Global Optimization of Mesh Quality

David Eppstein

Univ. of California, Irvine
Dept. of Information and Computer Science

Outline:

Introduction

Mesh quality issues, meshing steps

Connectivity optimization

Delaunay triangulation, edge insertion

Global point placement

Quadtrees, incremental Delaunay refinement

Individual point placement

Quasiconvex programming, sliver exudation

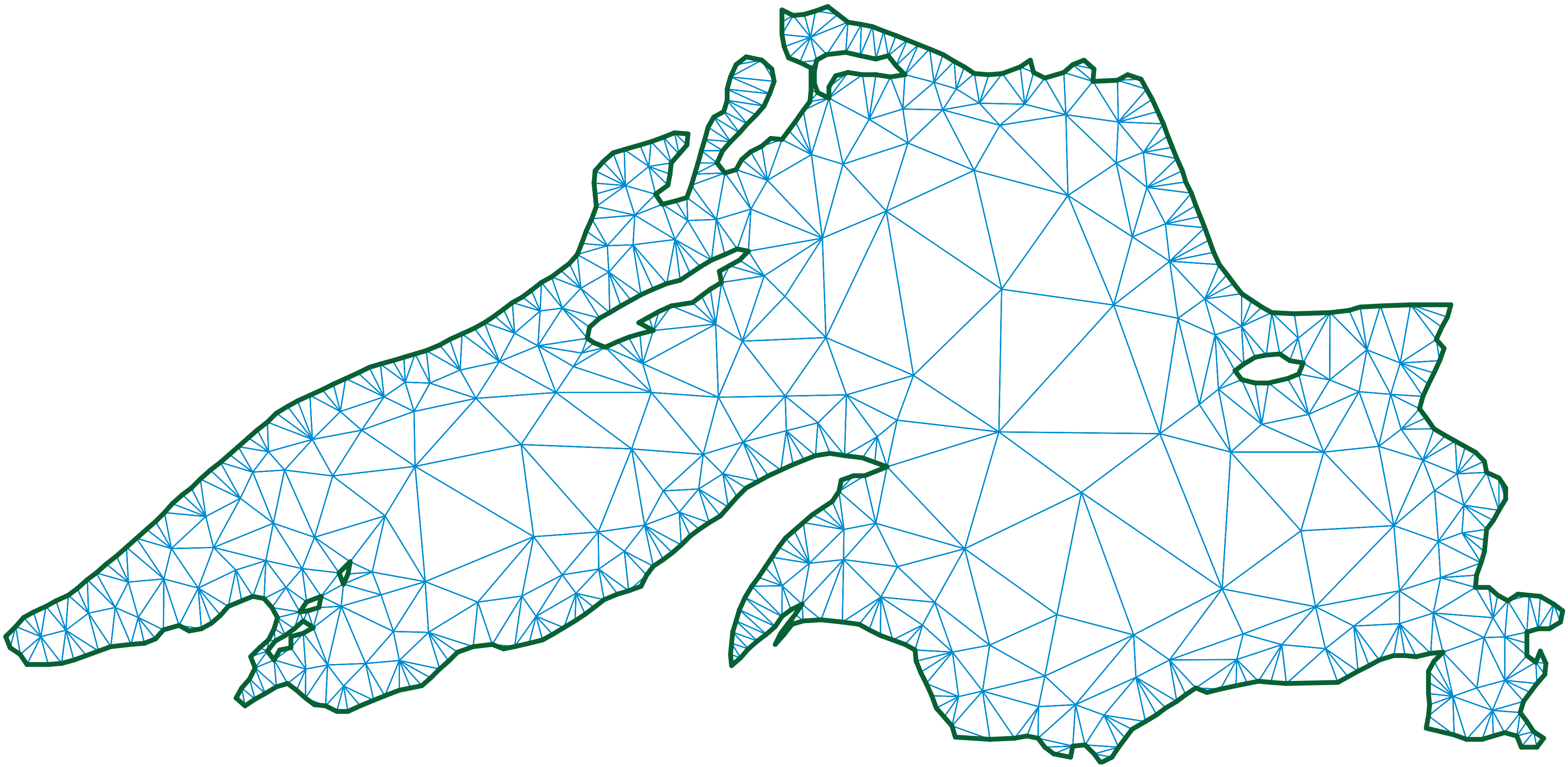
I. What is Meshing?

Given an input domain
(manifold with boundary or possibly non-manifold geometry)

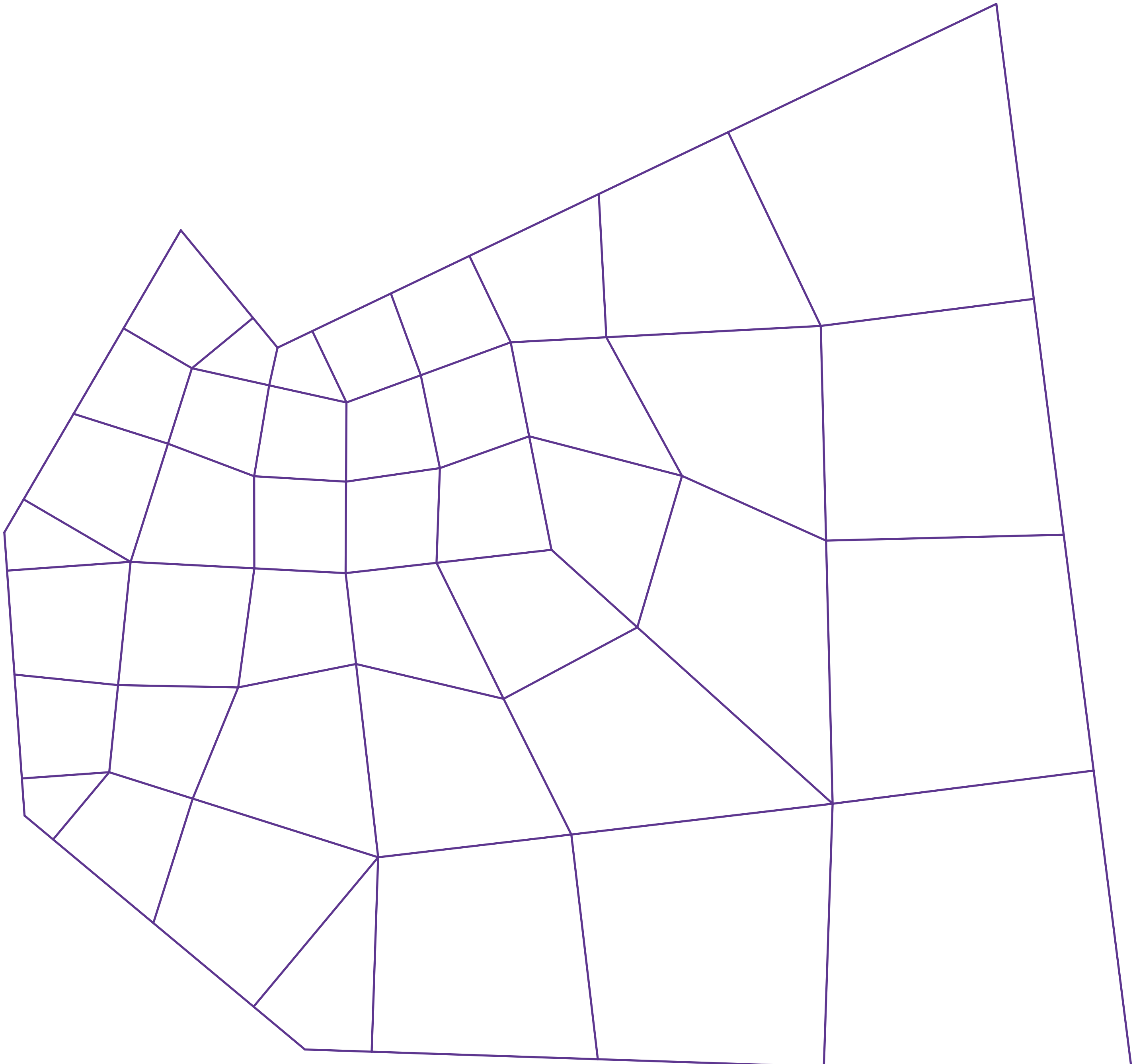
Partition it into simple cells
(triangles, quadrilaterals, tetrahedra, cuboids)

Essential preprocessing step for finite element method
(numerical solution of differential equations e.g. airflow)

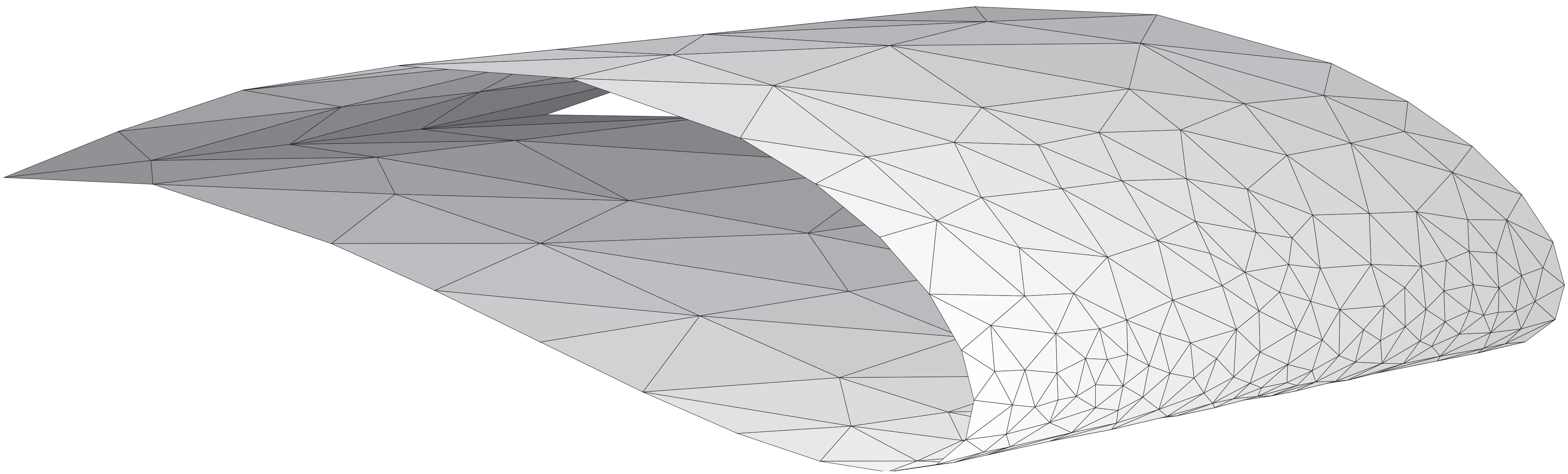
Other applications e.g. computer graphics



