Computer Graphics
Meshing & Optimisation

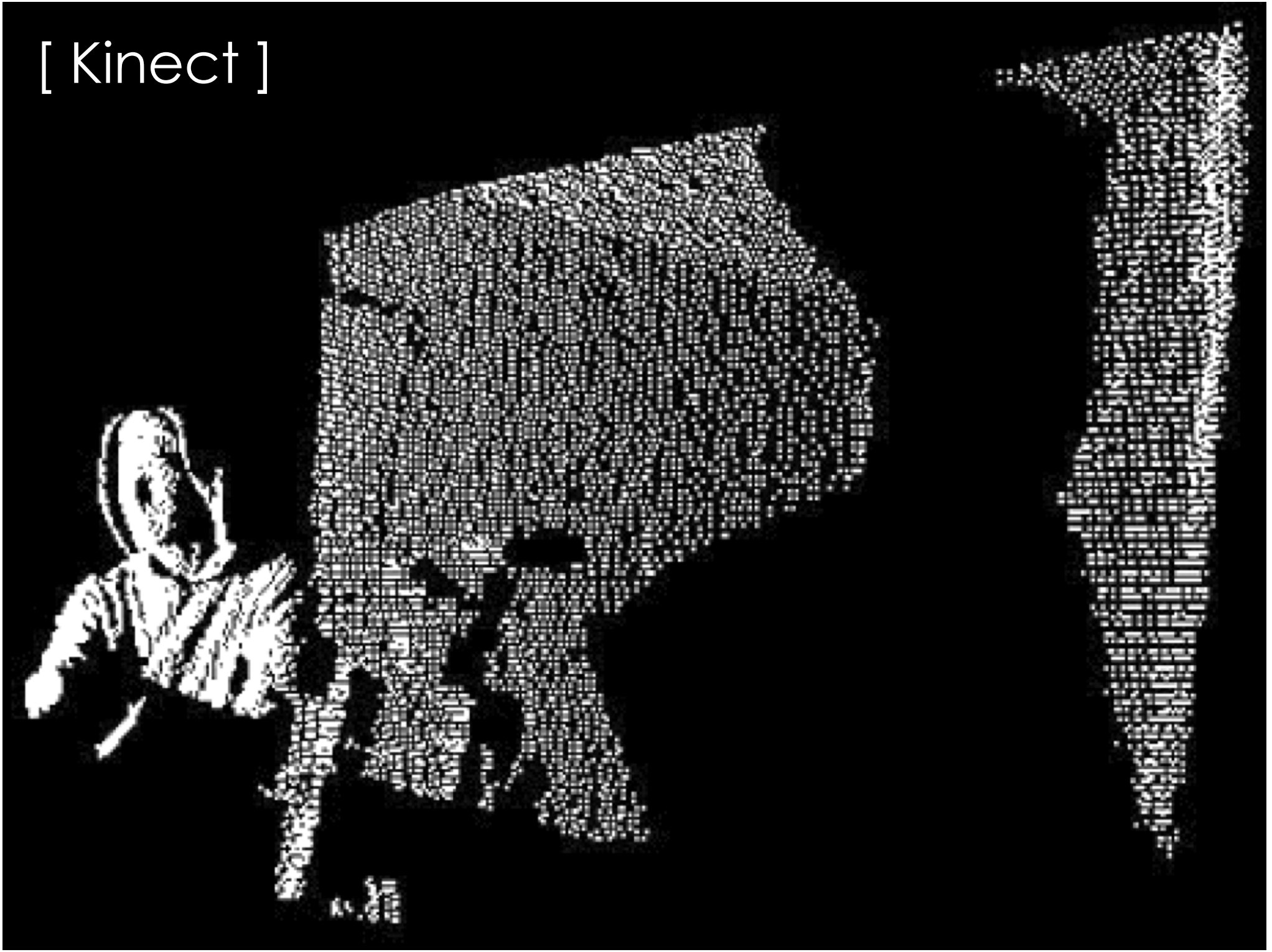
Dr Simon Lock
simon.lock@bristol.ac.uk

Overview

- We've been using triangular models for some time
- Looping through a predefined vector of triangles
- Rendering them with a raytracer / rasteriser

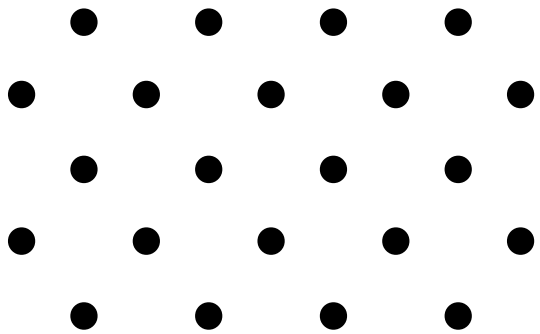
- We haven't done anything *to* this vector
- We haven't messed with its content
- We haven't manipulated the triangles themselves

- In this lecture we'll do just that:
  Creation, Simplification, Optimization, Adjustment


[Kinect]

The Meshing Problem

- The conversion of a point-cloud… into triangles !
- Problem is: which “triples” do we join as facets ?
- A given point-cloud can be meshed in multiple ways



- We’ll consider a couple of algorithms to do this

But first !

- In many of the following examples
- We'll demonstrate concepts/algorithms in 2D

- Mainly because it is simpler to grasp in a lecture
- And lot easier to see on a 2D screen !
- Also easier to interactively inject 2D points

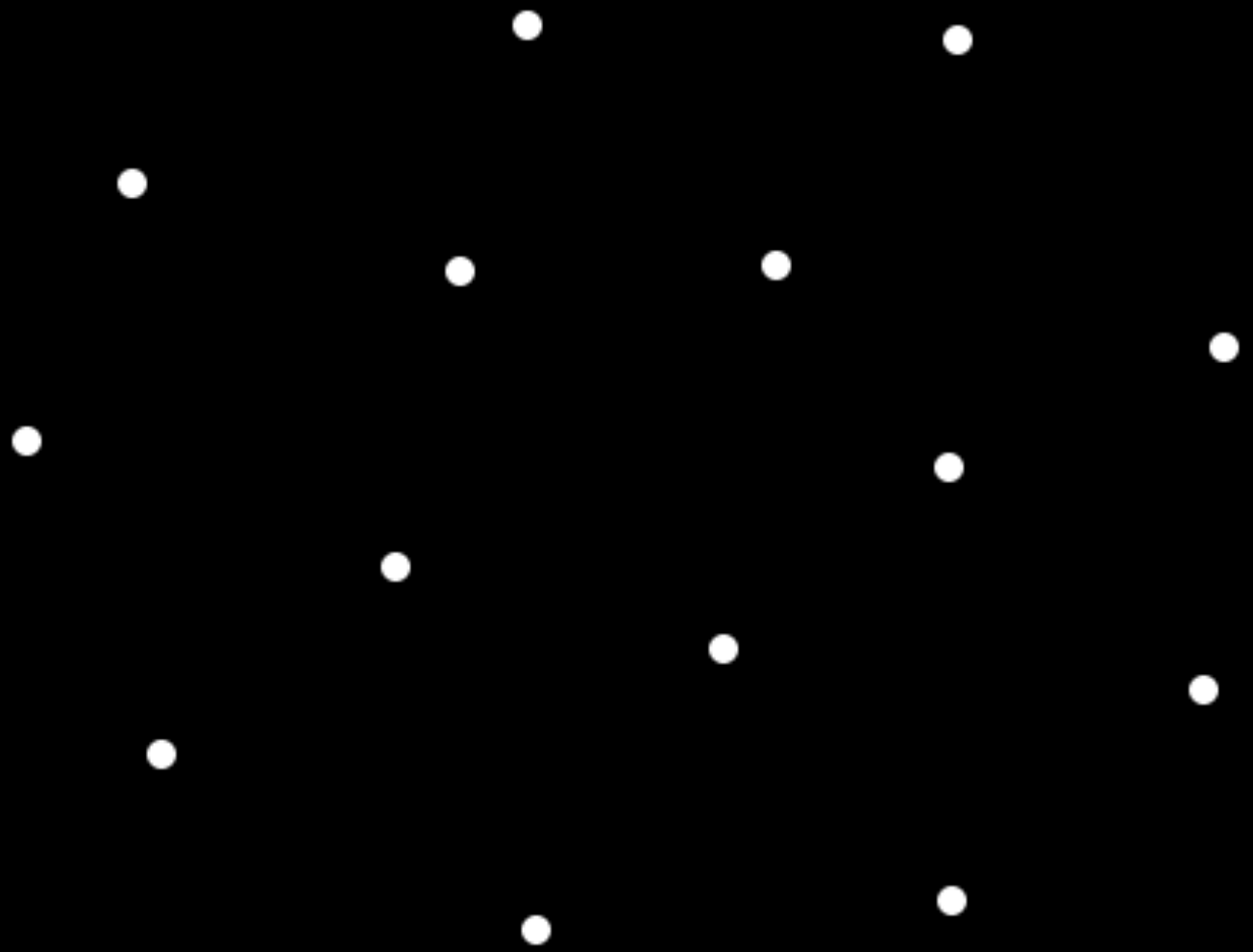
- Concepts generalise to 3D relatively easily
- Relatively ;o)

Advancing Front

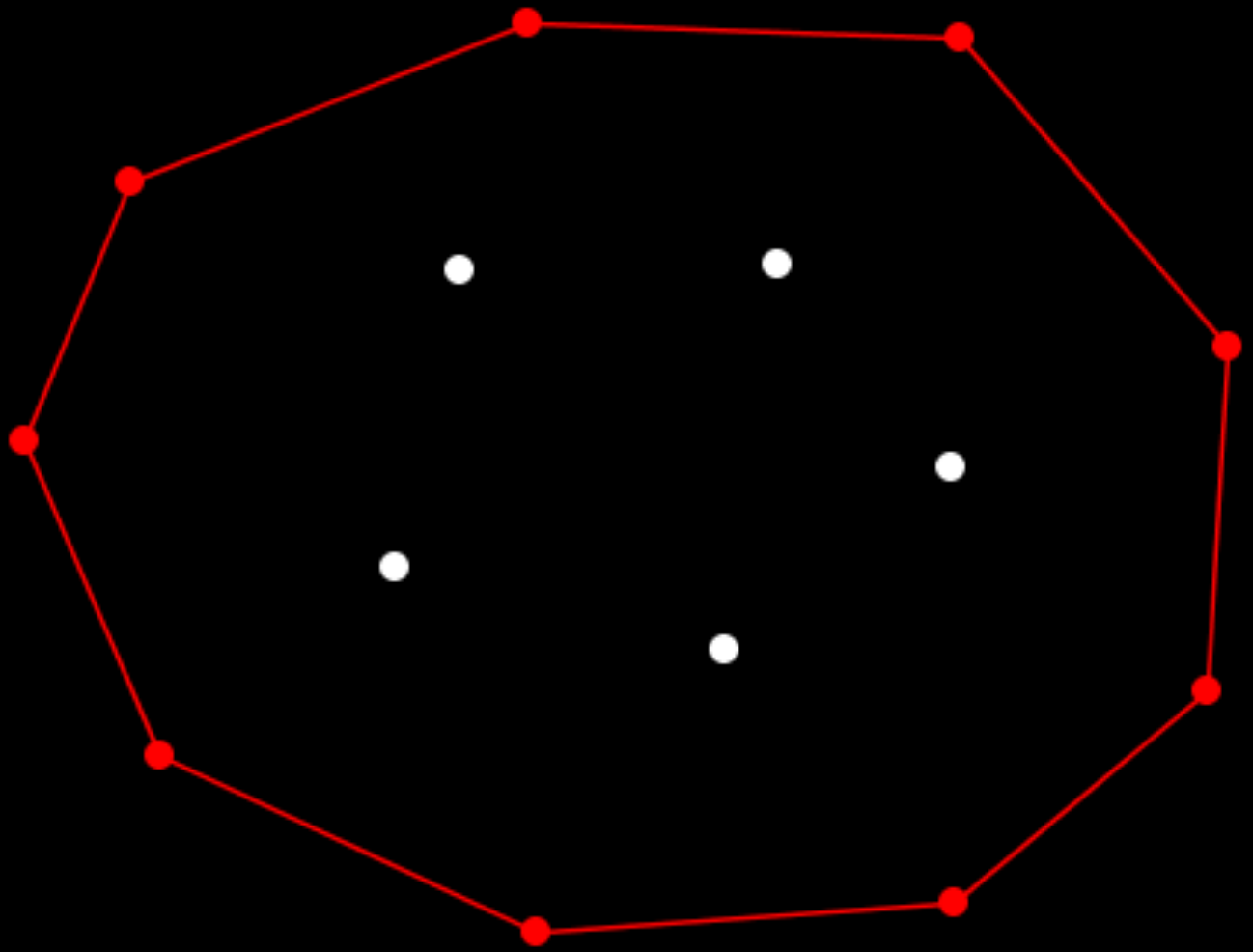
- A well-known approach to meshing
- Discussed in many text books and the literature
- Let's take a look at it in action…

