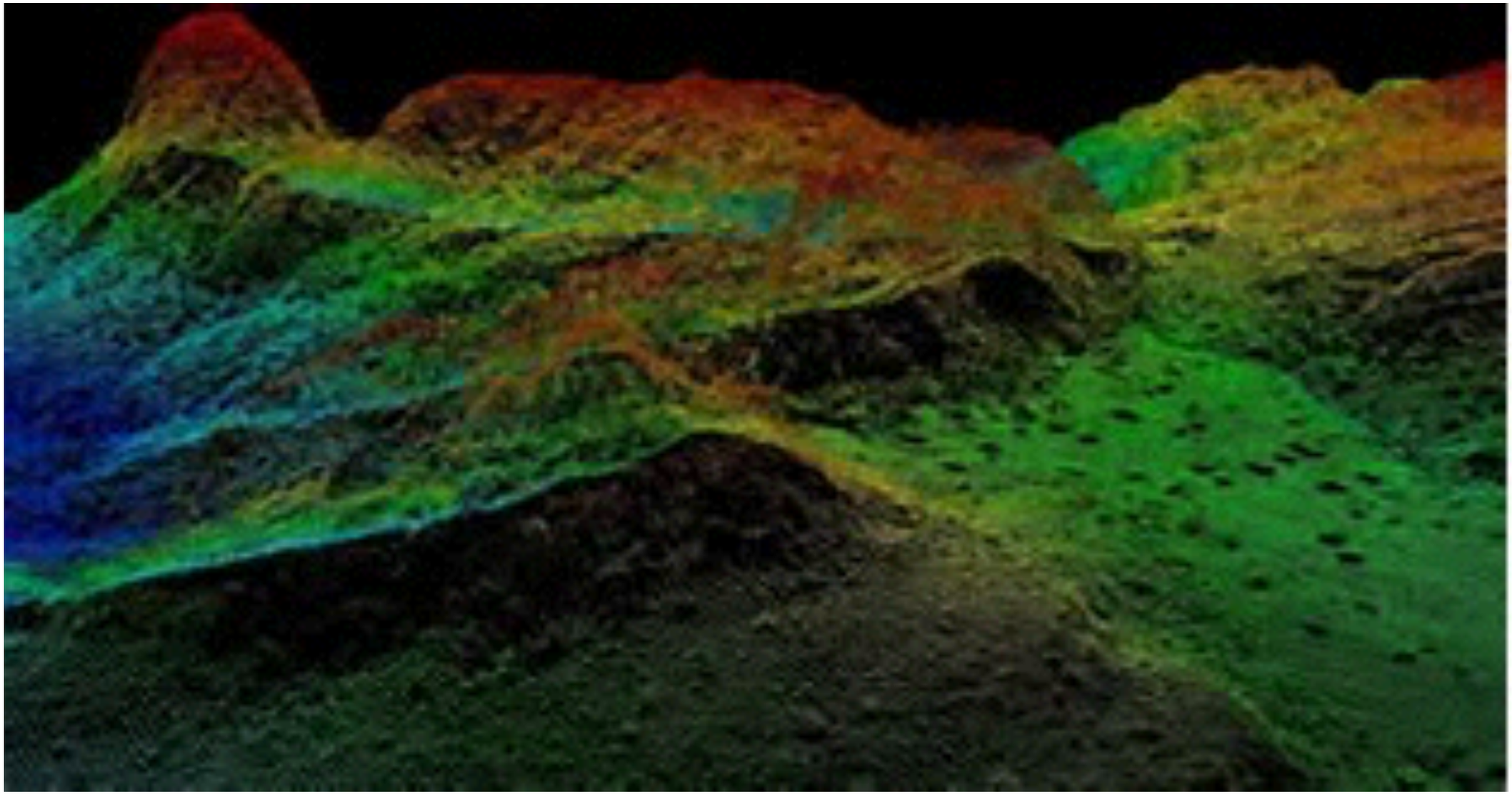
Data Sources: LIDAR point clouds



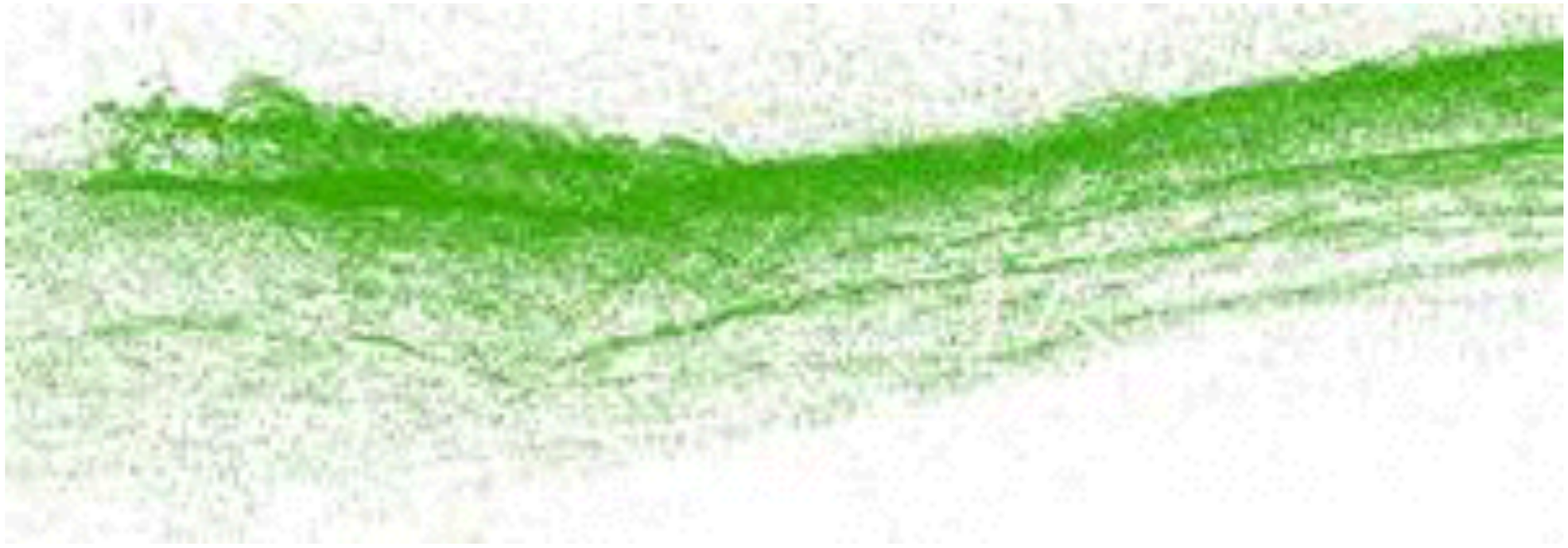
Data Sources: LIDAR point clouds



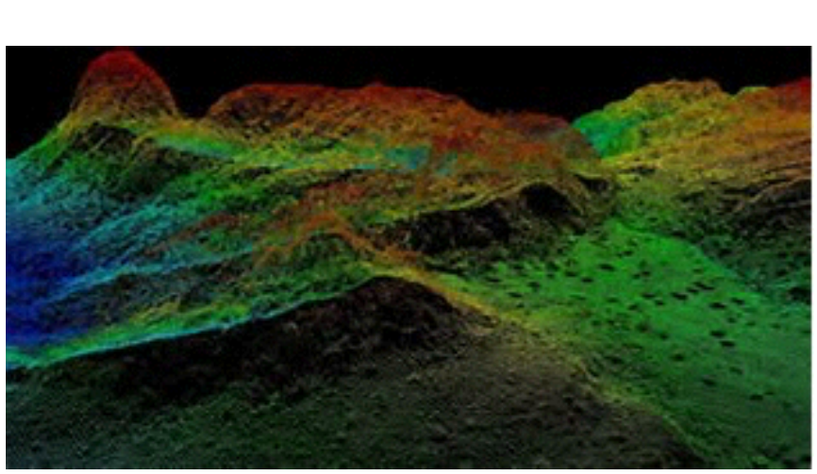
Data Sources: LIDAR point clouds



Data Sources: LIDAR point clouds

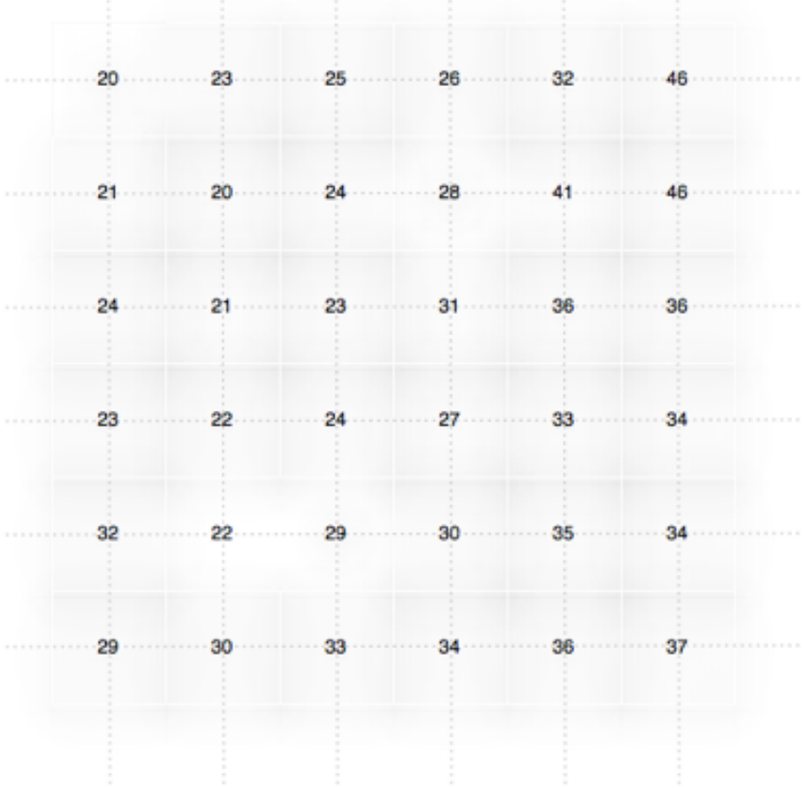


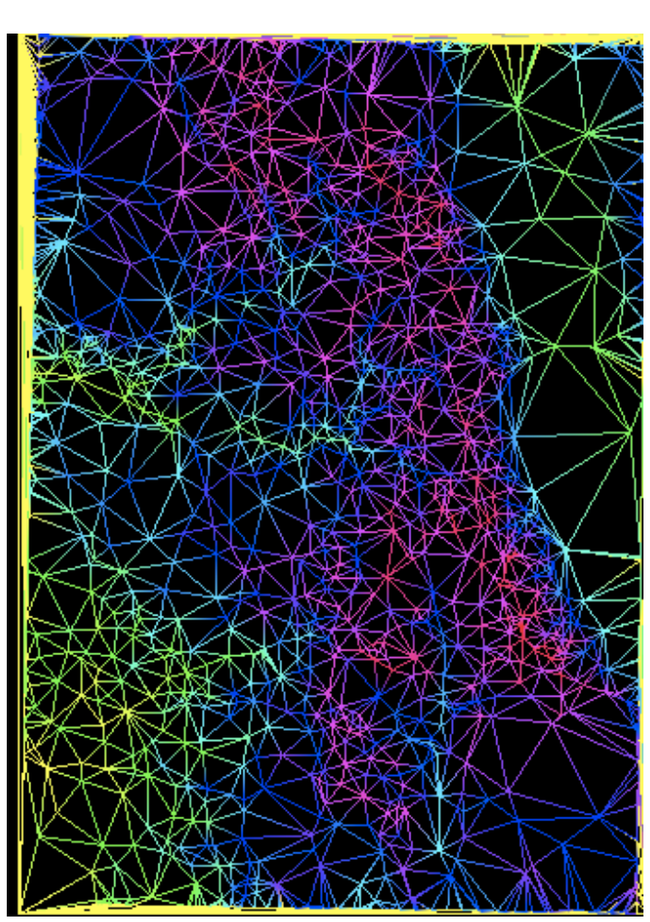
satellite imagery



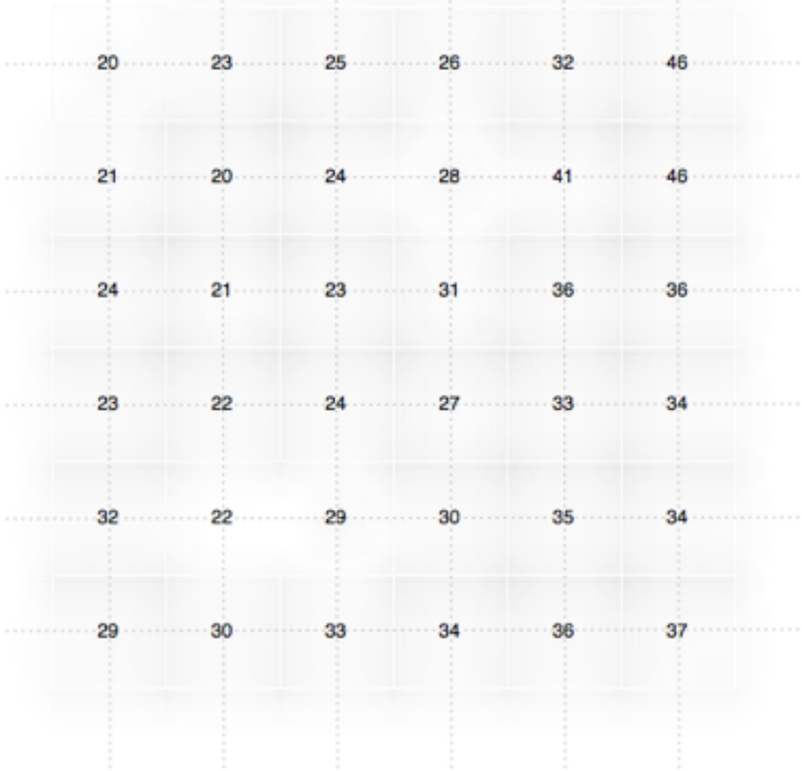
LIDAR point cloud

surface model

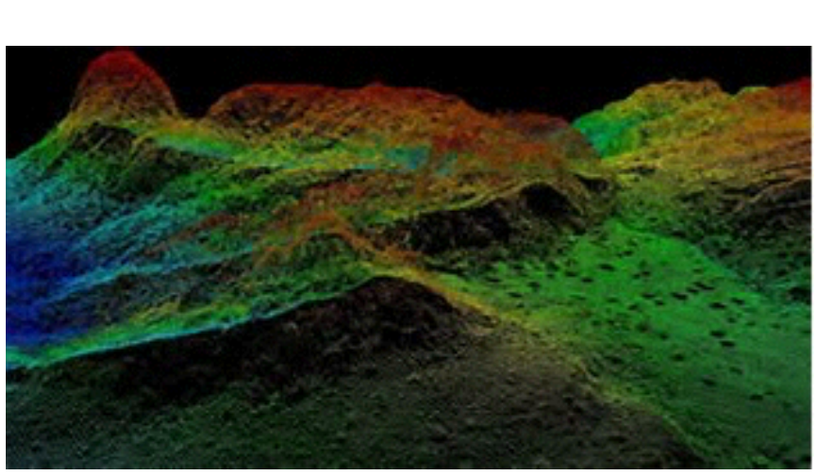




surface model

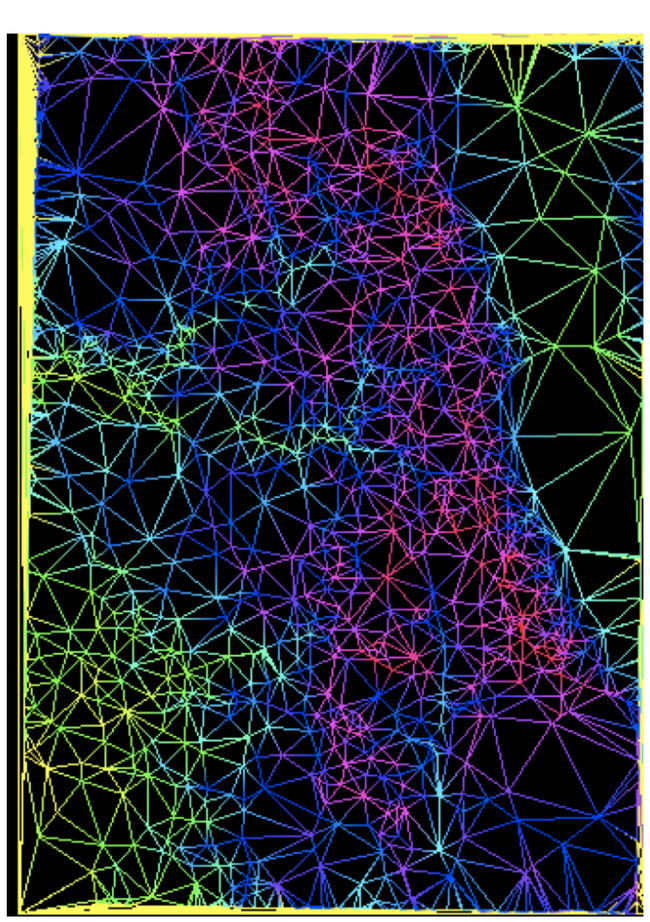


satellite imagery



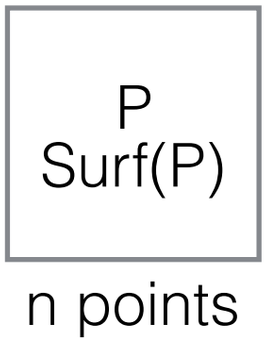
LIDAR point cloud

- point cloud to grid
- point cloud to TIN
- grid to TIN



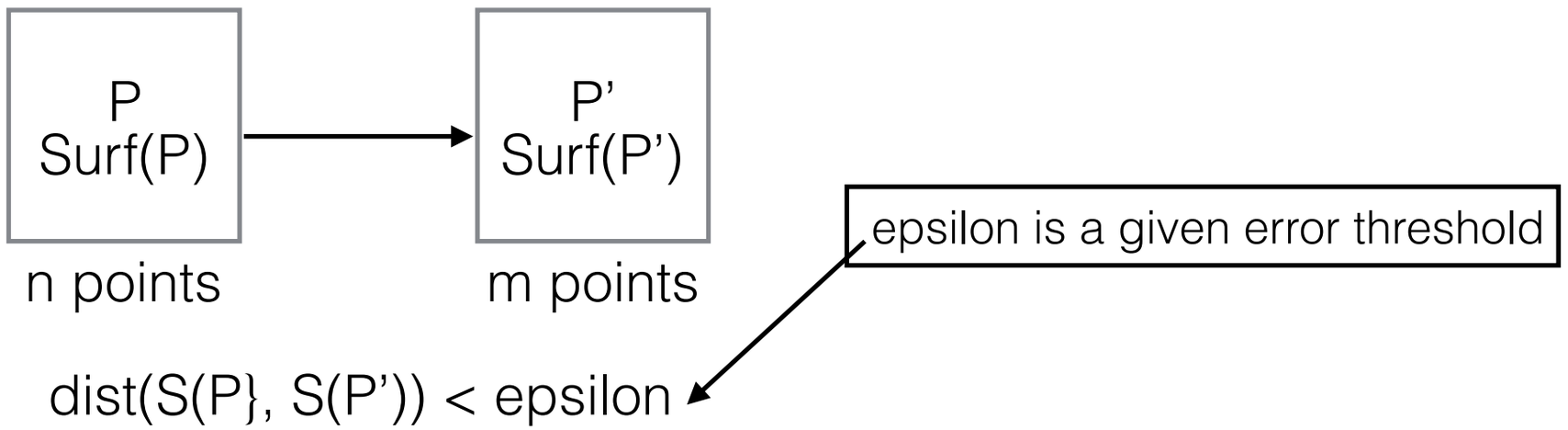
Terrain simplification

- P = { $(x_1, y_1, z_1), (x_2, y_2, z_2), \ldots\ldots, (x_n, y_n, z_n)$ } a set of terrain elevation samples