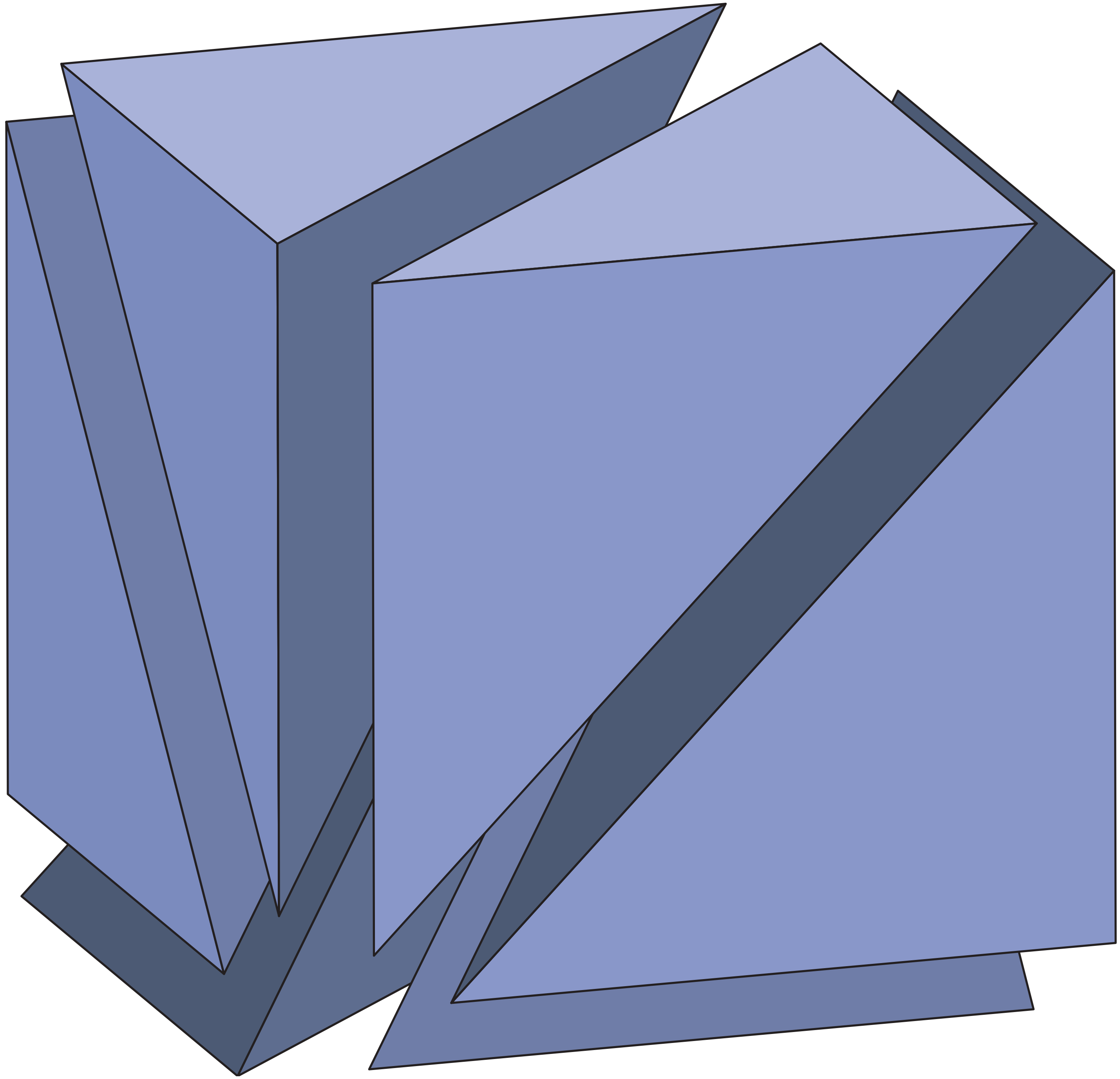
Triangle mesh of Lake Superior [Ruppert]



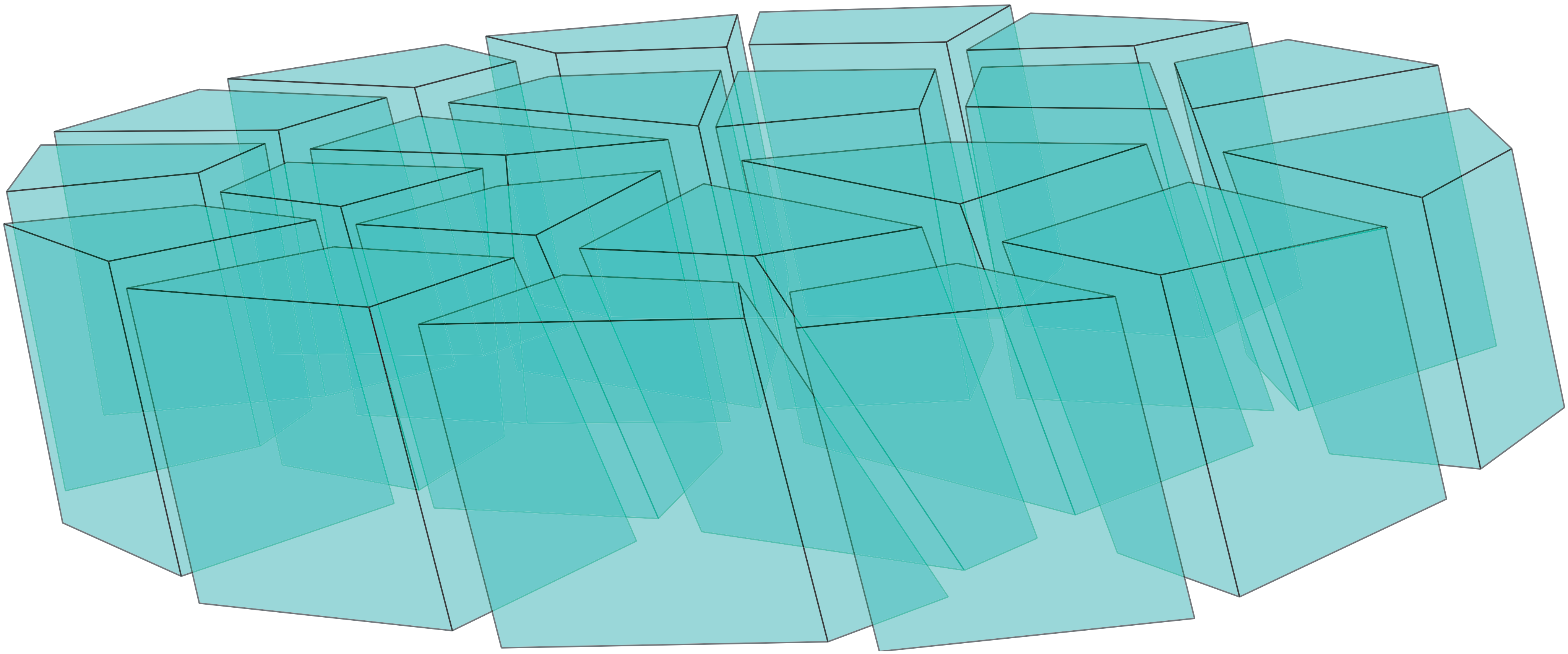
Quadrilateral mesh of an irregular polygon
(all quadrilaterals kite-shaped)



Triangle mesh on three-dimensional surface [Chew]



Tetrahedral mesh of a cube



Portion of hexahedral mesh of elbow pipe
[Tautges and Mitchell]

Mesh Quality Issues

Element type?

2d: triangles vs quadrilaterals
3d: tetrahedra vs hexahedra

This talk: primarily triangles and tetrahedra
Quality guarantees for quad/hex meshes much less developed

Element shape?

Avoid sharp angles, flat angles, distorted elements
Affects accuracy of numerical simulation

Element size?

Need small elements near small features or abrupt changes in solution
large elements ok in uninteresting parts of domain

Number of elements?

More elements = slower solution time

Elements on domain boundaries?

May be required to match existing domain boundary mesh
for quality reasons or to mesh multi-domain input

Theory vs. Practice

Practical emphasis: get system to work

Techniques often ad-hoc

Evaluation of result quality may be visual,
or by examining numerical quality of “typical” meshes

Works well in many cases
Unusual inputs may lead to bad mesh, slow construction, or crashing

Typical paper outlet: Meshing Roundtable
(should be familiar to this audience)

Theoretical emphasis: prove system works

Prove guarantee that method gives high-quality mesh
with small polynomial for worst-case running time

Leads to robust algorithms that work every time

Guarantees may be weaker than typical practice
Bigger problem: lack of implementation

Typical paper outlet: Symposium on Computational Geometry
(but also sometimes in Roundtable)

This tutorial mainly concentrates on theoretical approach

Goal: Synthesize best practices from theory and practice

Implement fast, guaranteed-quality algorithms
maintaining speed, robustness, mesh quality guarantees

Apply practical mesh improvements to lift quality above worst-case guarantees
without violating quality, robustness guarantees

Evaluate meshes and meshing algorithms
by both provable and observed meshing quality

Need continued research on appropriate quality measures
as well as meshing algorithm development

Also insufficiently studied:
What is a “typical” meshing input, and how can we use its structure
to improve results compared to worst-case inputs?

Maintain communications between SCG and Roundtable communities

What is the “right” quality measure?

Not fully settled, varies by meshing application

Avoid sharp angles? Allow sharp angles but avoid obtuse angles?

Use non-directional quality measures?
Align mesh elements with domain boundaries?
Concentrate quality near boundary, allow worse elements in interior?

A priori measures (depending only on domain geometry)
vs. measures depending on finite element solution

What to do when you don't know the right quality measure?