[Meshing – Advancing Front]

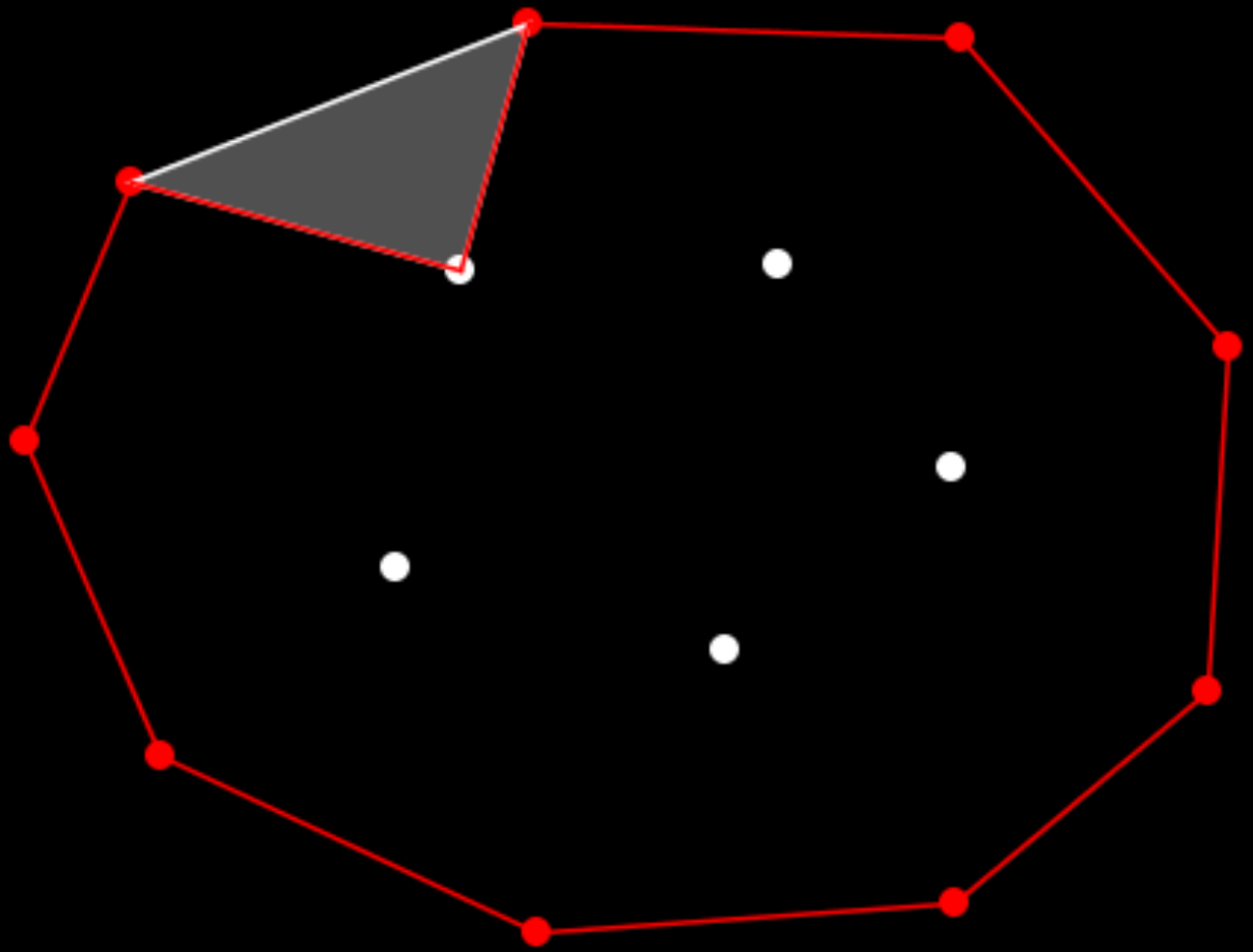
For a given point cloud



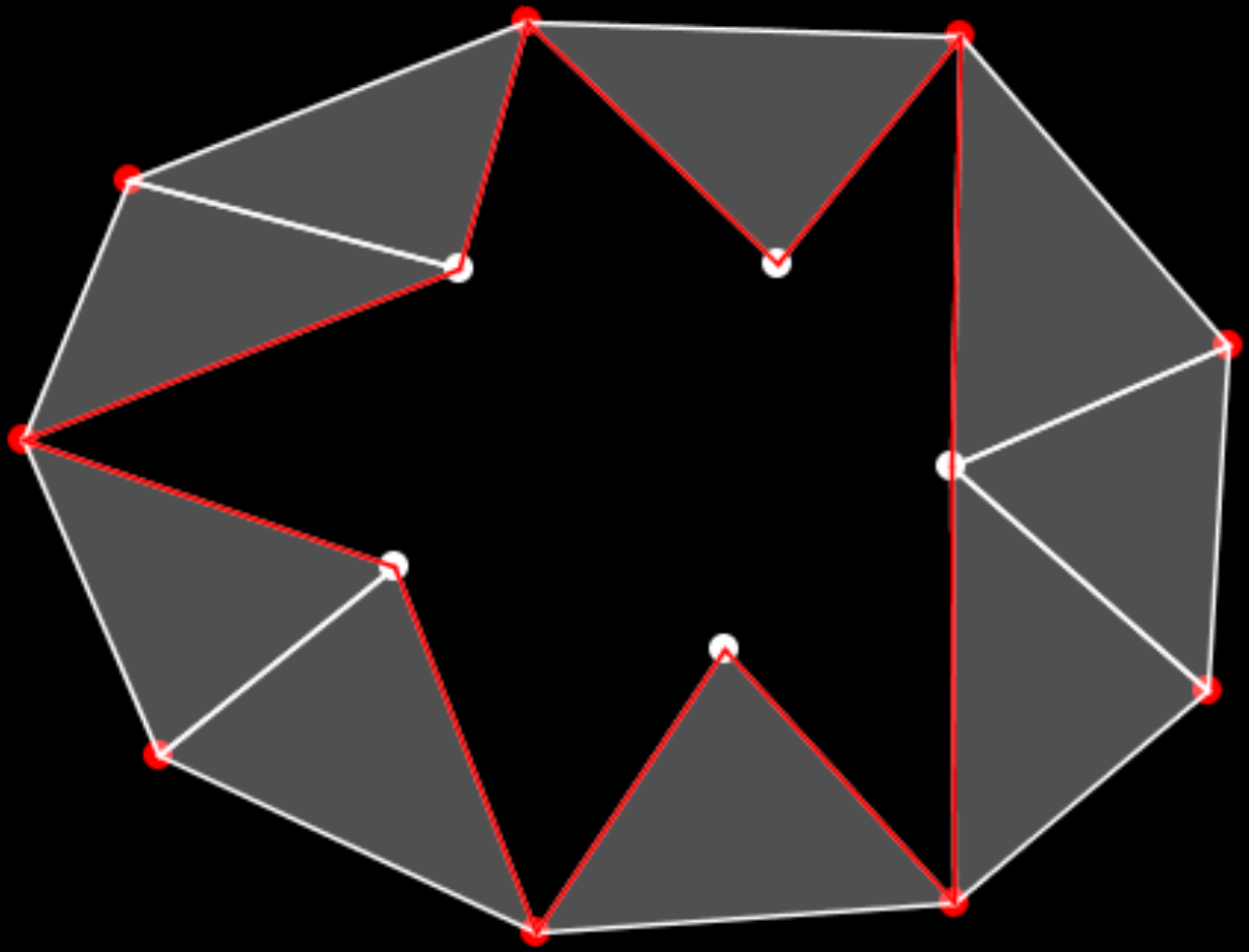
Identify outer “Envelope”



