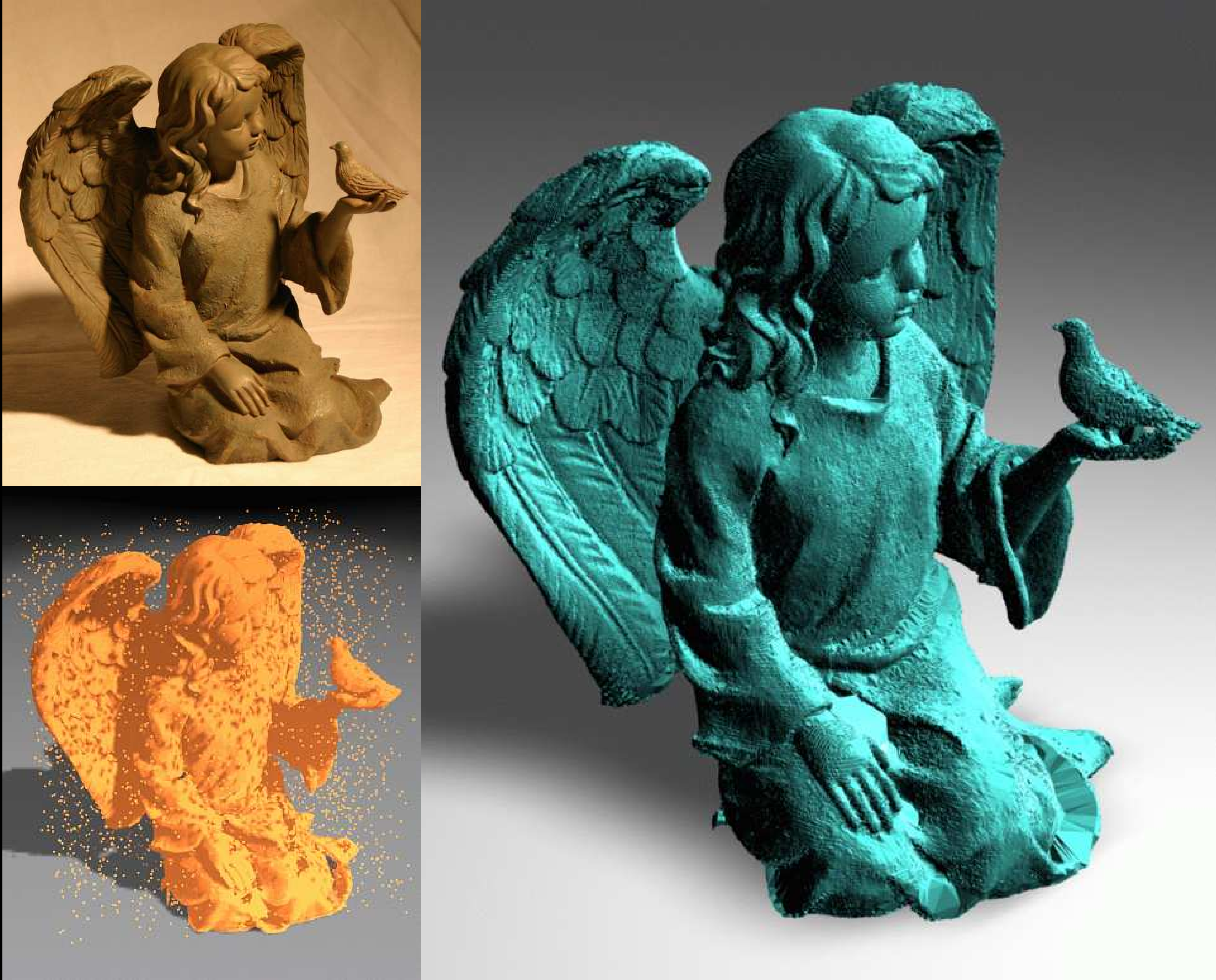
Spectral Surface Reconstruction from Noisy Point Clouds

Ravi Kolluri
Jonathan Shewchuk
James O'Brien

Computer Science Division
University of California
Berkeley, California



Surface Reconstruction from 3D Point Clouds

Input: Point cloud

Output: Surface triangulation

Previous Work

Pioneers:

Boissonnat (1984)

Hoppe−DeRose−Duchamp−McDonald−Stuetzle (1992)

Curless−Levoy (1996)

Implicit Surfaces:

Bittar−Tsingos−Gascuel (1995)

Carr−Beatson−Cherrie−Mitchell−Fright−et al. (2001)

Ohtake−Belyaev−Alexa−Turk−Seidel (2003)

Delaunay:

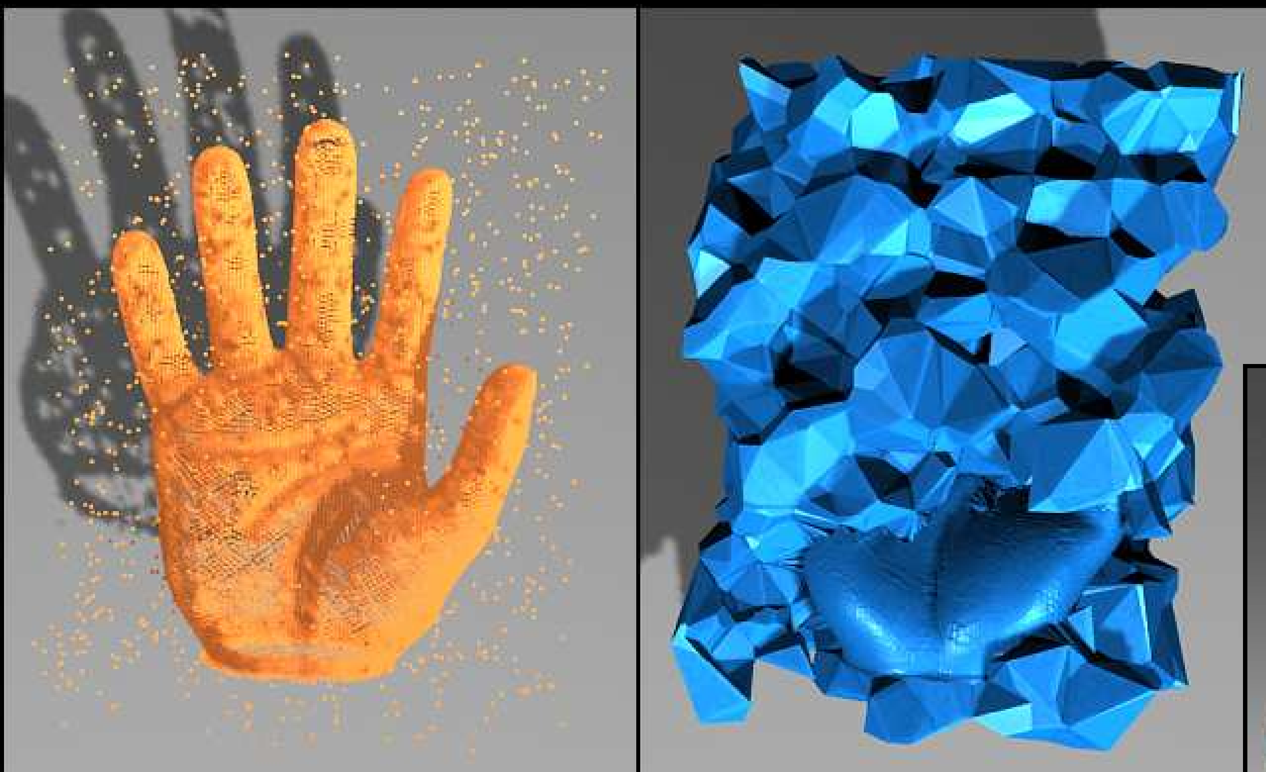
Amenta−Bern−Kamvysselis “Crust” (1998/1999)

Amenta−Choi−Kolluri “Powercrust” (2001)

Amenta−Choi−Dey−Leekha “Cocone” (2002)

Dey−Goswami “Tight Cocone” (2003)

Noise, Outliers, and Undersampling

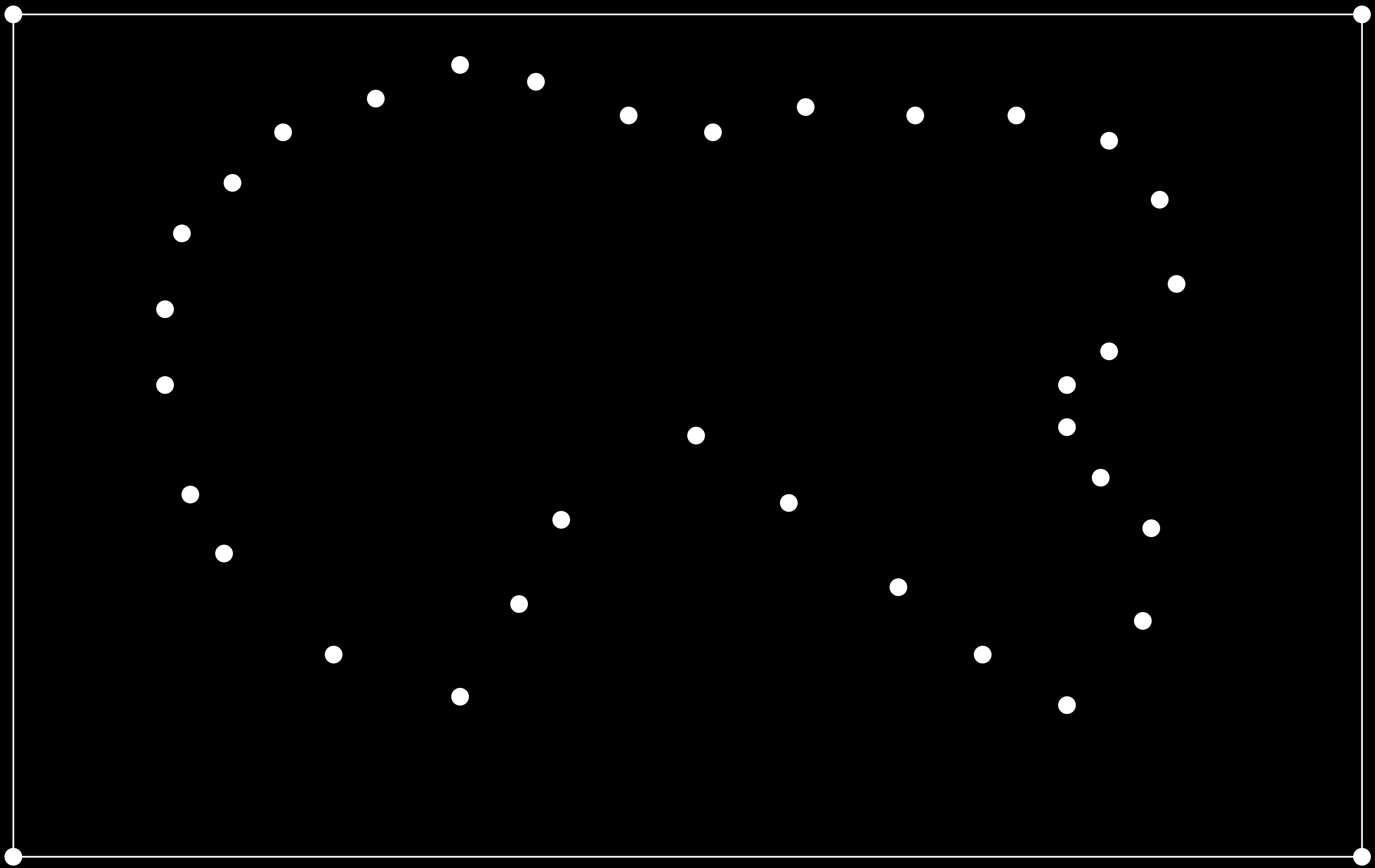


◀ Powercrust reconstruction of hand with outliers.

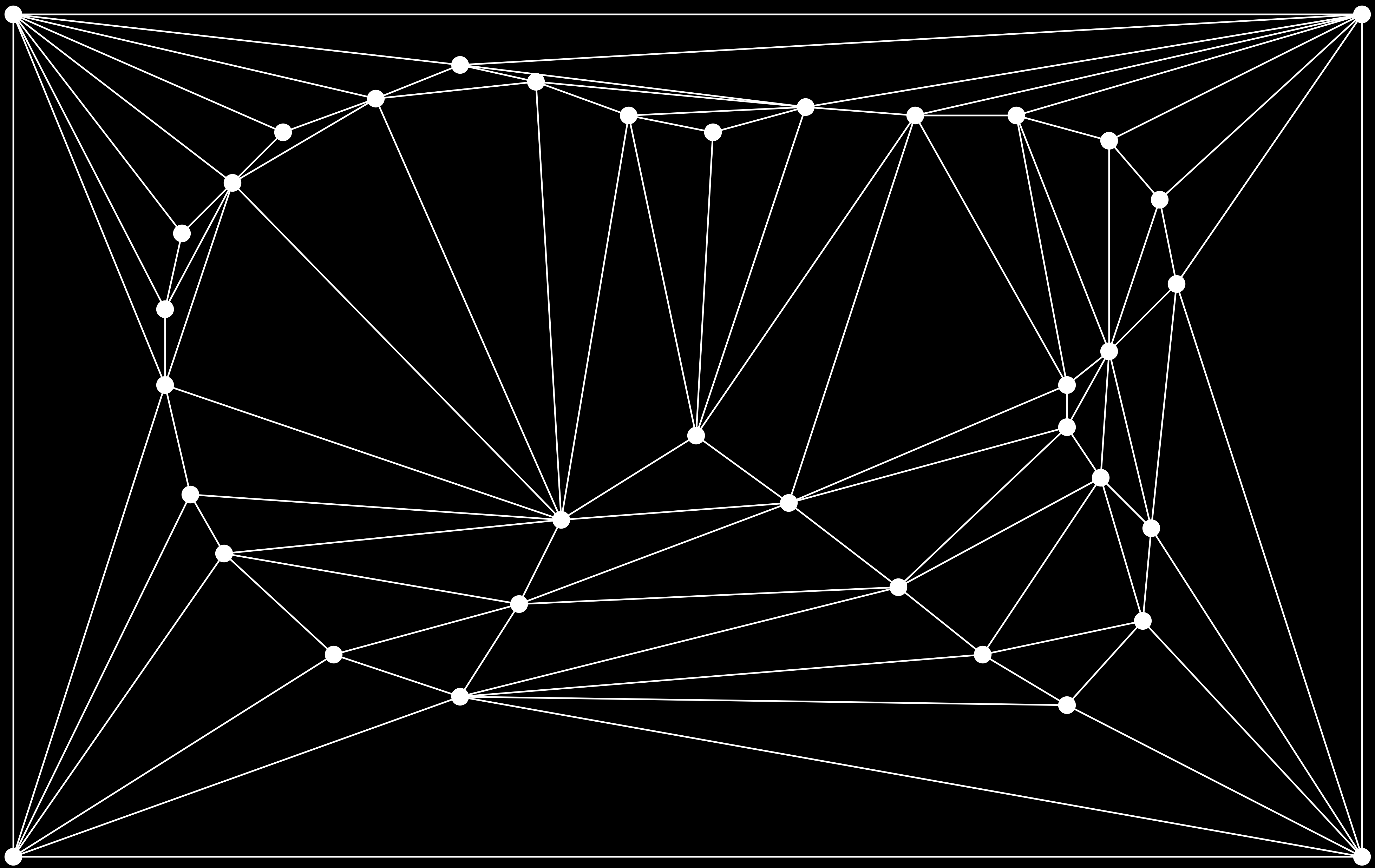
Tight Cocone reconstruction of Stanford Bunny with random noise in point coordinates. ▶