Use general-purpose algorithms that can
be adapted to different quality measures

Prove that certain algorithms (primarily, Delaunay triangulation)
simultaneously optimize many quality measures

Relate different measures to each other
allowing quality guarantees for one measure to apply to another

Typical mesh generation stages:

Generate initial point placement

Well spaced, other quality considerations

Determine mesh connectivity

Usually, Delaunay triangulation

May be intermixed with point placement stage

Iterate mesh improvement stages

Laplacian or optimization-based smoothing
Flips and other connectivity changes

Order for this talk:

Mesh connectivity
Initial placement
Mesh smoothing

Easier to understand justification for placement after seeing connectivity

Connectivity improvement similar to initial connectivity

Outline:

Introduction

Mesh quality issues, meshing steps

Connectivity optimization

Delaunay triangulation, edge insertion

Global point placement

Quadtrees, incremental Delaunay refinement

Individual point placement

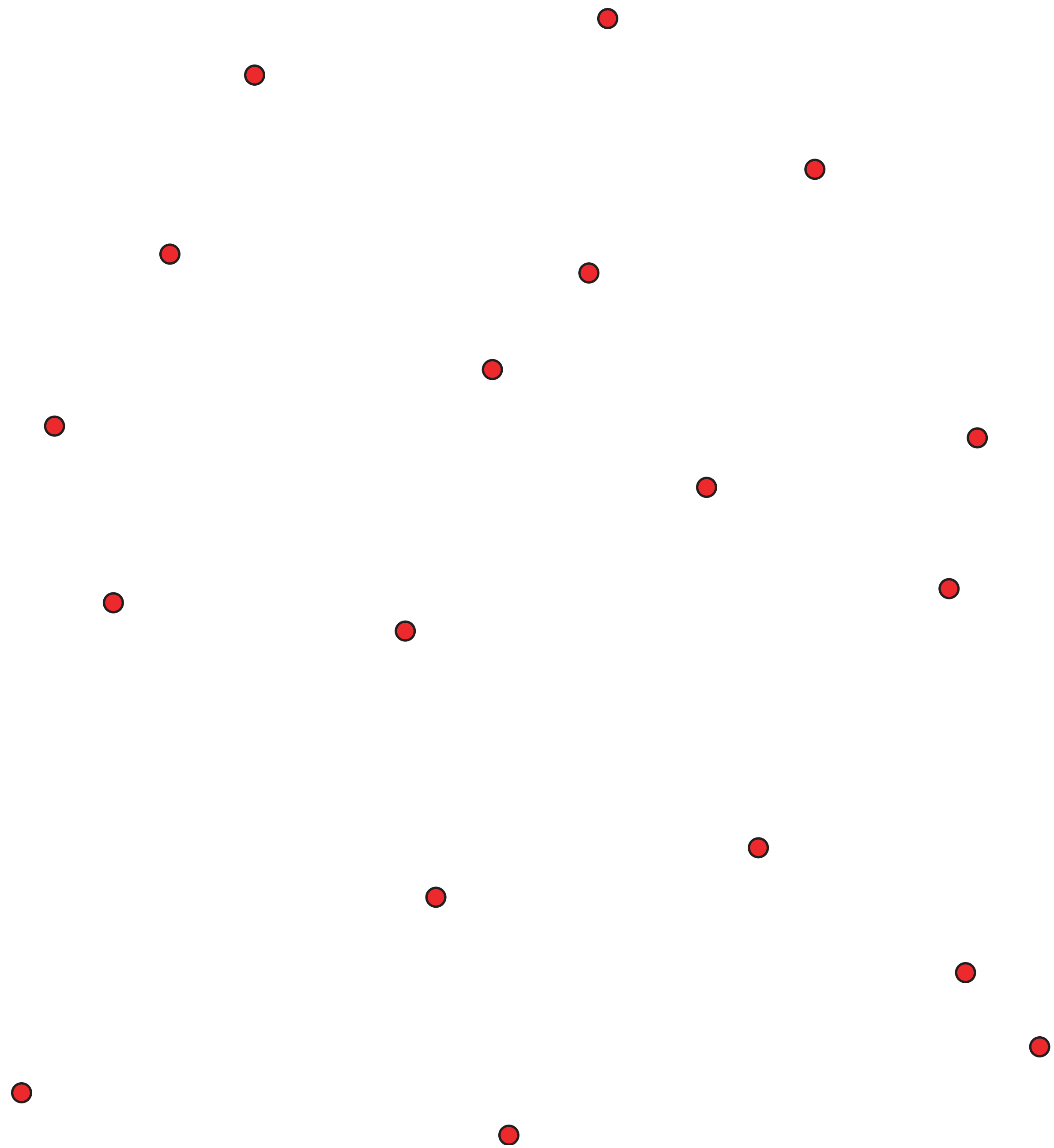
Quasiconvex programming, sliver exudation

Delaunay Triangulation:

Given planar point set

Connect two points by edge
if some circle exists
with them on boundary,
empty interior

Collection of all such edges
for points in general position
(no four cocircular)
forms triangle mesh
covering convex hull

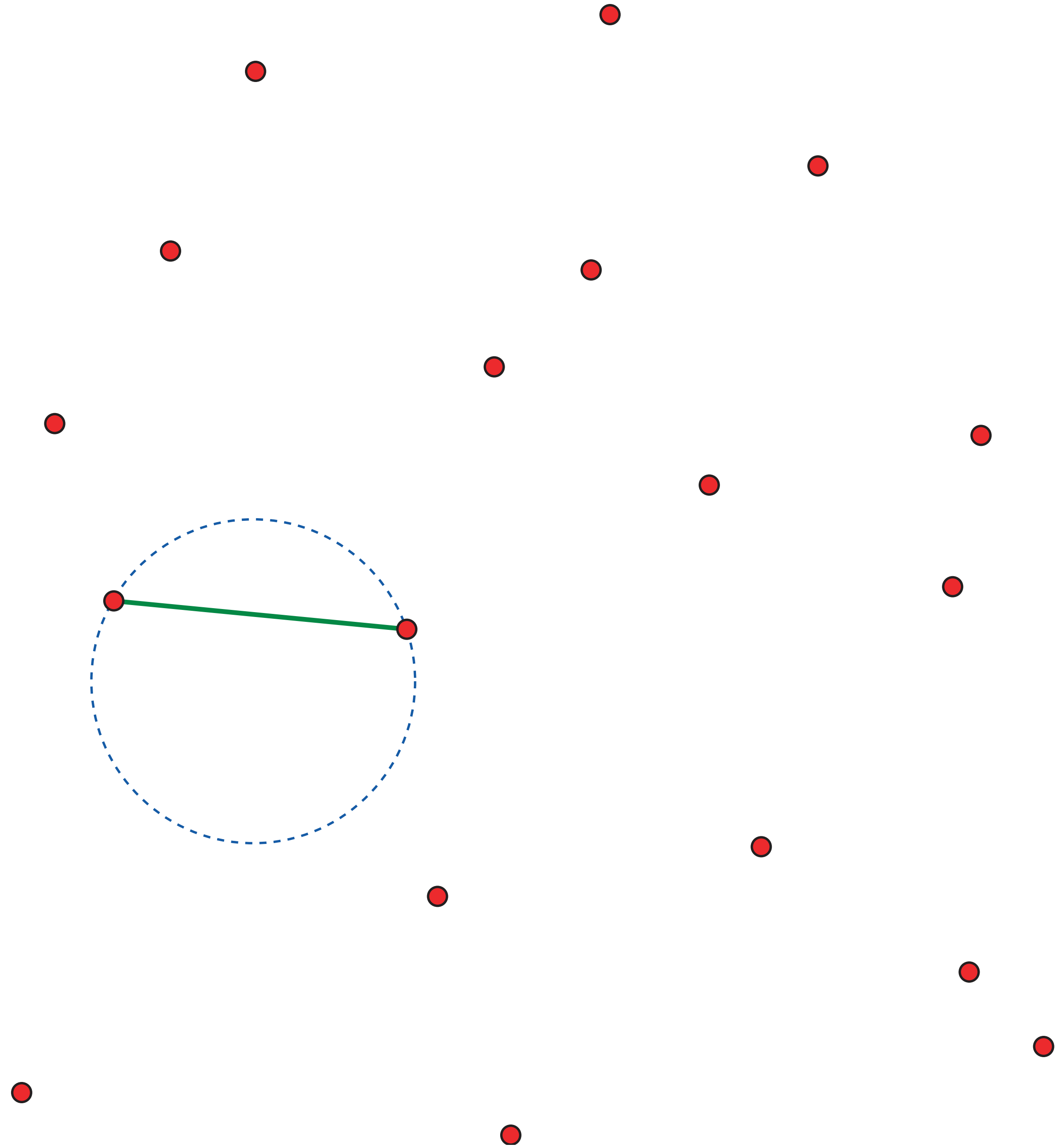


Delaunay Triangulation:

Given planar point set

Connect two points by edge
if some circle exists
with them on boundary,
empty interior

Collection of all such edges
for points in general position
(no four cocircular)
forms triangle mesh
covering convex hull