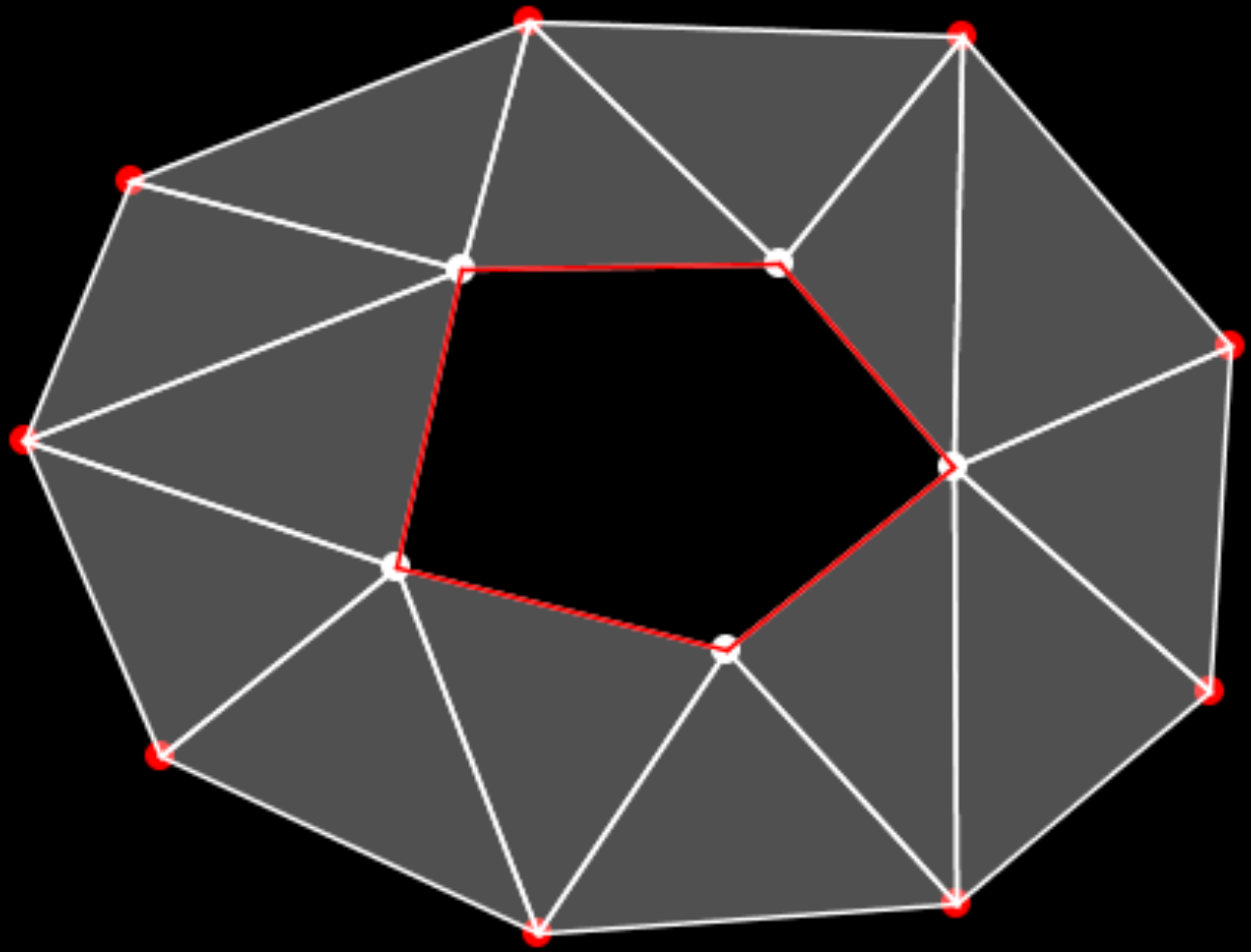
Advance this “Front line” step-by-step



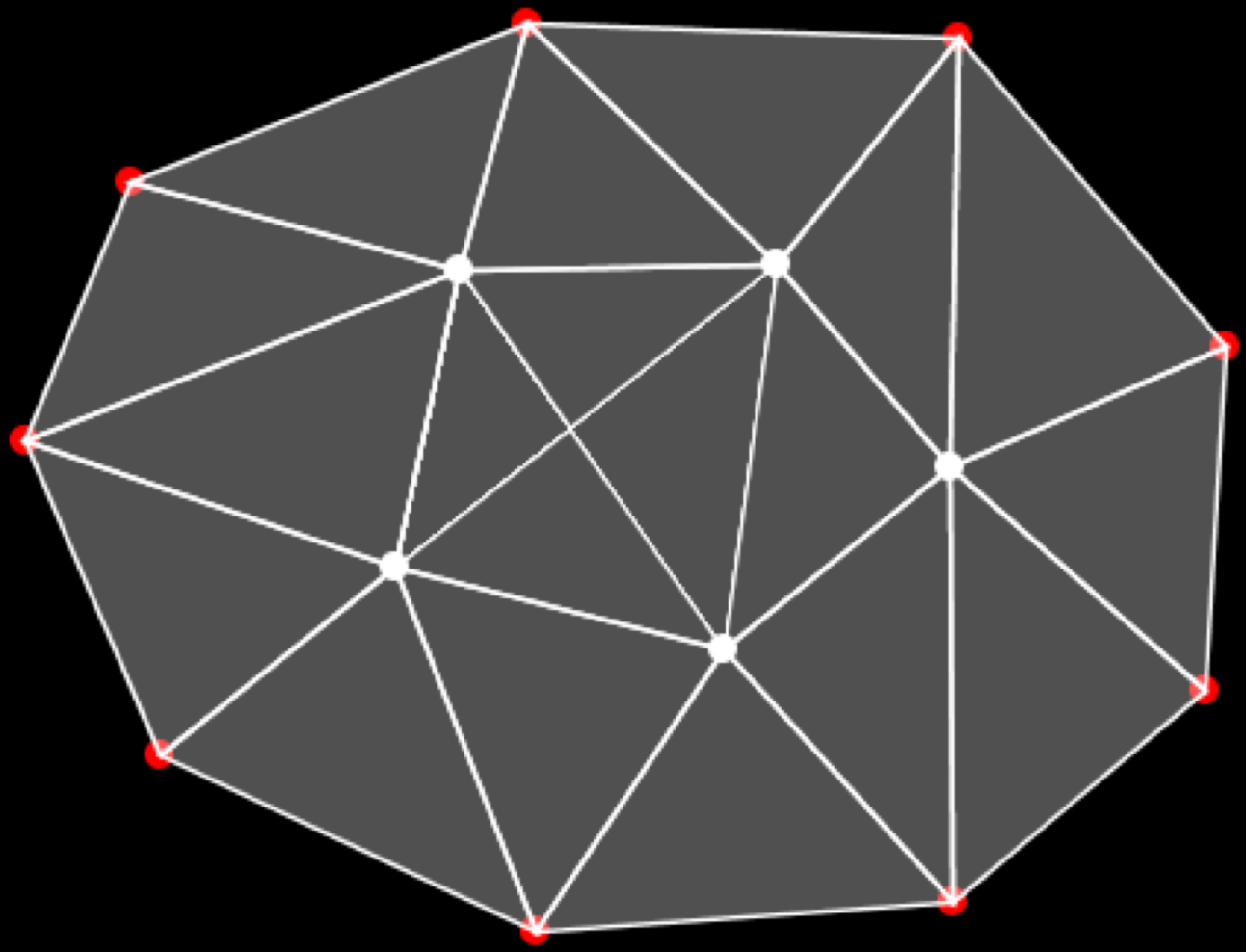
Advance this “Front line” step-by-step



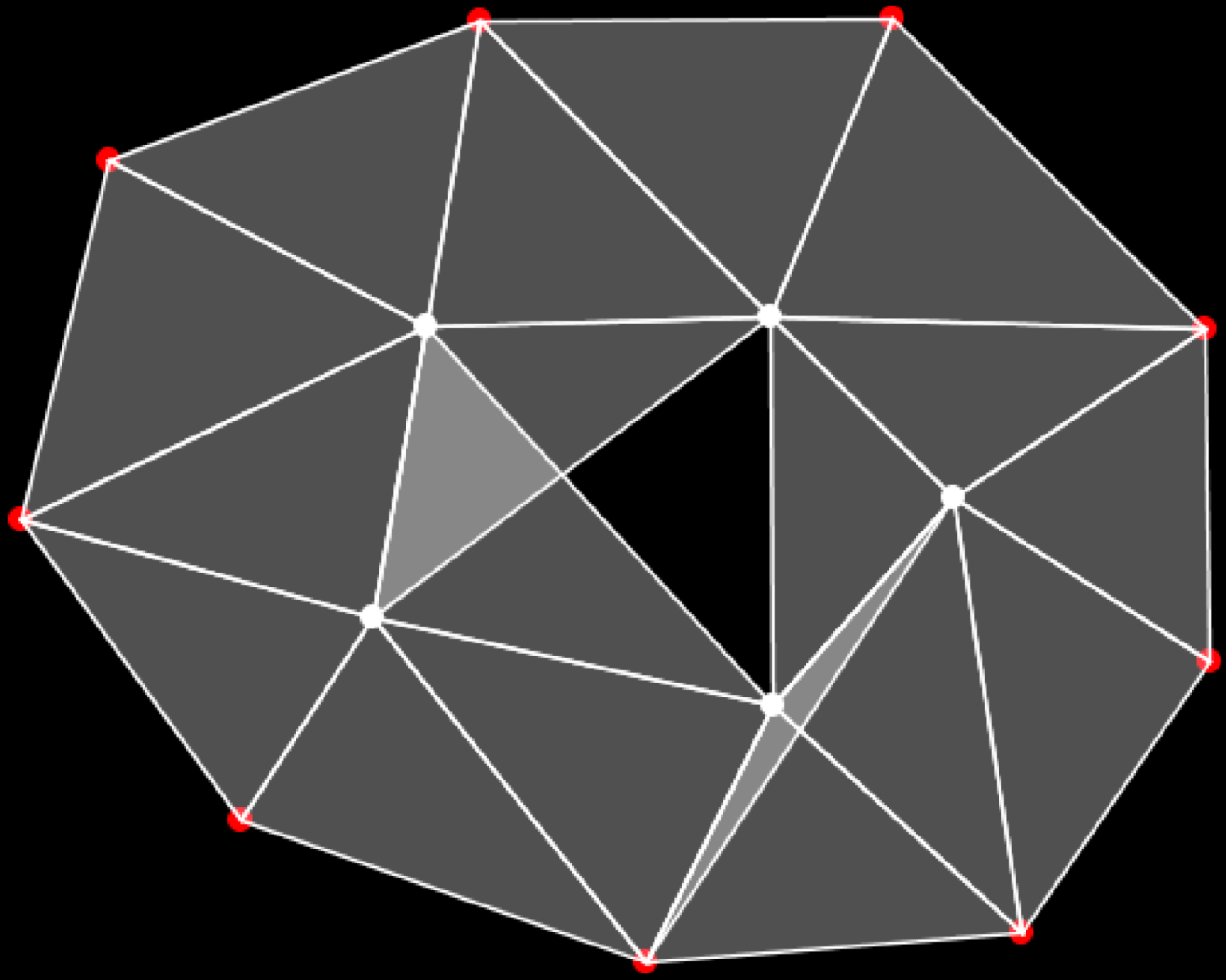
Advance this “Front line” step-by-step



Advance this “Front line” step-by-step



When meshing goes wrong



Bowyer–Watson algorithm

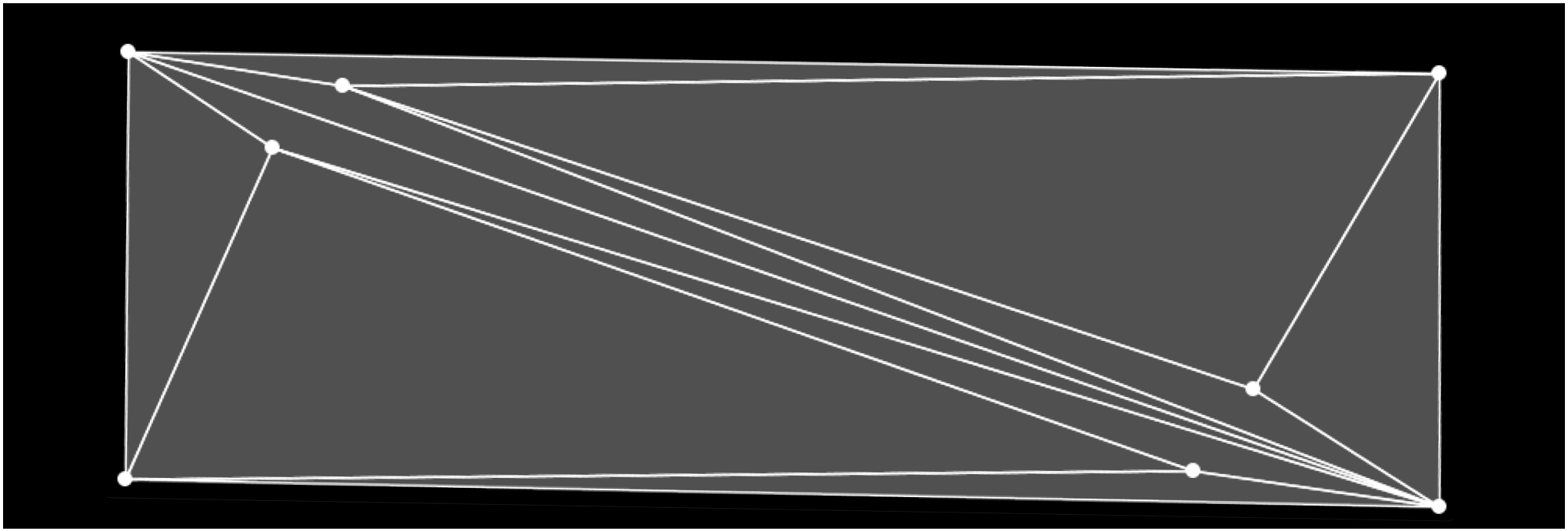
Another well-published approach
Conceptually simple, yet is a powerful technique
Algorithm operates as follows:

1. Read in a point at a time and add it to model
2. If the point is *outside* any existing triangle…
   Just add it to the model as part of a *new* triangle
3. If a point is *inside* an existing triangle…
   Split that triangle into three sub-triangles

[Meshing – Bowyer – Not Flippin Yet]

Problem

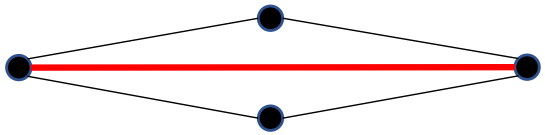
- Ideally we want a mesh with the finest granularity
- Small triangles, short edges
- These will maximise the *perceived* resolution
- Reduce the “pixelated” appearance of the mesh
- Our enemy is the “sliver” :



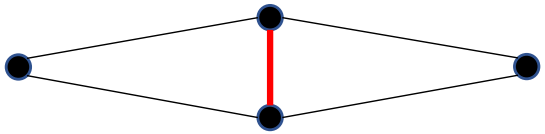
The Solution: Flipping

- ***Flipping*** is an easy way to minimise slivers
- Basically involves swapping vertices…

- Switching pairs of triangles from long, thin ones:



- Into short fat ones:



[Meshing – Bowyer – With Flipping !]

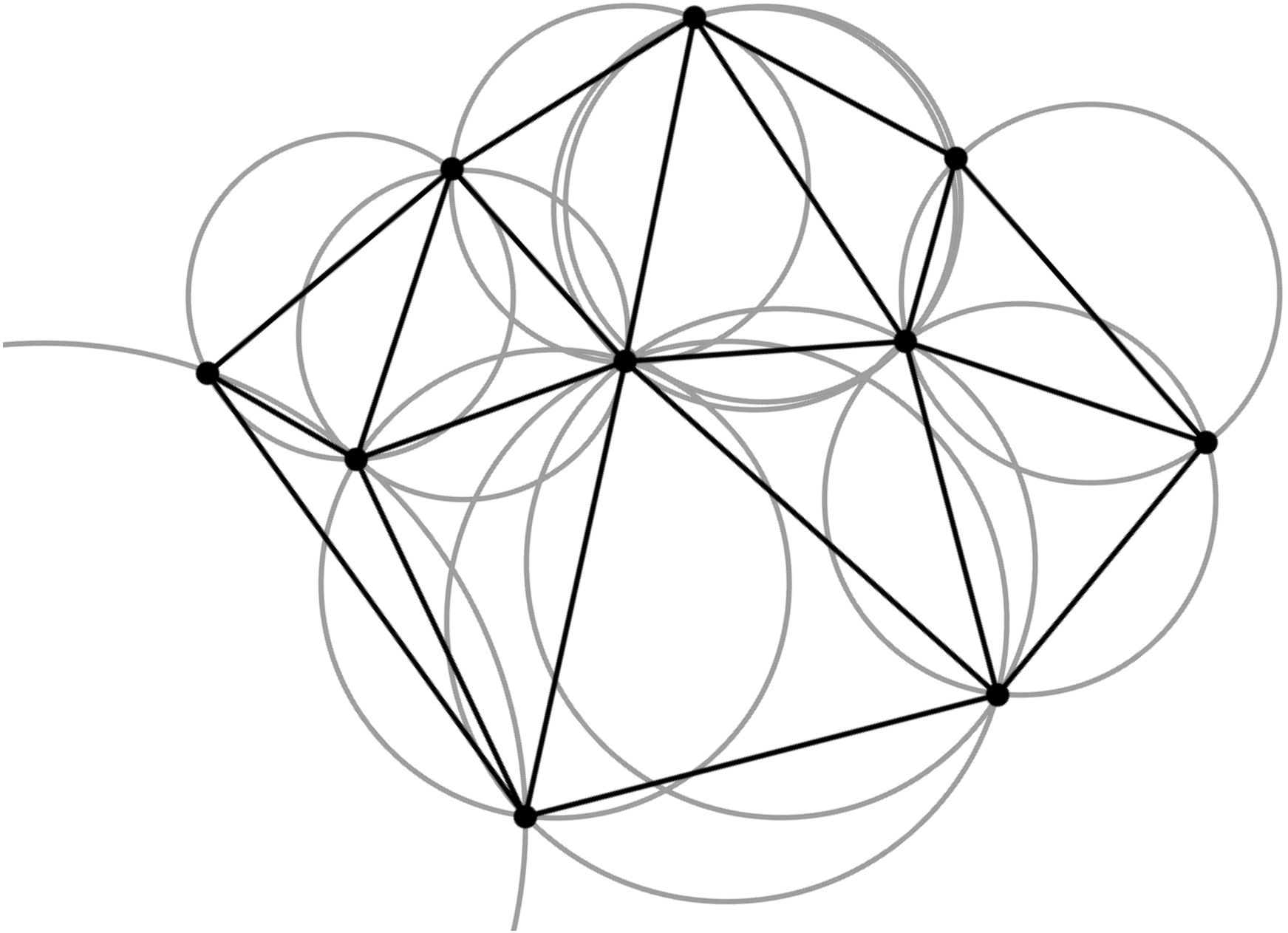
But when to flip ?

- How do we know when to flip ?
- I've been doing it manually "by eye"

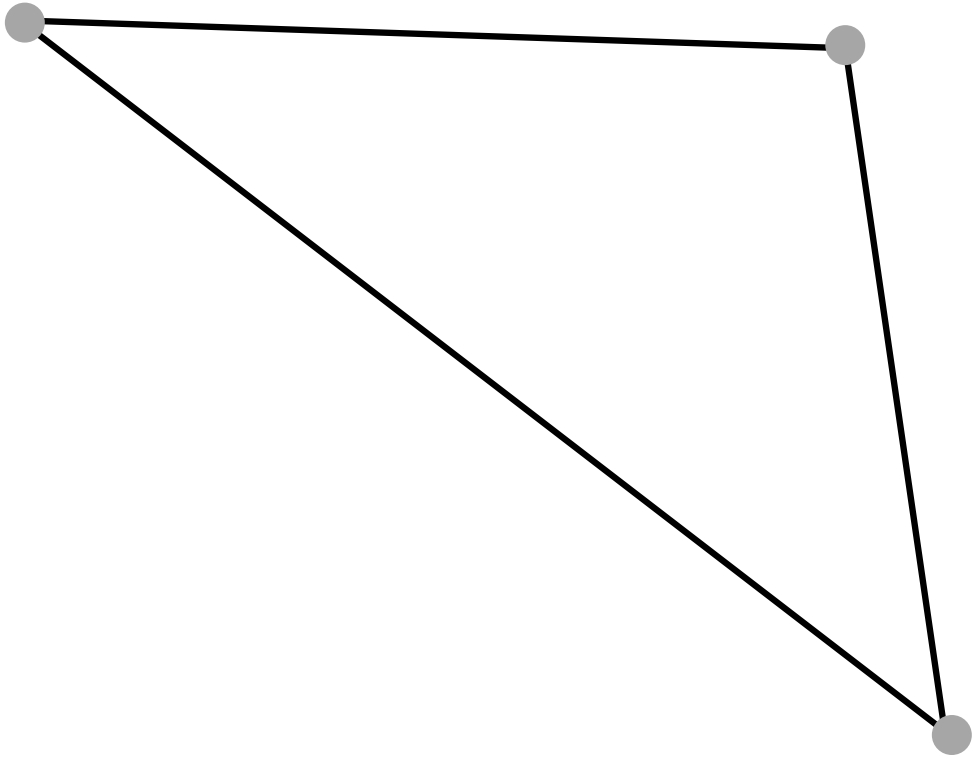
- What metric can we use to detect slivers ?
- When do we know it would be best to flip ?

Delaunay Triangulation…

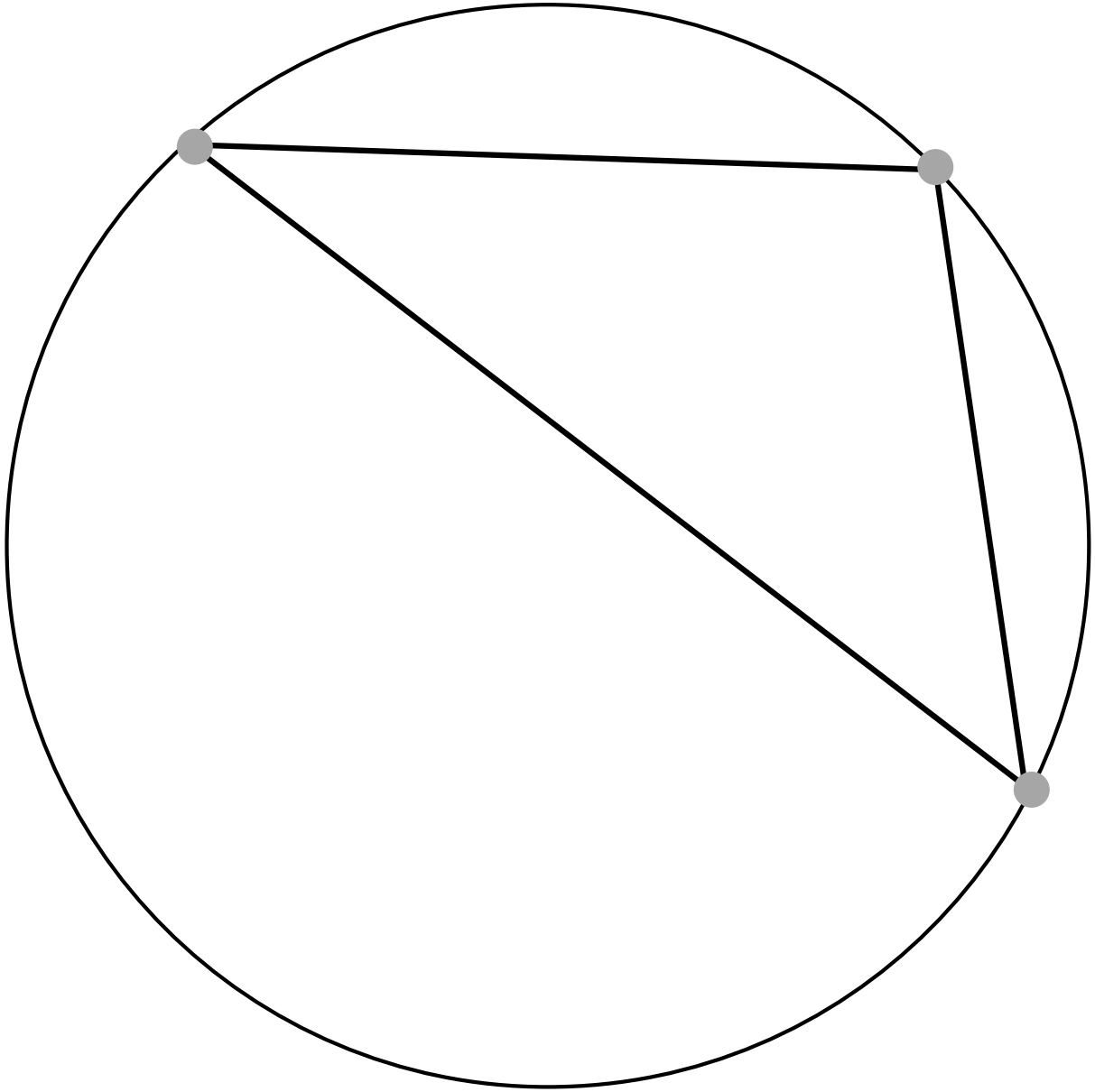
Delaunay condition: Each triangle's circumcircle must not contain vertices of another triangle