Our Approach

Add bounding box

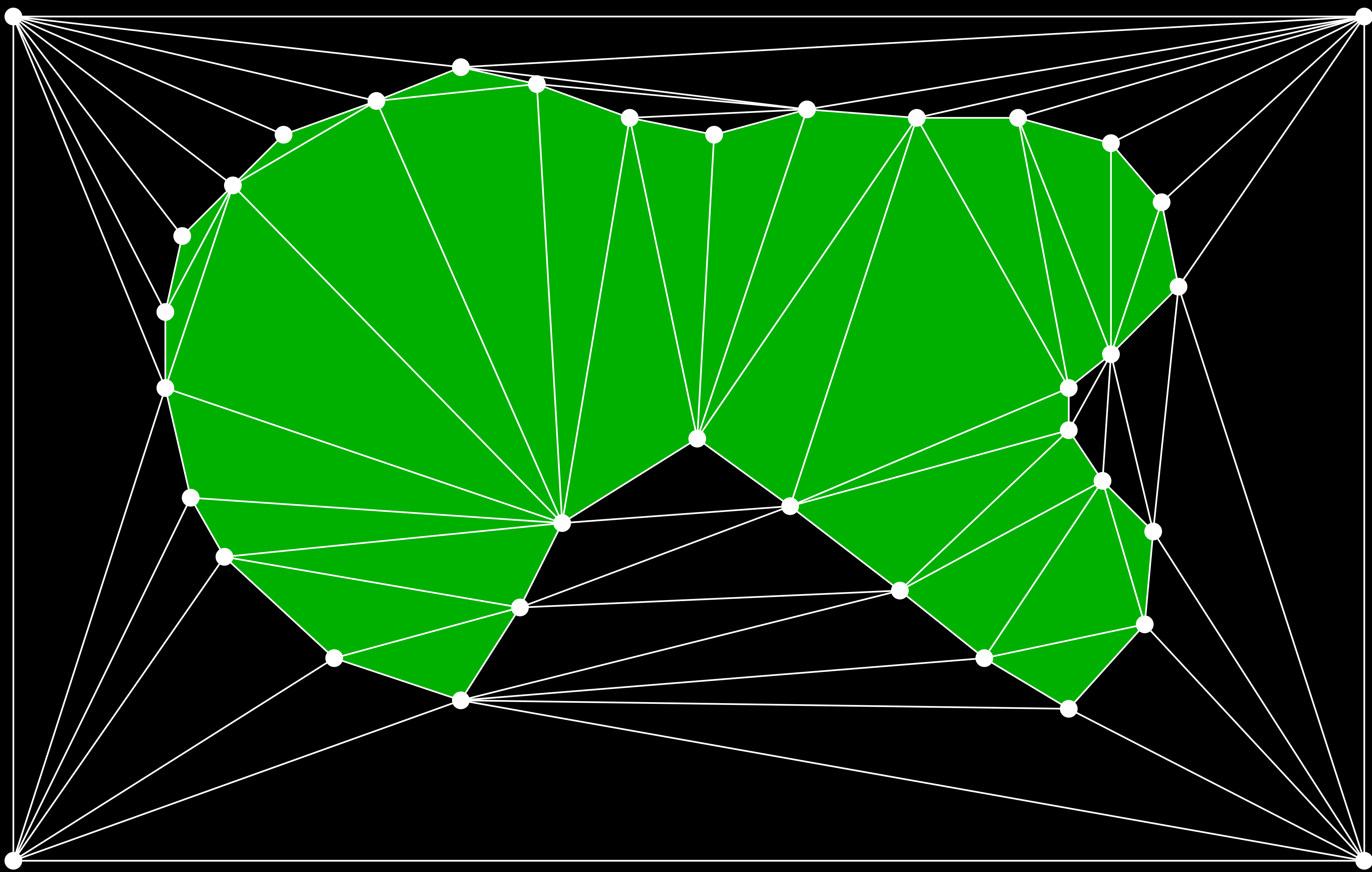


Our Approach Form Delaunay triangulation



Our Approach

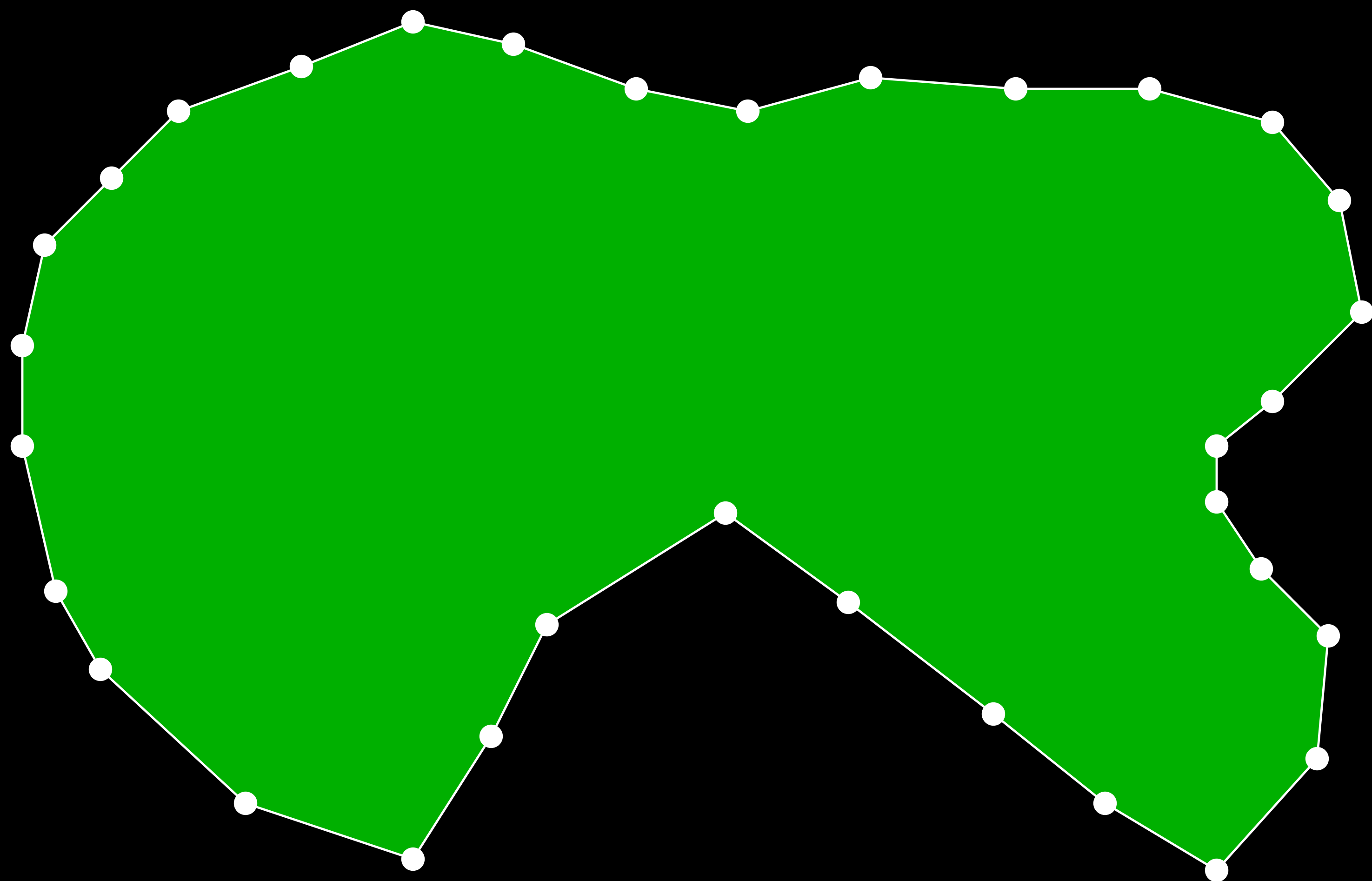
▲ : Inside △ : Outside



Our Approach (Boissonnat 1984)

Output surface

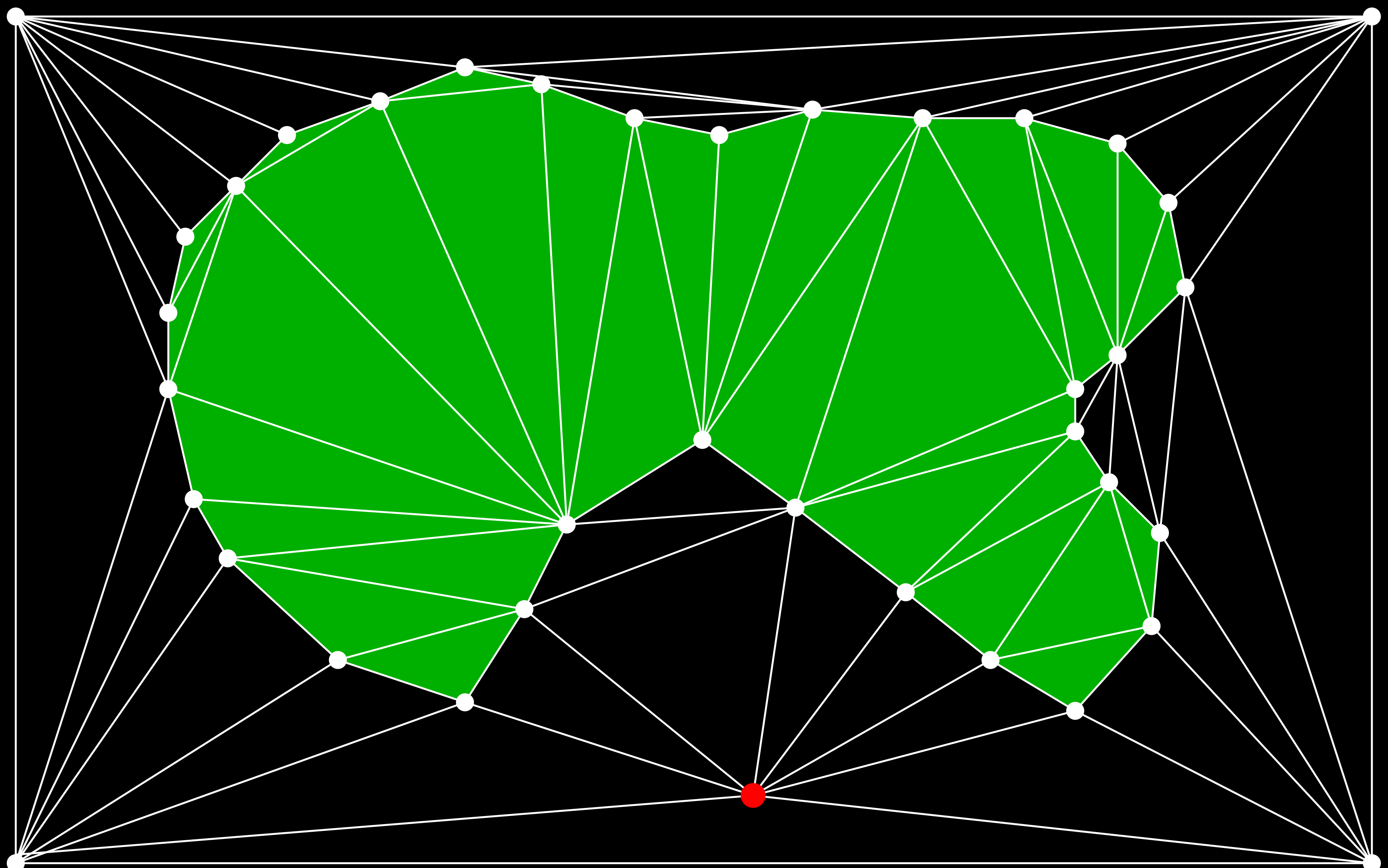
(Always watertight!)



Why Use Delaunay?

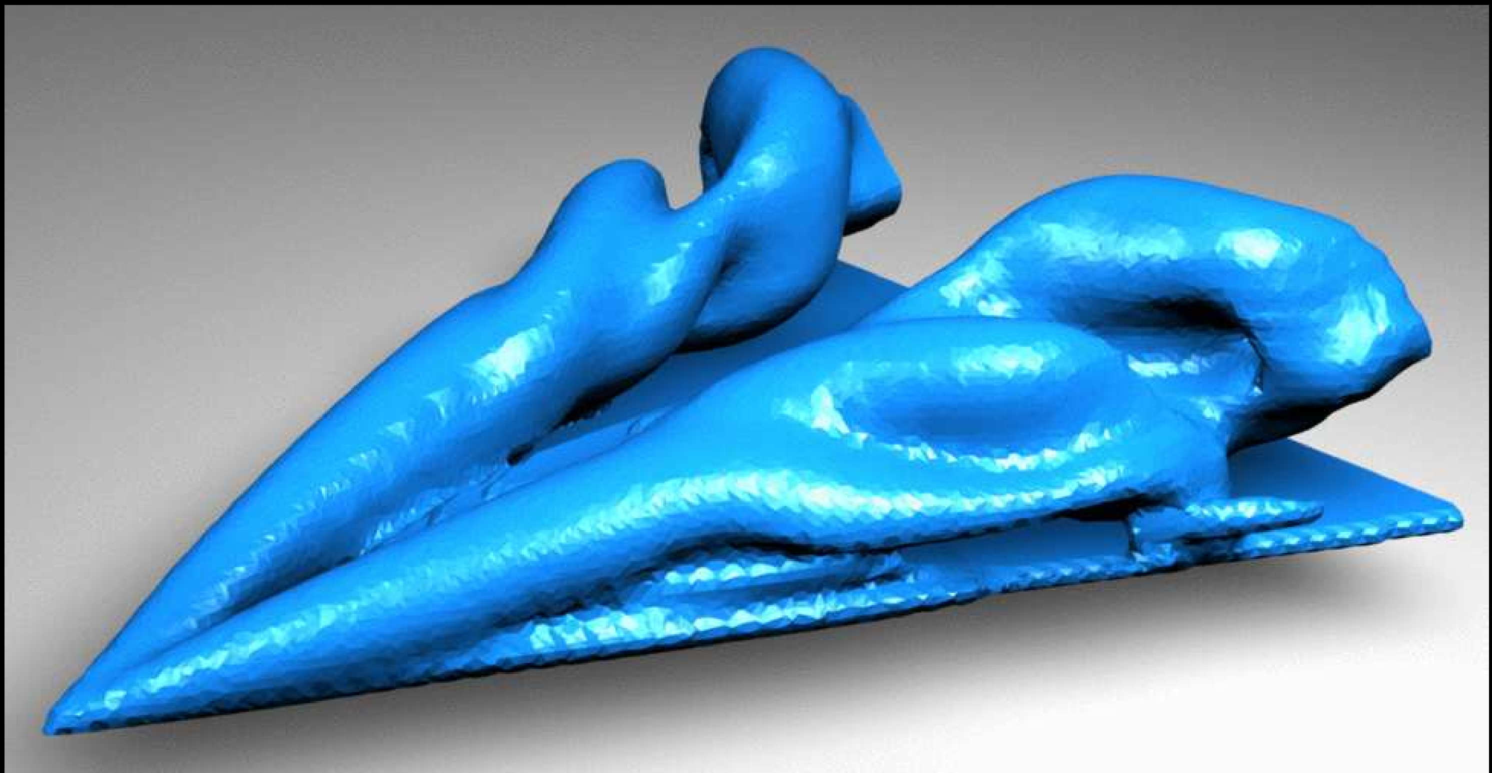
Why Use Delaunay?

- Effortless watertightness & outlier removal.



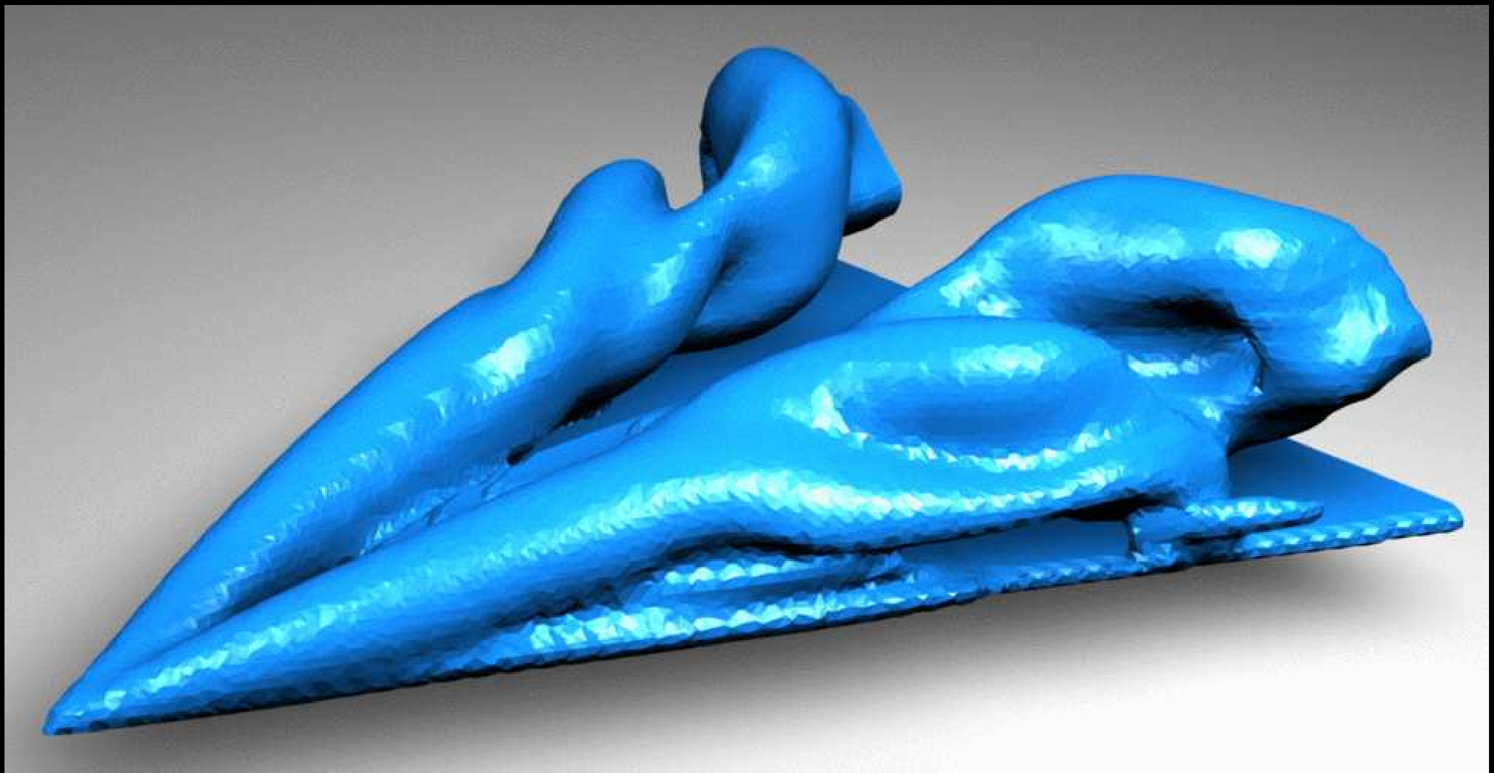
Why Use Delaunay?

- Effortless watertightness & outlier removal.
- Many Delaunay algorithms are provably correct.



Why Use Delaunay?

- Effortless watertightness & outlier removal.
- Many Delaunay algorithms are provably correct...in the absence of noise, outliers, undersampling.



Why Use Delaunay?

- Effortless watertightness & outlier removal.
- Many Delaunay algorithms are provably correct...in the absence of noise, outliers, undersampling.