  - For e.g. P could be a set of grid (aerial image) or in general a point cloud (from LIDAR)
  - sometimes called a “height field” (in graphics and vision)
- P + interpolation method ===> surface Surf(P) corresponding to P



Terrain simplification

- P = { $(x_1, y_1, z_1), (x_2, y_2, z_2), \ldots\ldots, (x_n, y_n, z_n)$ } a set of terrain elevation samples
  - For e.g. P could be a set of grid (aerial image) or in general a point cloud (from LIDAR)
  - sometimes called a “height field” (in graphics and vision)
- P + interpolation method ===> surface Surf(P) corresponding to P



Simplification:

find an approximation S(P’) which approximates S(P) within the desired error threshold using as few points as possible

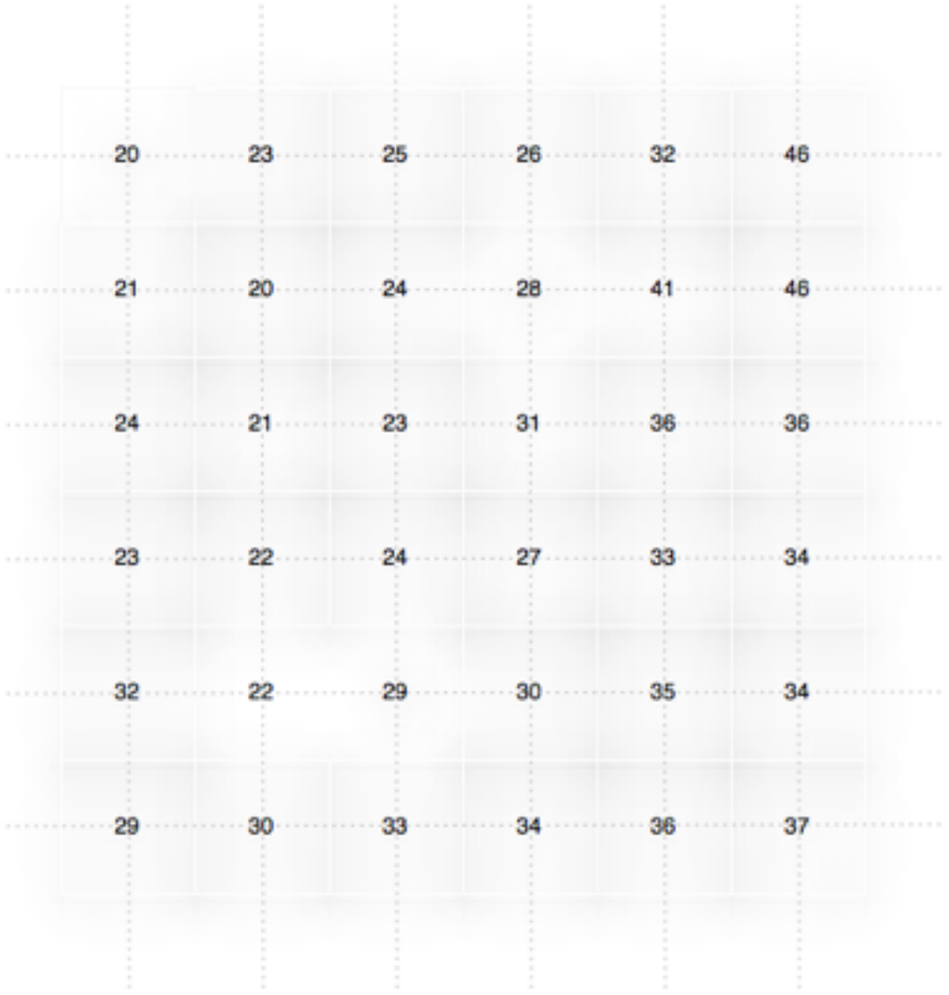
S(P) has n points ==> S(P’) has m points (m << n)