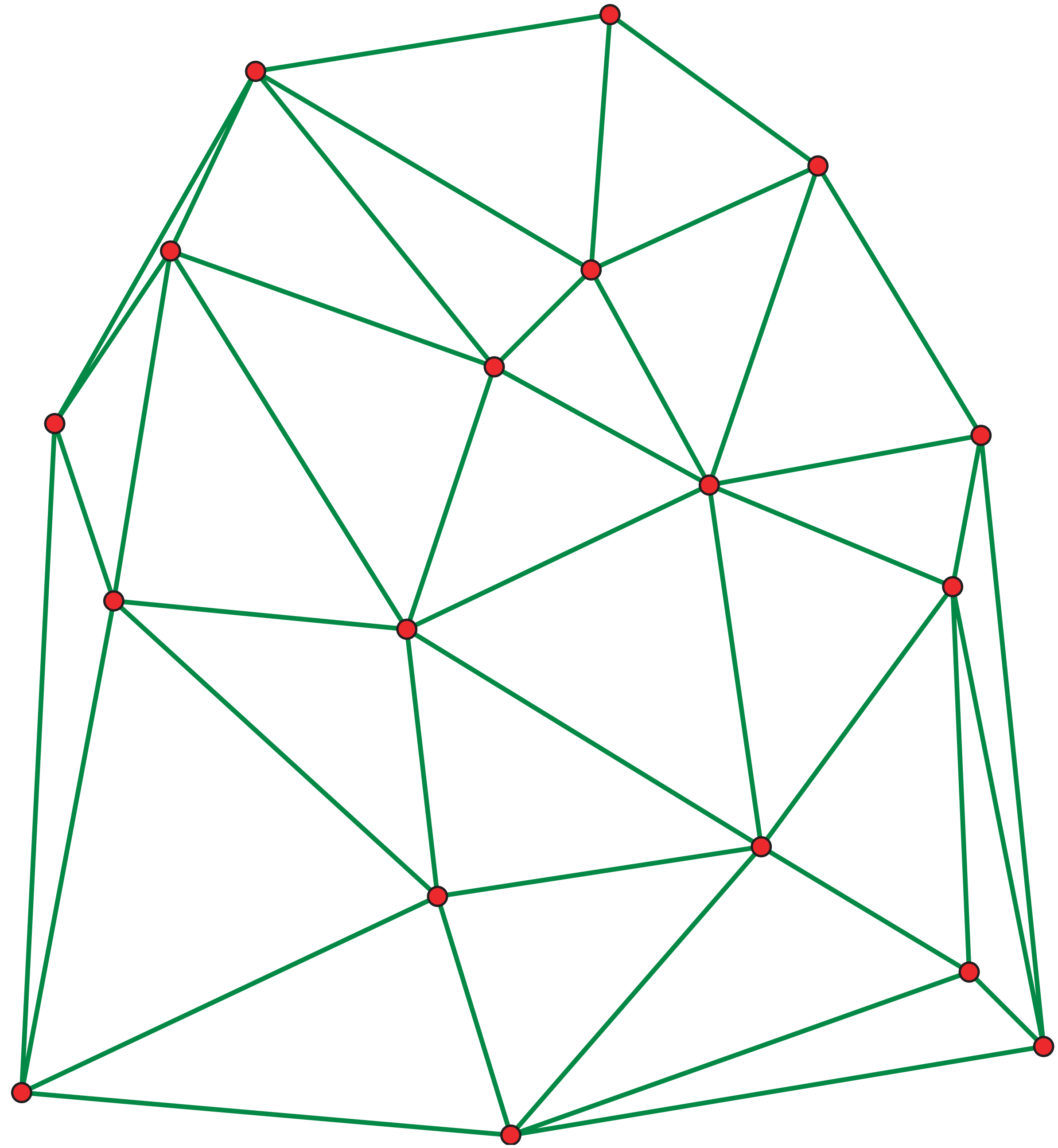


Delaunay Triangulation:

Given planar point set

Connect two points by edge
if some circle exists
with them on boundary,
empty interior

Collection of all such edges
for points in general position
(no four cocircular)
forms triangle mesh
covering convex hull



Delaunay Triangulation:

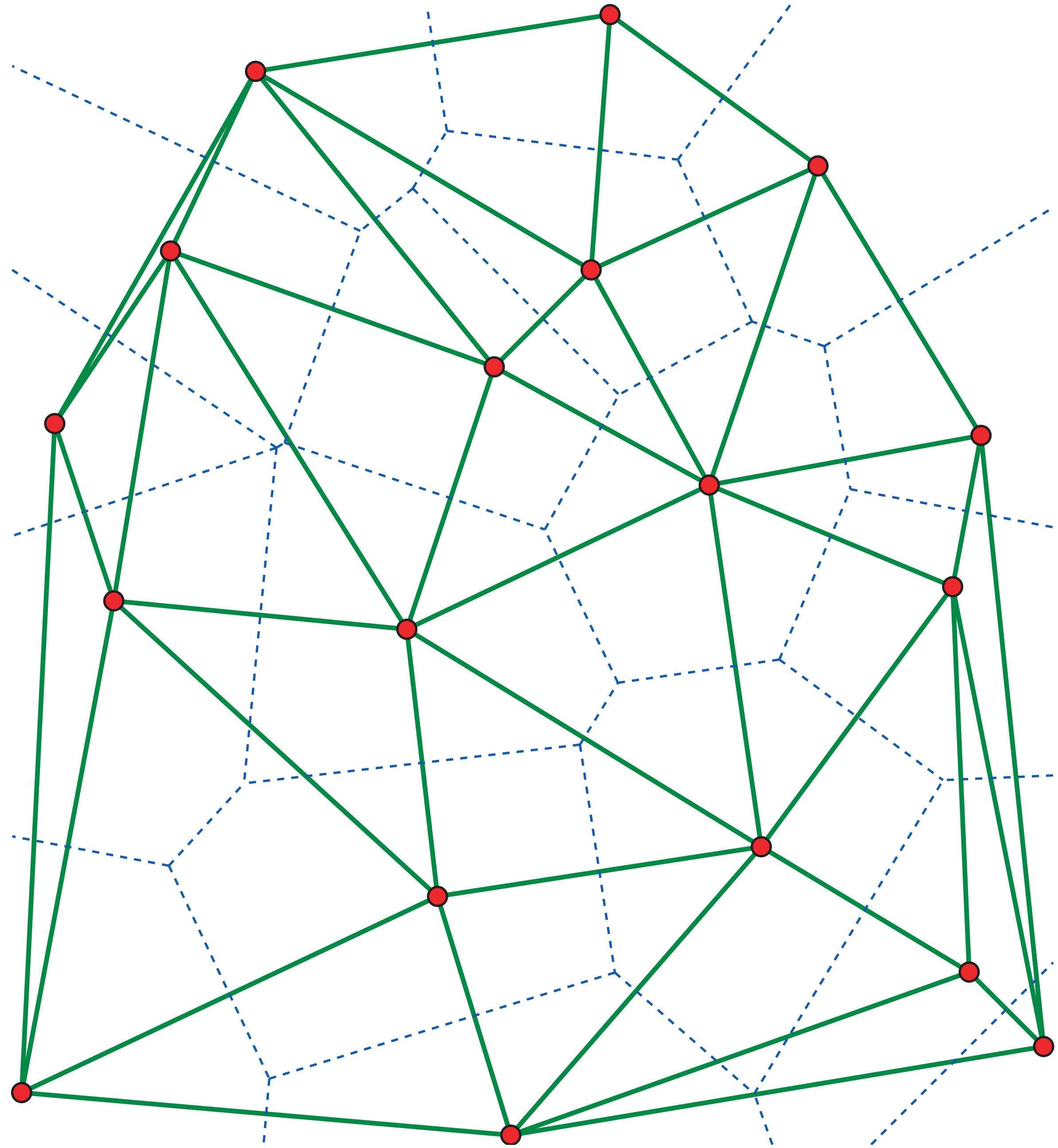
Alternate definition:

Voronoi diagram
is partition into cells
having given points
as nearest neighbors

Delaunay triangulation
is planar dual of VD

Voronoi edges =
centers of empty circles
touching two points

Unless points cocircular,
cells meet in threes
→ triangulation



Lifting Transformation

Map (x, y) plane
to paraboloid $(x, y, x^2 + y^2)$

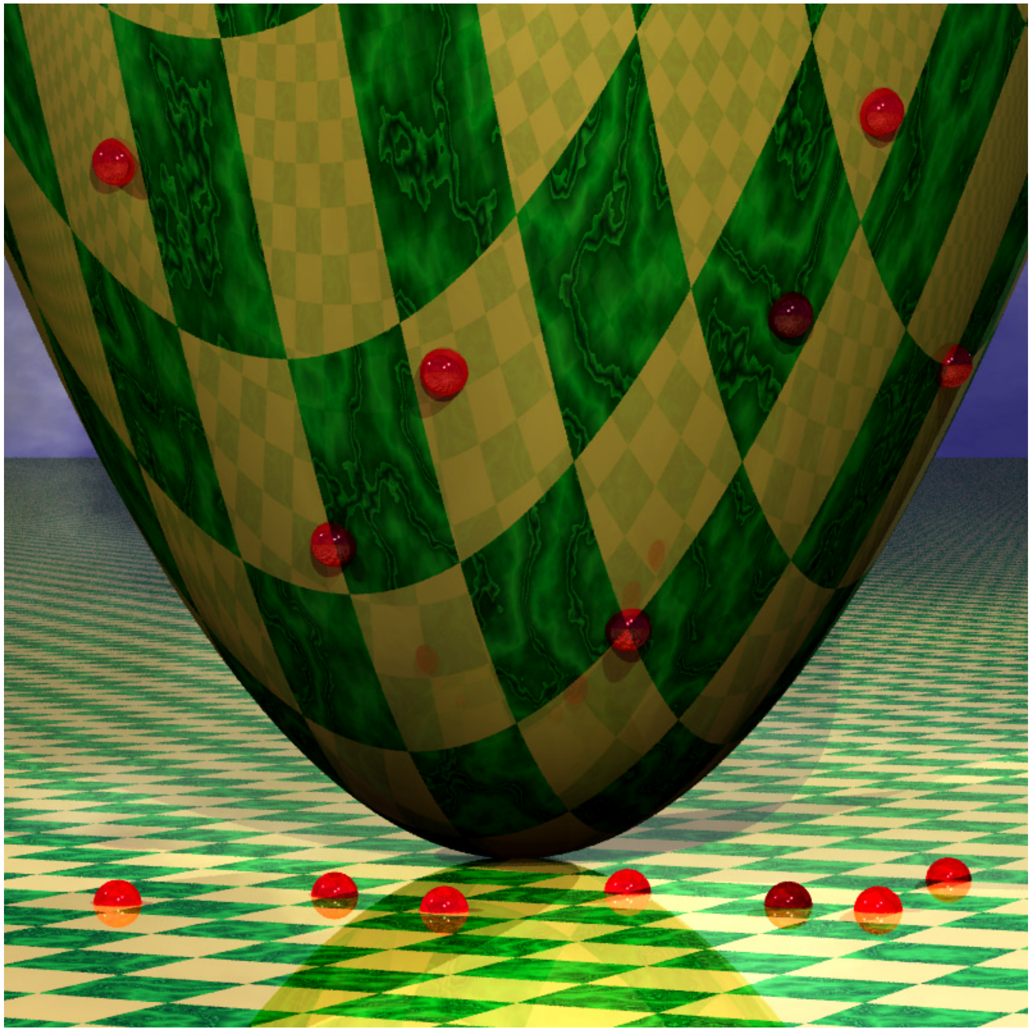
Triangulation lifts to triangulated surface