Our Goal

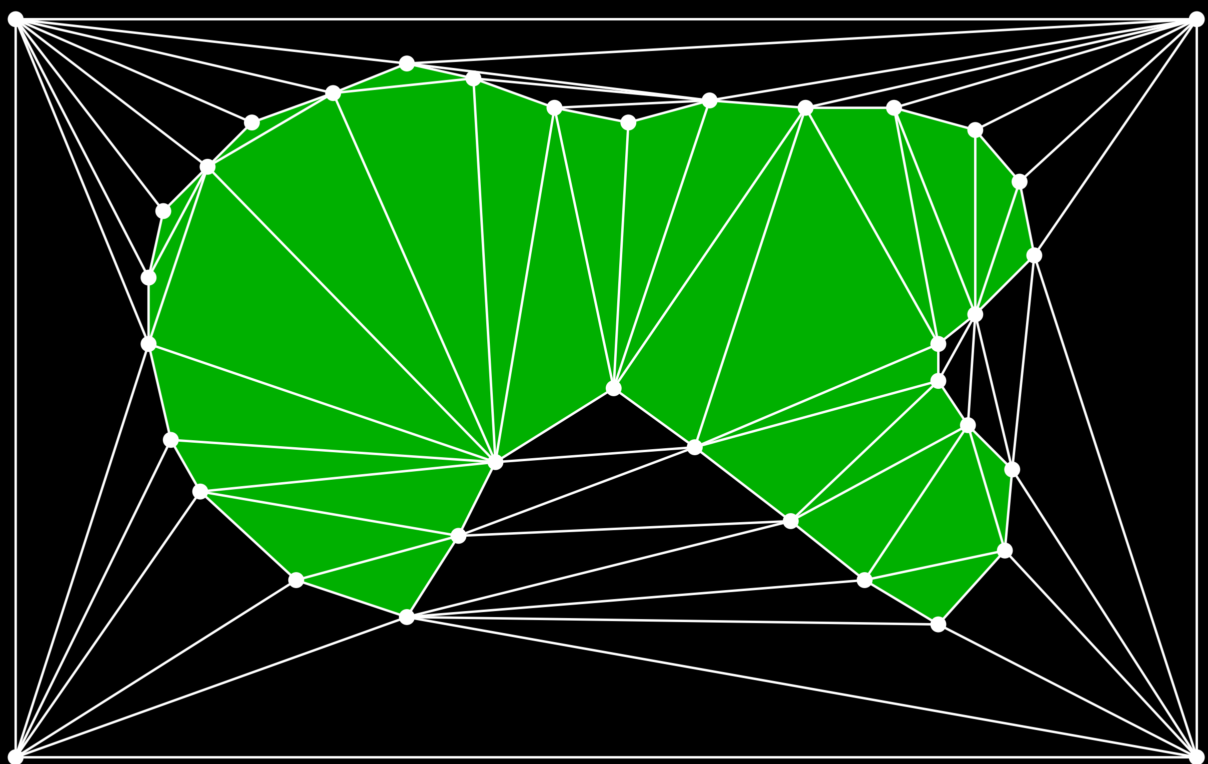
- Achieve same results (in practice) as Cocone algorithm on “clean” point clouds; better results otherwise.

Why Use Delaunay?

- Because we can make it robust against noise, outliers, and undersampling.

Central Idea

Use spectral graph partitioning to decide which Delaunay tetrahedra are inside/outside the object.



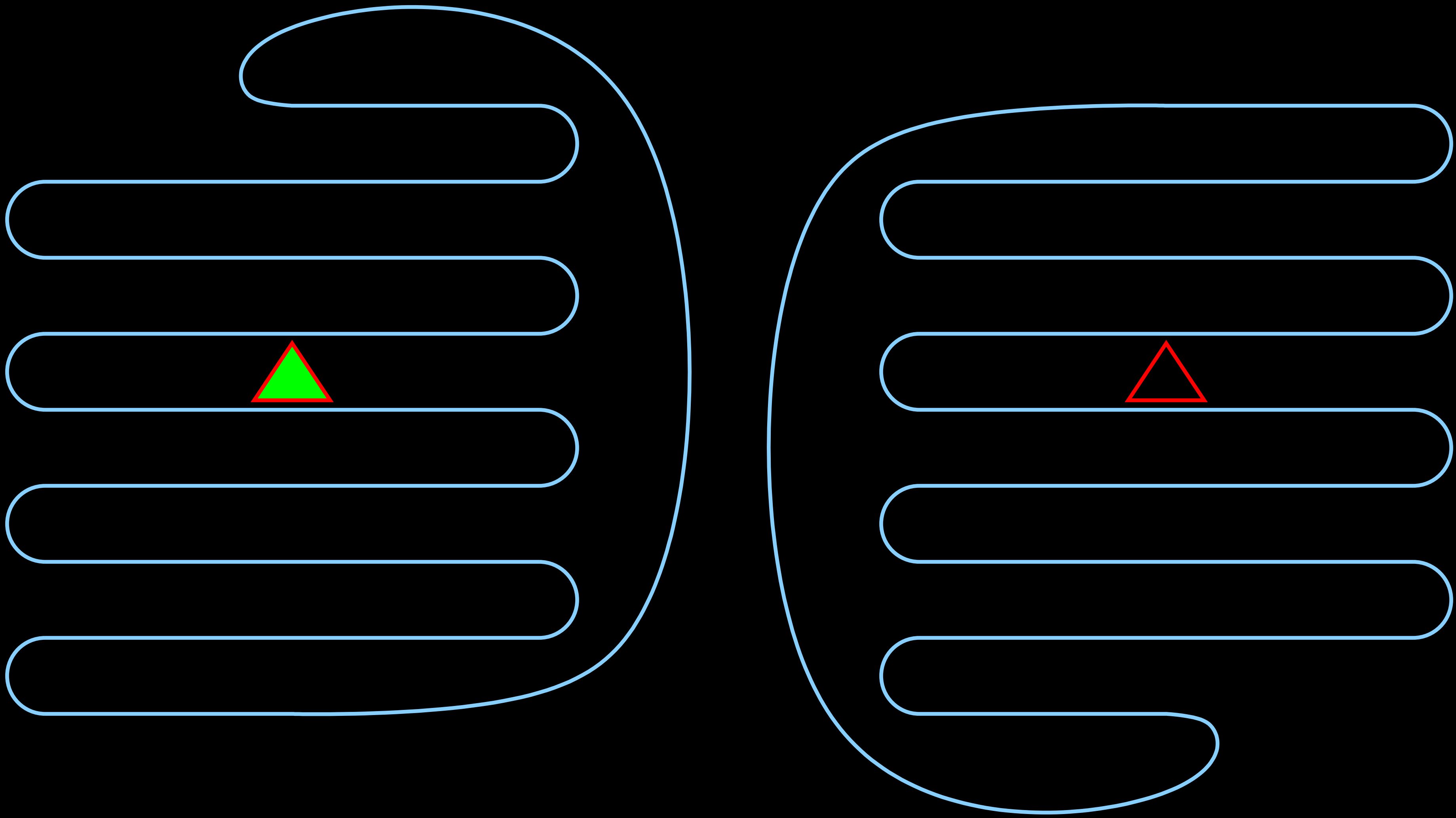
Central Idea

Use spectral graph partitioning to decide which Delaunay tetrahedra are inside/outside the object.

And One Little Idea

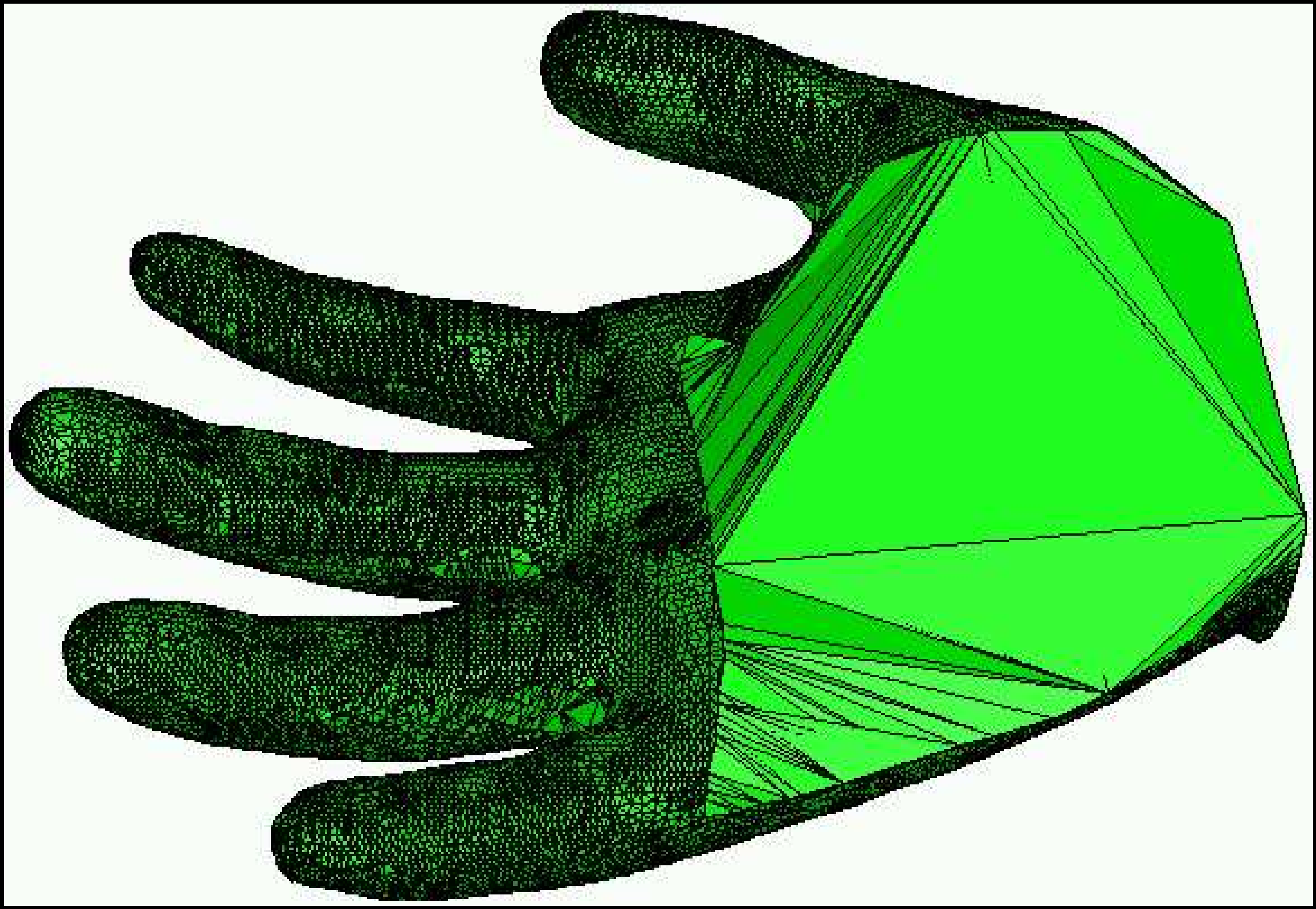
Use negative edge weights to make the partitioner robust and fast.

The Partitioner Has a Global View



Inside or outside?

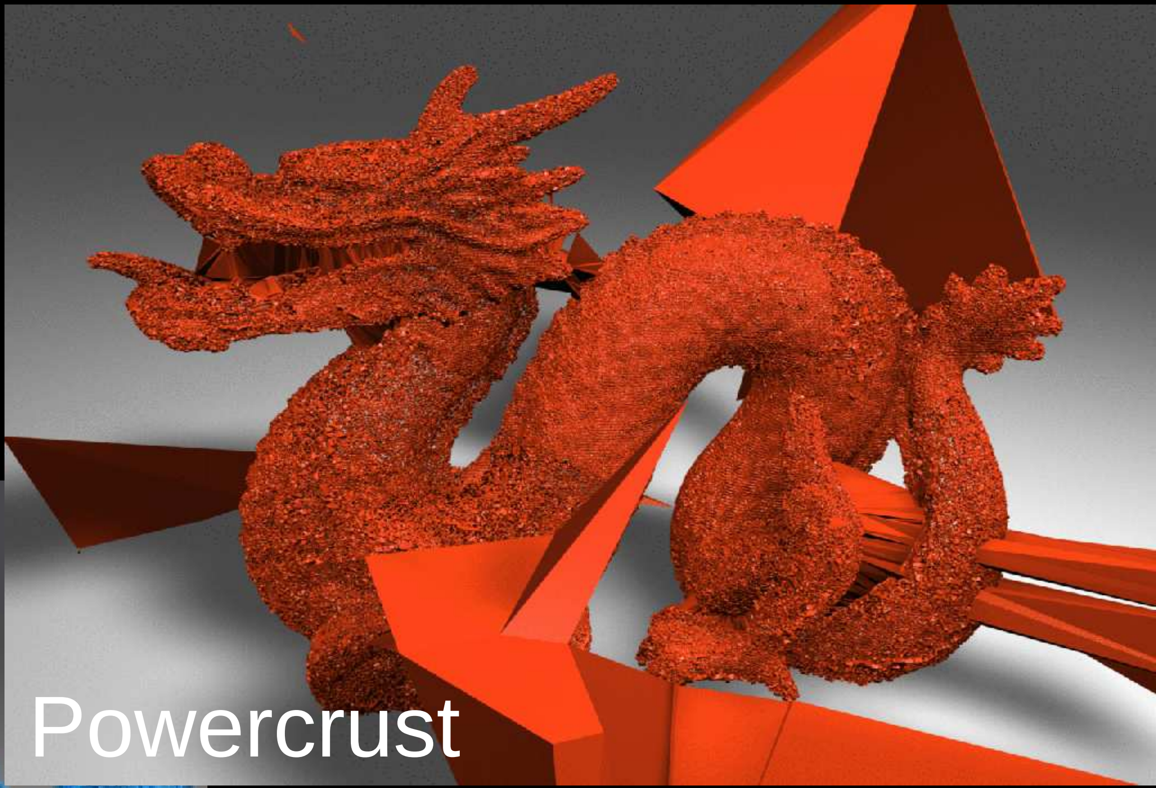
The Partitioner Has a Global View

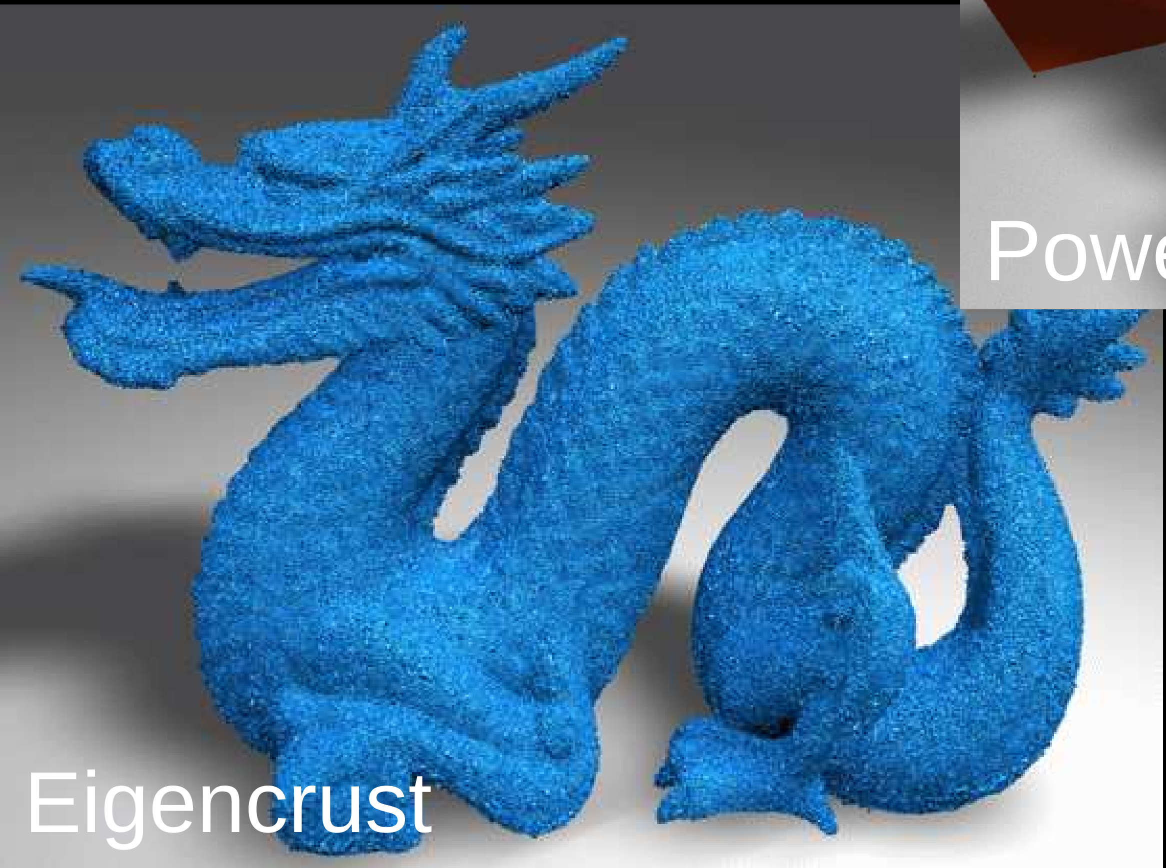


Eigencrust reconstruction of undersampled hand.

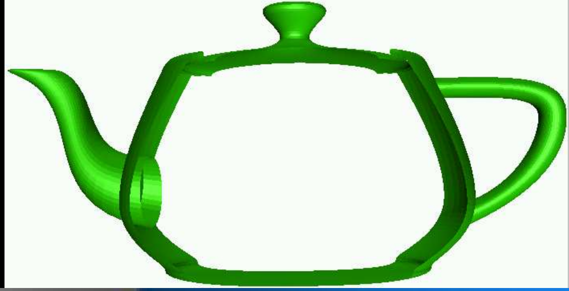
The Partitioner Has a Global View

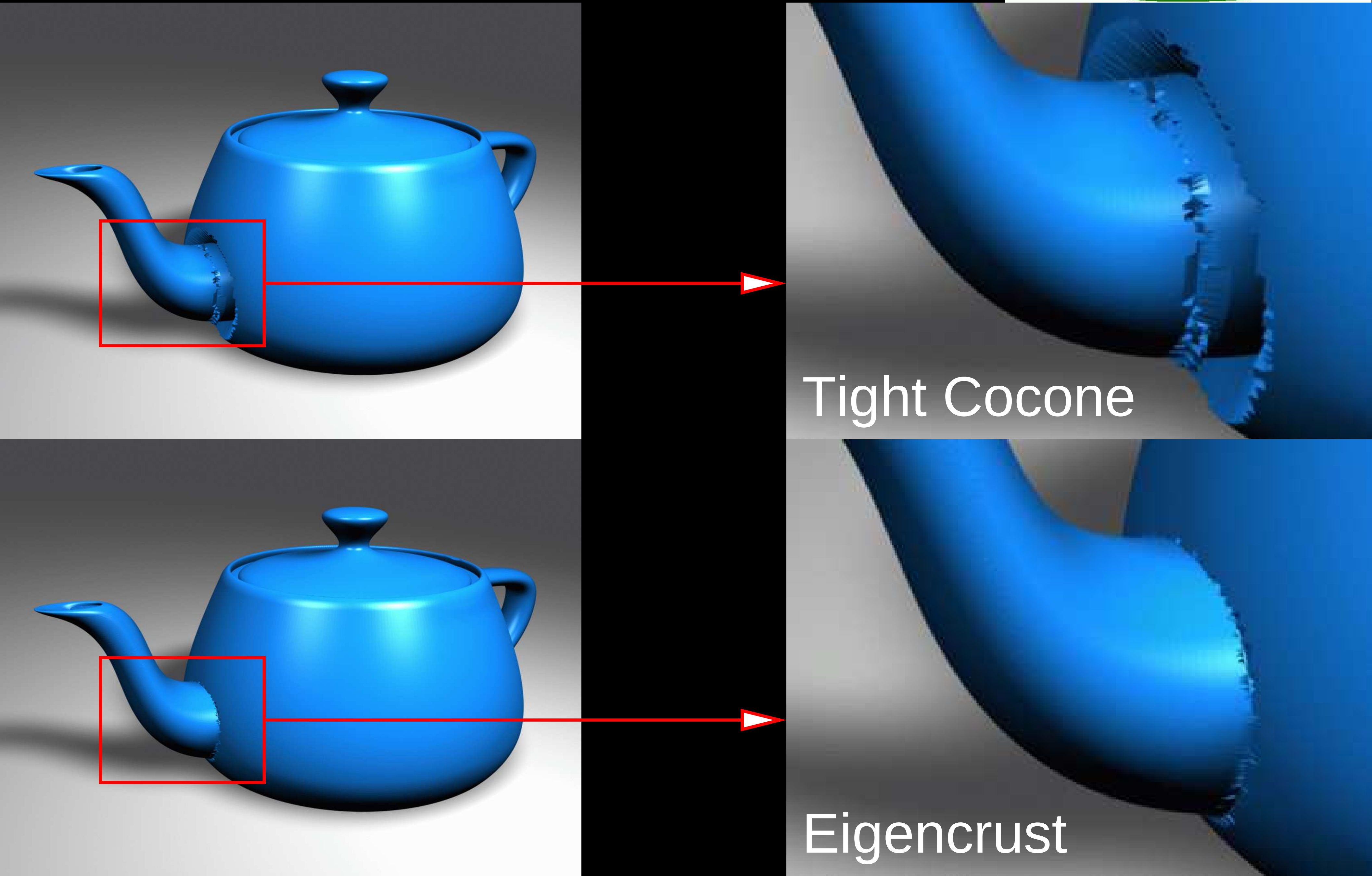
and can make better sense of outliers.