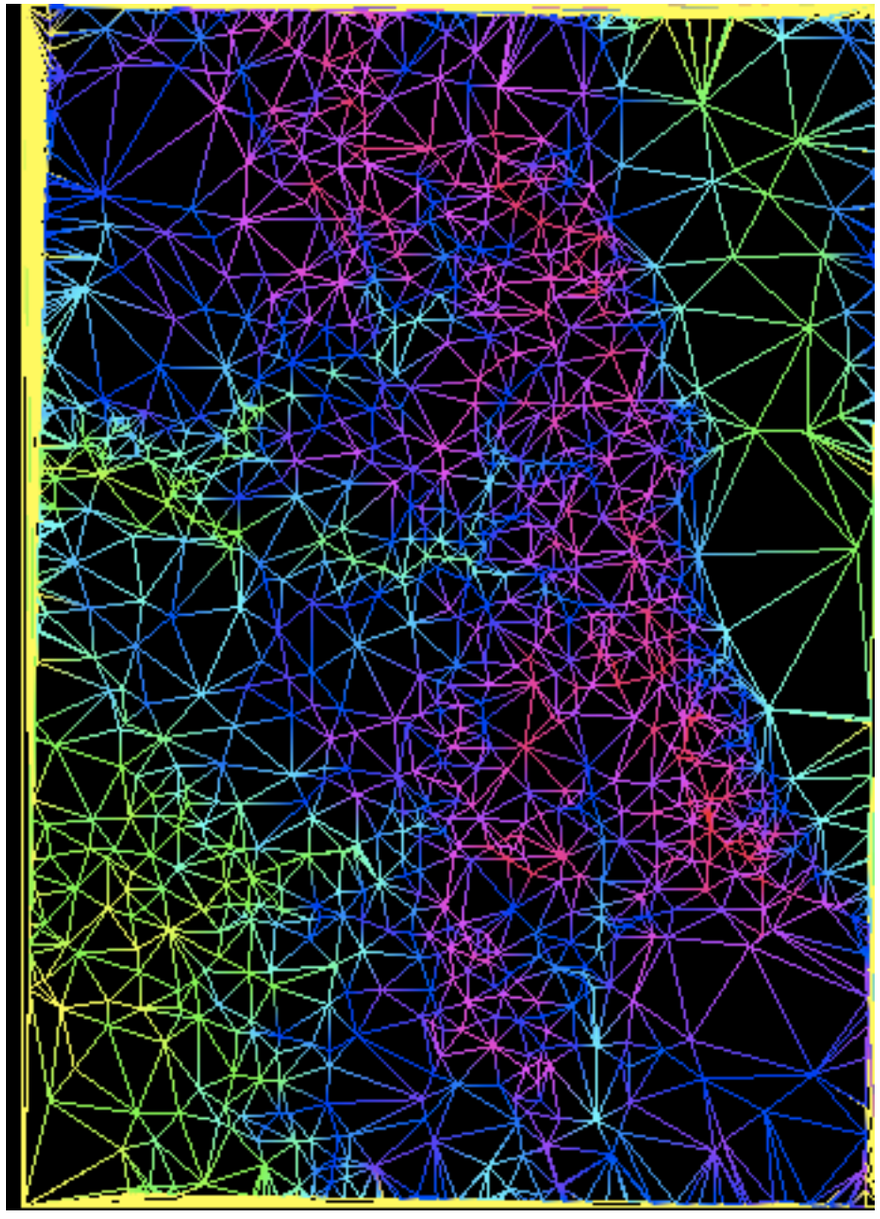
Outline

- Grid-to-TIN
  - We'll focus on grid-to-TIN simplification
  - The methods can be extended to deal with arbitrary (non-grid) data

- Point-cloud-to-TIN
- Point-cloud-to-grid

Grid to TIN


20 23 25 26 32 46
21 20 24 28 41 46
24 21 23 31 36 36
23 22 24 27 33 34
32 22 29 30 35 34
29 30 33 34 36 37



Motivation

- Grids
  - uniform resolution means a lot of data redundancy
  - grids get very large very fast

- Example:
  - Area if approx. 800 km x 800 km
  - Sampled at:
    - 100 resolution: 64 million points **(128MB)**
    - 30m resolution: 640 **(1.2GB)**
    - 10m resolution: 6400 = 6.4 billion **(12GB)**
    - 1m resolution: 600.4 billion **(1.2TB)**

Grid-to-TIN simplification

- Methods
  - Multi-pass decimation methods
    - start with P and discard points (one by one)
    - E.g.: Lee's drop heuristic
  - Multi-pass refinement methods
    - start with an initial approximation and add points one by one
    - greedy insertion (e.g. Garland & Heckbert)
  - One-pass methods
    - pre-compute importance of points
    - select points that are considered important features and triangulate them
    - based on quad trees or kd-trees

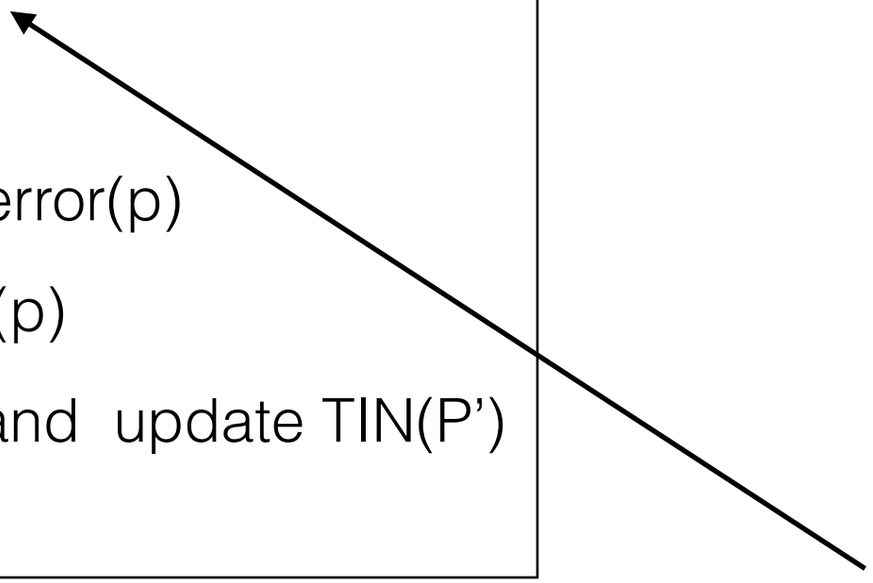
Decimation: Lee's drop heuristic

Refinement: Greedy insertion

- Notation:
  - P = set of grid points
  - P' = set of points in the TIN
  - TIN(P'): the TIN on P'

Algorithm:

- P = {all grid points}, P' = {4 corner points}
- Initialize TIN to two triangles with corners as vertices
- while not DONE() do
  - for each point p in P, compute error(p)
  - select point p with largest error(p)
  - insert p in P' , delete p from P, and update TIN(P')

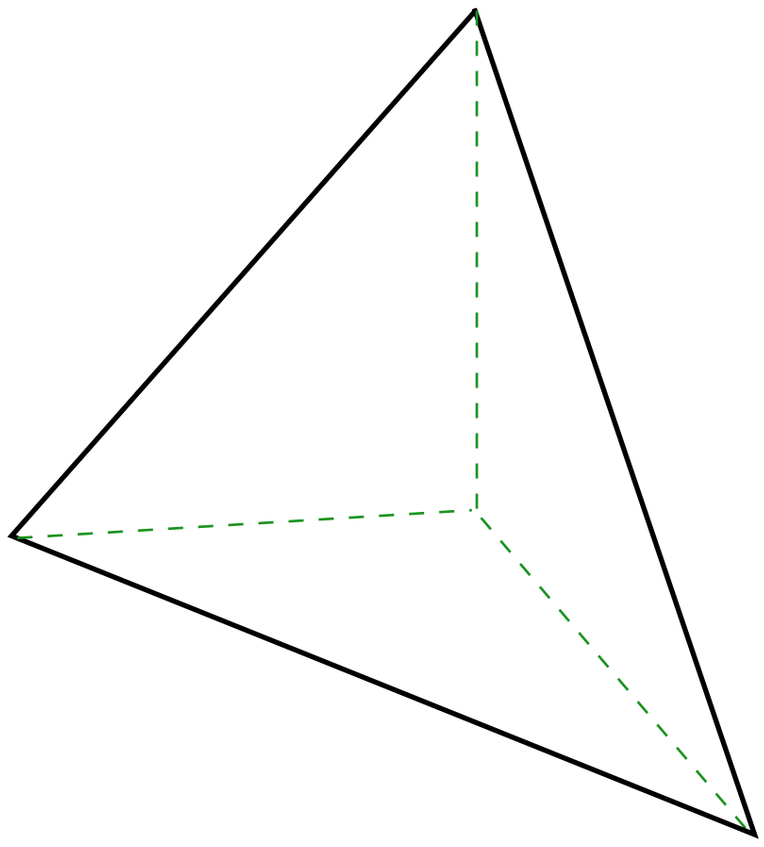


DONE() :: return (max error below given epsilon) ? TRUE; FALSE;

Greedy insertion

- Come up with a straightforward implementation of the generic greedy insertion and analyze its running time.

- Assume straightforward triangulation:
  - when inserting a point in a triangle, split the triangle in 3



Greedy insertion

	| P |	| P' |
	n	4 => O(1)
iteration 1	n-1	1 + O(1)
iteration 2	n-2	2 + O(1)
	.	.
	.	.
iteration k	n-k	k
at the end	n-m	m

- Note:
  - m = nb of vertices in the simplified TIN at the end (when error of P' falls below epsilon)
  - usually m is a fraction of n (e.g. 5%)

Greedy insertion— VERSION 1

Algorithm:

- P = {all grid points}, P' = {4 corner points}
- Initialize TIN to two triangles with 4 corners as vertices
- while not DONE() do
  - for each point p in P, compute error(p)
  - select point p with largest error(p)
  - insert p in P', delete p from P and update TIN(P')
    - create 3 new triangles

find triangle that contains p and compute the vertical difference in height between p and its interpolation on the triangle

Greedy insertion— VERSION 1

RE-CALCULATION

SELECTION

INSERTION

Algorithm:

- P = {all grid points}, P' = {4 corner points}
- Initialize TIN to two triangles with 4 corners as vertices
- while not DONE() do
  - for each point p in P, compute error(p)
  - select point p with largest error(p)
  - insert p in P', delete p from P and update TIN(P')
    - create 3 new triangles

ANALYSIS: At iteration k: we have O(n-k) points in P, O(k) points in P'

- RE-CALCULATION
  - compute the error of a point: must search through all triangles to see which one contains it ==> worst case O(k)
  - compute errors of all points ==> O(n-k) x O(k)
- SELECTION: select point with largest error: O(n-k)
- INSERTION: insert p in P', update TIN ==> O(1)
  - unless each point stores the triangle that contains it, need to find the triangle that contains p
  - for a straightforward triangulation: split the triangle that contains p into 3 triangles ==> O(1) time

Greedy insertion— VERSION 1

Analysis worst case:

- iteration k: $O((n-k) \times k) + O(n-k) + O(1)$

- overall: SUM { $(n-k) \times k$ } = …= $O(m^2n)$

- Note: dominant cost is re-calculation of errors (which includes point location)

- More on point location:
  - to locate the triangle that contains a given point, we “walk” (traverse) the TIN from triangle to triangle, starting from a triangle on the boundary (aka DFS on the triangle graph).
  - we must be very unlucky to always take O(k)
  - simple trick: start walking the TIN **from the triangle that contained the previous point.**
    - because points in the grid are spatially adjacent, most of the time a point will fall in the same triangle as the previous point or in one adjacent to it
  - average time for point location will be O(1)

Greedy insertion— VERSION 1

Worst-case: $O(m^2n)$

- iteration k: $O(n-k) \times O(k) + O(n-k) + O(1)$

  RE-CALC | SELECT | INSERT

- overall: SUM $\{O(n-k) \times k\} = O(m^2n)$

Average case: $O(mn)$

- trick to seed up point location ==> average time for point location will be $O(1)$
- iteration k: $O(n-k) \times O(1) + O(n-k) + O(1)$

  RE-CALC | SELECT | INSERT

- SUM $\{O(n-k)\} = O(mn)$

Greedy insertion— VERSION 2

Observation: Only the points that fall inside triangles that have changed need to re-compute their error.

- Re-compute errors ONLY for points whose errors have changed
- Each point p in P stores its error, error(p)
- Each triangle stores a list of points inside it

Algorithm:

- P = {all grid points}, P' = {4 corner points}
- Initialize TIN to two triangles with 4 corners as vertices
- while not DONE() do
  - ~~for each point p in P, compute error(p)~~
  - select point p with largest error(p)
  - insert p in P', delete p from P and update TIN(P')
    - create 3 new triangles
    - for all points in triangle that contains p:
      - find the new triangles where they belong, re-compute their errors

Greedy insertion— VERSION 2

Worst-case: O(mn)

- iteration k: - + O(n-k) + O(1) + O(n-k) x O(1)

RE-CALC | SELECT | INSERT + re-calc

- overall: SUM {O(n-k) } = O(mn)

Average case: O(mn)

- if points are uniformly distributed in the triangles ==> O((n-k)/k) points per triangle
- iteration k: - + O(n-k) + O(1) + O((n-k)/k) x O(1)

RE-CALC | SELECT | INSERT + re-calc

- SUM {O(n-k) + O((n-k)/k} = O(mn)

SELECTION will be dominant!

Greedy insertion— VERSION3

- Version2, re-calculation goes down and selection becomes dominant

- Version 3: improve selection
  - store a heap of errors of all points in P

Algorithm:

- P = {all grid points}, P' = {4 corner points}
- Initialize TIN to two triangles with 4 corners as vertices
- while not DONE() do
  - use heap to select point p with largest error(p)
  - insert p in P', delete p from P and update TIN(P')
    - for all points in the triangle that contains p:
      - find the new triangles where they belong, re-compute their errors
      - update new errors in heap

Greedy insertion— VERSION 3

Worst-case: O(mn lg n)

- iteration k: - + O(lg (n-k)) + O(1) + O(n-k) x O(lg (n-k))

RE-CALC | SELECT | INSERT + re-calc

- overall: SUM {(n-k) lg (n-k)} = O(mn lg n)

Average case: $O((m+n) \lg^2 n)$

- if points are uniformly distributed in the triangles ==> O((n-k)/k) points per triangle
- iteration k: - + O(lg (n-k)) + O(1) + O((n-k)/k) x O(lg (n-k))

RE-CALC | SELECT | INSERT + re-calc

- SUM {lg (n-k) + O((n-k)/k} = $O((m+n) \lg^2 n)$

heap updates will be dominant!