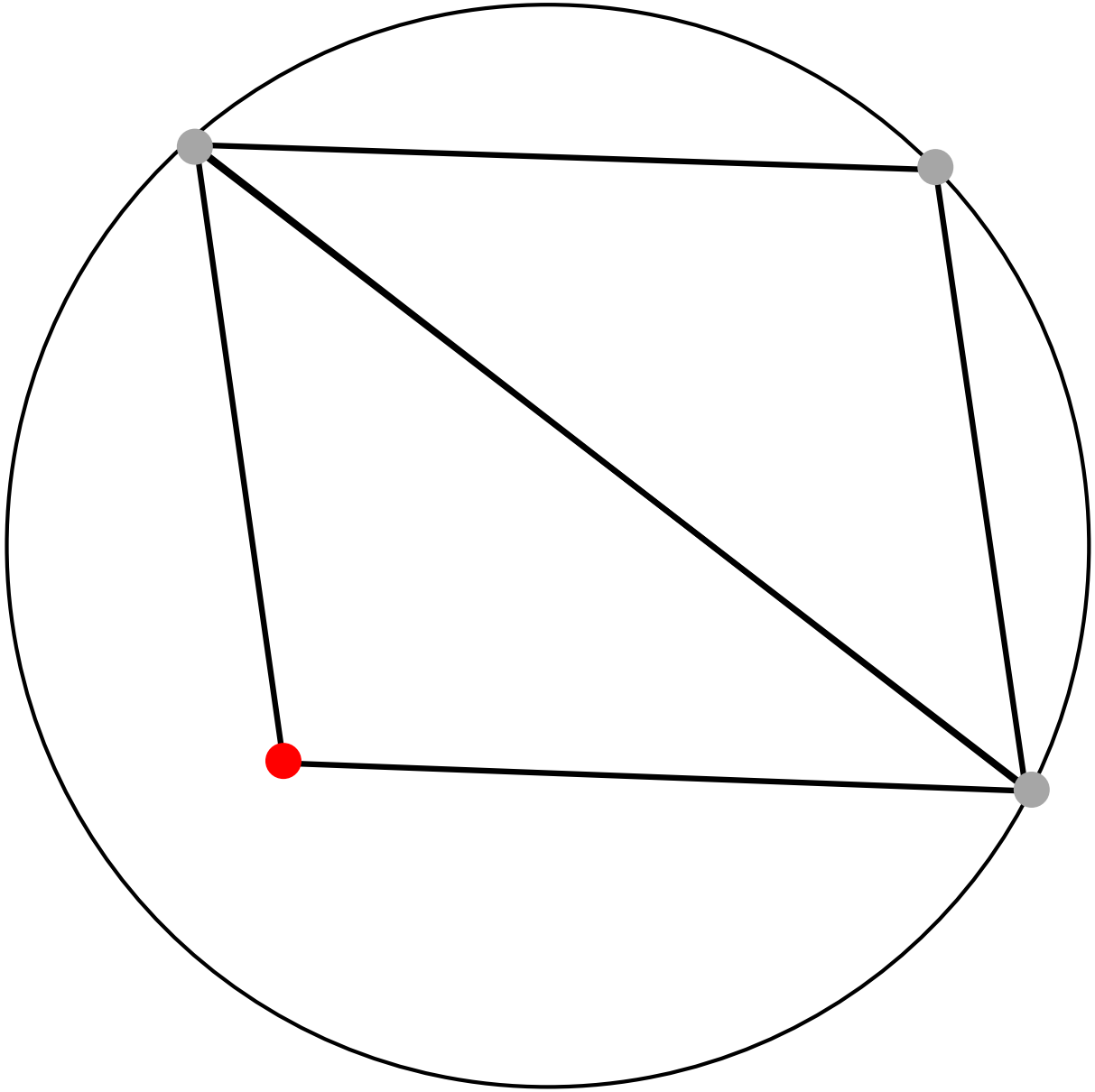
Triangle



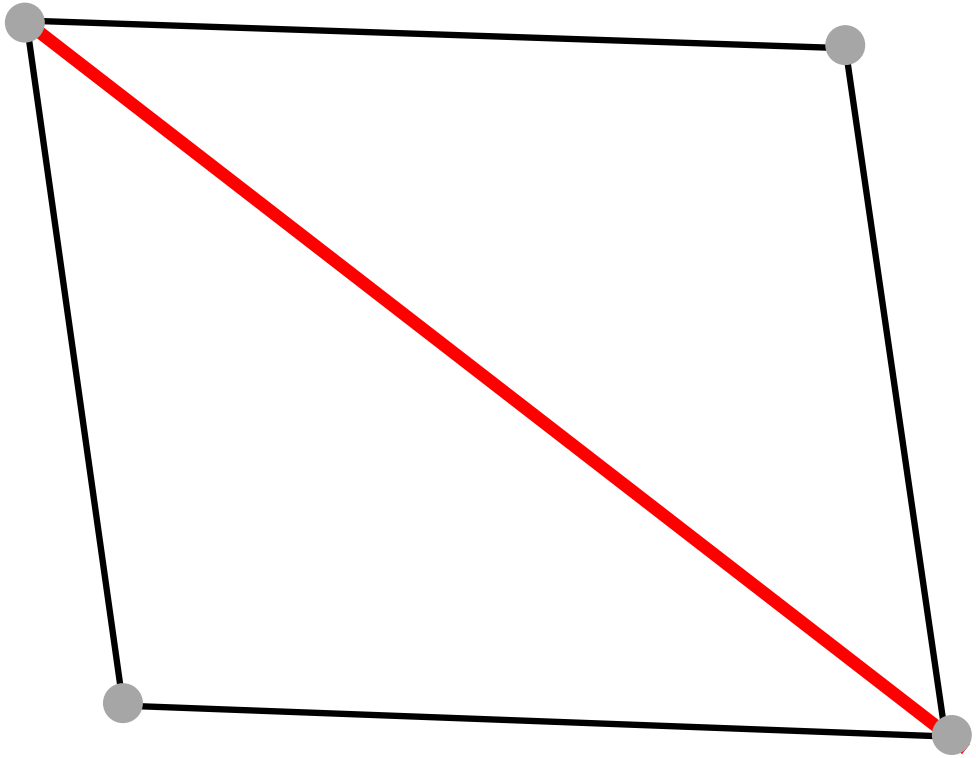
Circumcircle



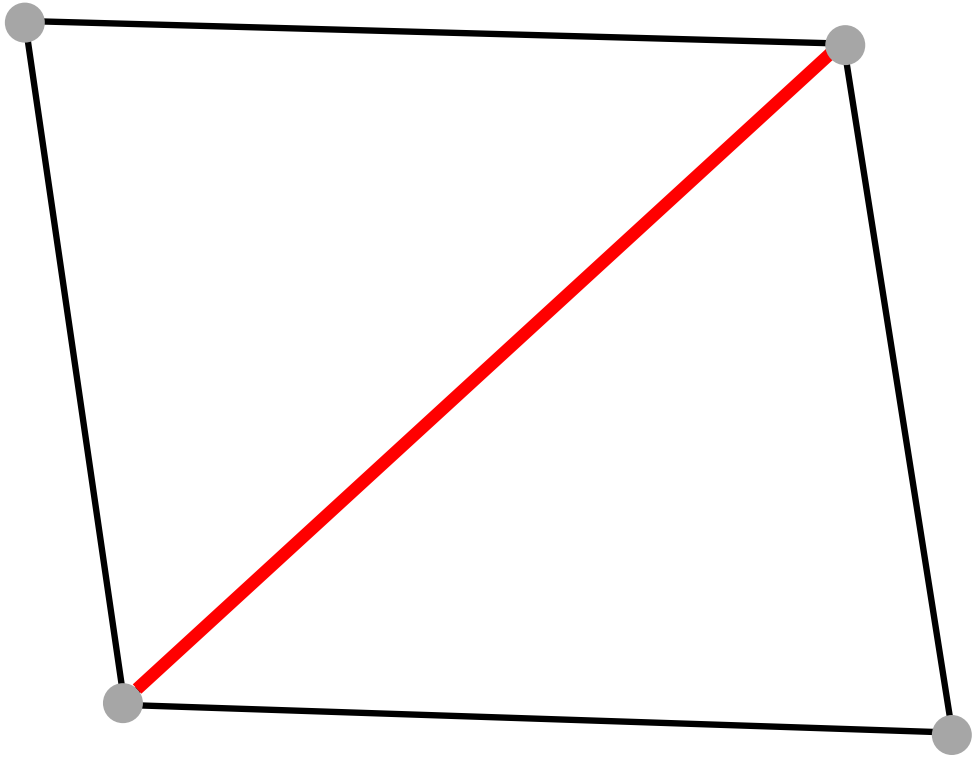
Broken Condition !