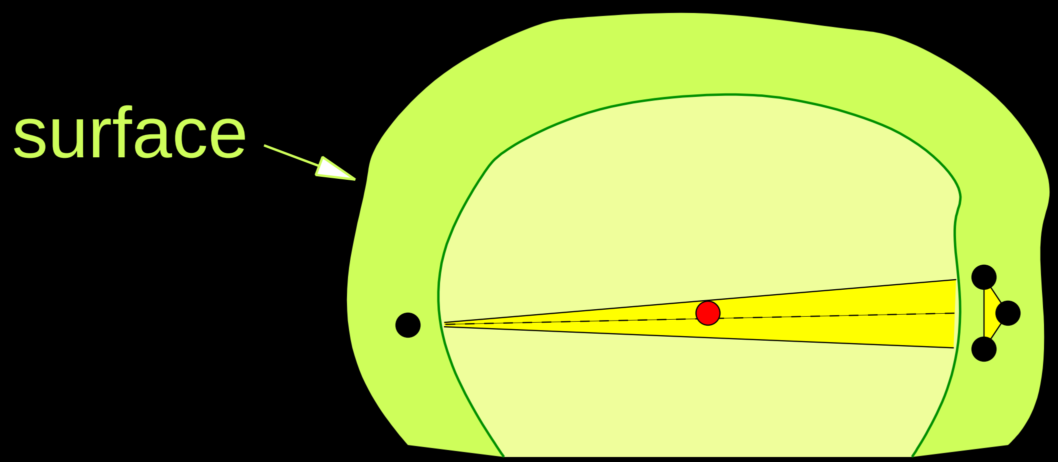
The Partitioner Has a Global View





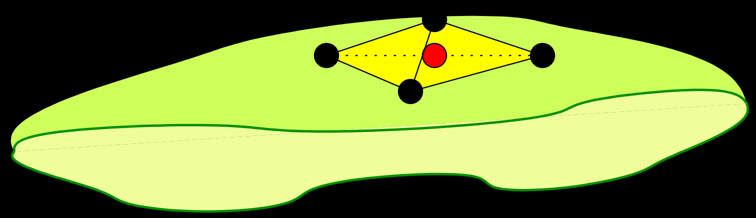
Eigencrust

Some tetrahedra are easy to classify.



Obviously inside.

Some are ambiguous.

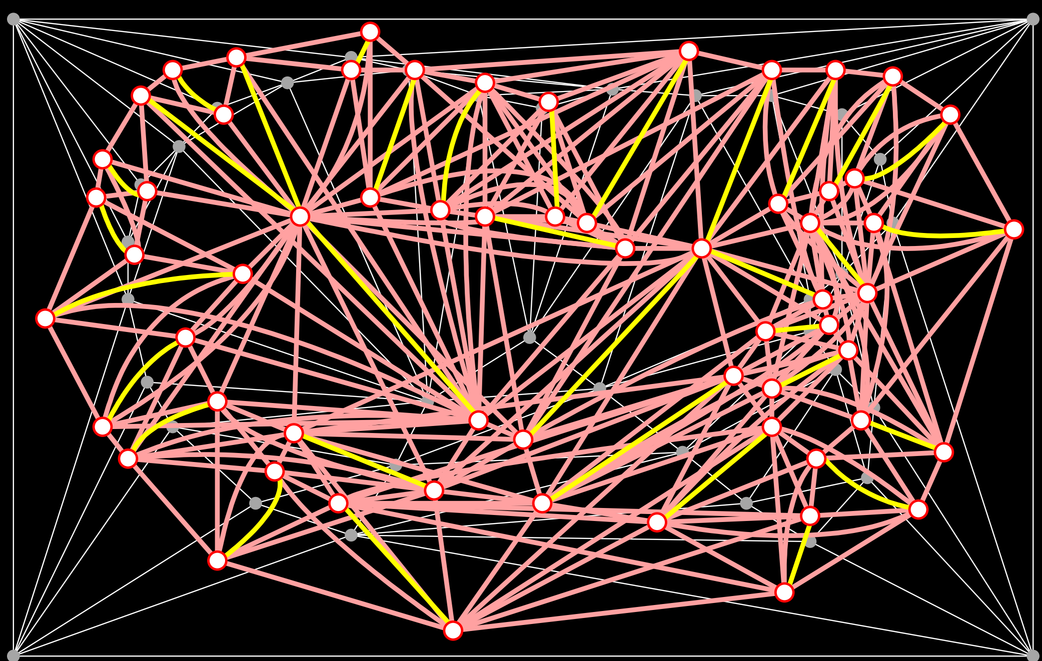


Could be labeled inside or outside.

Eigencrust Algorithm

Stage 1:

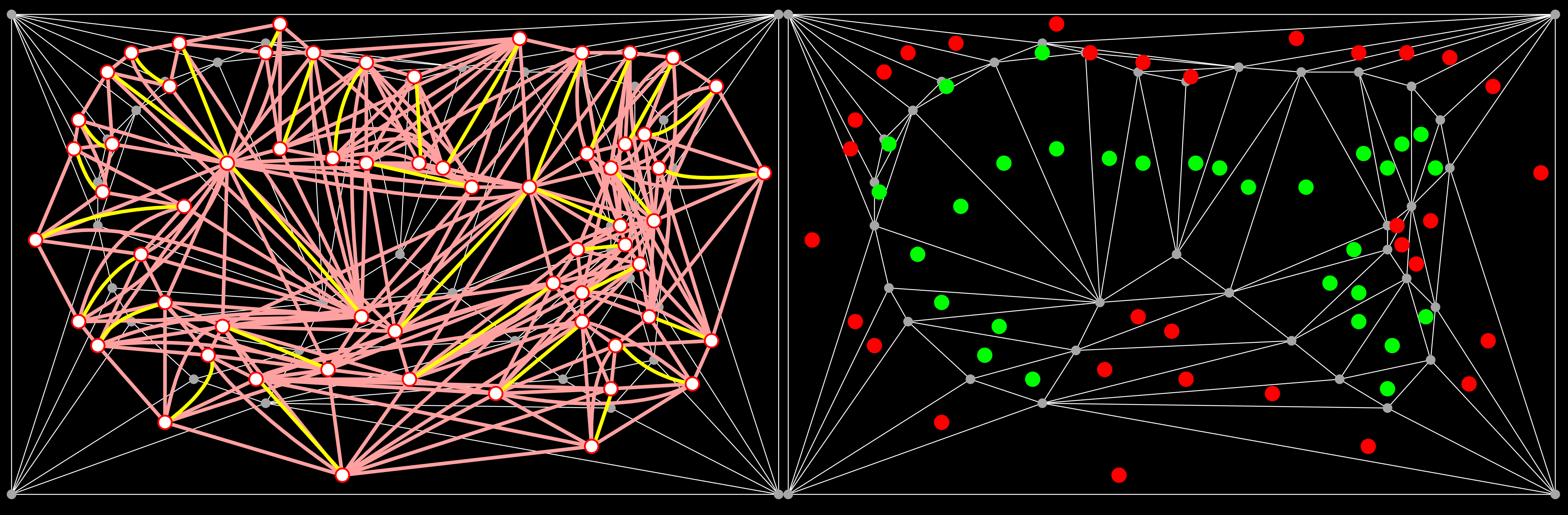
- Identify non−ambiguous tetrahedra called “poles”.
- Form a graph whose vertices are the poles.
- Assign edge weights based on geometry.



Eigencrust Algorithm

Stage 1:

- Identify non−ambiguous tetrahedra called “poles”.
- Form a graph whose vertices are the poles.
- Assign edge weights based on geometry.
- Partition graph.



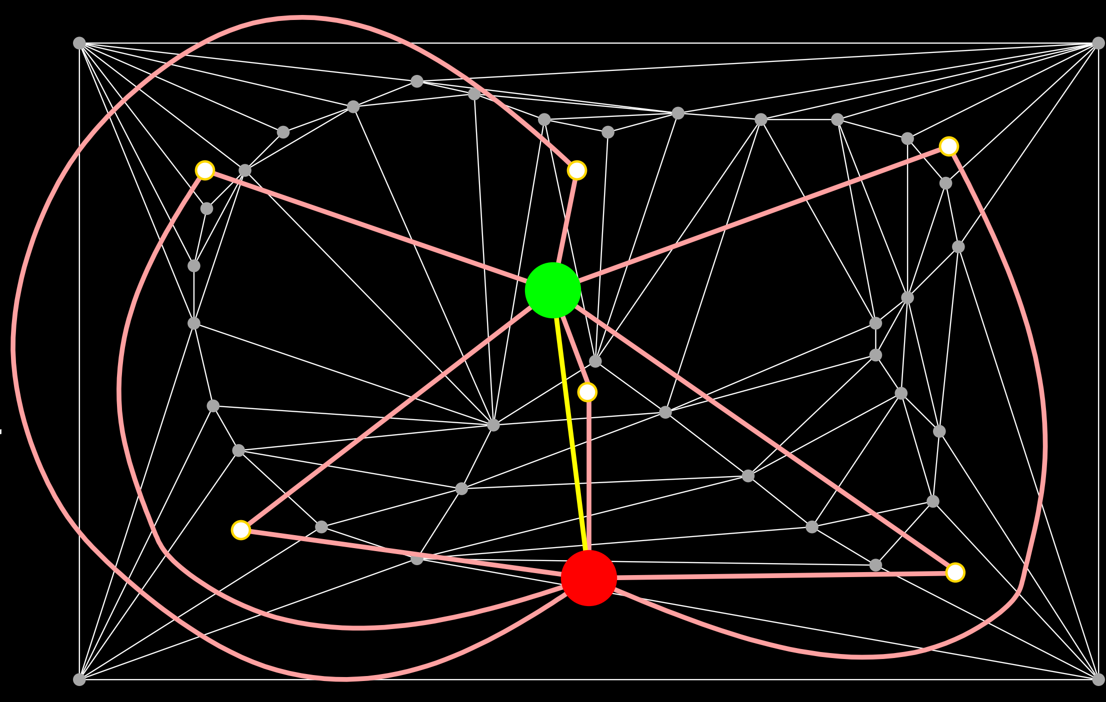
Eigencrust Algorithm

Stage 1:

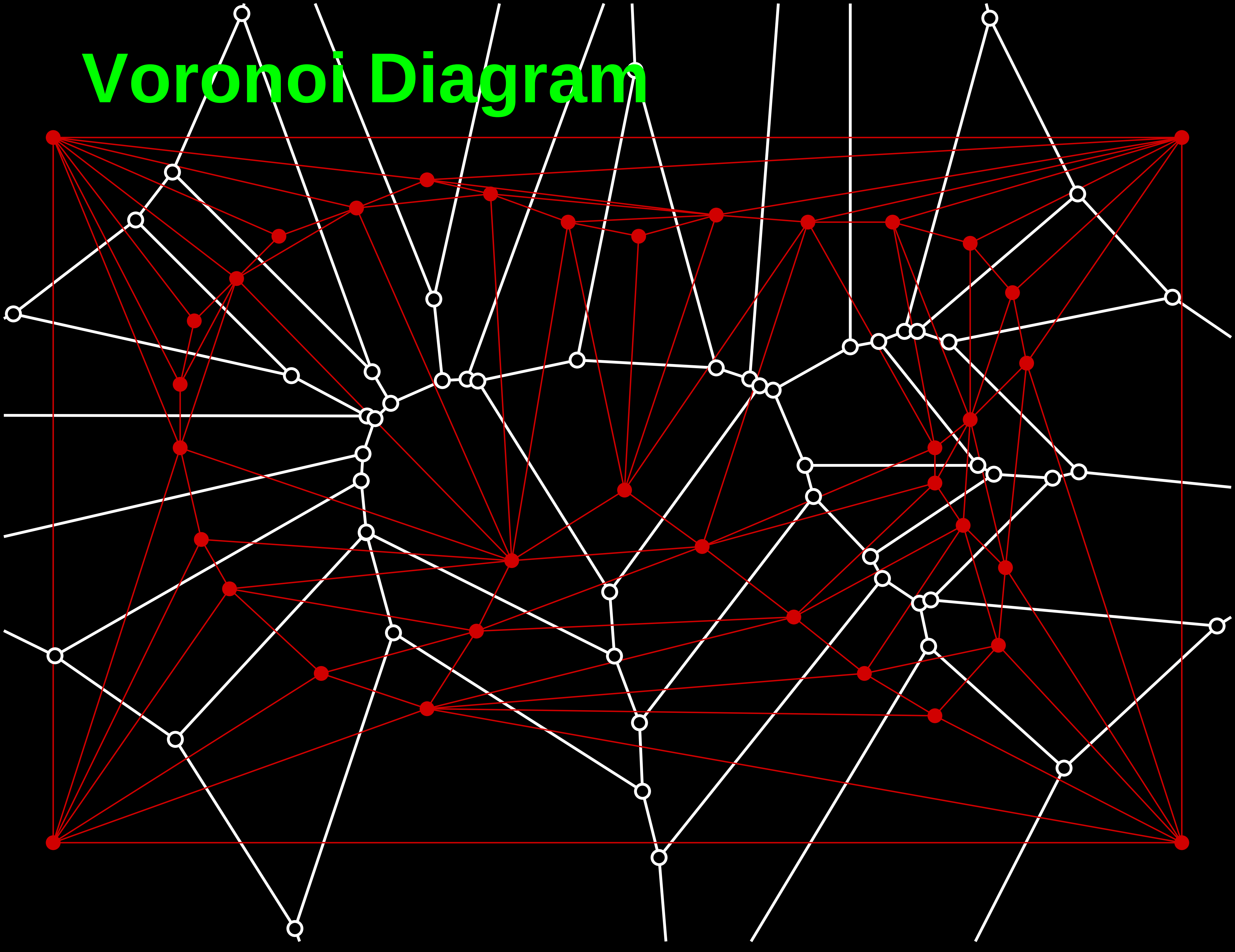
- Identify non−ambiguous tetrahedra called “poles”.
- Form a graph whose vertices are the poles.
- Assign edge weights based on geometry.
- Partition graph.

Stage 2:

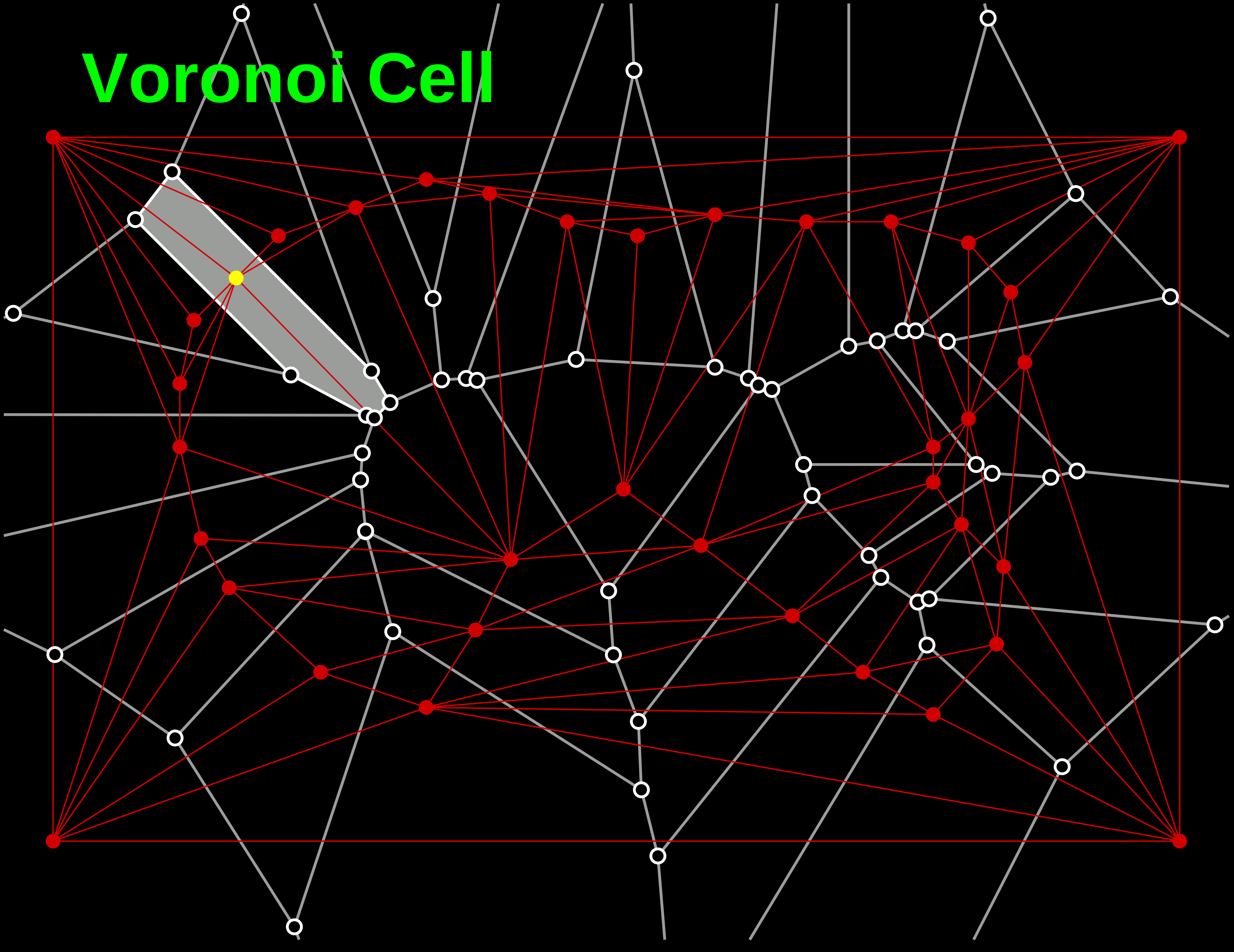
- Form a graph whose vertices are the ambiguous tetrahedra (non−poles).
- Form graph, partition.