Greedy insertion— VERSION 4

- Version 3: selection is down, but updating the heap is now dominant

- Version 4: store in heap only one point per triangle (point of largest error)

Algorithm:

- P = {all grid points}, P' = {4 corner points}
- Initialize TIN to two triangles with 4 corners as vertices
- while not DONE() do
  - use heap to select point p with largest error(p)
  - insert p in P', delete p from P and update TIN(P')
    - for all points in the triangle that contains p:
      - find the new triangles where they belong, re-compute their errors
      - find point with largest error per triangle
      - add these points (one per triangle) to the heap

Greedy insertion— VERSION 4

Worst-case: O(mn)

- iteration k: - + O(lg k) + O(1) + O(n-k)xO(1) + O(1) x O(lg k)

RE-CALC | SELECT | INSERT + re-calc

- overall: SUM {lg k + O(n-k) } = O(mn)

Average case: O((m+n) lg n)

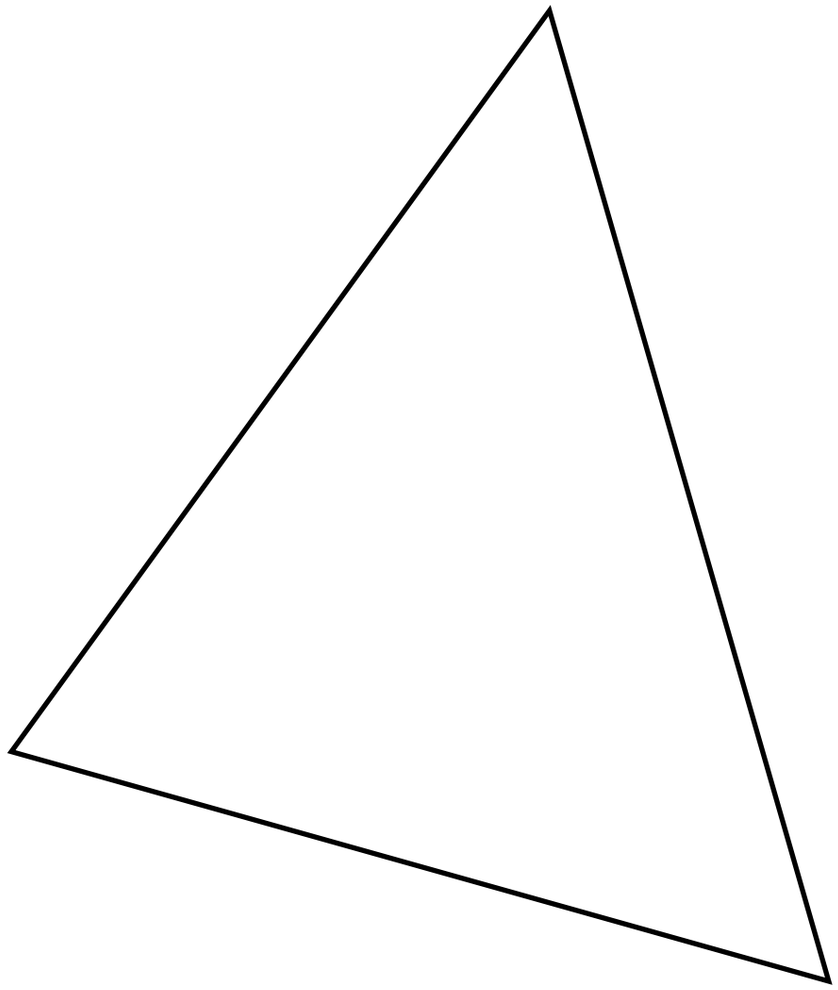
- if points are uniformly distributed in the triangles ==> O((n-k)/k) points per triangle
- iteration k: - + O(lg k) + O(1) + O((n-k)/k)x O(1) + O(1) x O(lg k)

RE-CALC | SELECT | INSERT + re-calc

- SUM {lg k + O((n-k)/k} = O((m+n) lg n)

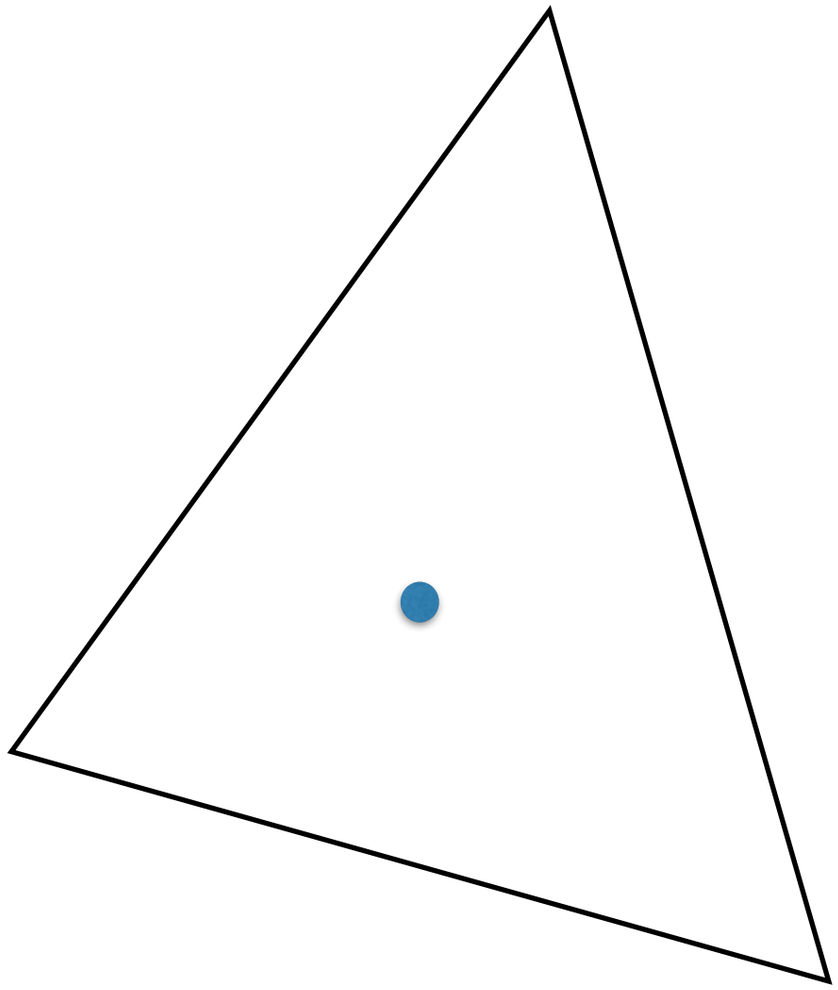
Triangulations

- The straightforward way to triangulate when adding new points runs in O(1) time but will create long and skinny triangles



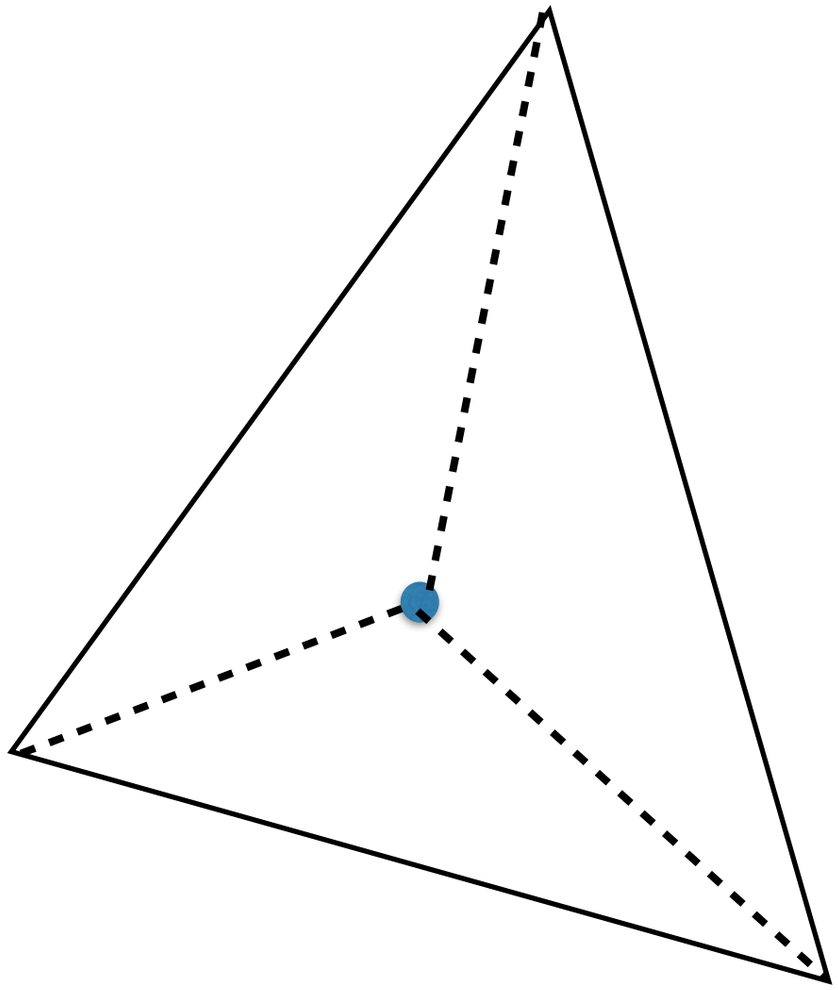
Triangulations

- The straightforward way to triangulate when adding new points runs in O(1) time but will create long and skinny triangles



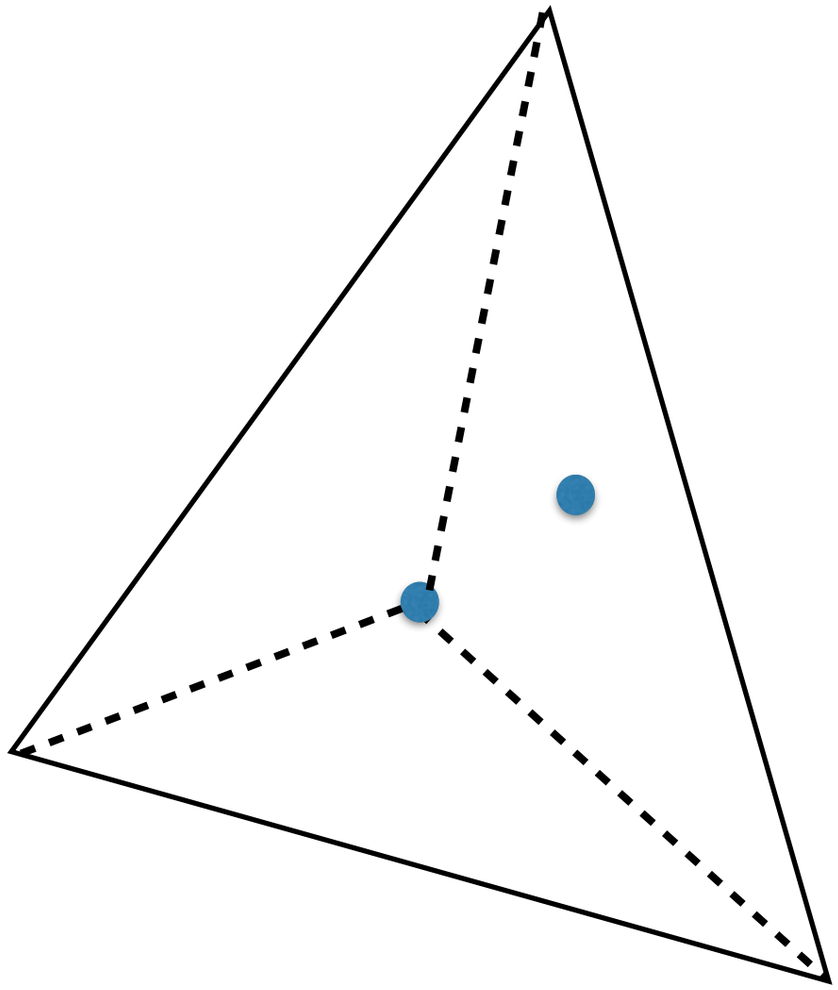
Triangulations

- The straightforward way to triangulate when adding new points runs in O(1) time but will create long and skinny triangles