Circle lifts to set of coplanar points
on the paraboloid (and conversely)

Empty circle lifts to empty halfspace
below the corresponding plane

Delaunay triangulation lifts to
lower facets of convex hull

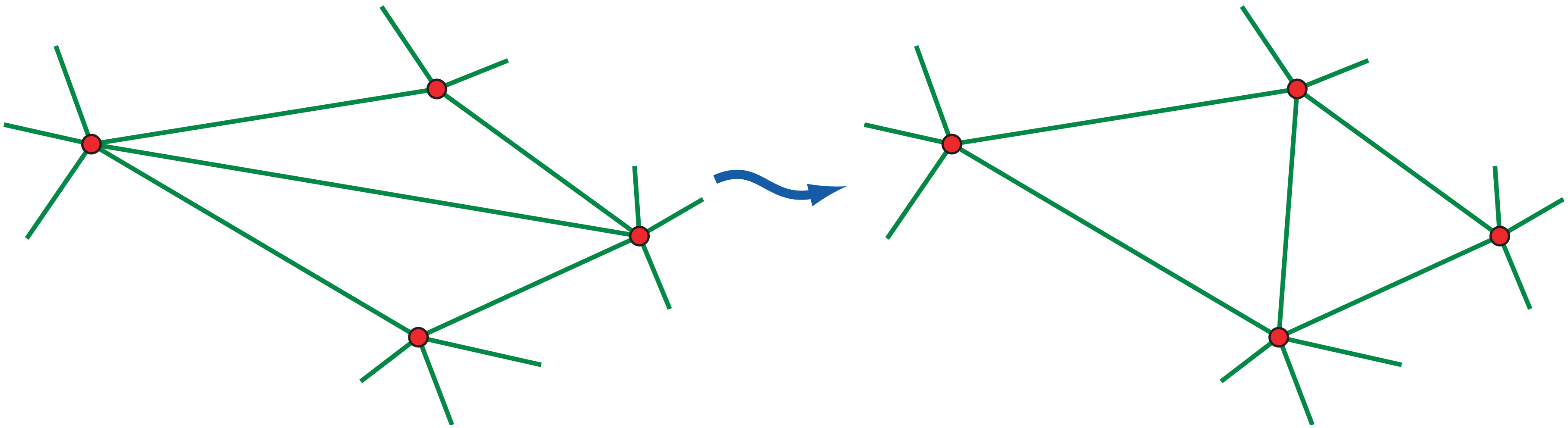
i.e. triangulation is Delaunay
if and only if its lifted surface is convex



Flipping

Algorithm for computing Delaunay triangulations [Sibson, Computer J. 1978]

Not so useful directly as a practical implementation technique
Very useful for proving optimality properties



Start with any (non-optimal) triangulation

Find any two adjacent triangles that form a convex quadrilateral
and that are not the Delaunay triangulation of the four corners

Replace with the Delaunay triangulation, repeat until stuck

If non-Delaunay, lifted surface has a concave edge, always flippable
So can't get stuck until correct Delaunay triangulation found

Flipping always lowers lifted surface, so can't get into infinite loops

Flipping and Optimal Triangulation

Suppose you have a quality measure that depends only on the shape of each individual triangle

Further suppose that, for every convex quadrilateral, the Delaunay triangulation is at least as good as the other triangulation

Then, Delaunay triangulation is optimal for all point sets

Proof:

Suppose that some triangulation T is optimal and non-Delaunay

Start flipping from T

Each flip improves the triangulation,
so when the Delaunay triangulation is reached,
it must be at least as good as T

Example DT optimality result:

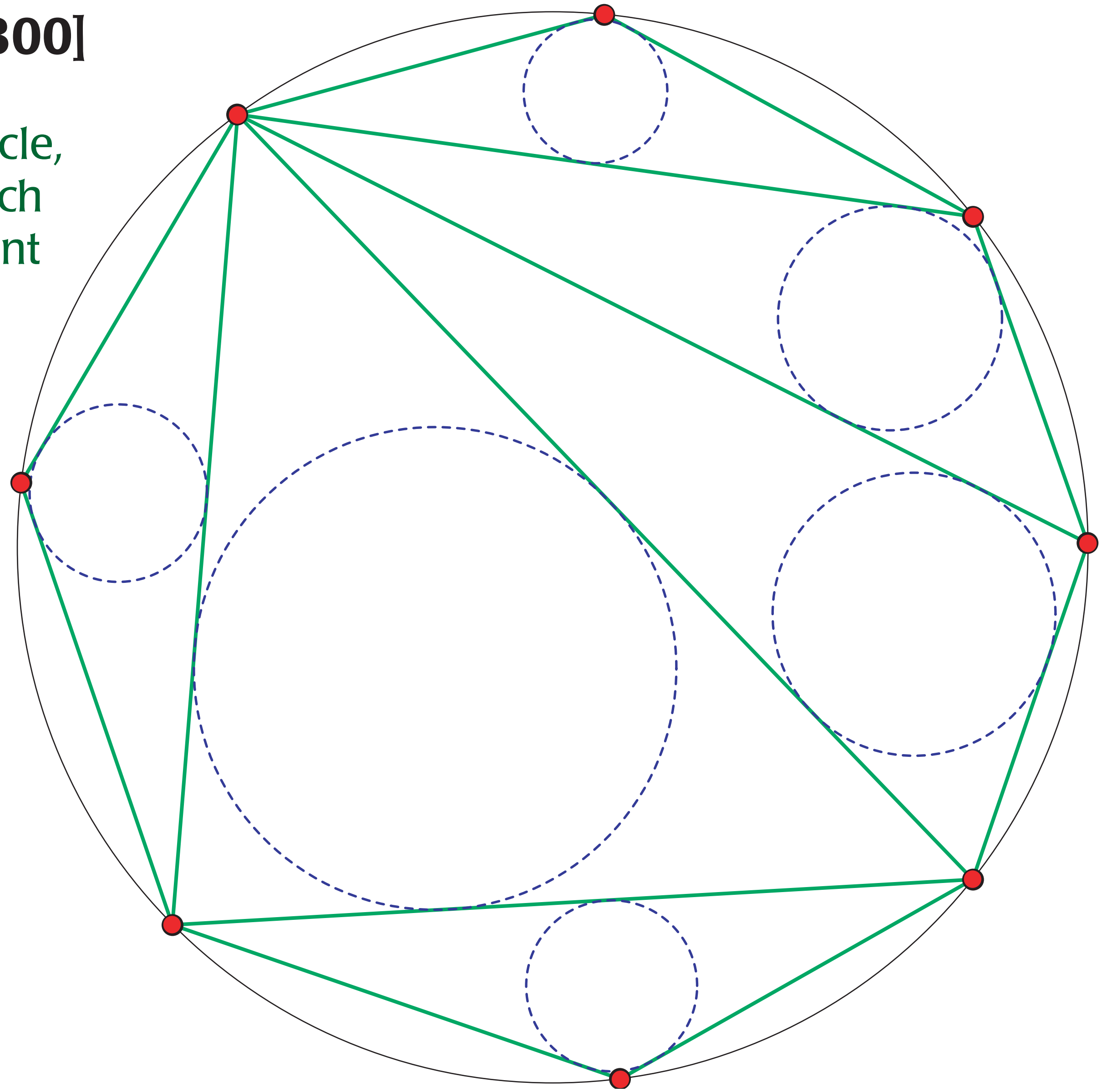
Japanese Temple Theorem [ca 1800]

If a convex polygon is inscribed in a circle, triangulated, and circles inscribed in each triangle, then sum of radii is independent of which triangulation is chosen
[e.g. http://www.cut-the-knot.com/proofs/jap.html]

So, if quality measure = sum of inradii
then for convex quadrilateral,
two triangulations equally good
exactly when DT is ambiguous

When only one triangulation is DT,
it has the smaller sum of inradii

So, more generally,
Delaunay Triangulation maximizes
the sum of inscribed circle radii
among all triangulations of a point set
[Lambert, 1994]



General Delaunay Optimality

Delaunay Triangulation is the triangulation that...

Maximizes the minimum triangle angle
Maximizes the lexicographic vector of triangle angles
[Lawson, 1977]

Minimizes the maximum triangle circumcircle
Minimizes the maximum containing circle
[D'Azevedo and Simpson, SIAM J. Sci. Stat. Comp. 1989]

Minimizes the sum of inscribed circle radii [Lambert, 1994]

Minimizes roughness (integral of gradiant squared) for 3d surfaces
[Rippa, CAGD 1990]

Efficient Delaunay triangulation construction algorithms

Plane Sweep
Sweep line left-right across point set
Build DT behind beach line:
union of parabolae generated by points and sweep line

Divide and Conquer
Divide points into equal subsets by vertical or horizontal line
Recursively construct DT of each half
Merge two halves into single triangulation

Randomized Incremental Flipping
Add points one at a time, order by randomly chosen permutation
After adding each point, flip until have DT again
Trace through flipping history to find where to add each point

All have time O(n log n), optimal
divide and conquer can be O(n) for evenly spaced points

Efficient implementations available
Shewchuk's *Triangle*, http://www-2.cs.cmu.edu/~quake/triangle.html

Extensions of 2d Delaunay Triangulation

Non-convex polygons

Constrained Delaunay triangulation

Has same optimality properties as DT

Cocircular points

DT is ambiguous, not all possible choices may be optimal

Max-min angle triangulation still efficient [Mount and Saalfeld, SCG 1988]

Curved surfaces in three dimensions

Various ways of defining DT

Most successful [Chew, SCG '93] based on empty circle property
but only valid for sufficiently closely spaced points on surface

Riemannian metrics

I.e., for each point of 2d domain,
specify ellipsoid defining local definition of distance
[Leibon and Letscher, SCG 2000]

Generalizes curved surface, non-isotropic meshes

What about quality measures not optimized by Delaunay?

Flipping seldom works
usually gets stuck at local optima

Define more powerful local improvement operator:
Edge Insertion
[Bern et al, Discrete & Comput. Geom. 1993]

Generalization of flipping, so harder to
get stuck at local optima

But, slower running times...