Voronoi Diagram

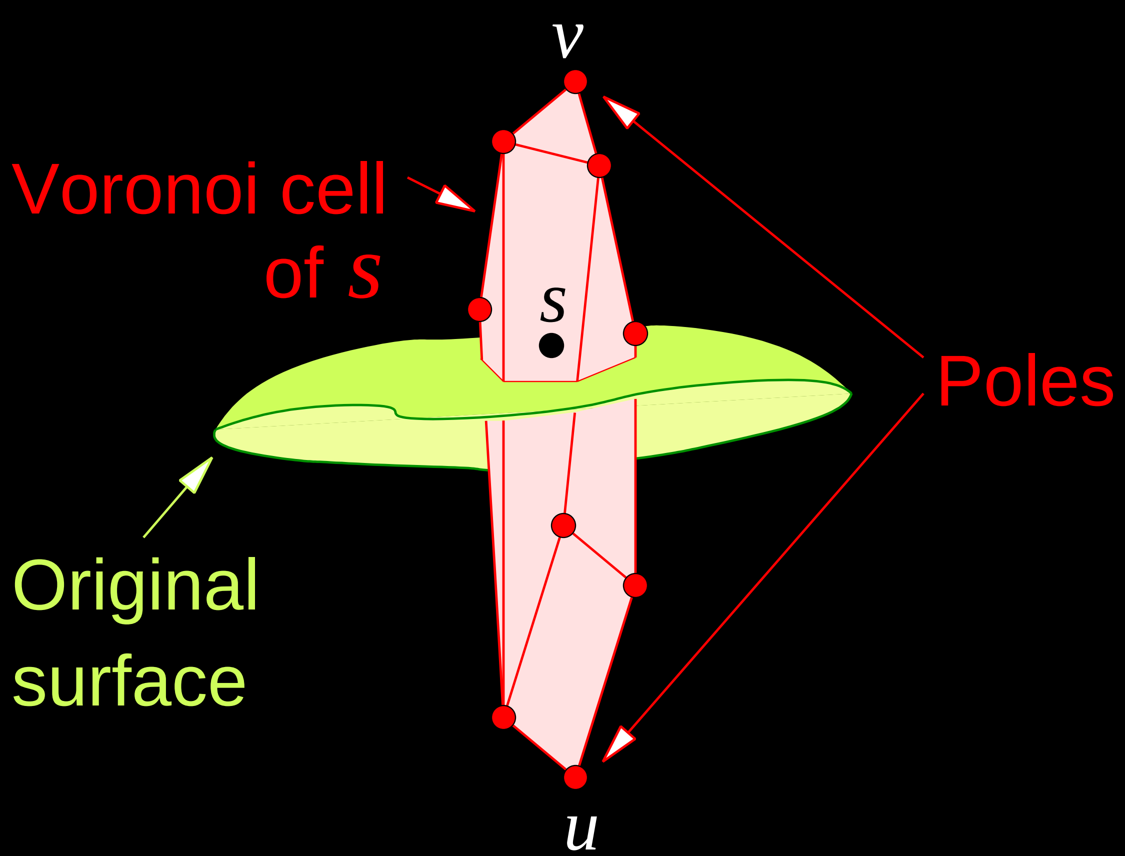


Voronoi Cell



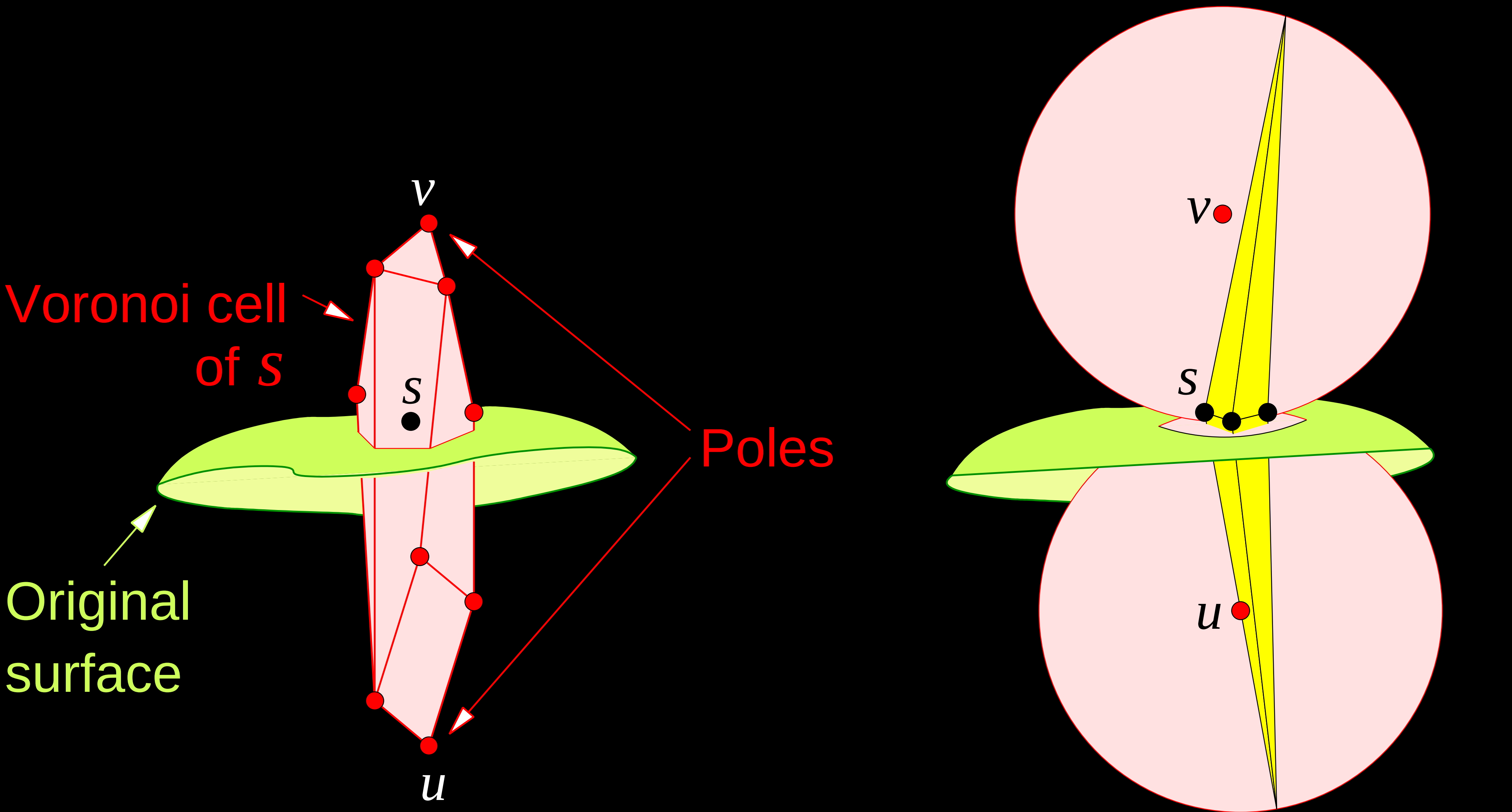
Poles

Poles of a sample point are likely to be on opposite sides of surface. (Amenta–Bern 1999.)



Poles

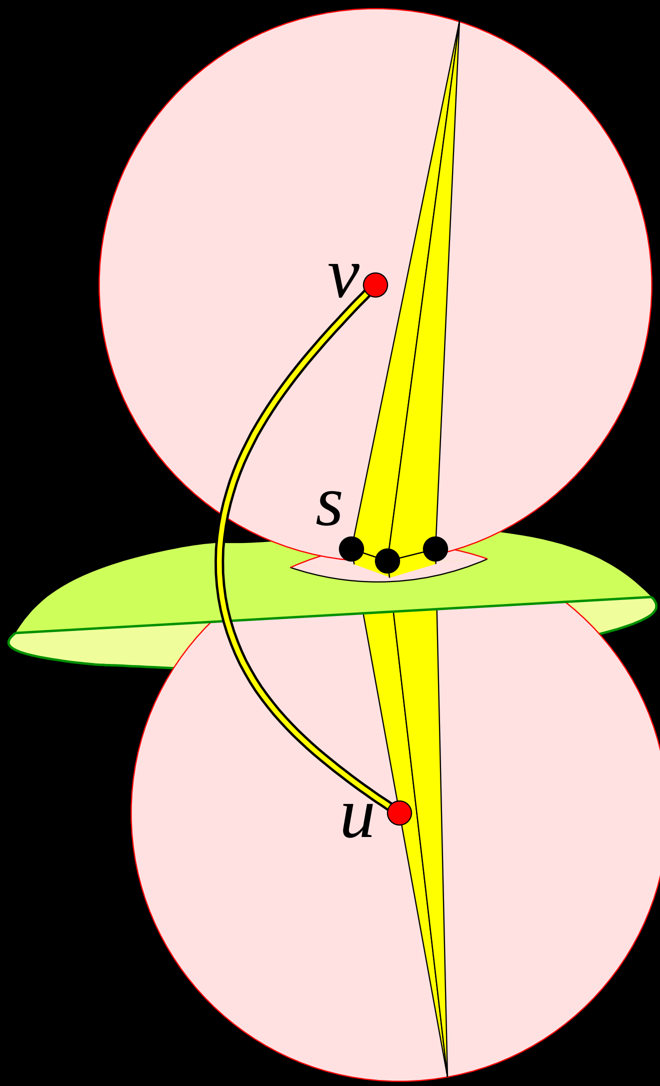
Poles of a sample point are likely to be on opposite sides of surface.



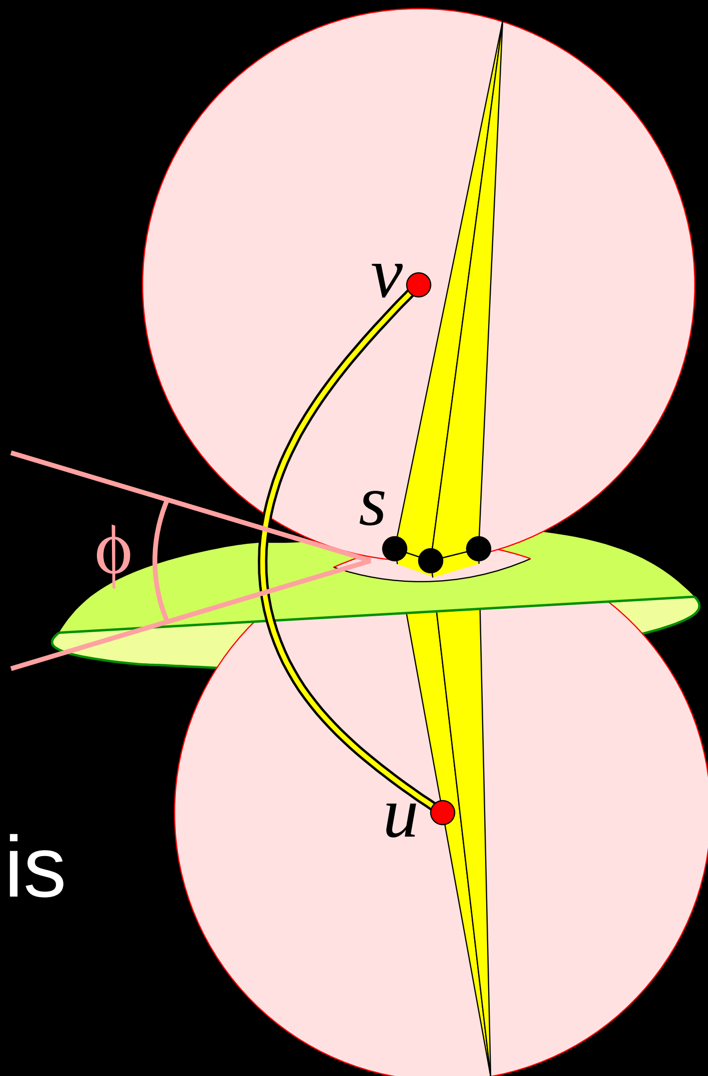
Poles

Poles of a sample point are likely to be on opposite sides of surface.

Connect them them with a negative−weight edge.