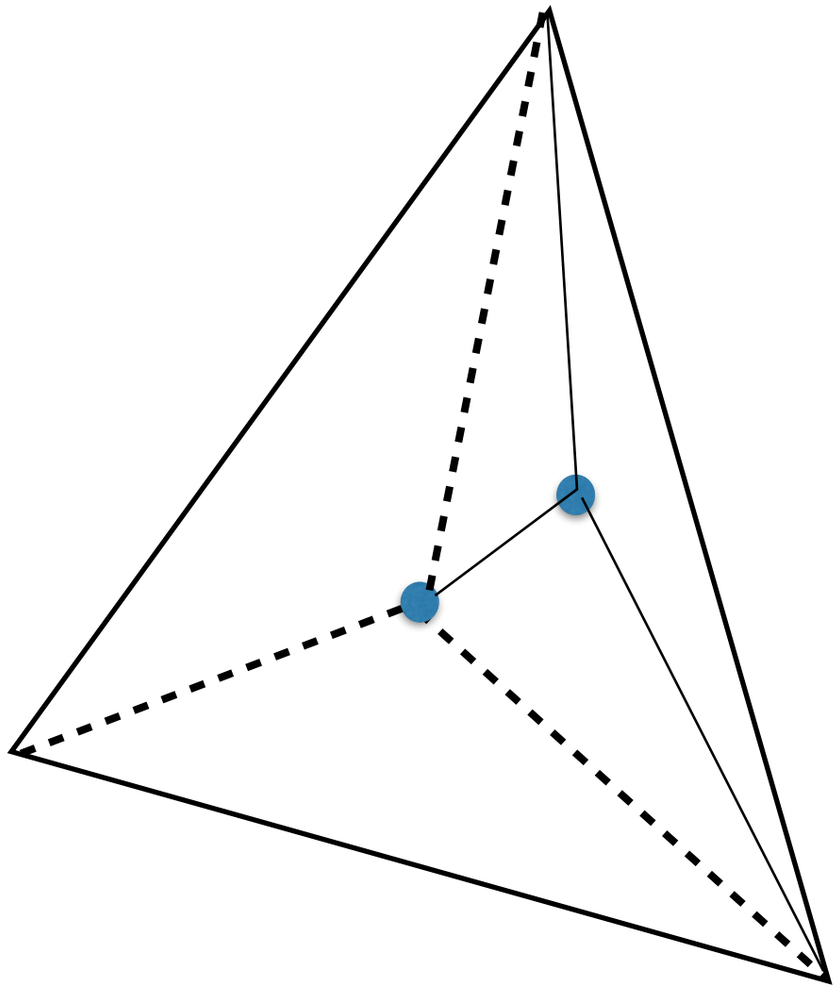
Triangulations

- The straightforward way to triangulate when adding new points runs in O(1) time but will create long and skinny triangles



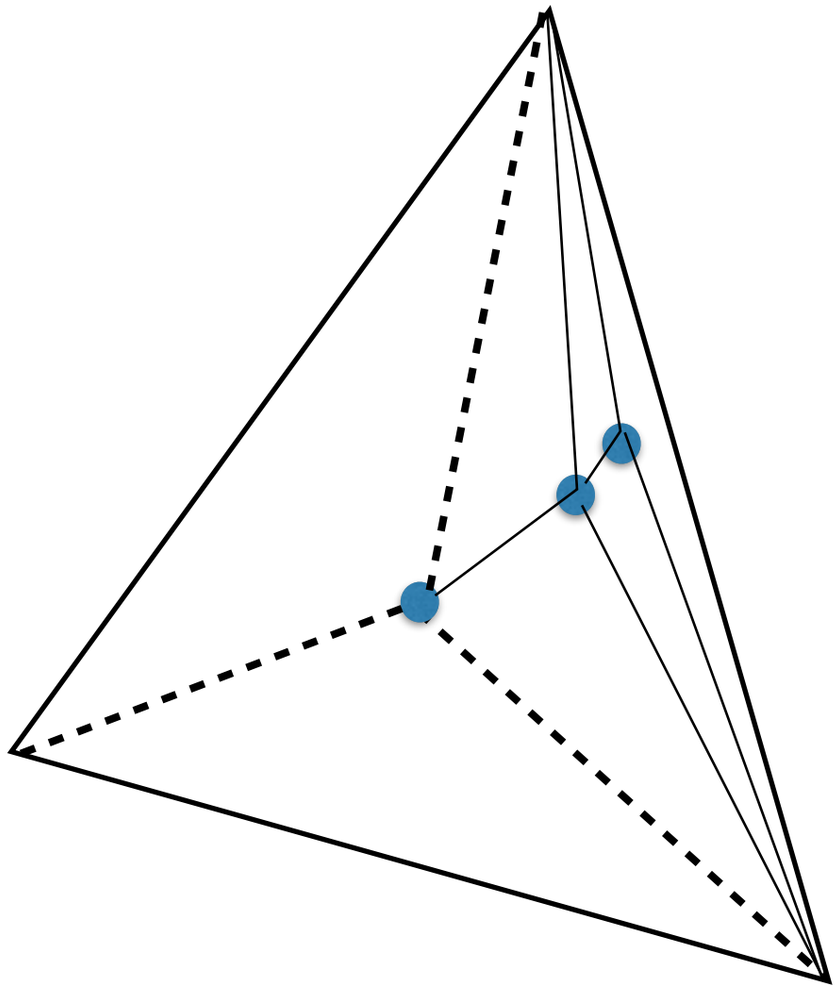
Triangulations

- The straightforward way to triangulate when adding new points runs in O(1) time but will create long and skinny triangles



Triangulations

- The straightforward way to triangulate when adding new points runs in O(1) time but will create long and skinny triangles



Triangulations

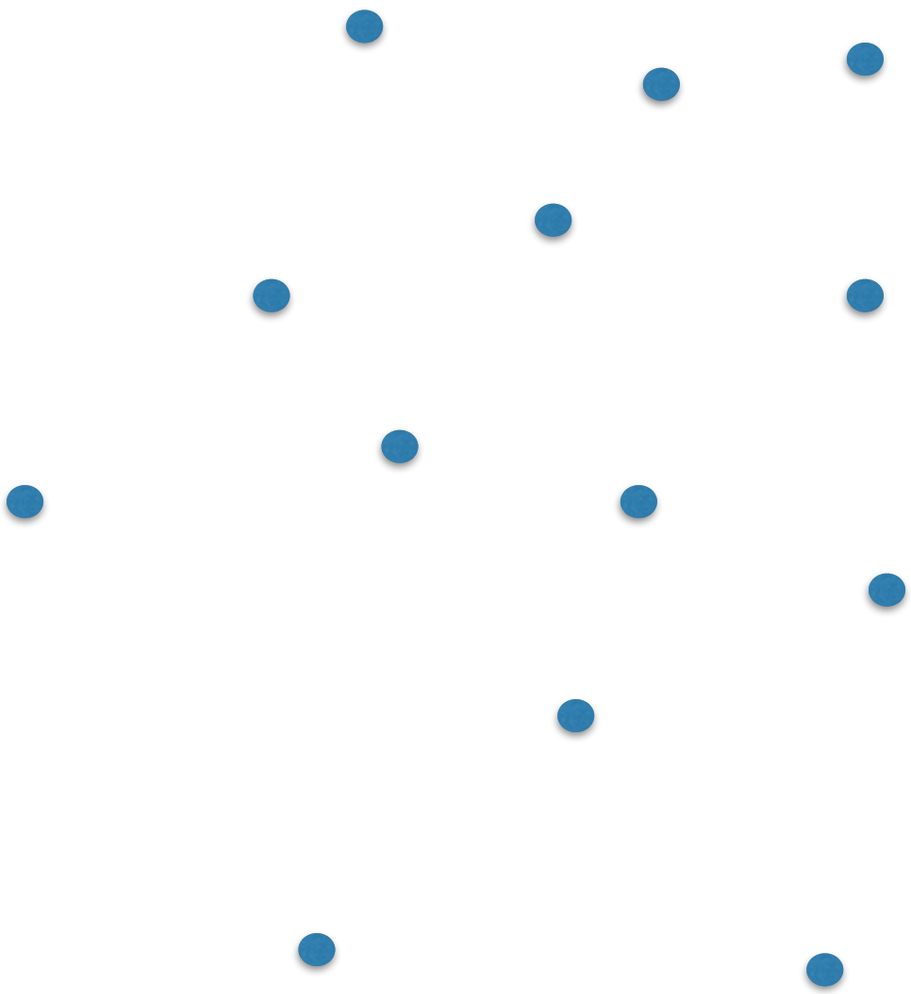
- The straightforward way to triangulate when adding new points runs in O(1) time but will create long and skinny triangles
- Small angles cause troubles!

Triangulations

- The straightforward way to triangulate when adding new points runs in O(1) time but will create long and skinny triangles
- Small angles cause troubles!
- Good meshes have uniform triangles and angles that are neither too small nor too large

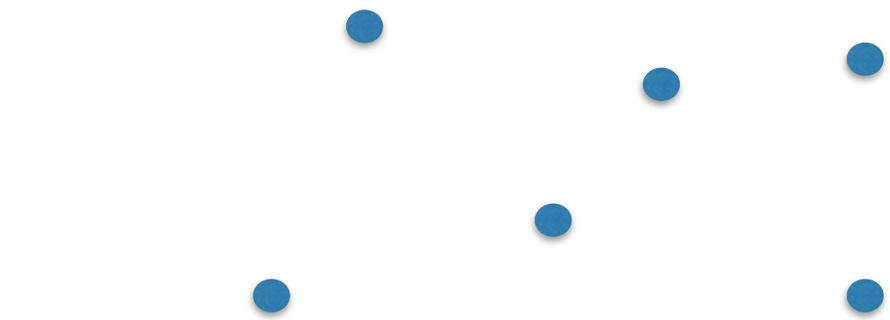
http://doc.cgal.org/latest/Surface_mesh_simplification/Illustration-Simplification-ALL.jpg

Triangulation



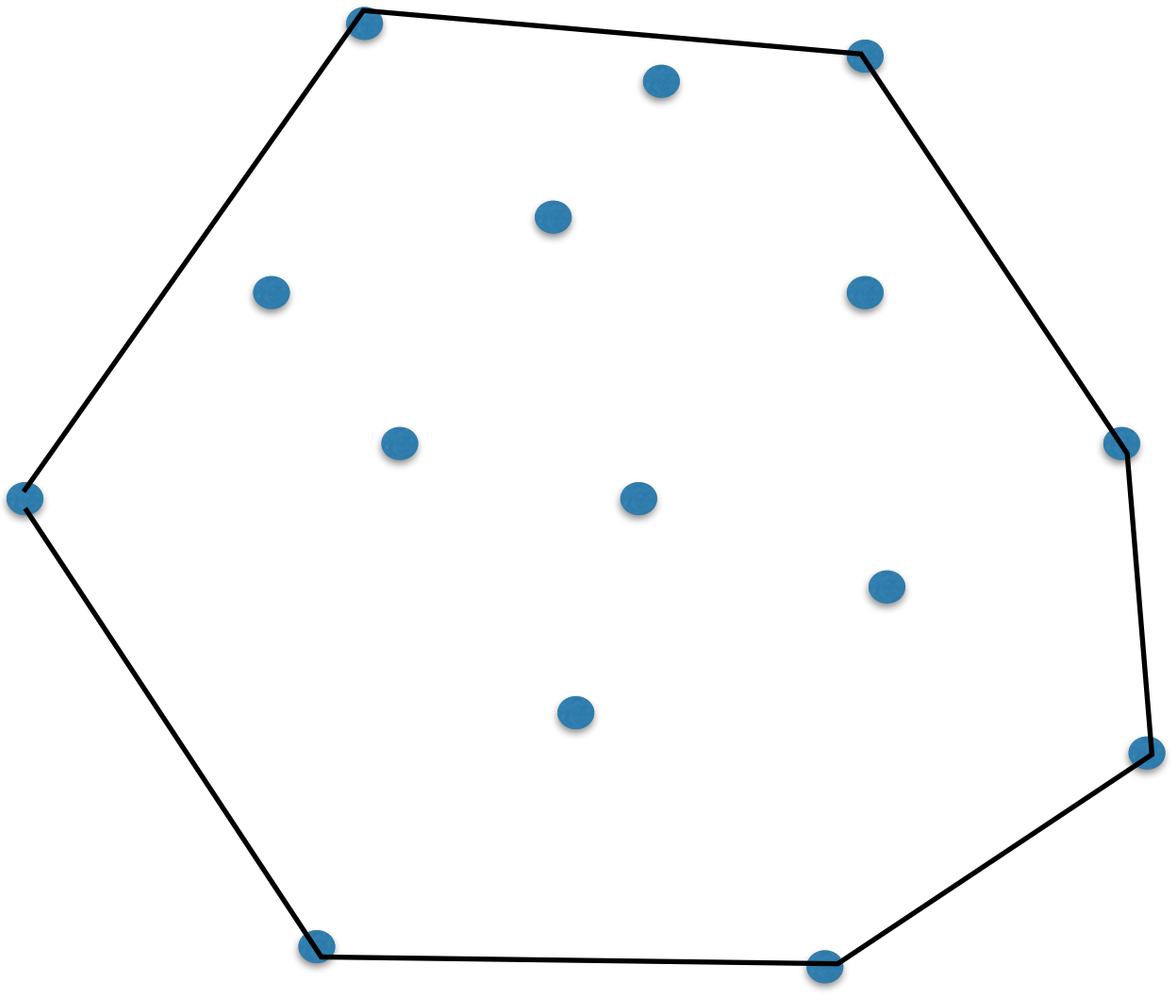
Triangulation

- A triangulation of a point set P in 2D is a triangulation of the convex hull of P