Edge insertion procedure

Start with any triangulation

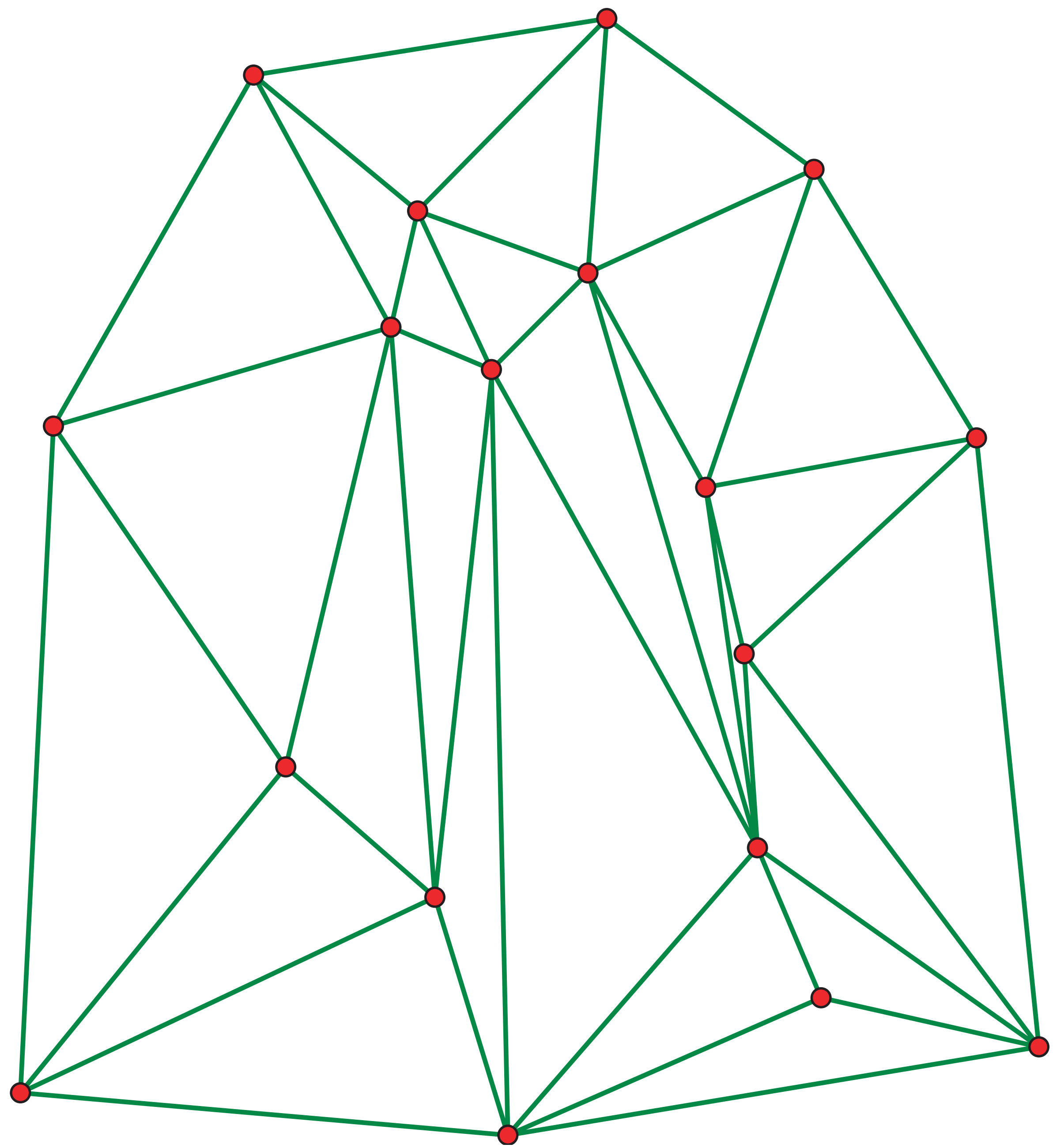
Choose a pair of vertices to be connected by a new edge

Remove all edges crossed by new edge

Retriangulate cleared-out region by greedily choosing ears (triangles having two sides on boundary of region) with quality better than worst removed triangle

If retriangulation succeeds, keep changed triangulation, else back out and try again

Continue improving triangulation until no more improvement possible



Edge insertion procedure

Start with any triangulation

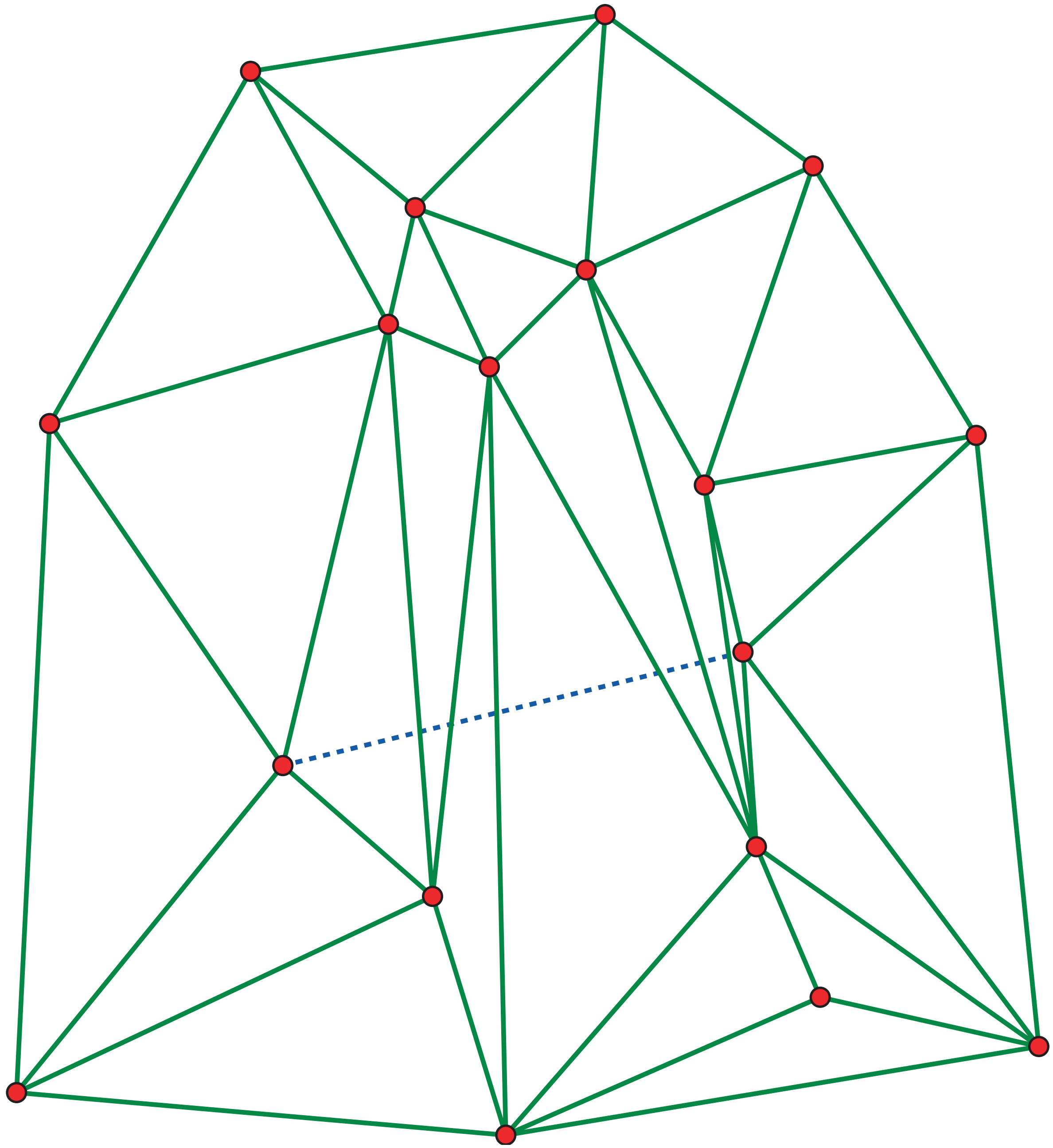
Choose a pair of vertices to be connected by a new edge

Remove all edges crossed by new edge

Retriangulate cleared-out region
by greedily choosing ears
(triangles having two sides on boundary of region)
with quality better than worst removed triangle

If retriangulation succeeds,
keep changed triangulation,
else back out and try again

Continue improving triangulation until no more improvement possible



Edge insertion procedure

Start with any triangulation

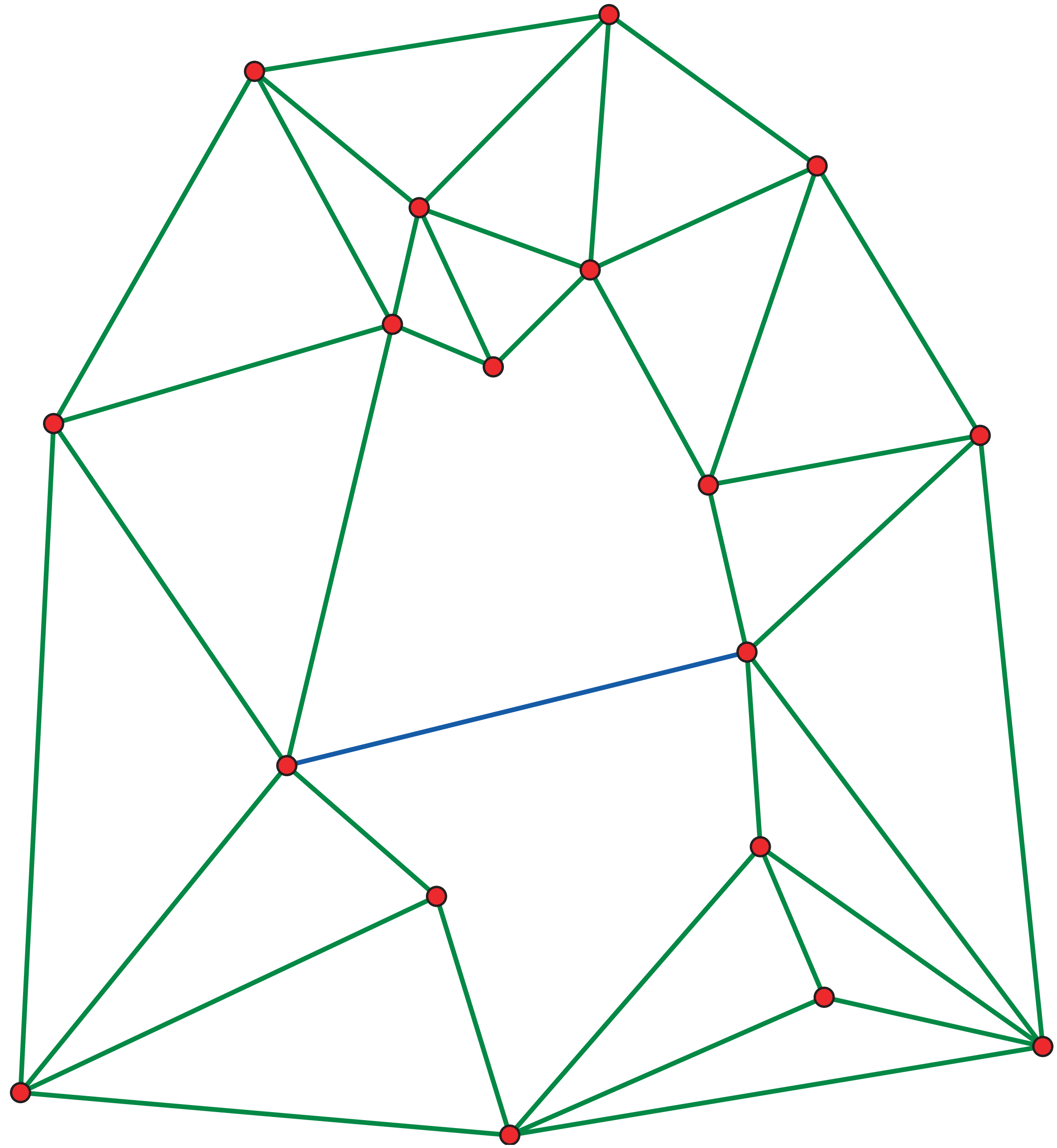
Choose a pair of vertices to be connected by a new edge