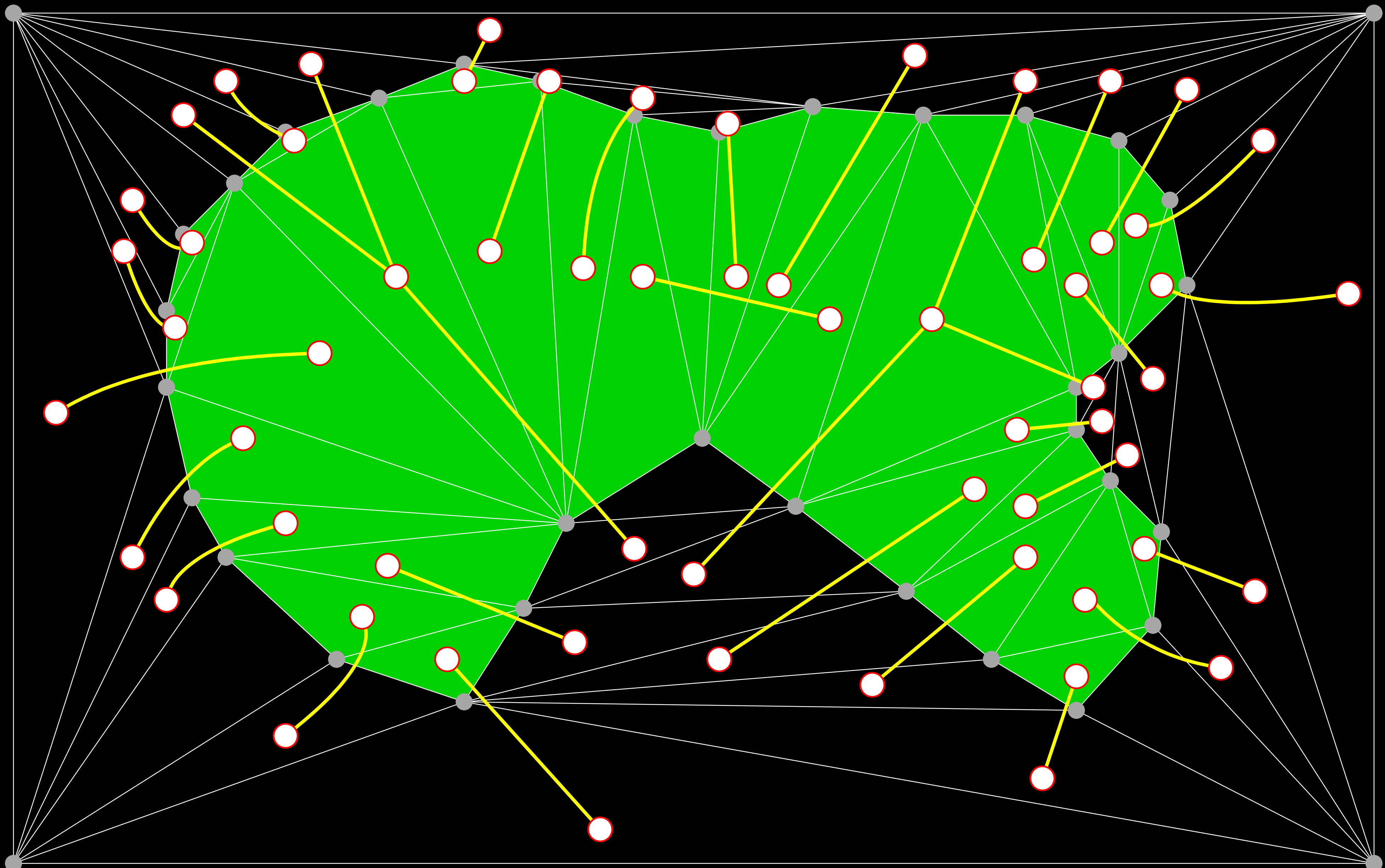
Poles



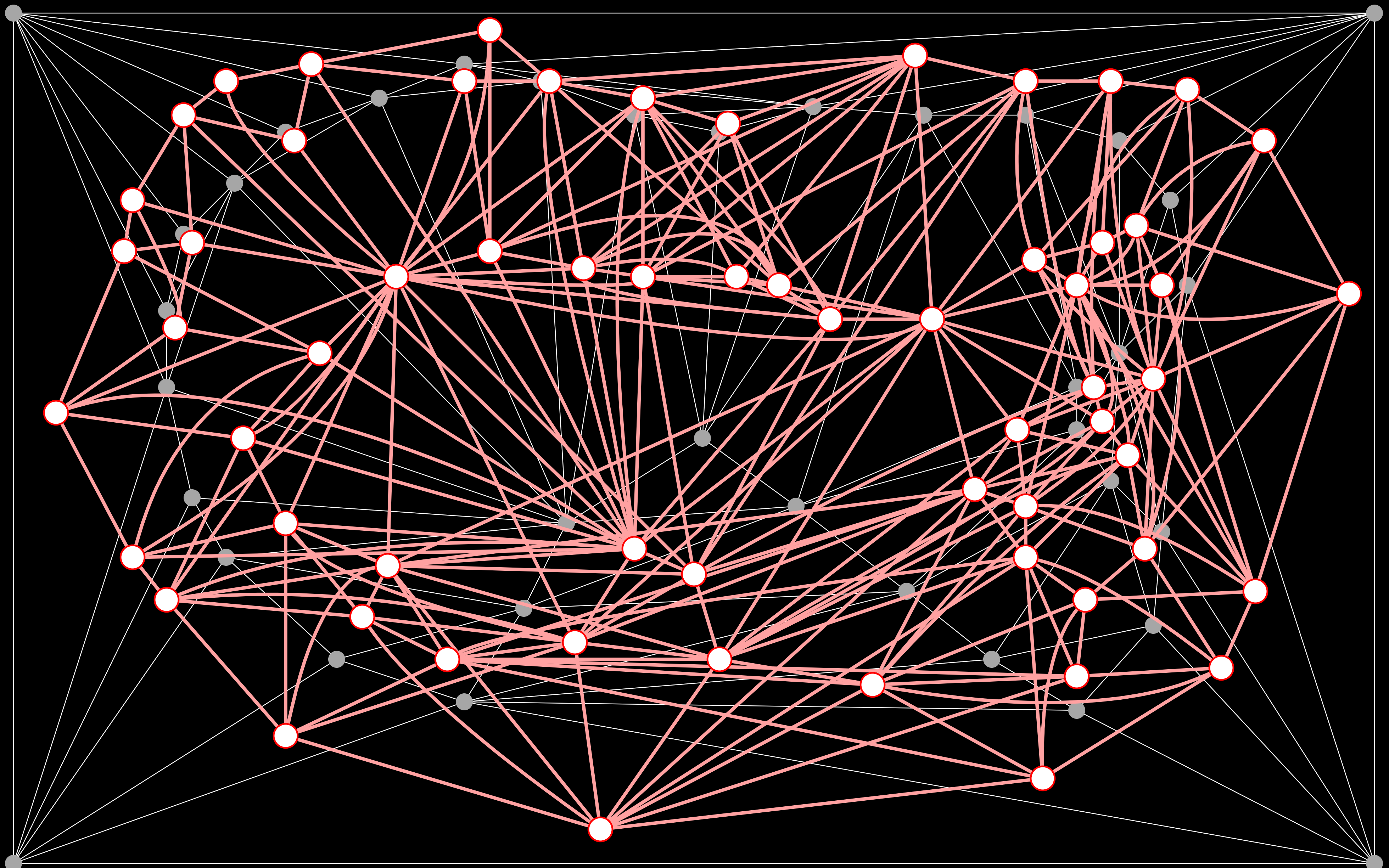
Weight of edge is

$$-e^{4+4\cos\phi}$$

Negative weight edges

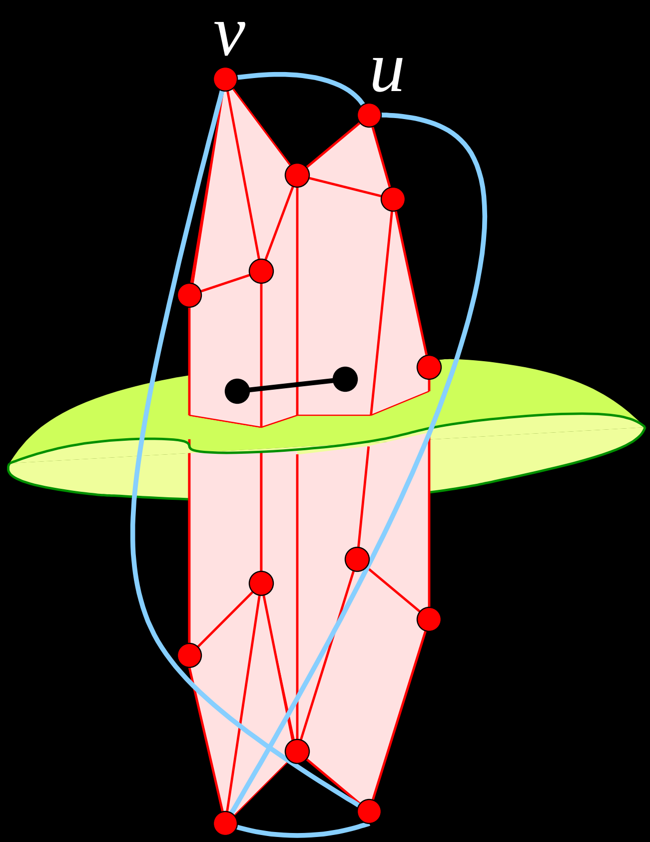


Positive weight edges



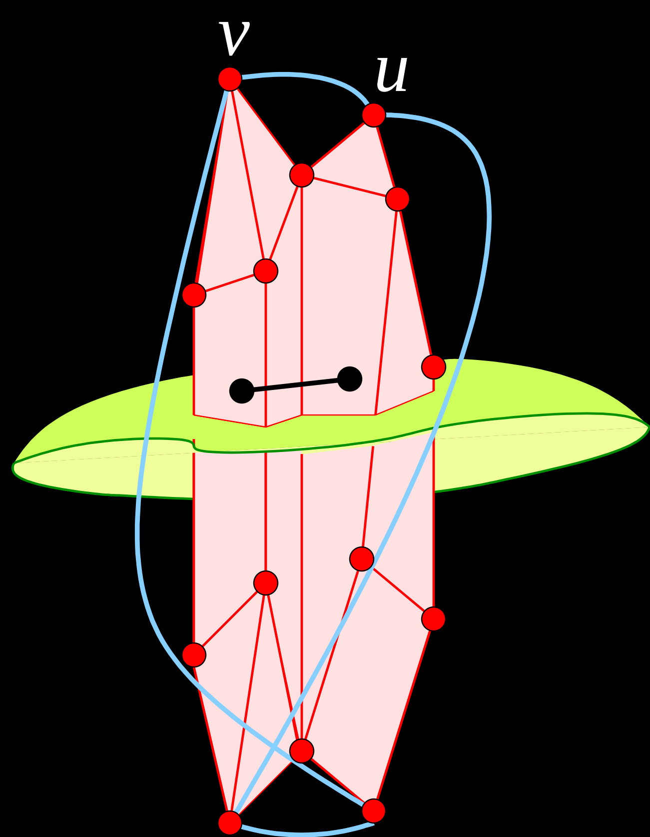
Positive Weight Edges

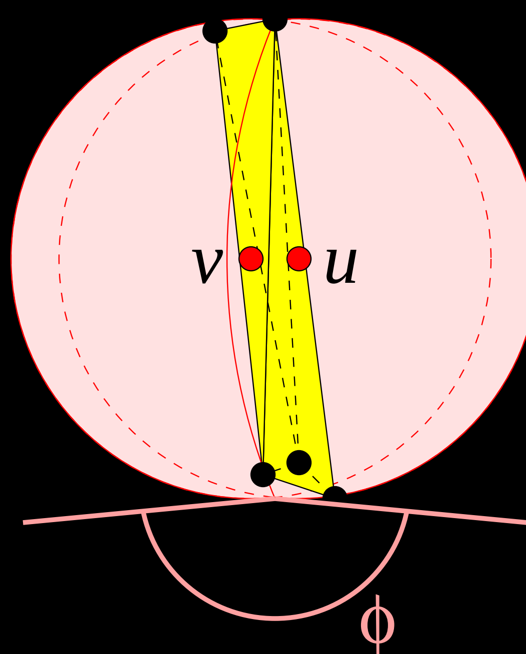
If two samples are connected by Delaunay edge, hook their poles together with positive weights.



Positive Weight Edges

Weight is large if the circumscribing spheres intersect deeply.

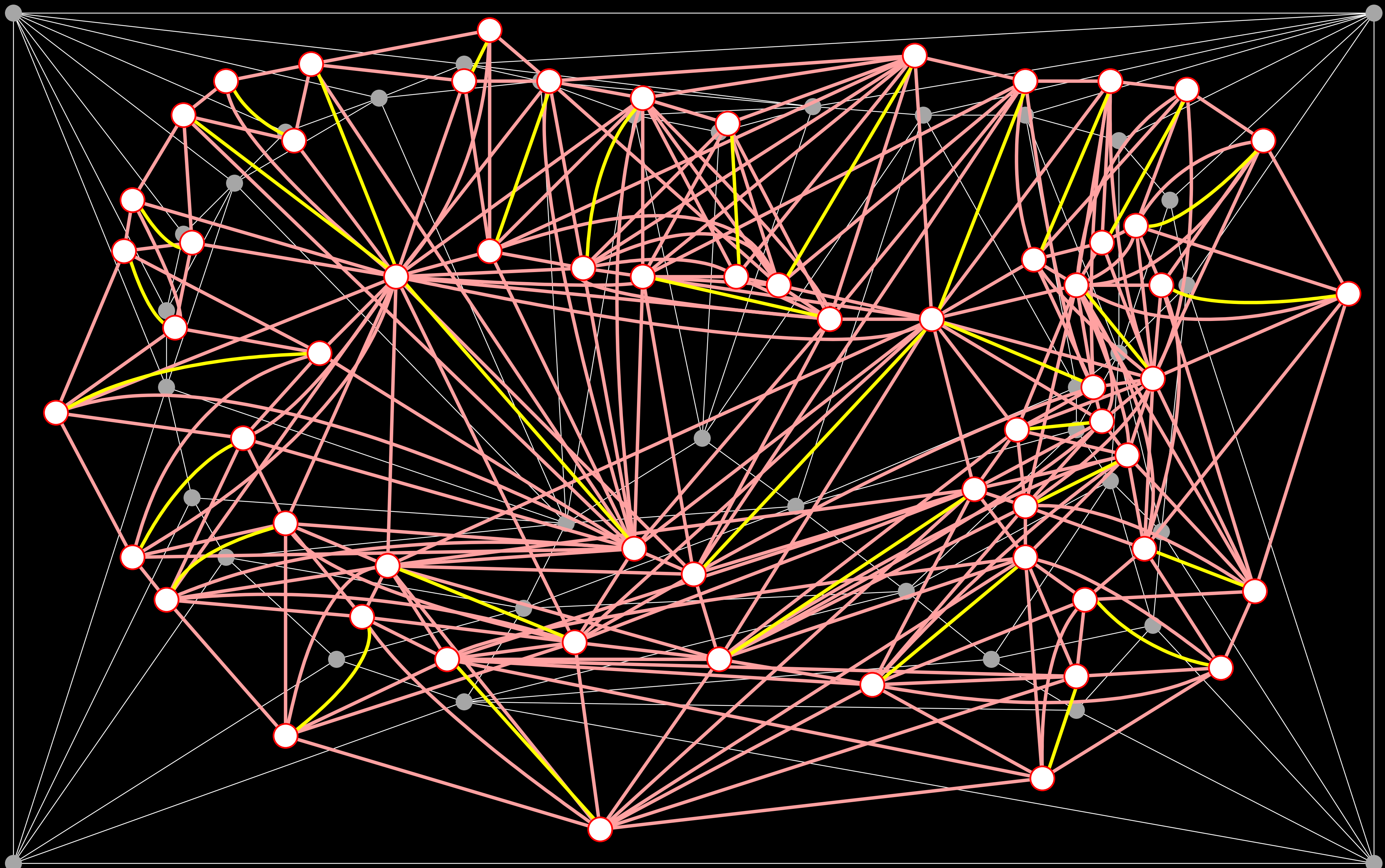




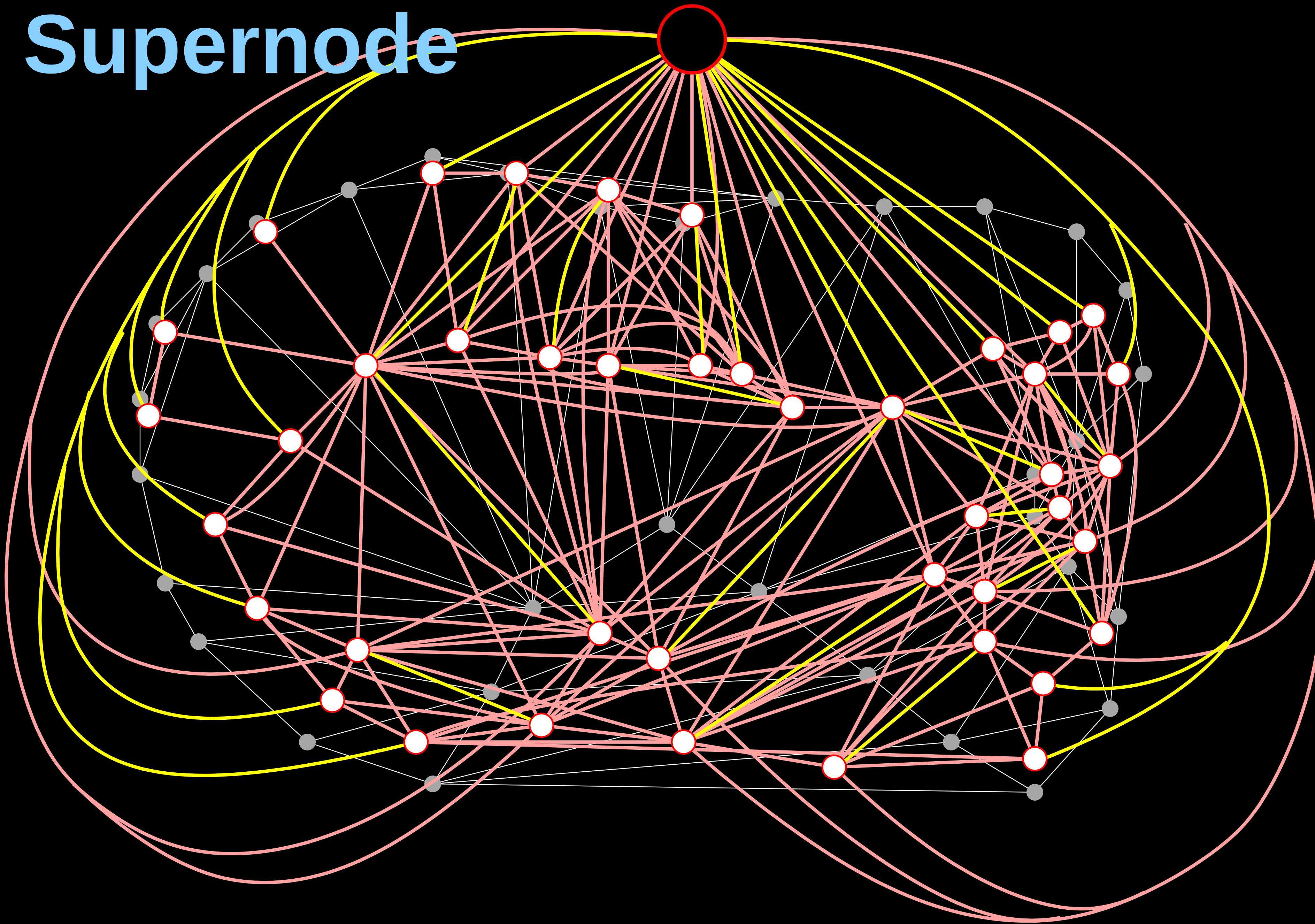
Weight of edge is

$$e^{4-4\cos\phi}$$

Pole graph

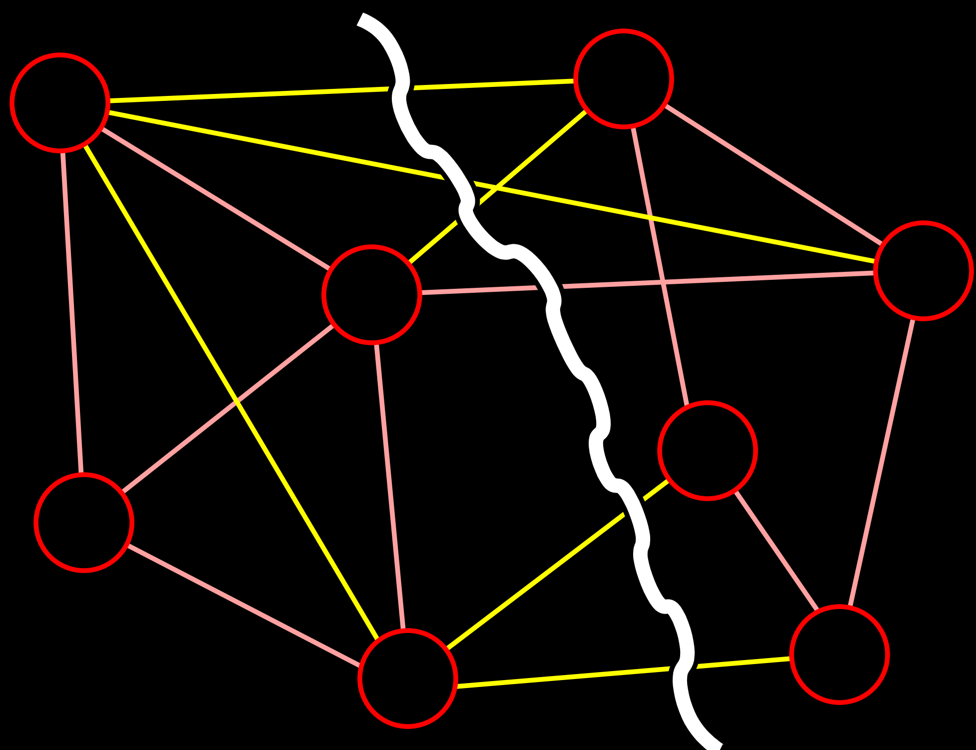


Supernode



Graph Partitioning with (Modified) Normalized Cuts

- Balances two criteria:
  - Minimizing sum of weights of cut edges.
  - Cutting graph into roughly "equal" pieces.

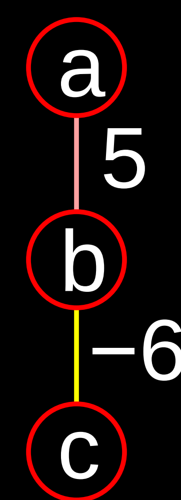


Graph Partitioning with (Modified) Normalized Cuts

- Balances two criteria:
  - Minimizing sum of weights of cut edges.
  - Cutting graph into roughly "equal" pieces.
- Pole Matrix L is weighted adjacency matrix of pole graph.