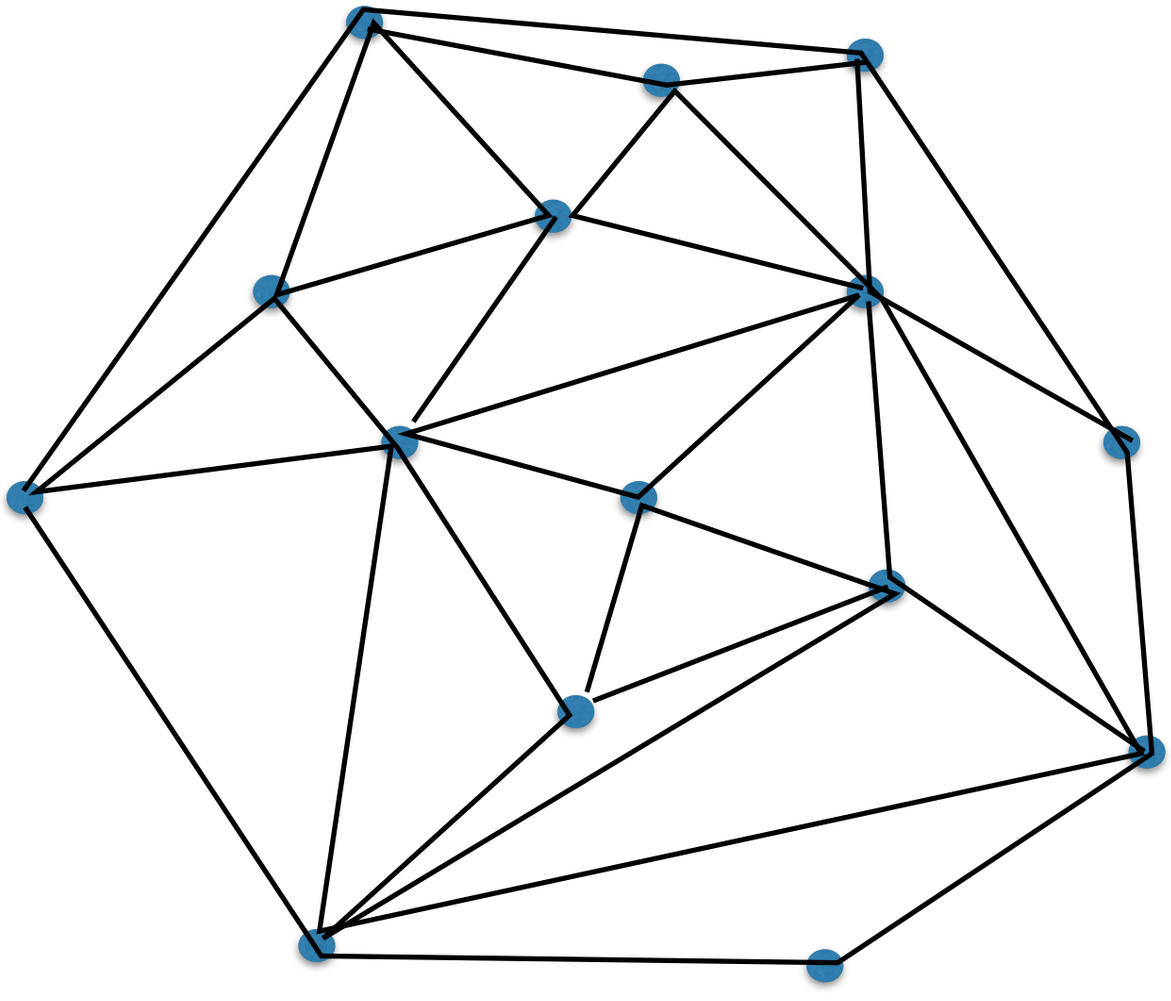
Triangulation

- A triangulation of a point set P in 2D is a triangulation of the convex hull of P



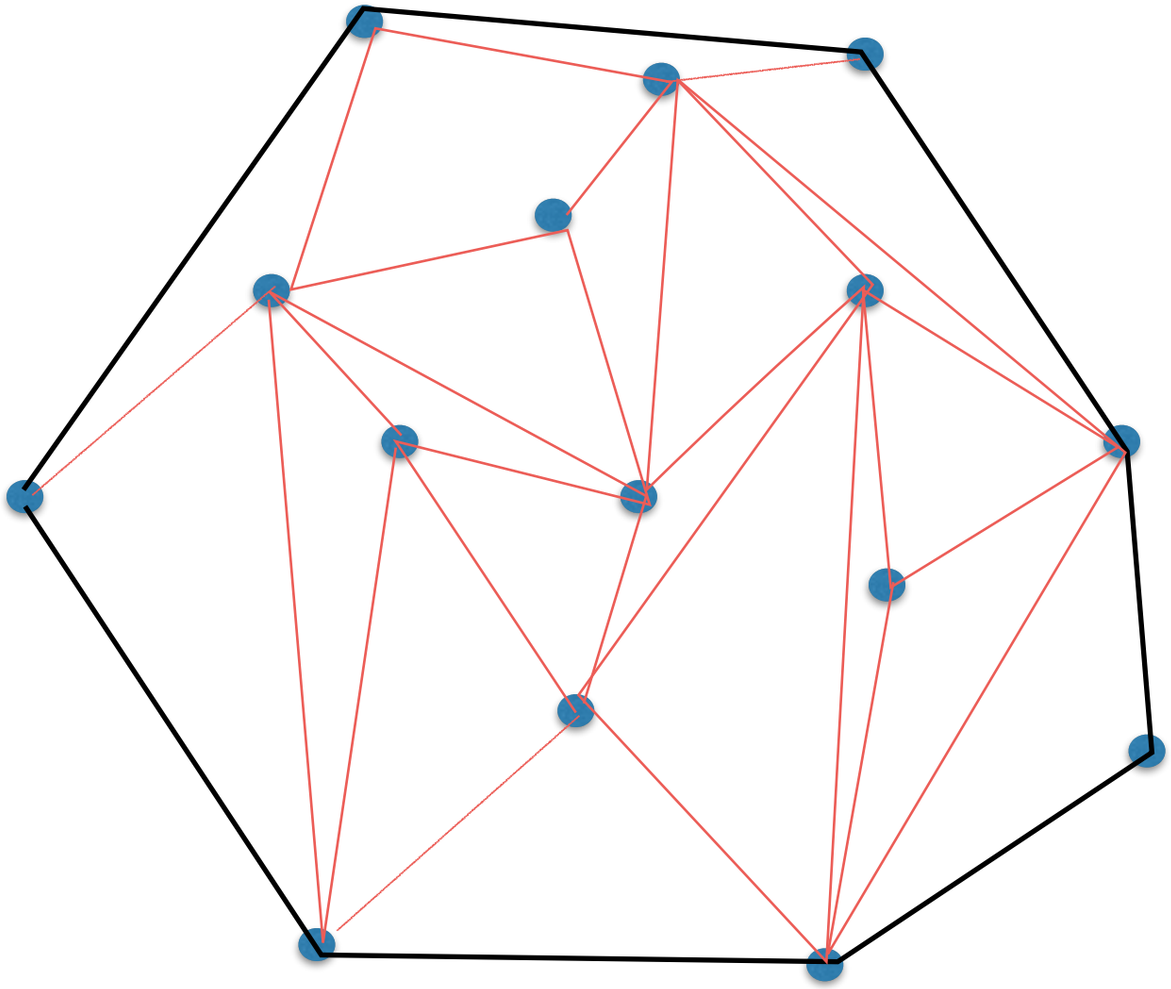
Triangulation

- A triangulation of a point set P in 2D is a triangulation of the convex hull of P



Triangulation

- Many ways to triangulate a set of points P



Triangulation

- Many ways to triangulate a set of points P
- Different ways to evaluate a triangulation
  - minimum angle
  - maximum degree
  - sum of edge lengths
  - …
- Algorithms for various kinds of optimal triangulations are known.

- A triangulation that maximizes the minimum angle across all triangles is called the Delaunay triangulation and can be computed in O(n lg n) time.

Greedy insertion with Delaunay triangulation

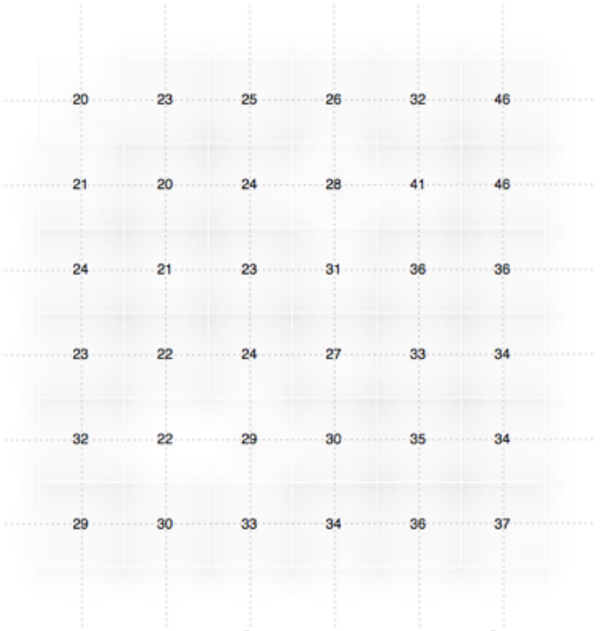
Algorithm:

- P = {all grid points}, P' = {4 corner points}
- Initialize TIN to two triangles with corners as vertices
- while not DONE() do
  - for each point p in P, compute error(p)
  - select point p with largest error(p)
  - insert p in P' , delete p from P, and update TIN(P')

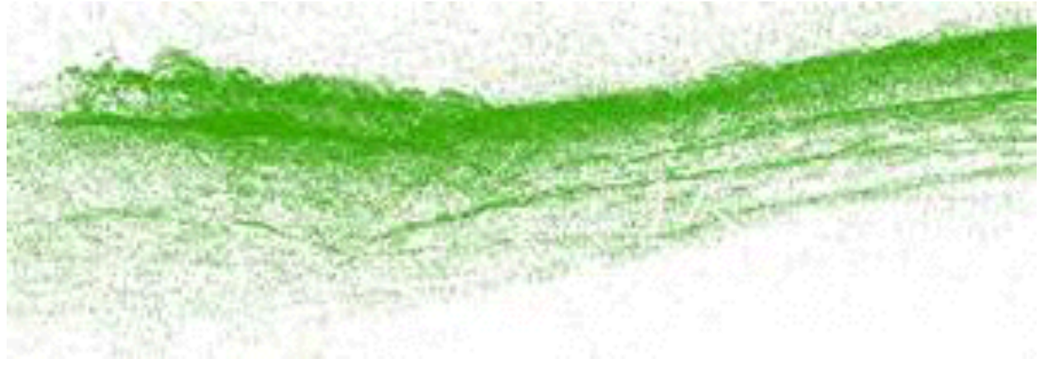
maintain TIN as a Delaunay triangulation of P'