Remove all edges crossed by new edge

Retriangulate cleared-out region by greedily choosing ears (triangles having two sides on boundary of region) with quality better than worst removed triangle

If retriangulation succeeds, keep changed triangulation, else back out and try again

Continue improving triangulation until no more improvement possible



Edge insertion procedure

Start with any triangulation

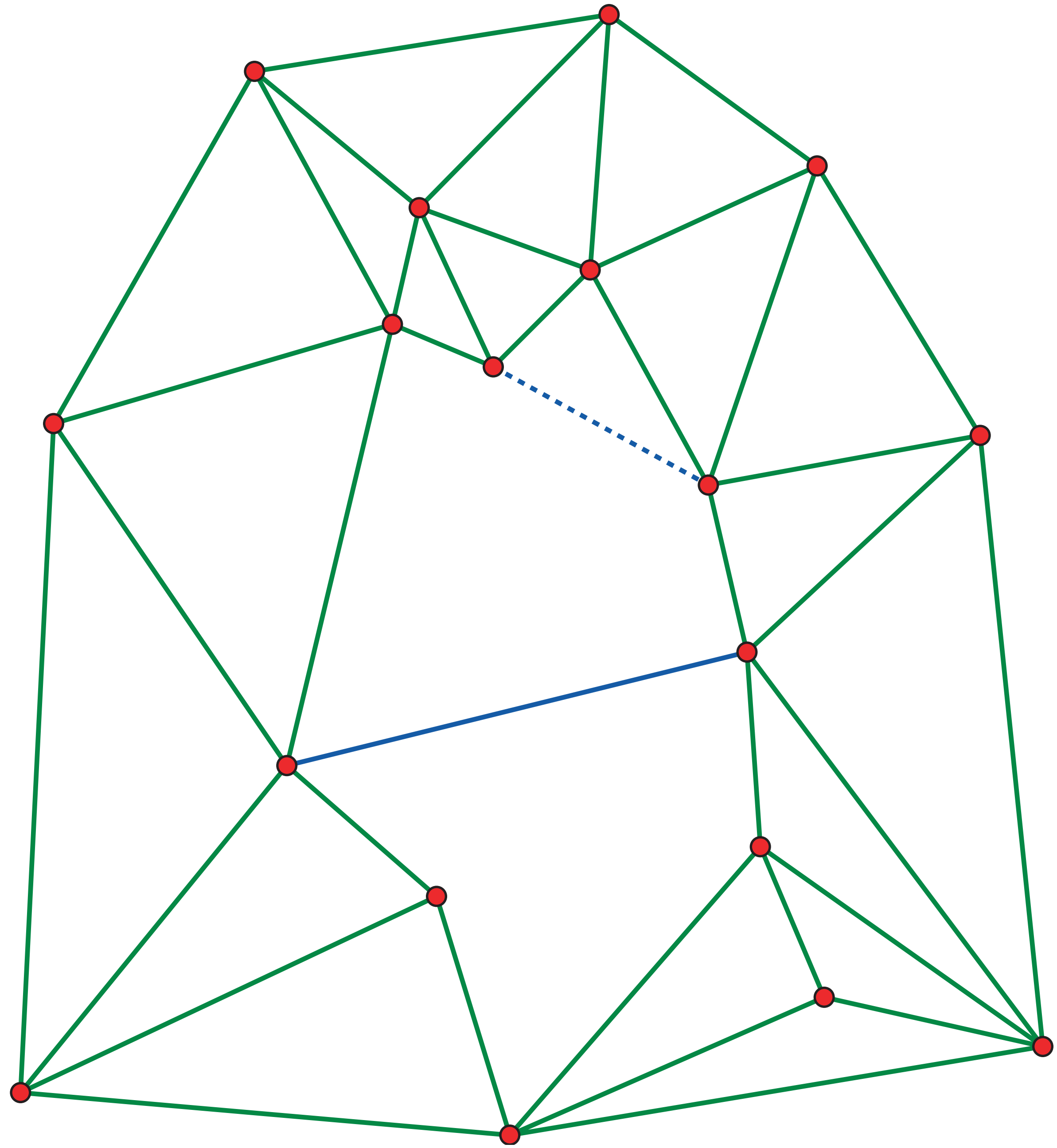
Choose a pair of vertices to be connected by a new edge

Remove all edges crossed by new edge

Retriangulate cleared-out region by greedily choosing ears
(triangles having two sides on boundary of region)
with quality better than worst removed triangle

If retriangulation succeeds, keep changed triangulation, else back out and try again

Continue improving triangulation until no more improvement possible



Edge insertion procedure

Start with any triangulation

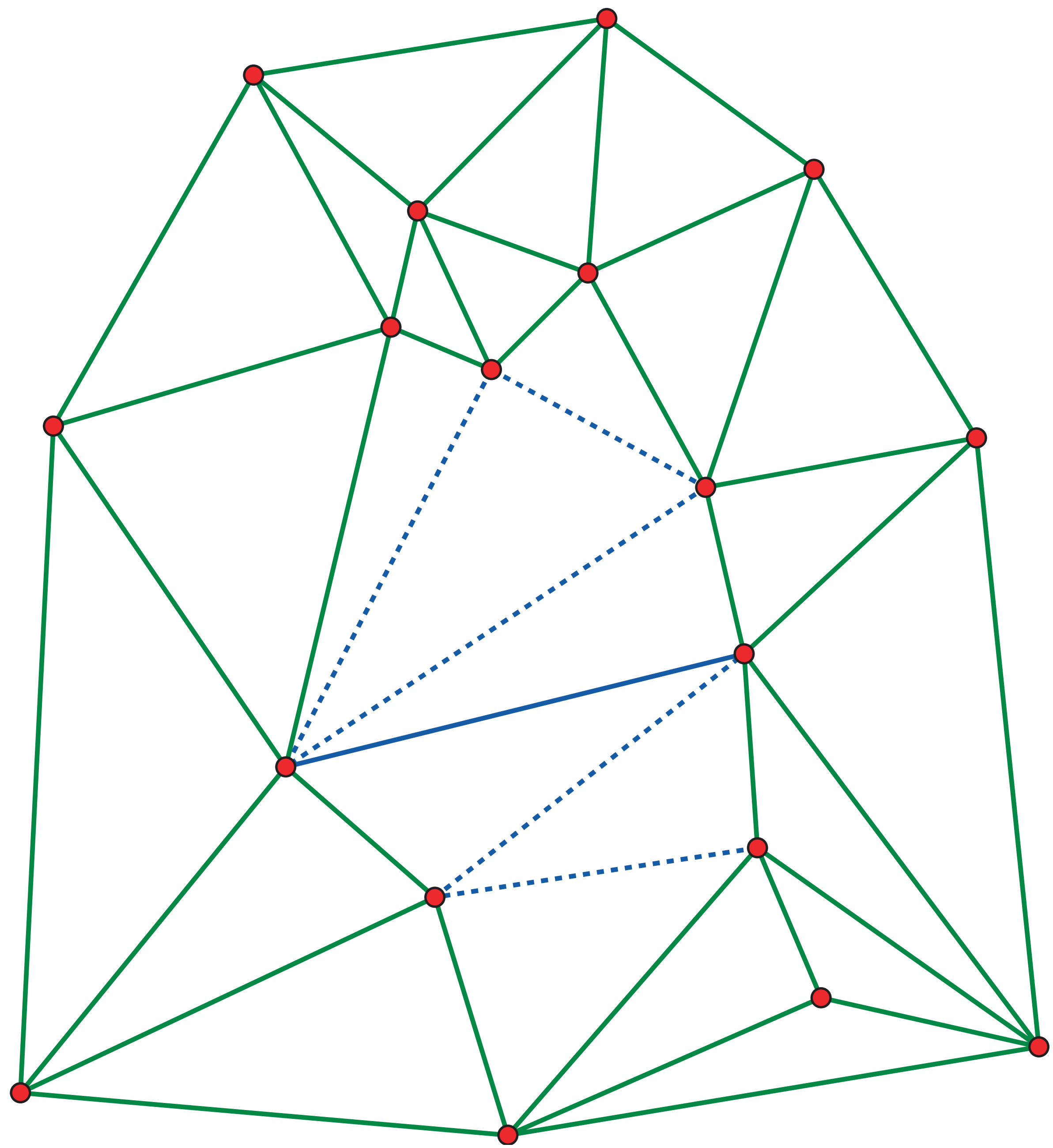
Choose a pair of vertices to be connected by a new edge

Remove all edges crossed by new edge

Retriangulate cleared-out region
by greedily choosing ears
(triangles having two sides on boundary of region)
with quality better than worst removed triangle

If retriangulation succeeds,
keep changed triangulation,
else back out and try again

Continue improving triangulation until no more improvement possible



Anchors

An anchor of a triangle Δ is a vertex v, such that:

If any triangulation T has better quality than Δ
then T has at least one edge vx cutting across Δ

Intuitively: which triangle vertex to blame for bad quality

Example:

For quality measure = minimize maximum angle in triangulation
then vertex of Δ with largest angle is its anchor

Weak Anchor Property:

In any triangulation, worst triangle has an anchor

So e.g. when performing edge insertion,
only need to consider candidate insertion edges incident to the anchor

Edge insertion always finds global optimum in $O(n^3)$ time

Strong Anchor Property:

In any triangulation, every triangle has an anchor

Allows binary search techniques to eliminate candidate insertion edges quickly
Edge insertion always finds global optimum in $O(n^2 \log n)$ time

Problems solved by edge insertion

Strong anchor property, near-quadratic time

Triangulation minimizing maximum angle
(or lexicographic vector of angles)

Triangulation maximizing minimum height
(distance from vertex to opposite side)

Triangulation minimizing maximum eccentricity
(distance from circumcenter to triangle, measure of obtuseness)

Weak anchor property, cubic time

Triangulated surface in 3d minimizing maximum slope

What about higher dimensions?