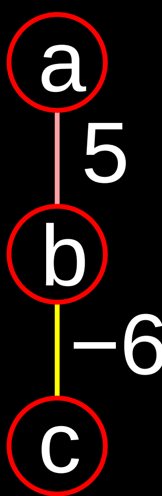
$$L = \begin{array}{ccc} \text{a} & \text{b} & \text{c} \\ \end{array}$$

$$L = \begin{array}{c} \\ \\ \\ \end{array} \begin{bmatrix} & -5 & 0 \\ -5 & & 6 \\ 0 & 6 & \end{bmatrix} \begin{array}{c} \text{a} \\ \text{b} \\ \text{c} \end{array}$$



Graph Partitioning with (Modified) Normalized Cuts

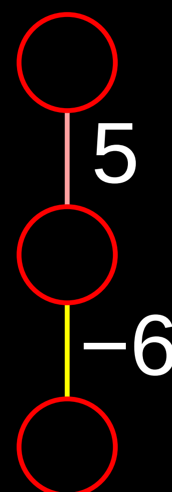
- Balances two criteria:
  - Minimizing sum of weights of cut edges.
  - Cutting graph into roughly “equal” pieces.
- Pole Matrix L is weighted adjacency matrix of pole graph. Diagonal D of L is the row sums of absolute off−diagonals.



$$L = \begin{array}{c} \begin{array}{ccc} a & b & c \end{array} \\ \begin{bmatrix} 5 & -5 & 0 \\ -5 & 11 & 6 \\ 0 & 6 & 6 \end{bmatrix} \end{array} \begin{array}{c} \\ a \\ b \\ c \end{array}$$

Graph Partitioning with (Modified) Normalized Cuts

- Balances two criteria:
  - Minimizing sum of weights of cut edges.
  - Cutting graph into roughly “equal” pieces.
- Pole Matrix L is weighted adjacency matrix of pole graph. Diagonal D of L is the row sums of absolute off−diagonals.
- Compute eigenvector x of $Lx = \lambda Dx$ with smallest eigenvalue (Lanczos iterations).



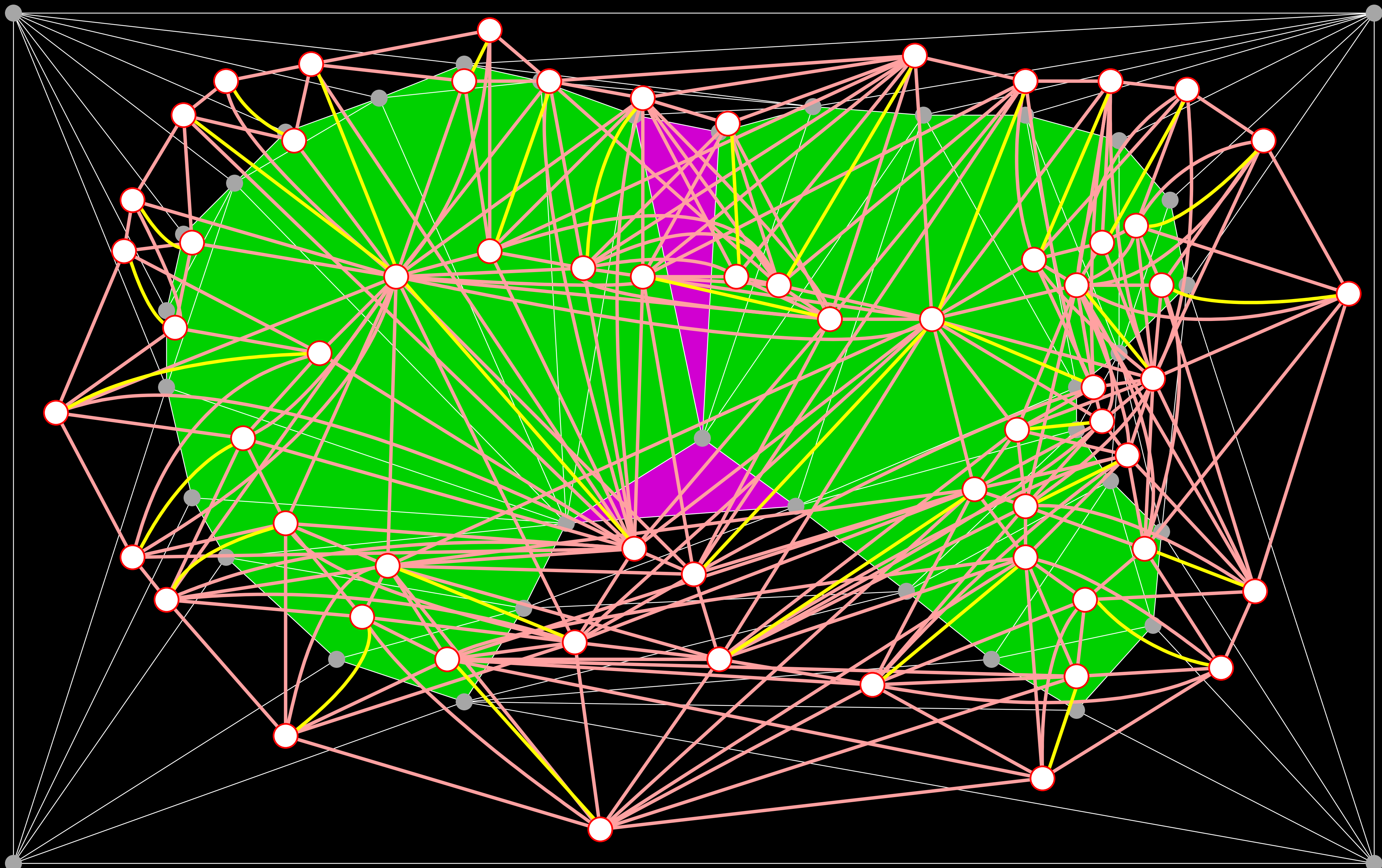
$$L = \begin{bmatrix} 5 & -5 & 0 \\ -5 & 11 & 6 \\ 0 & 6 & 6 \end{bmatrix} \qquad x = \begin{bmatrix} 3 \\ 2 \\ -4 \end{bmatrix}$$

Graph Partitioning with (Modified) Normalized Cuts

- Balances two criteria:
  - Minimizing sum of weights of cut edges.
  - Cutting graph into roughly "equal" pieces.
- Pole Matrix L is weighted adjacency matrix of pole graph. Diagonal D of L is the row sums of absolute off−diagonals.
- Compute eigenvector x of $Lx = \lambda Dx$ with smallest eigenvalue (Lanczos iterations).
- Each component of x corresponds to a pole/tetrahedron. Positive = inside; negative = outside.

End of Stage 1

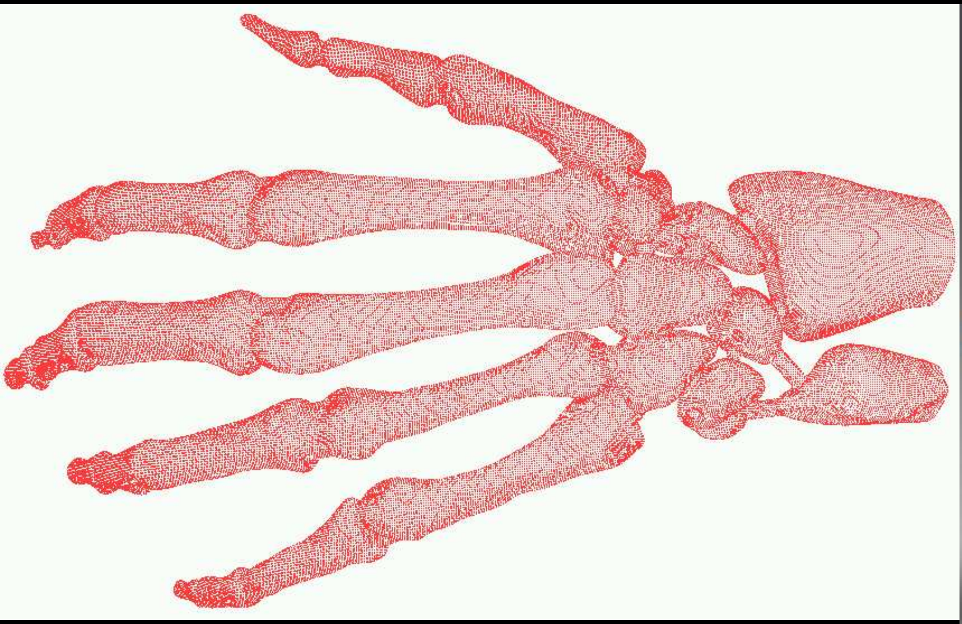
▲ : Inside   △ : Outside

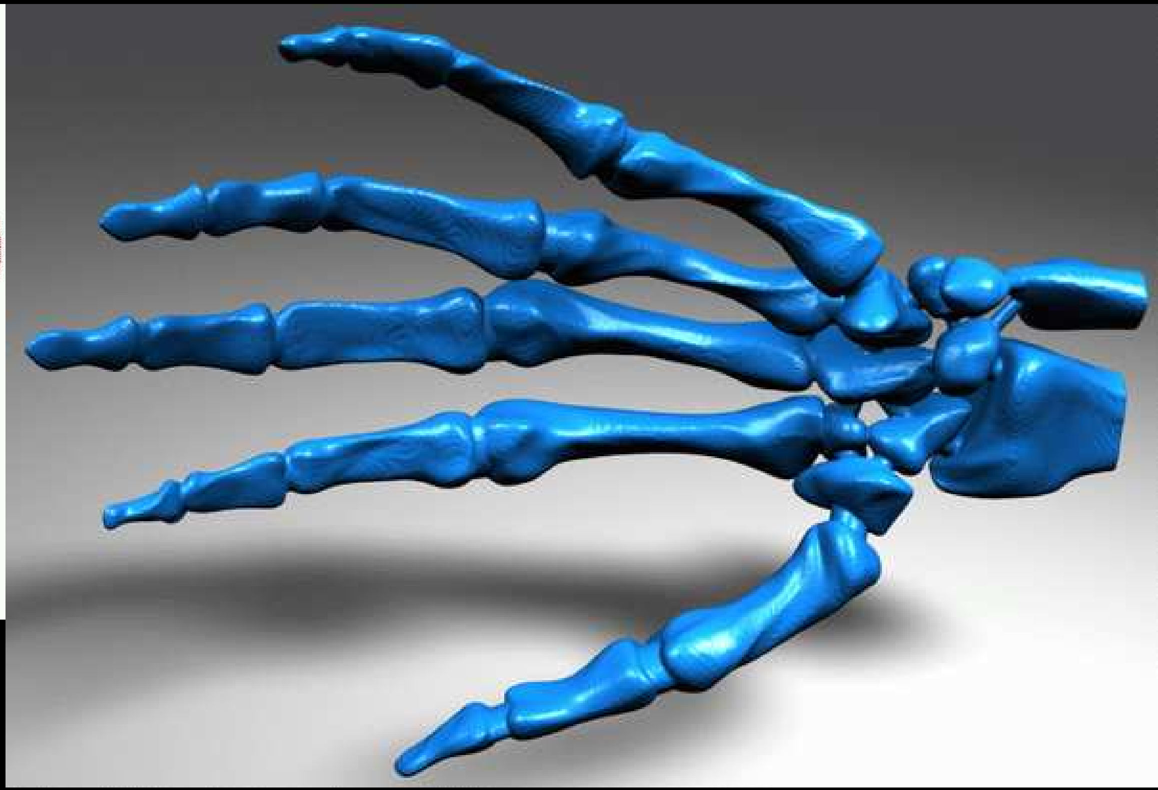


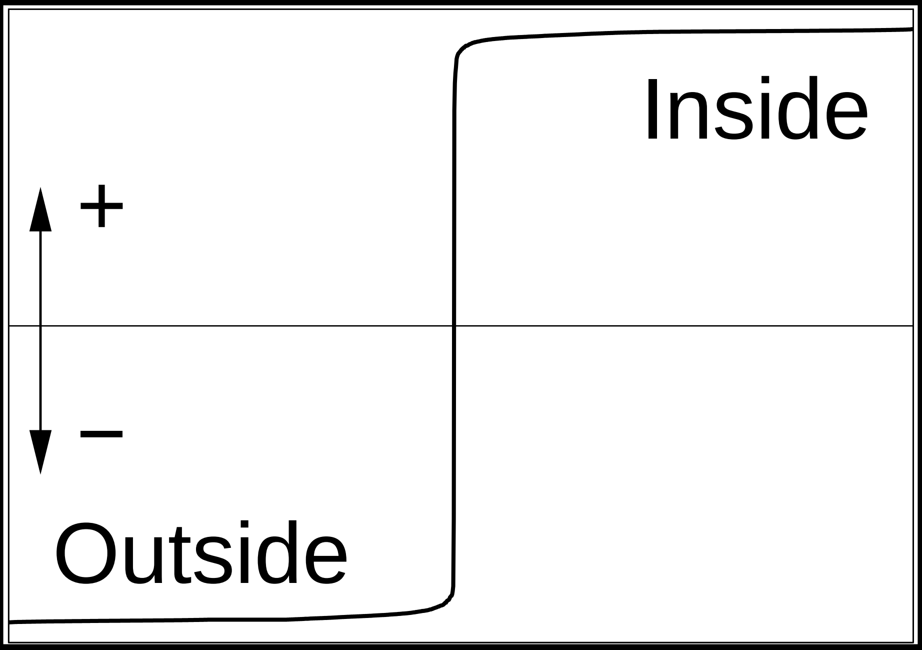
▲ : Not pole

Results

A Clean Point Cloud



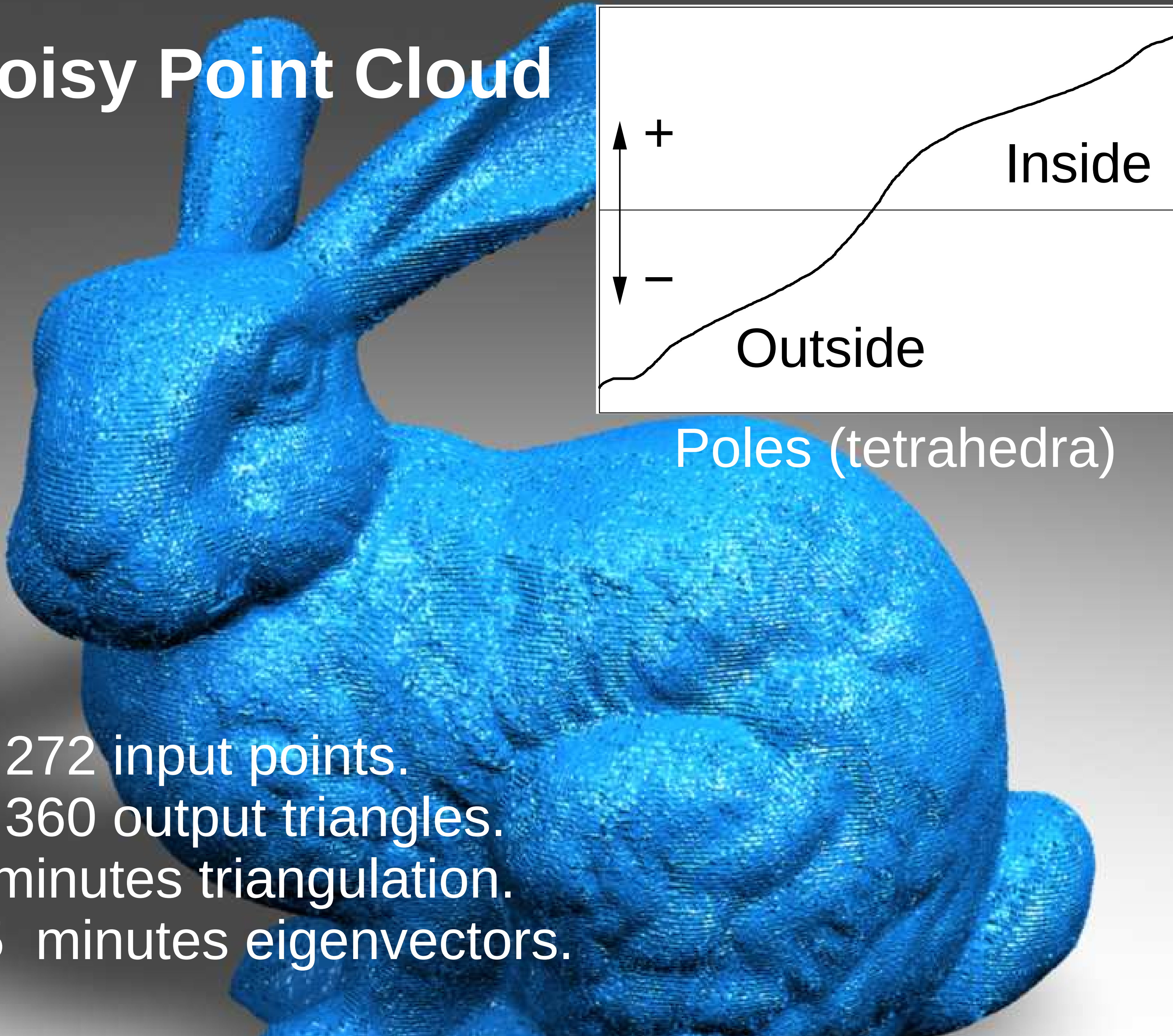




Poles (tetrahedra)

327,323 input points.
654,496 output triangles.
2.8 minutes triangulation.
9.3 minutes eigenvectors.