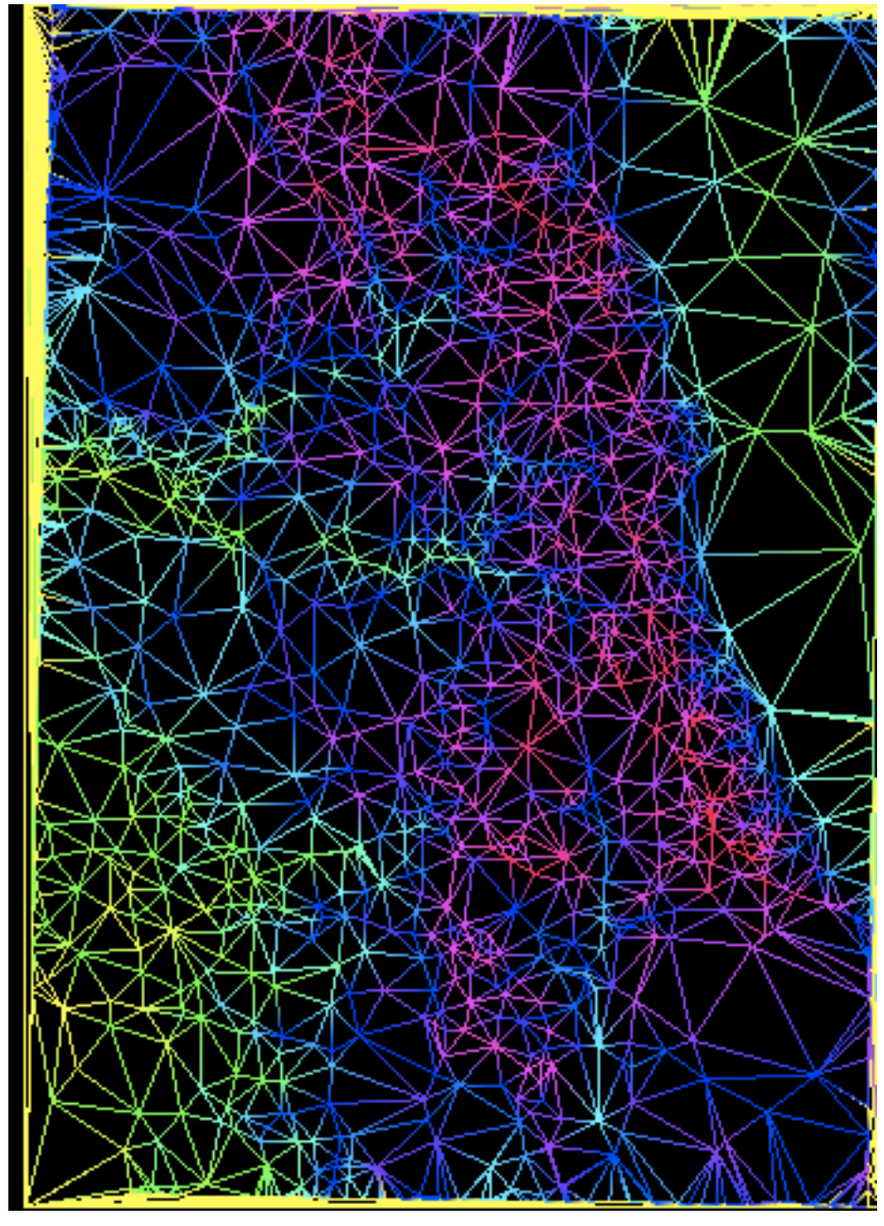
Brainstorming: Point-cloud-to-TIN ?

Brainstorming: Point-cloud-to-TIN ?



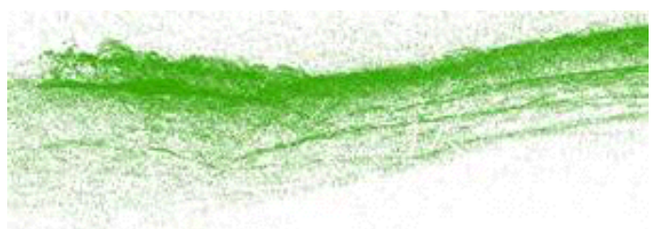
grid (raster)

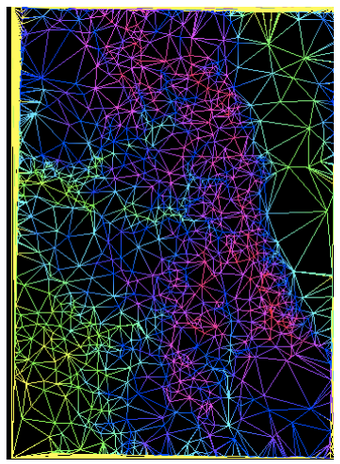




TIN

Brainstorming: Point-cloud-to-TIN ?





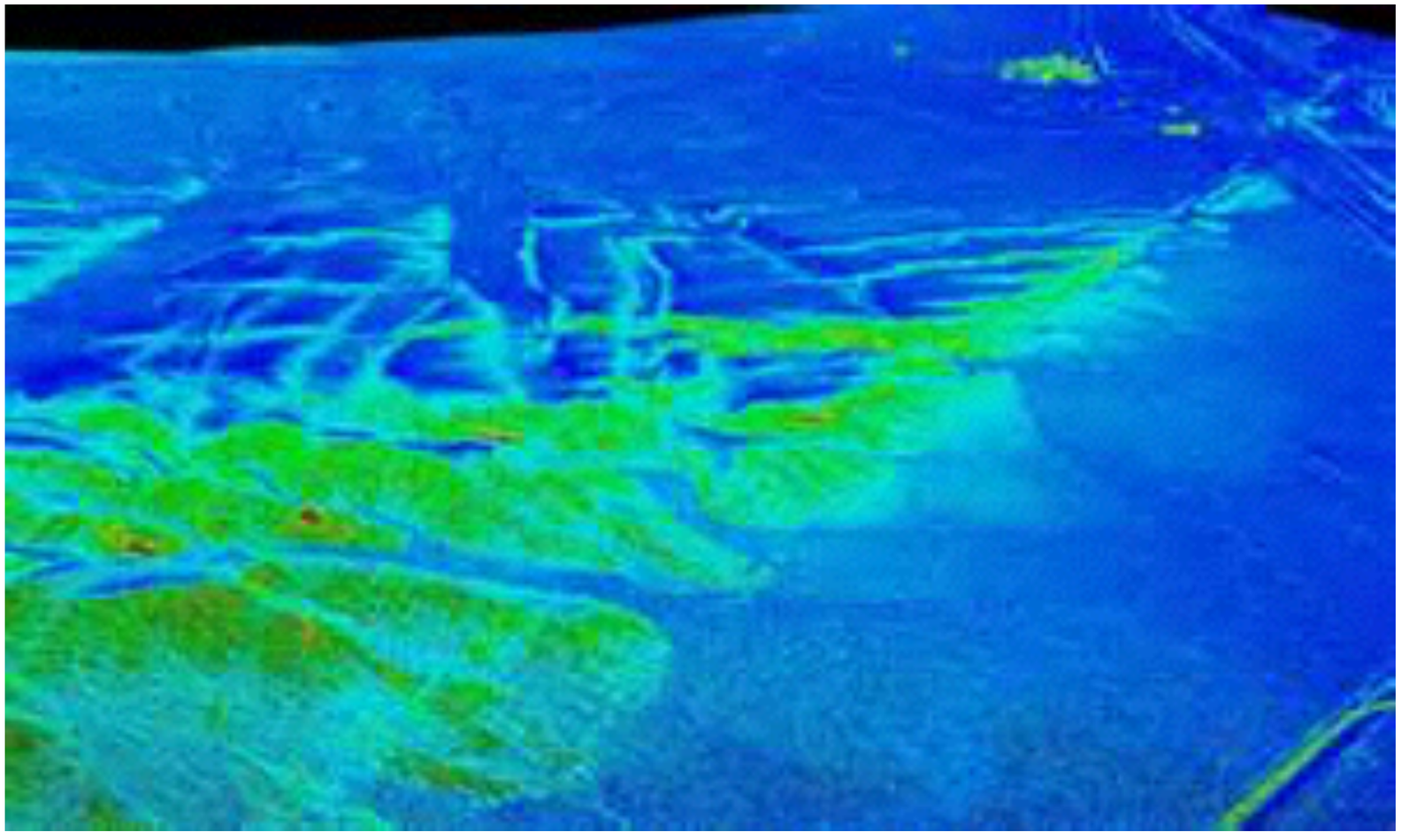
What needs to change?

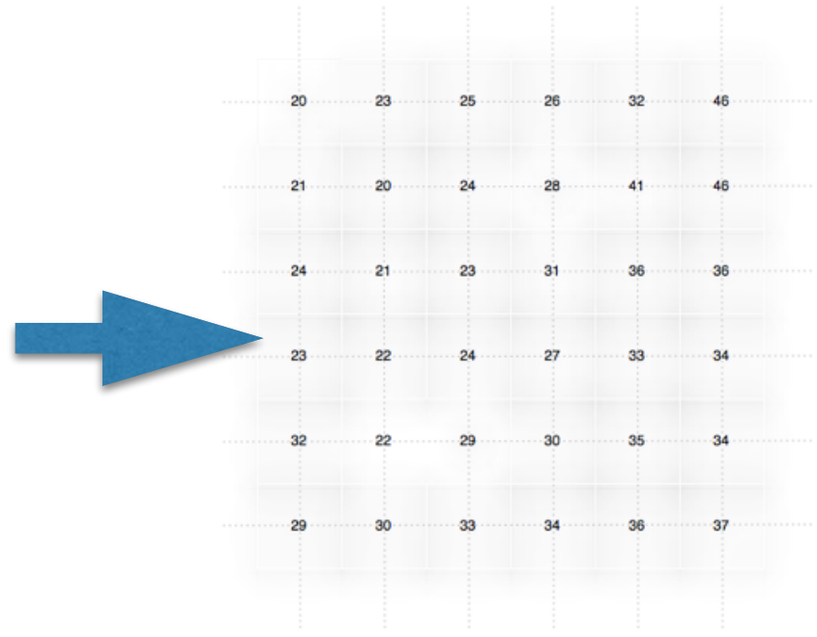
Algorithm:

- P = {all grid points}, P' = {4 corner points}
- Initialize TIN to two triangles with corners as vertices
- while not DONE() do
  - for each point p in P, compute error(p)
  - select point p with largest error(p)
  - insert p in P', delete p from P, and update TIN(P')

Brainstorming: Point-cloud-to-grid ?

Brainstorming: Point-cloud-to-grid ?