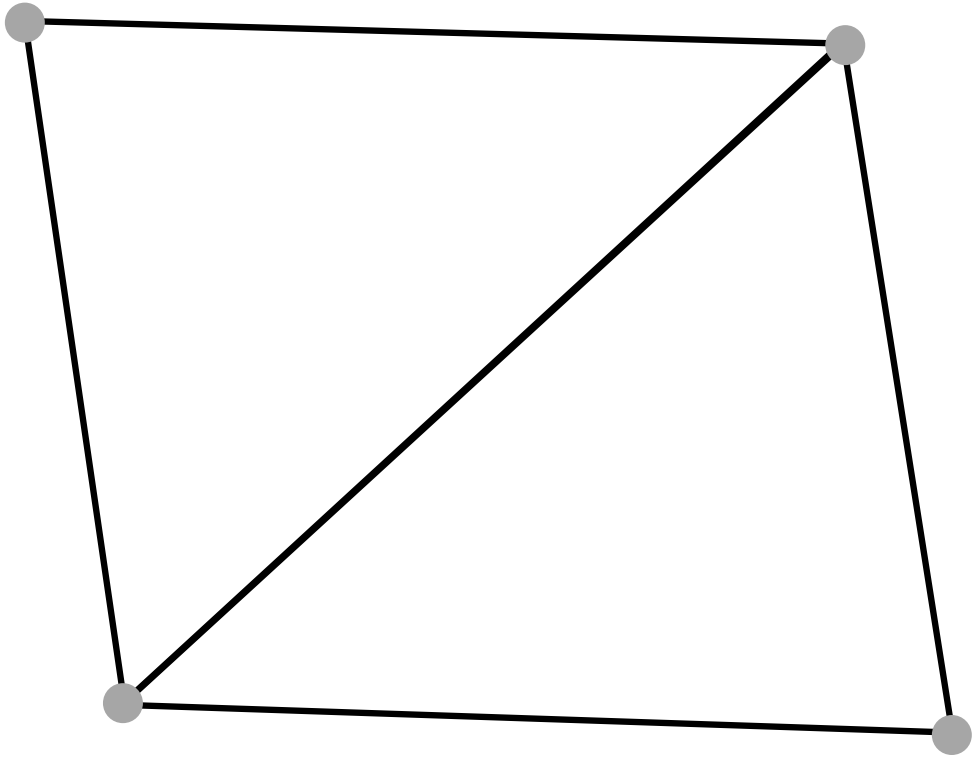
Take the shared edge...



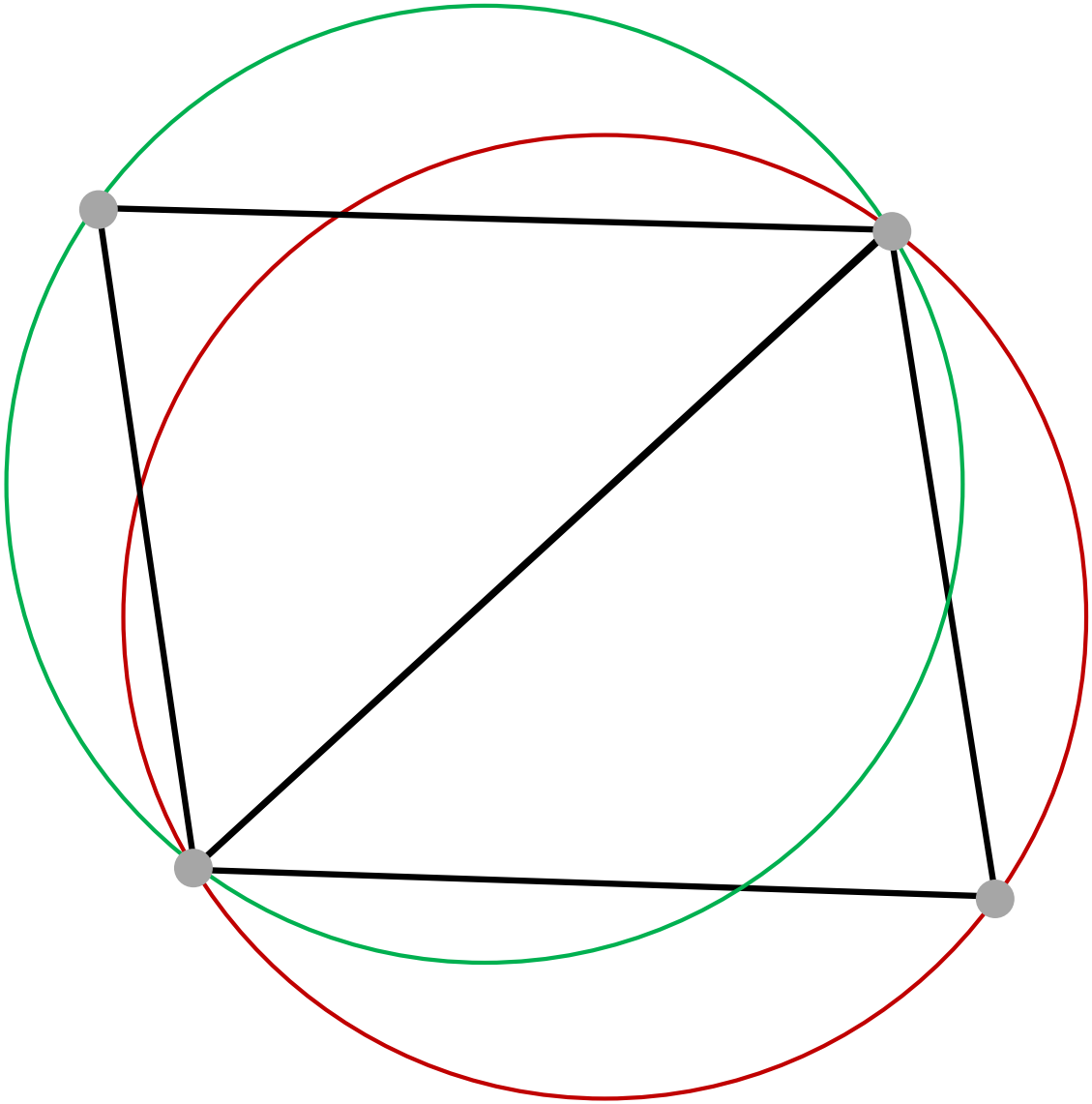
And flip it !



Resulting in two new triangles



Valid !!!



Hints for implementing in 3D

- Triangulation in this way is a little bit trickier in 3D
- New points *near* rather than *on* triangle surfaces
- Circumspheres (rather than Circumcirles)

- Think carefully before attempting as an extension !

Mesh Simplification

The Problem

- The more vertices, the more triangles
- The more triangles, the more iterations
- The more iterations, the slower the rendering

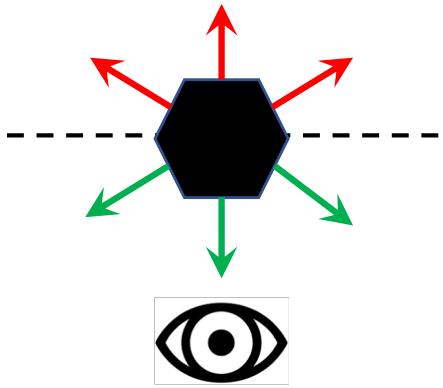
- What if we reduced the number of vertices ?
- That would speed everything up !

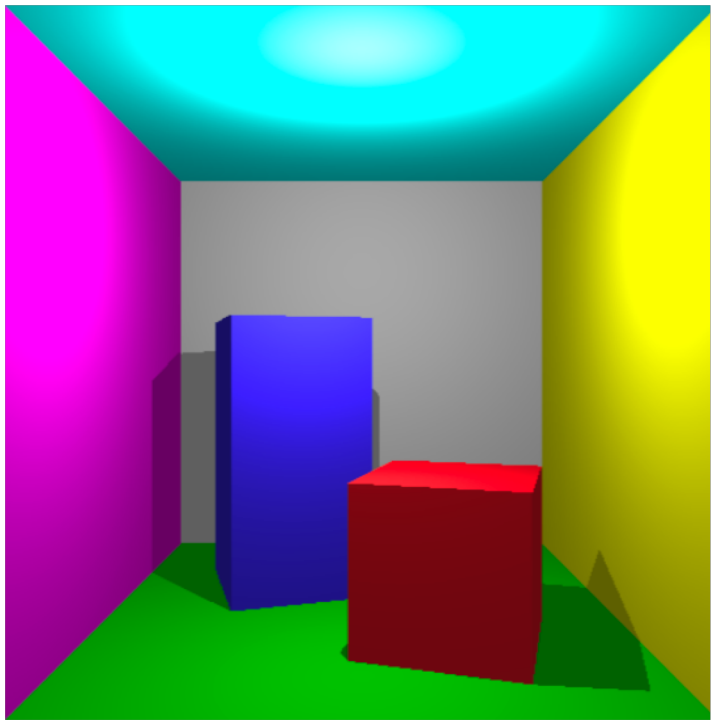
- But at the expense of model fidelity ?

Back-Face Culling

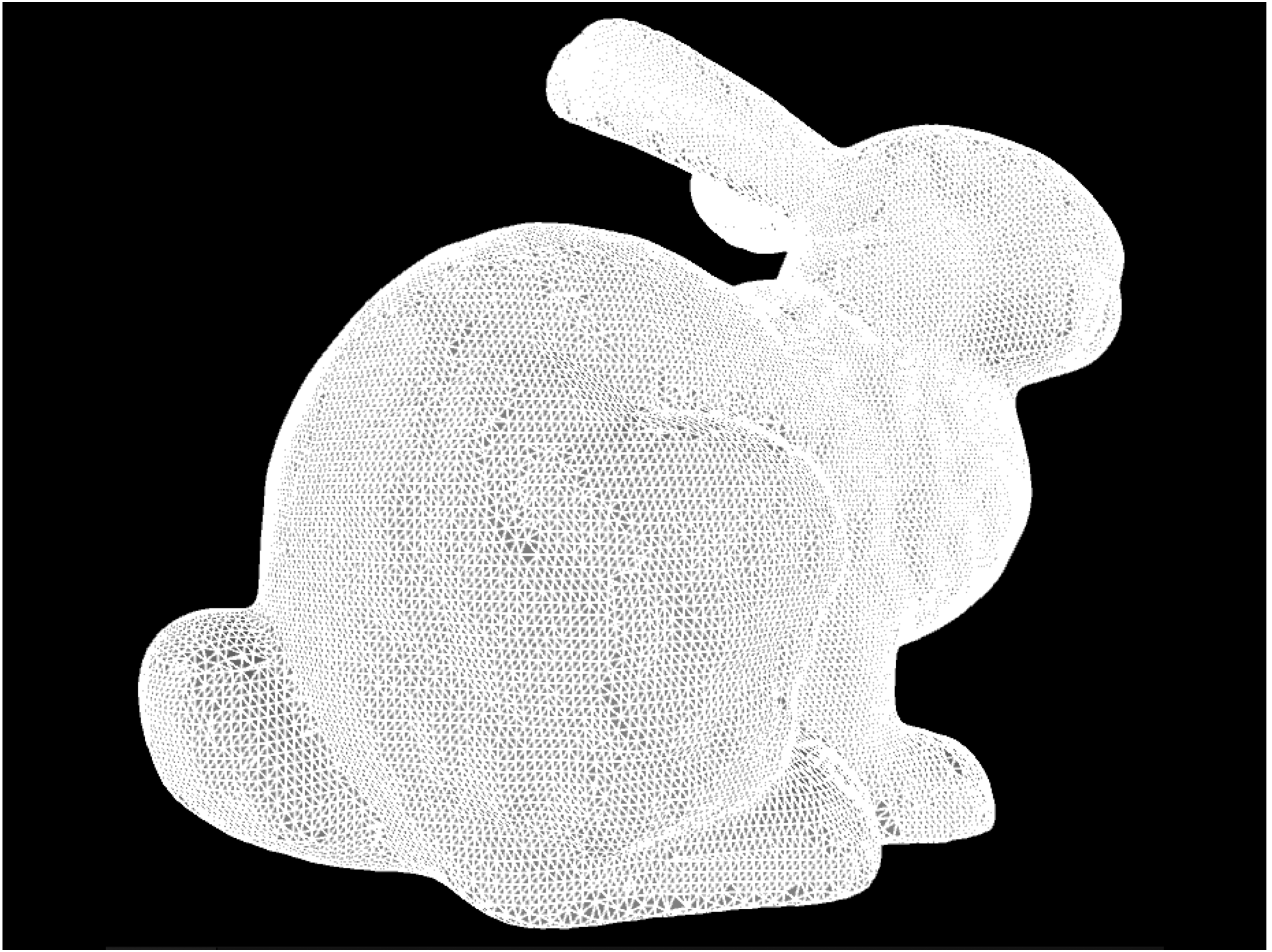
- Already seen this for removing unseen faces…

- Check each triangle's normal…
- If it faces away from camera…
- Remove it from the render !