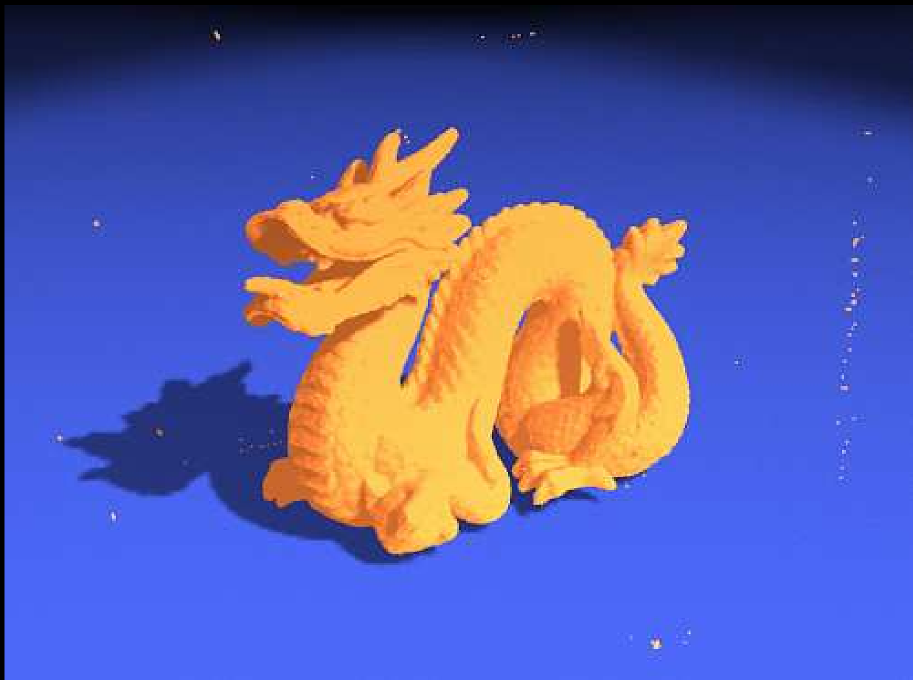
A Noisy Point Cloud

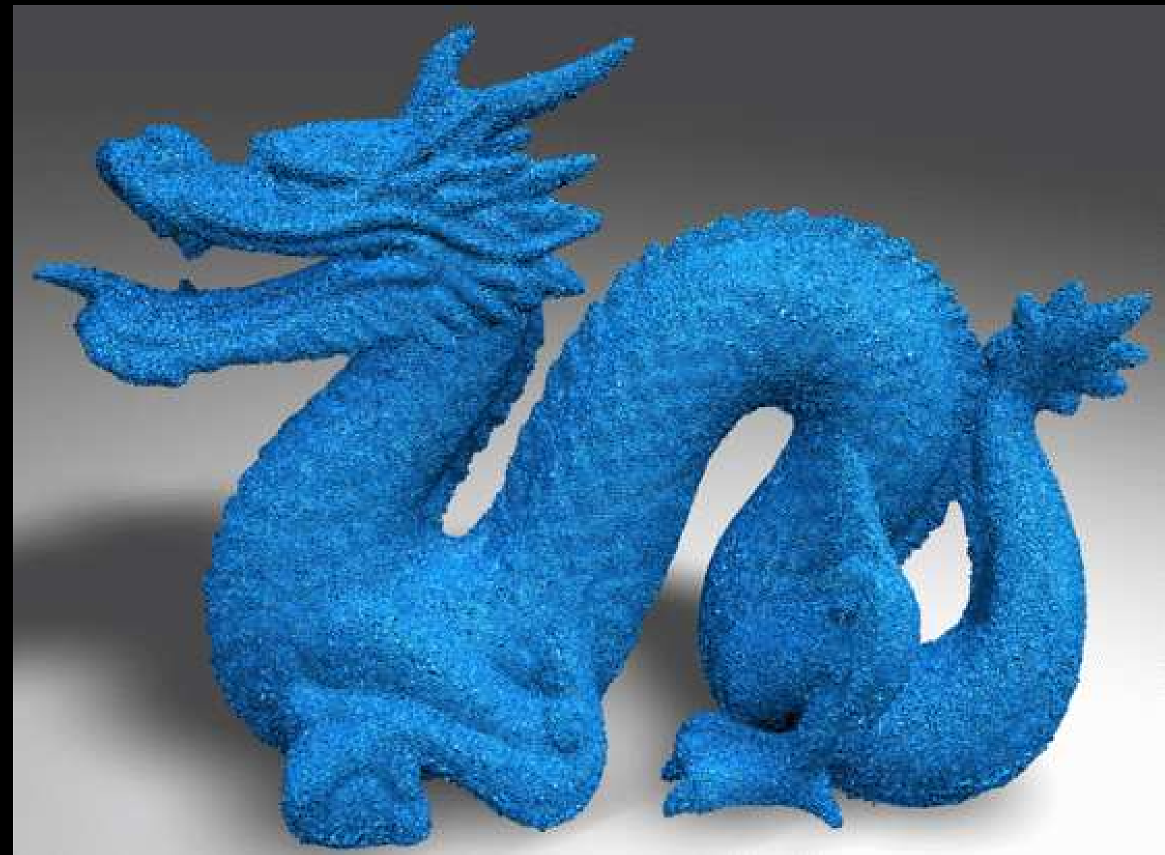


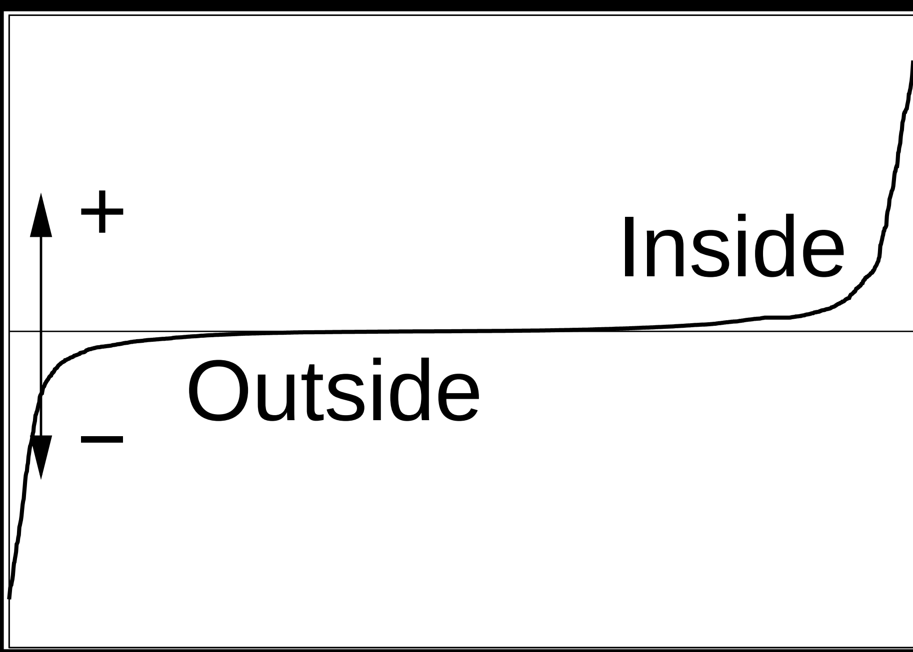
Poles (tetrahedra)

362,272 input points.
679,360 output triangles.
1.5 minutes triangulation.
17.5 minutes eigenvectors.

Stanford Dragon

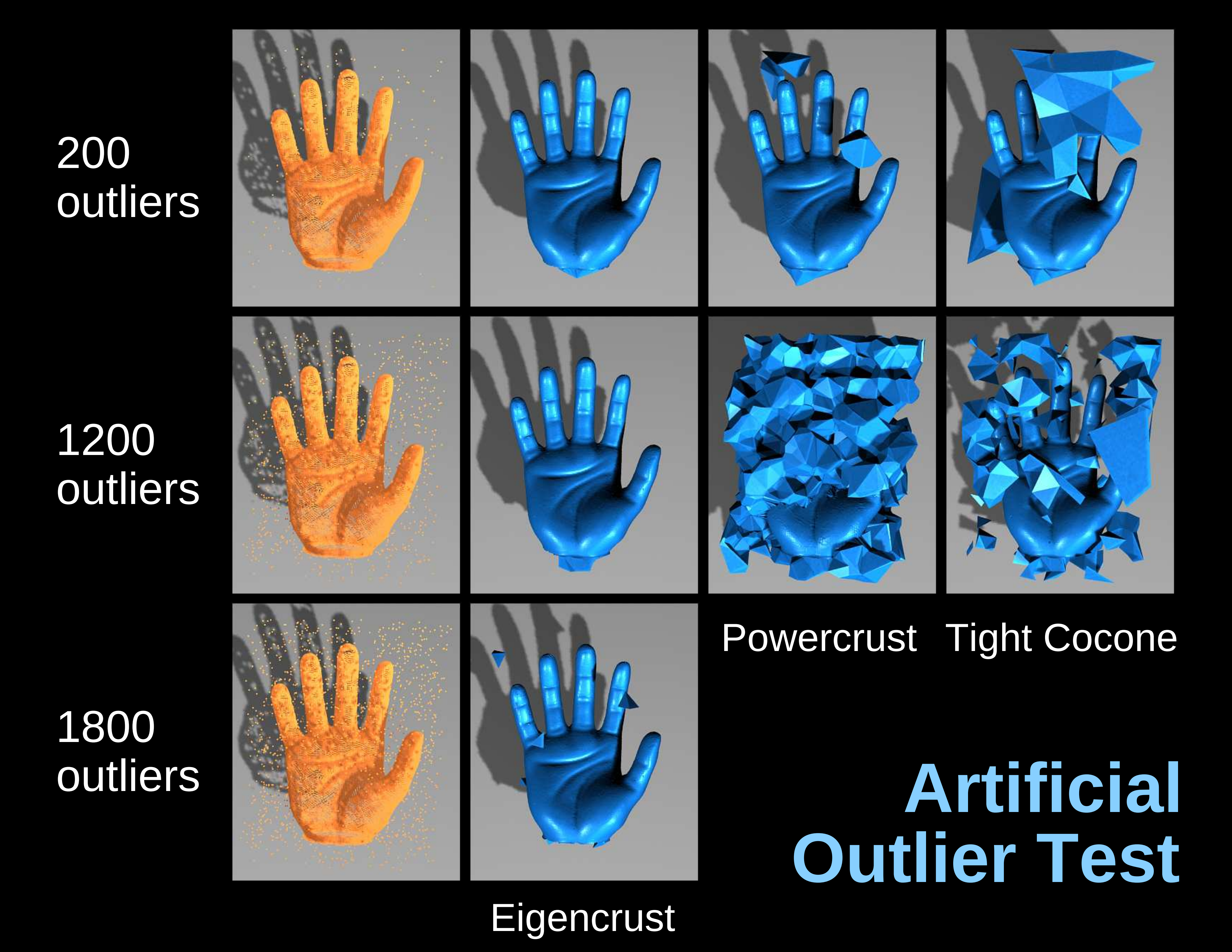


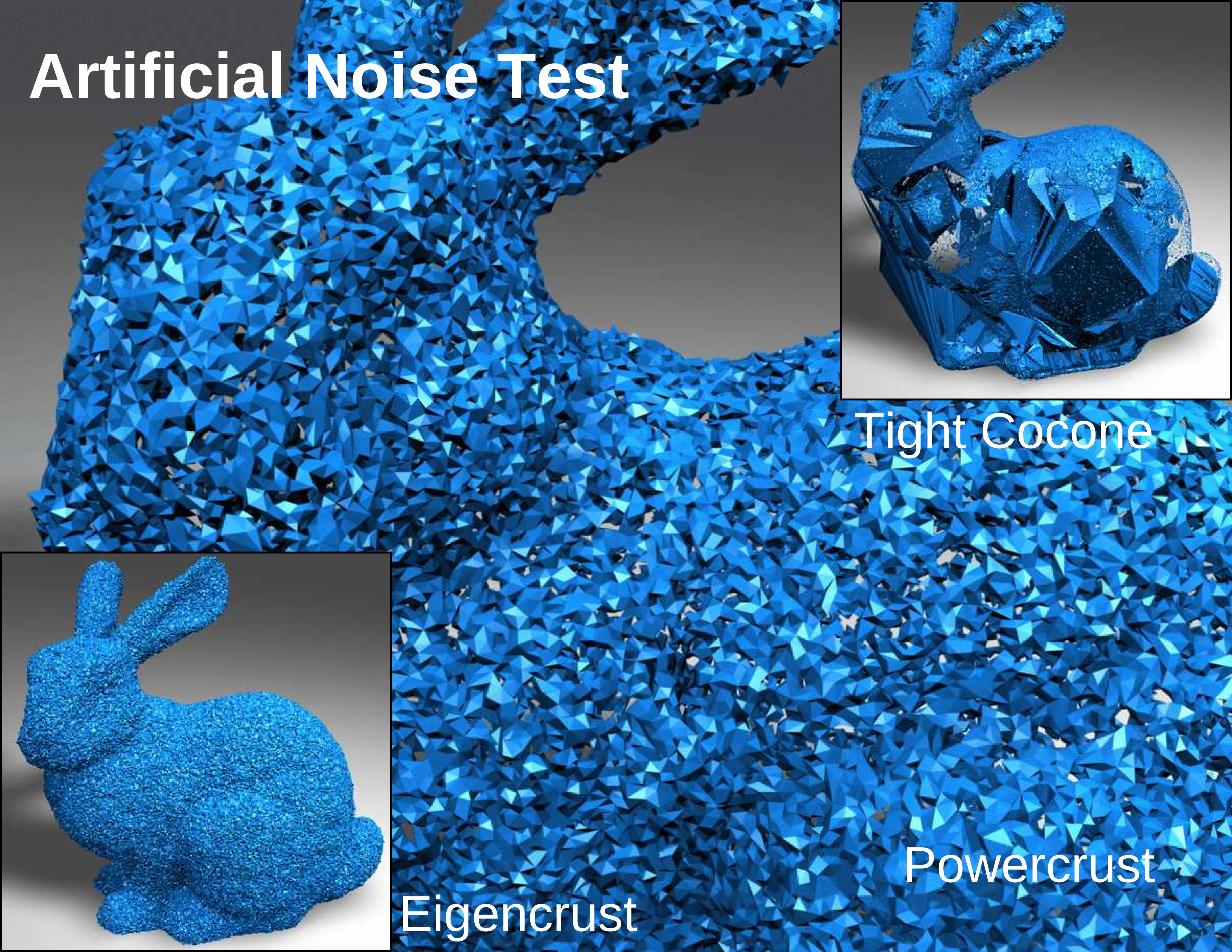




Poles (tetrahedra)

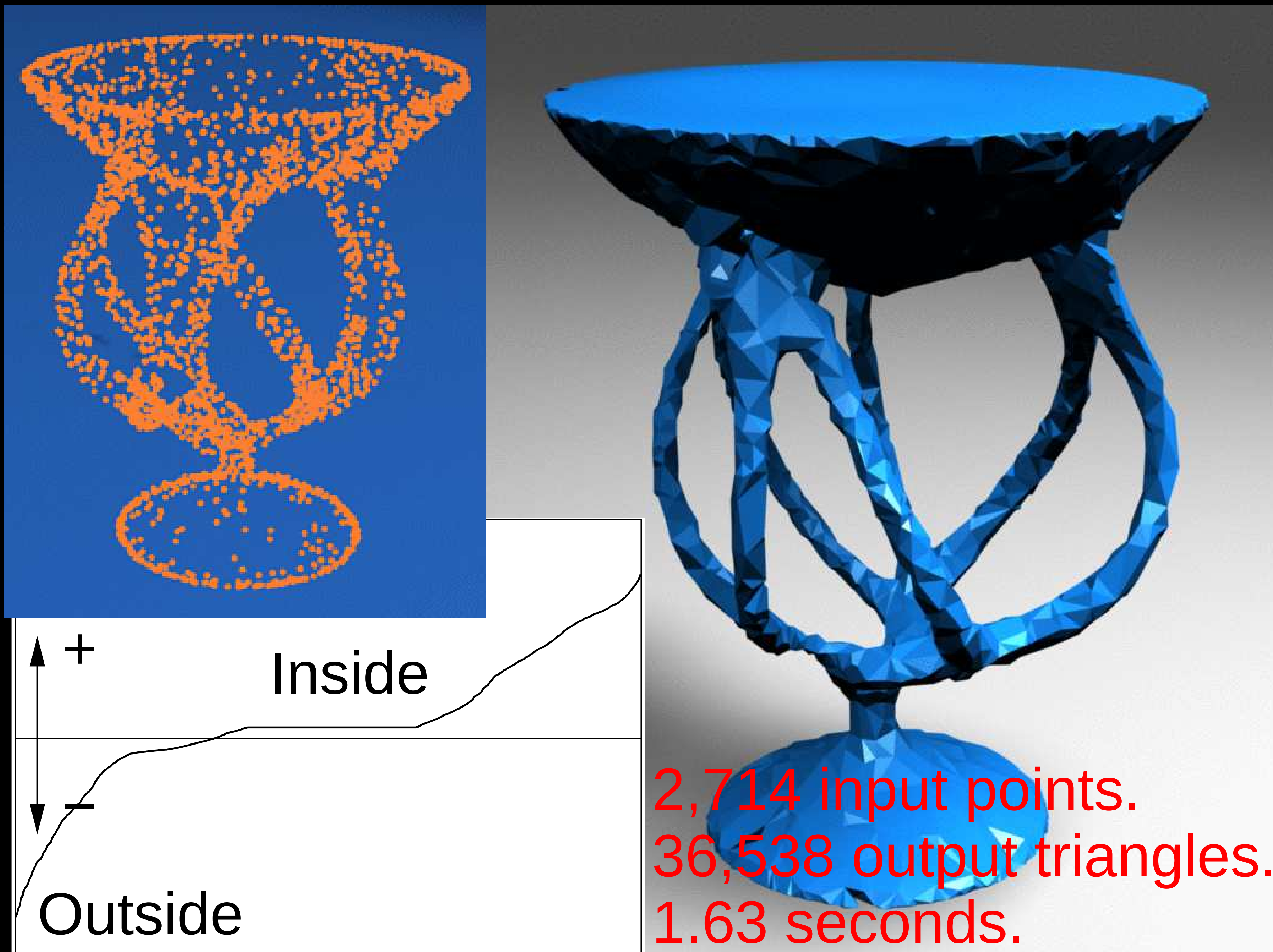
1,769,513 input points.
2,599,114 output triangles.
197 minutes.


200 outliers
1200 outliers
1800 outliers
Eigencrust
Powercrust
Tight Cocone
Artificial Outlier Test



Artificial Noise Test
Tight Cocone
Powercrust
Eigencrust

Undersampled Goblet



Conclusion

- Spectral partitioning is robust against noise, outliers, and undersampling.
- Handles raw data of real range finders.

Thanks

- Nina Amenta & Tamal Dey for their surface reconstruction programs.
- Chen Shen for rendering help.
- Stanford data repository.