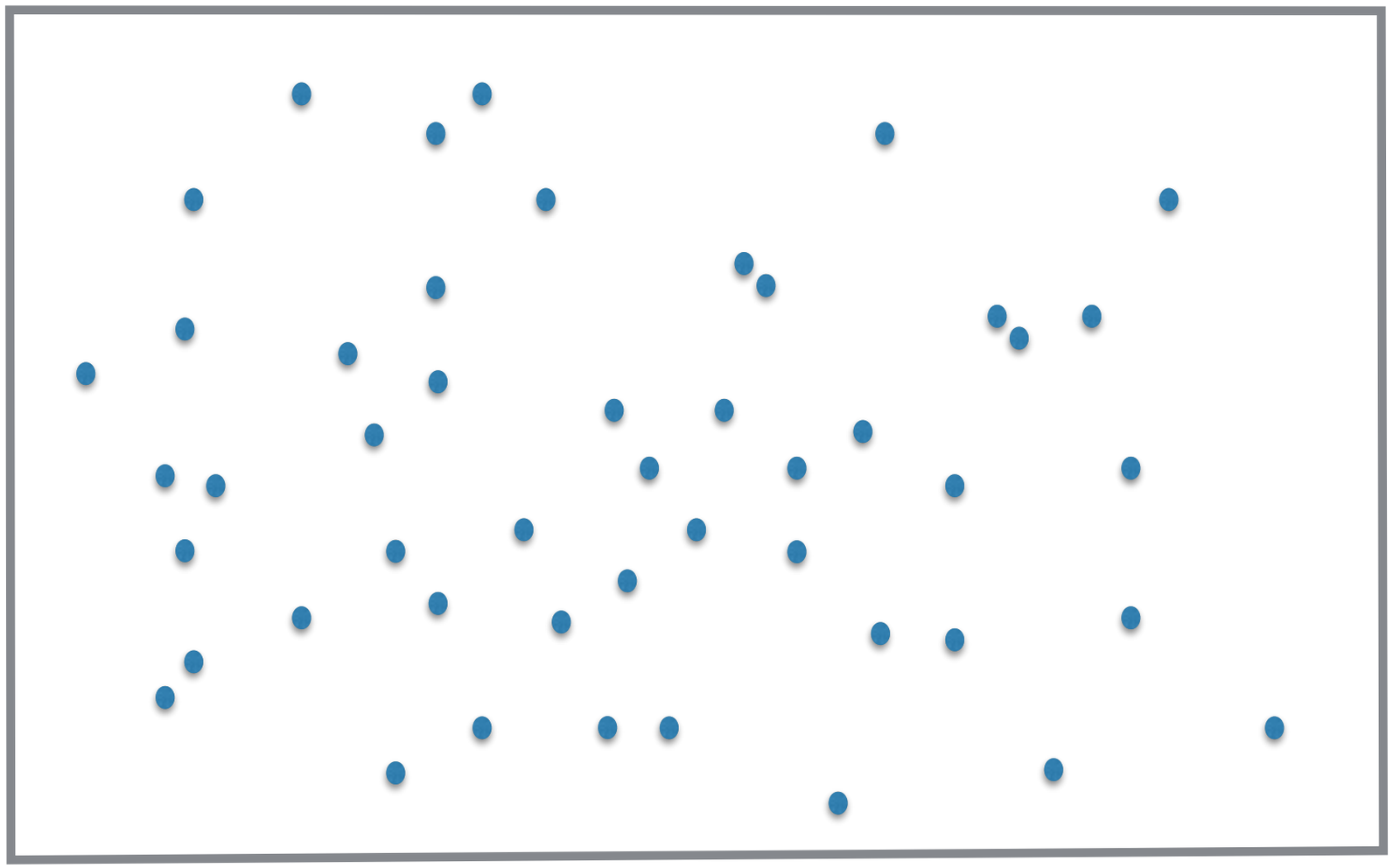




California Lidar data

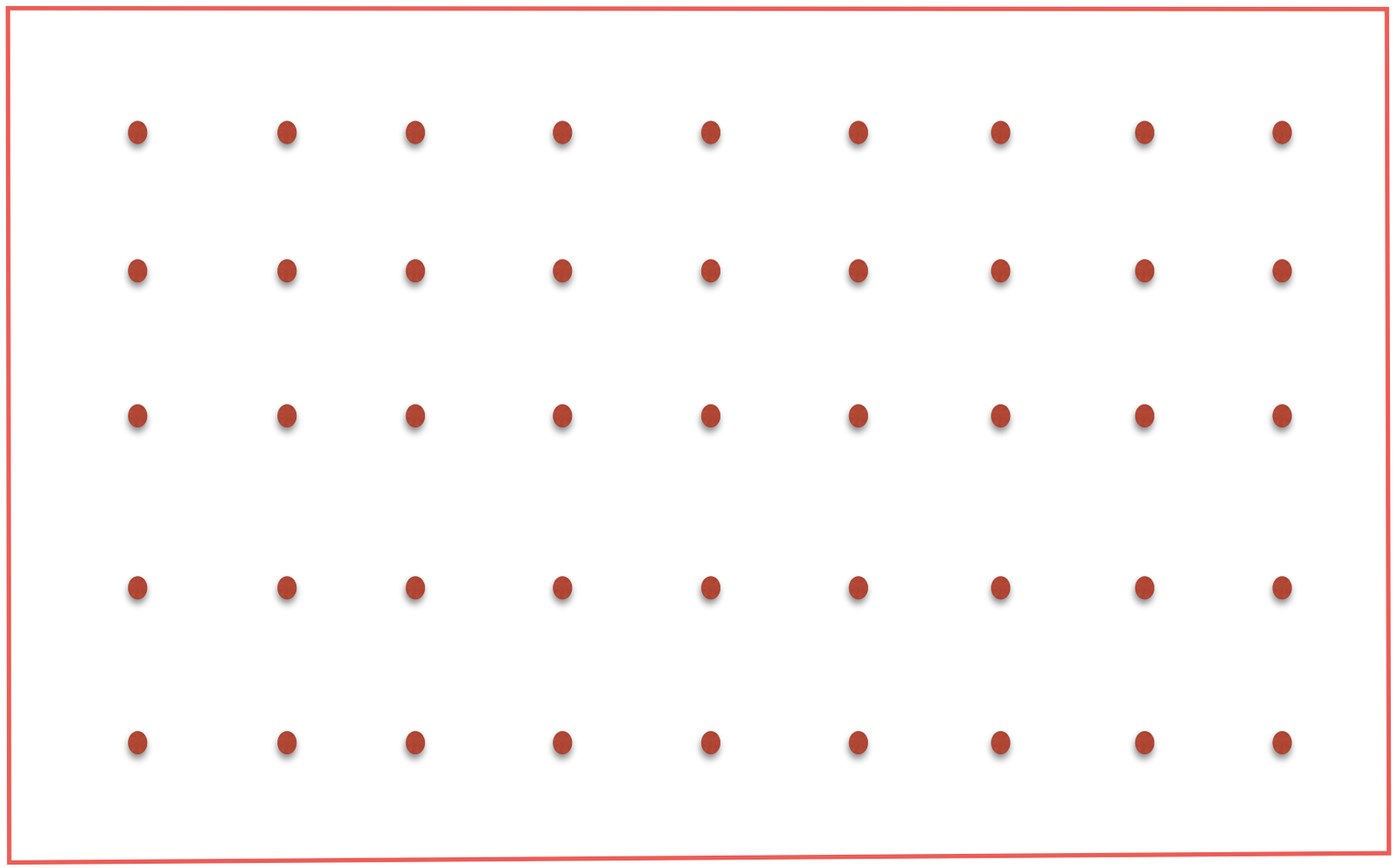
http://www.opentopography.org/images/opentopo_images/garlock_slope.jpg

Brainstorming: Point-cloud-to-grid ?



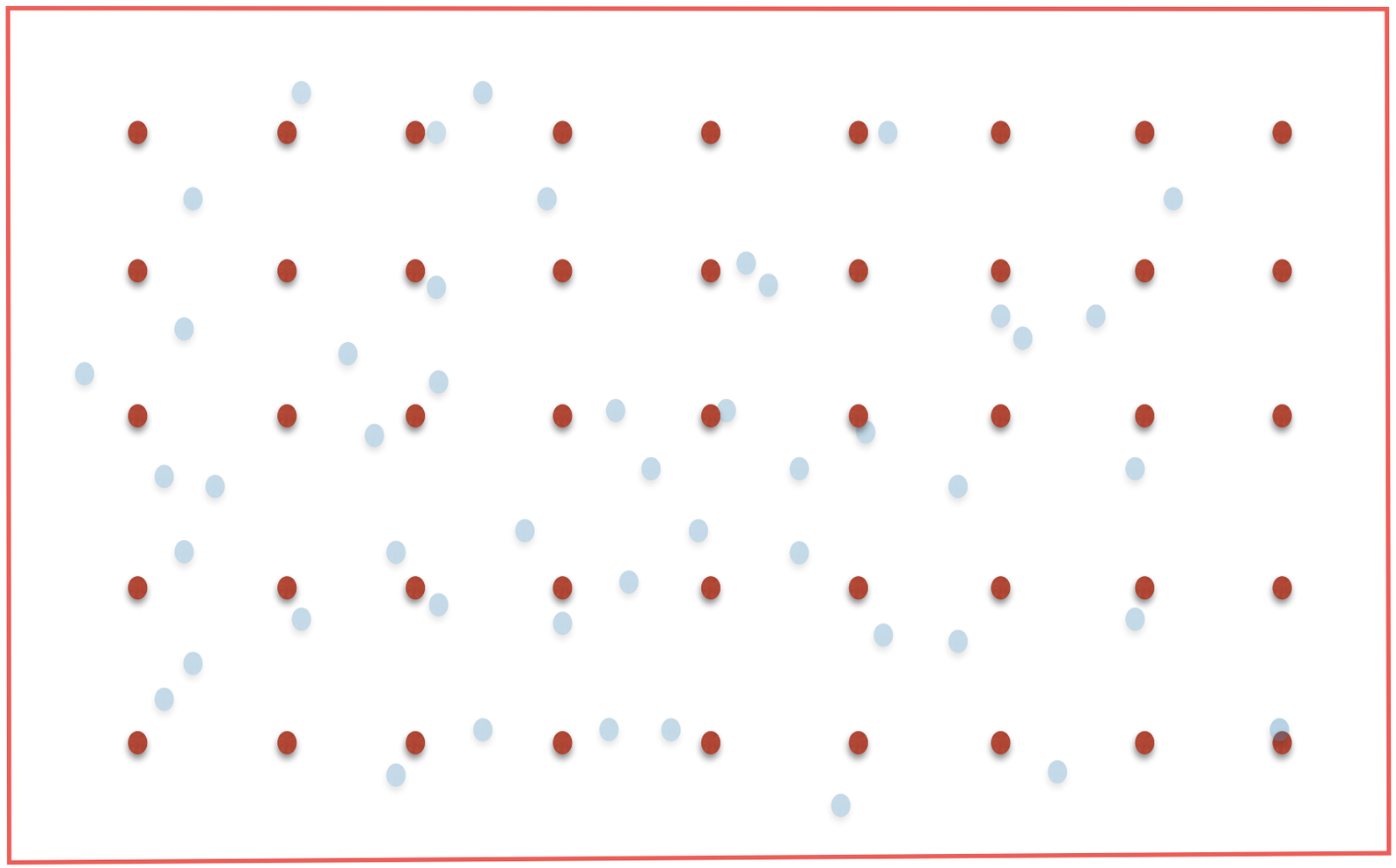
Given a point-cloud P and a desired grid spacing, compute a grid that represents P.

Brainstorming: Point-cloud-to-grid ?



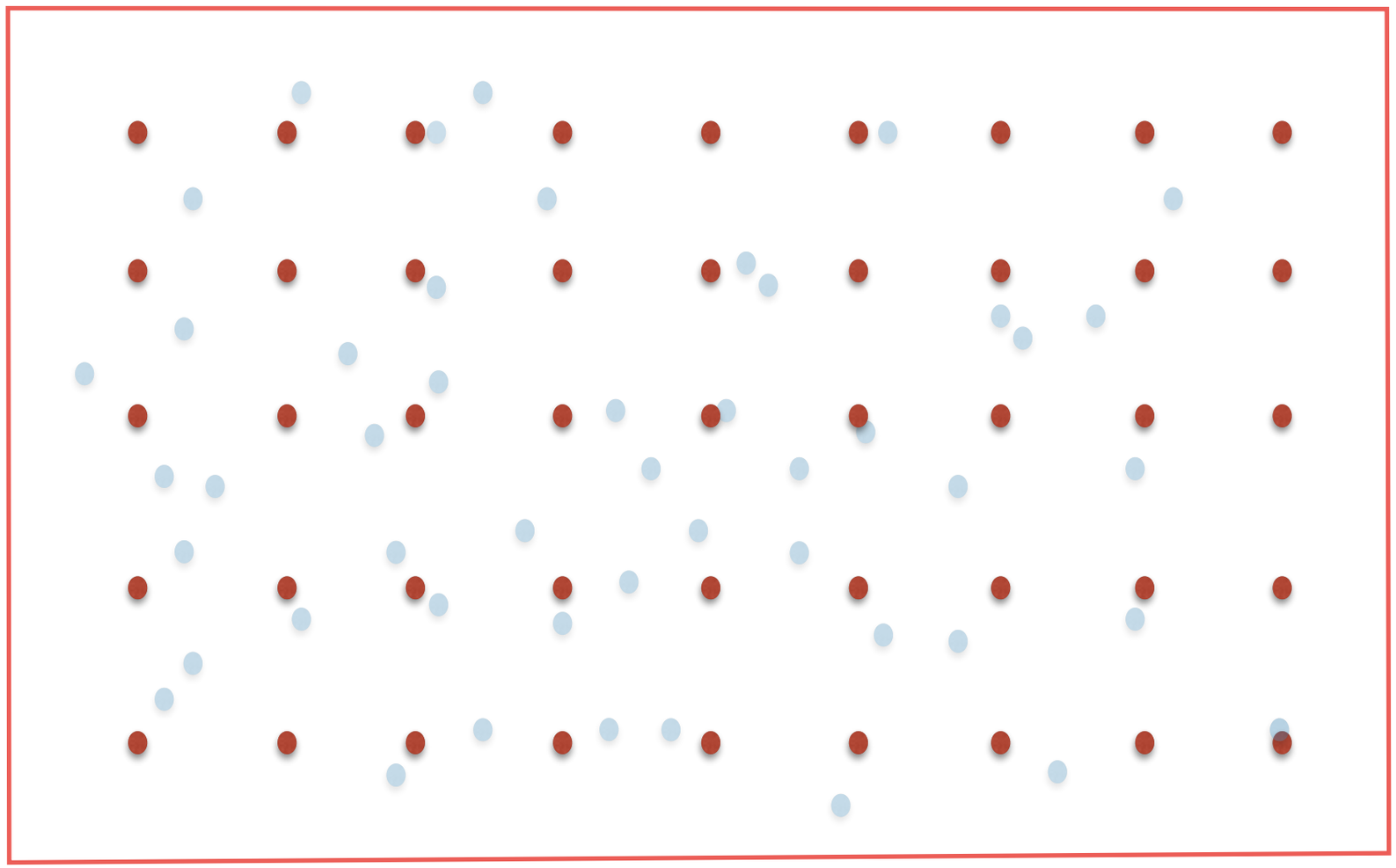
Given a point-cloud P and a desired grid spacing, compute a grid that represents P.

Brainstorming: Point-cloud-to-grid ?



Given a point-cloud P and a desired grid spacing, compute a grid that represents P.

Brainstorming: Point-cloud-to-grid ?



Sketch an algorithm to compute a grid given a point cloud and a desired resolution. Analyze it.