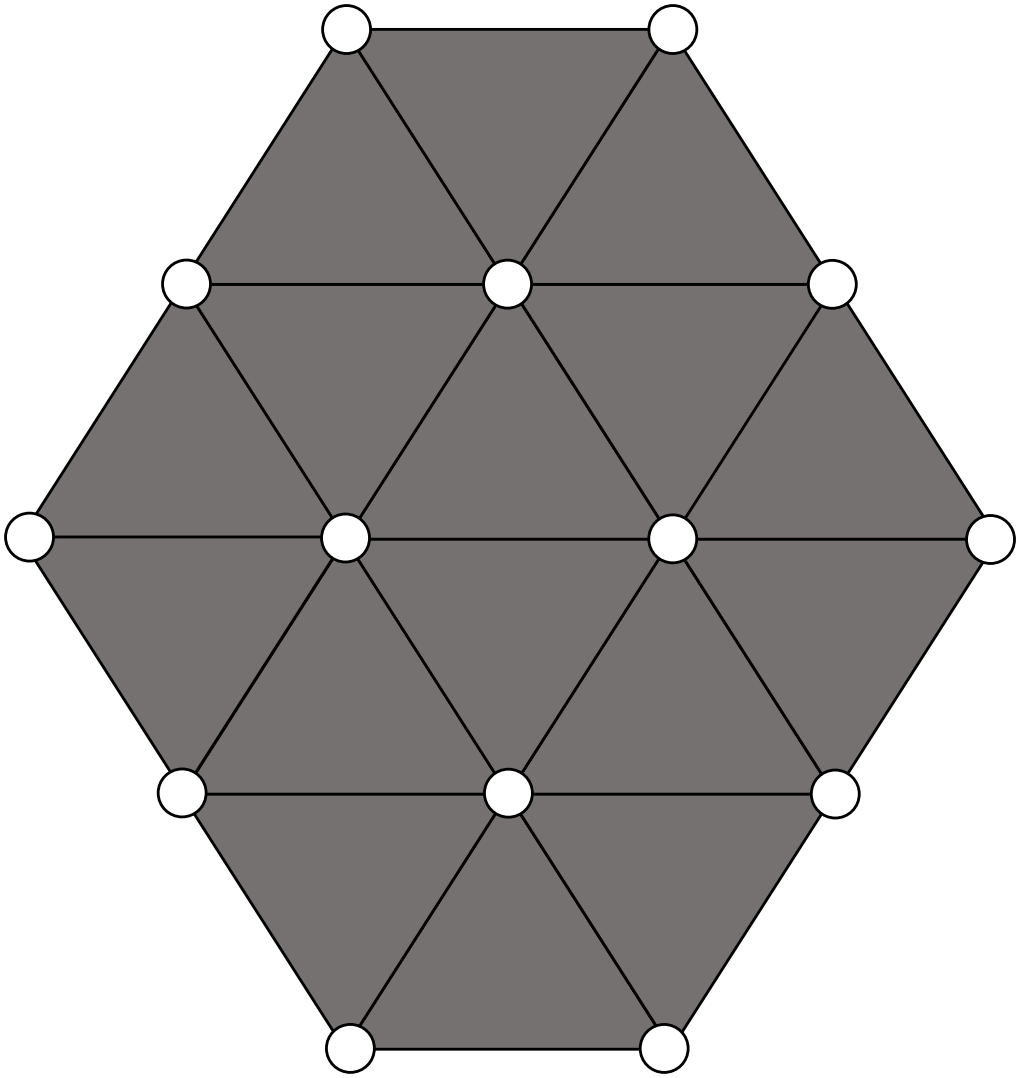




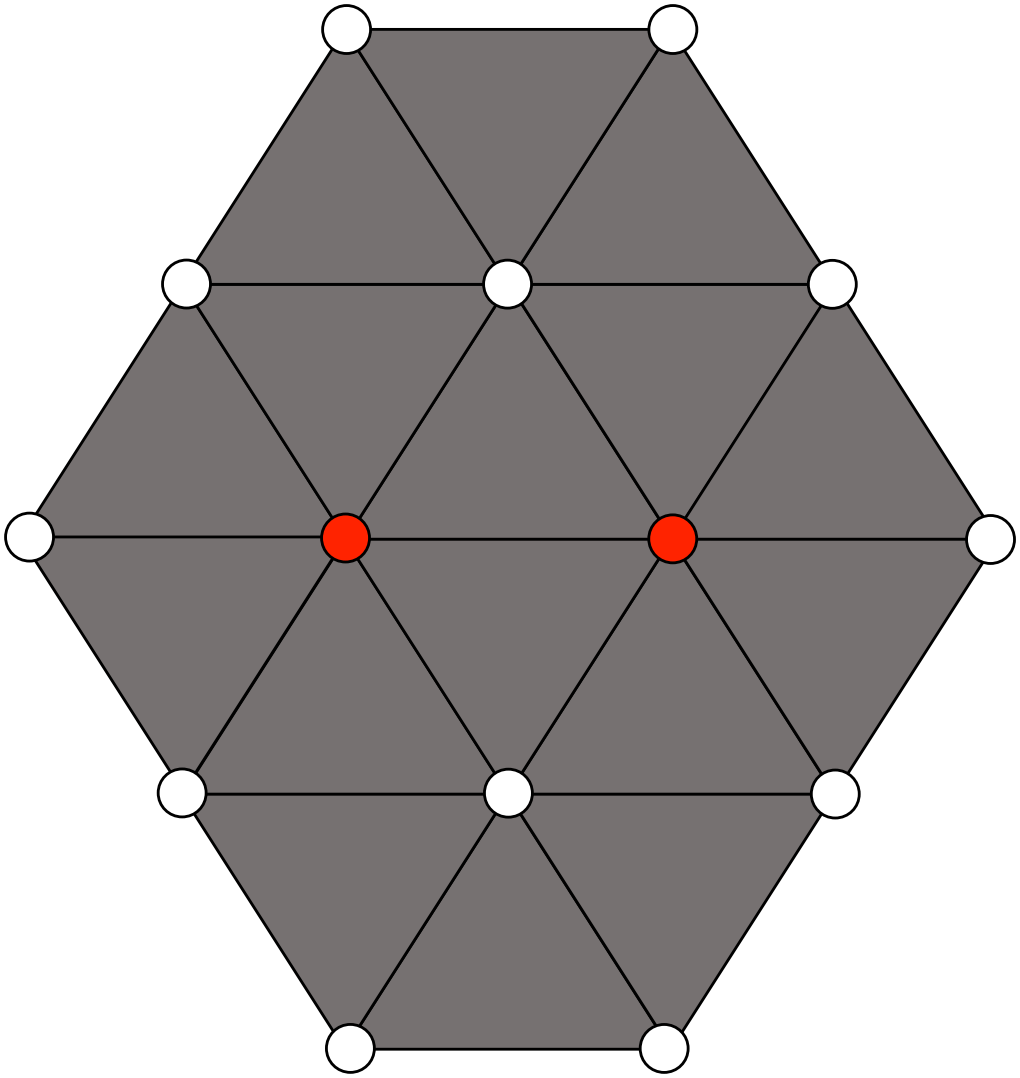
- But this will only takes us so far…



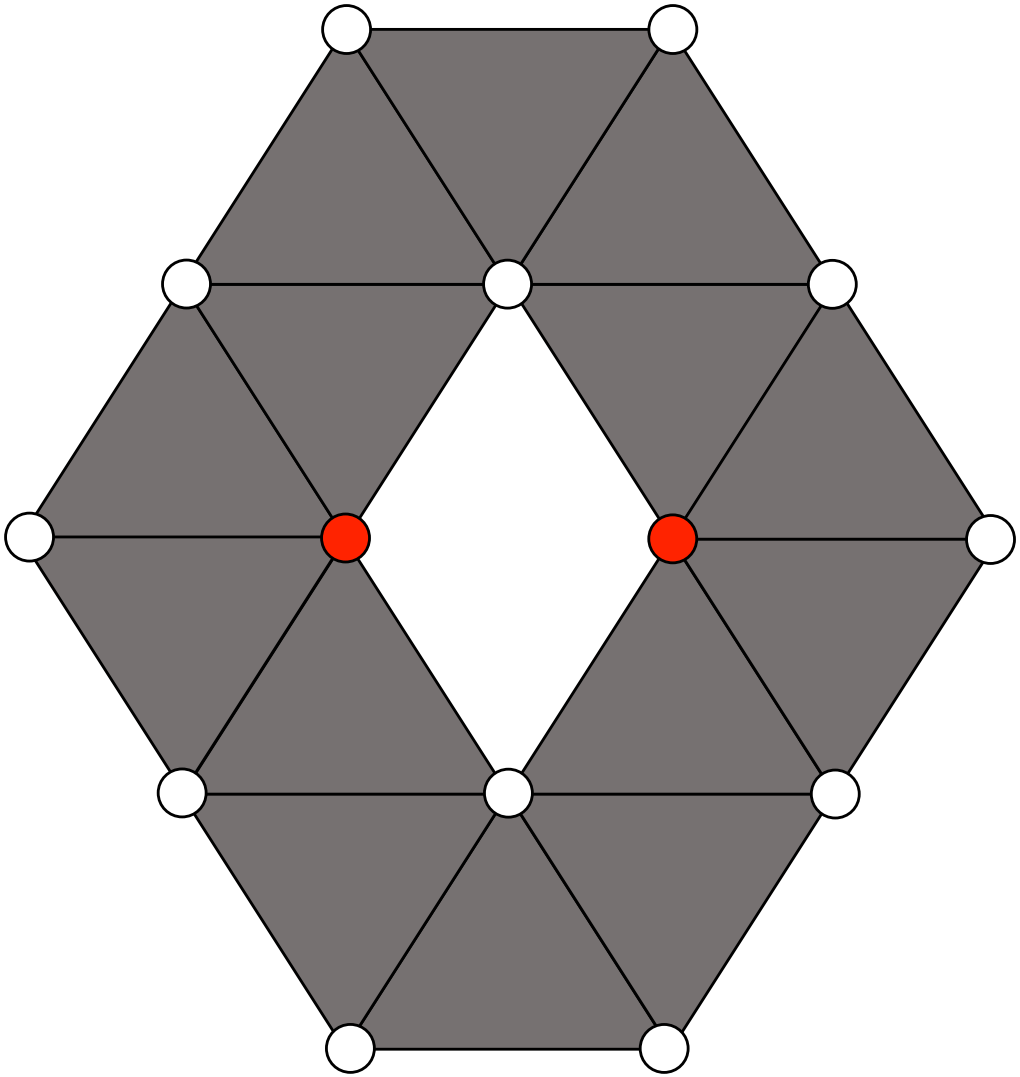
Model Simplification – Vertex Merging



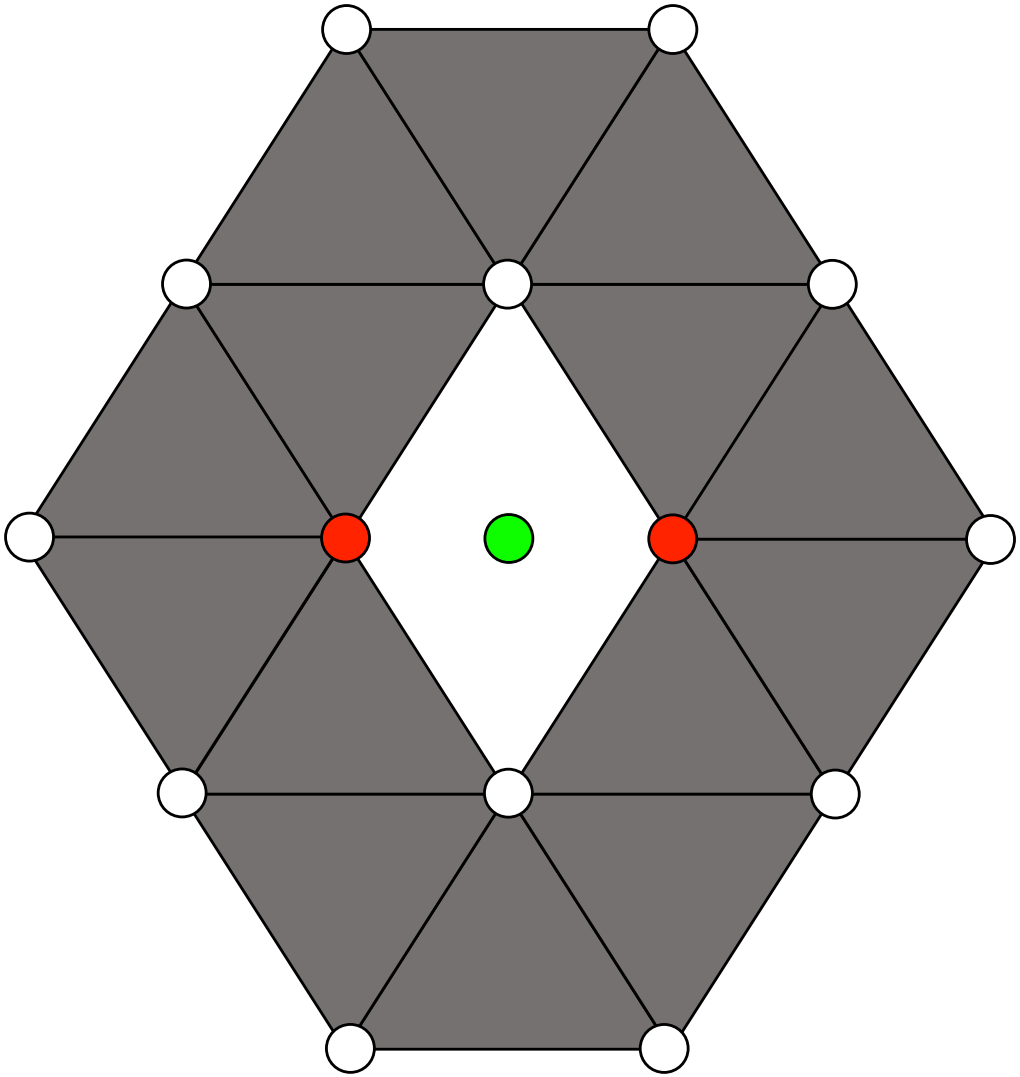
Select Pair of Linked Vertices



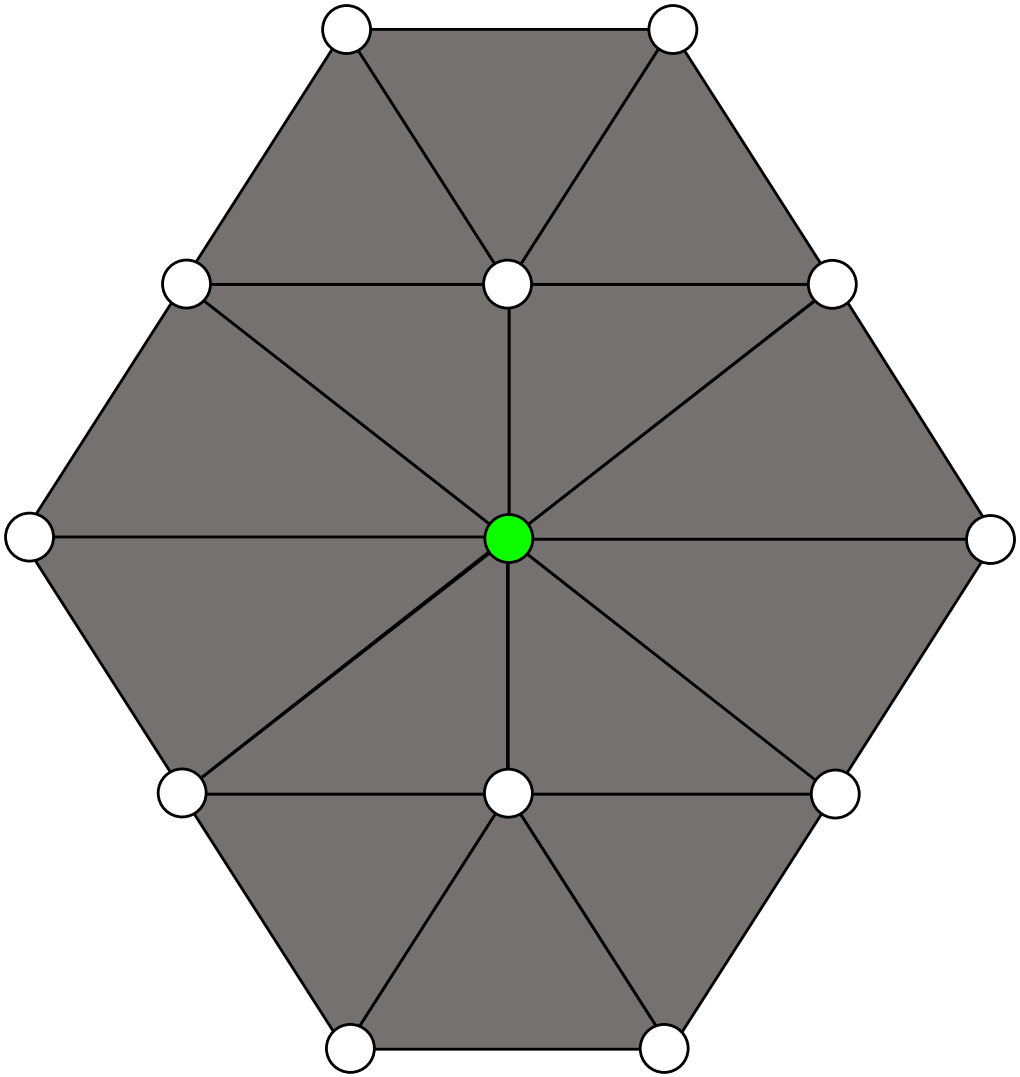
Remove Triangle Pair



Calculate Average Point



Merge Vertices to Repair Hole

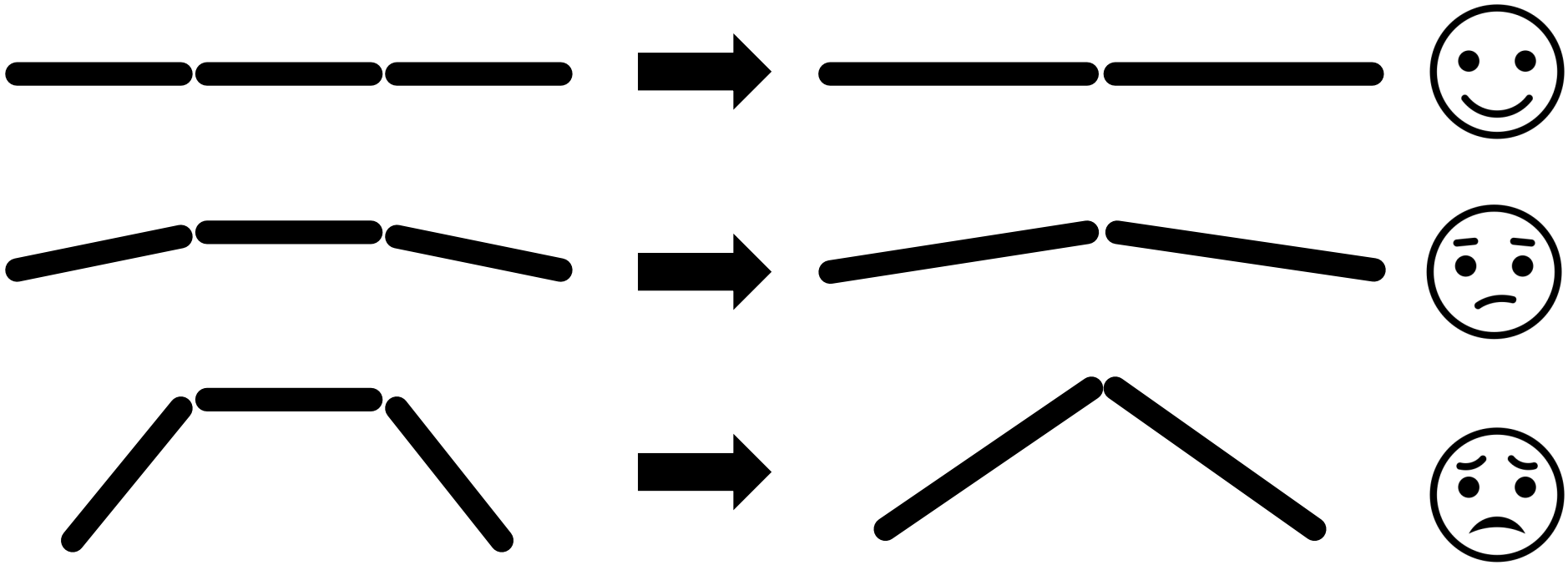


Vertex Merging

[MeshOpt (<-) remove]

Removal Selection

- Selection of triangles to remove is very important
- We must select those that are “most” redundant
- If we just remove arbitrarily, we risk loosing fidelity…



“Gravitational” Vertex Merging

- An elegant category of simplification algorithms
- Each vertex is attracted towards all others
- The closer the vertices, the stronger the pull…
- So that the closest vertices will eventually coalesce

- Which is exactly what we want !
- Small triangles and slivers are lost
- Large, important triangles are kept

Gravitational Vertex Merging

[Gravity2D]

Anchors

- By anchoring vertices to their original location
- We minimise the deformation of the model

- Adjusting the length of the anchor “chains”
- Allows us to reach a balance between…

- Simplification (reduction of vertices/triangles)
- Fidelity (faithfulness to the original model)

END ?