Delaunay definition, lifting transformation remain valid in any dimension

But flipping can get stuck
So, not very many optimality properties

Even simple-sounding optimal triangulation questions can be NP-hard
e.g. find triangulation of convex polyhedron using minimum number of tetrahedra
[Below, De Loera, Richter-Gebert, SODA 2000]

Delaunay triangulation can have many tetrahedra in 3d

E.g., points on moment curve (t, t^2, t^3)
or even uniformly spaced points on a cylinder [Erickson, SCG 2001]
can have quadratic-complexity DT

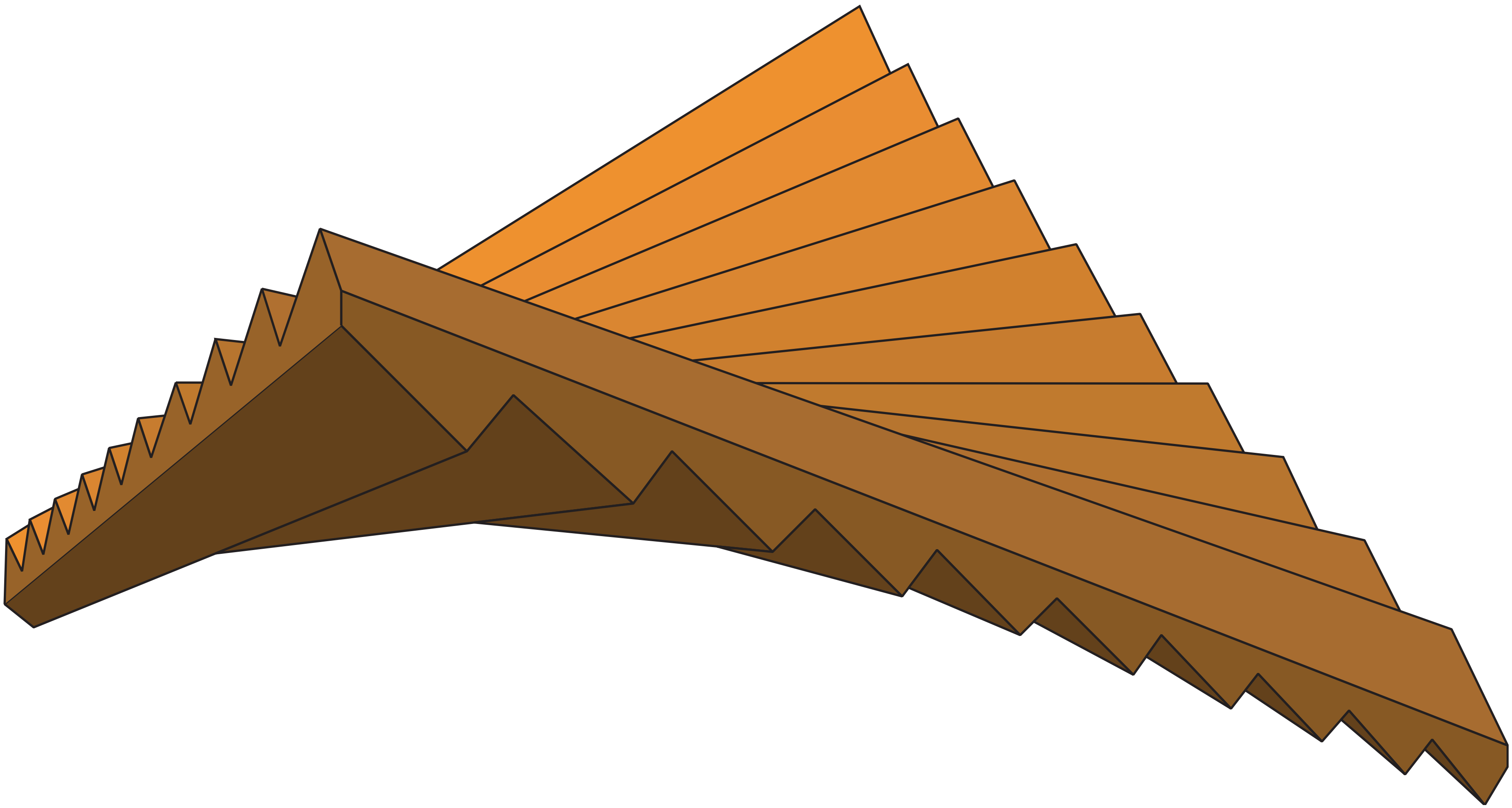
So, worst case for Delaunay construction time is not so good, $O(n^2)$

Current focus of research: understanding why typical inputs have sparse DT
[Erickson, SCG 2001 and SODA 2002; Attali and Boissonat, unpublished,
http://www-sop.inria.fr/prisme/personnel/boissonnat/Articles/complexiteDT.ps.gz]

Some polyhedra do not have tetrahedralizations

(without additional vertices)

so constrained Delaunay triangulation does not always exist



...but see [Shewchuk, SCG 1998 and SCG 2000] for sufficient conditions

Open problems in optimal triangulation

Improve time bound for edge insertion

Linear or near-linear time needed to make it practical
E.g., perhaps Delaunay is “close” to optimal so few insertions needed?

Minimum-weight triangulation

I.e., find triangulation minimizing sum of edge lengths
One of very few problems neither known to be poly time nor NP-hard

Some connection with other problems (e.g. ray tracing data structures)
but primarily of theoretical interest

Minimum area triangulation

For 2d triangulation of points with elevations in 3d
E.g. reconstruction of landforms from scattered data

Outline:

Introduction
Mesh quality issues, meshing steps

Connectivity optimization
Delaunay triangulation, edge insertion

Global point placement
Quadtrees, incremental Delaunay refinement

Individual point placement
Quasiconvex programming, sliver exudation

Goal: find global best point placement. But...

Computational complexity seems too high

Unlike optimal connectivity, no problems known with polytime solutions

For some criteria, global optimum is not even known to exist

E.g. min total edge length, possibly adding more points always increases quality

How to trade off number of added points vs mesh quality?

What quality measure to use?

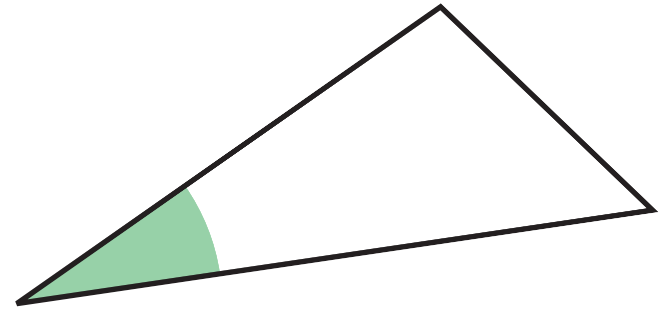
Instead, provably good quality

Absolute guarantee (not depending on input domain) on mesh quality simultaneously for many different quality measures

Number of points within a constant factor of *any* guaranteed-quality mesh

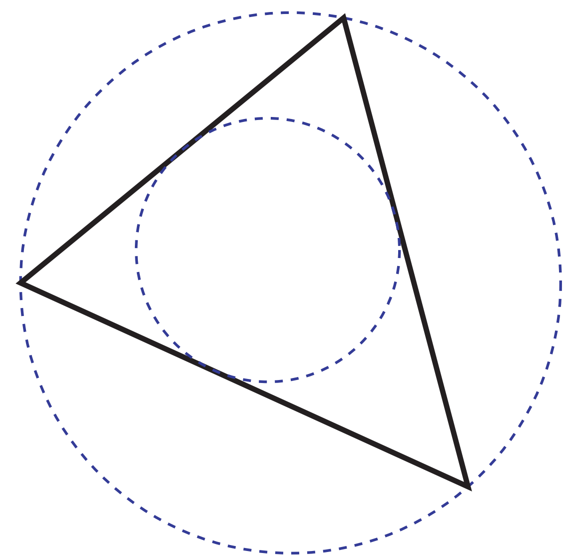
Key ideas: local feature size, well spaced points

Equivalent definitions of mesh quality



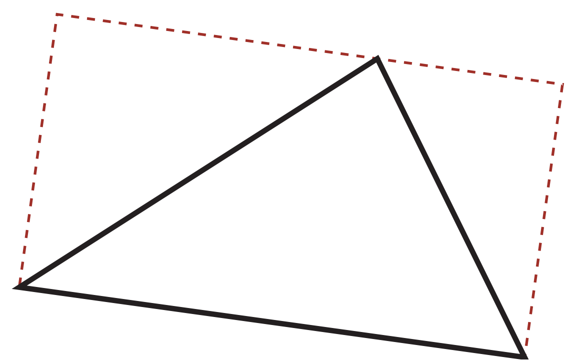
Minimum angle
bounded below by a constant

Ratio of perimeter^2 to area
bounded above by a constant



Ratio of circumcircle and incircle radii
bounded above by a constant

Semiperimeter is longer than longest side
by a factor of at least $1+\varepsilon$ for some constant ε



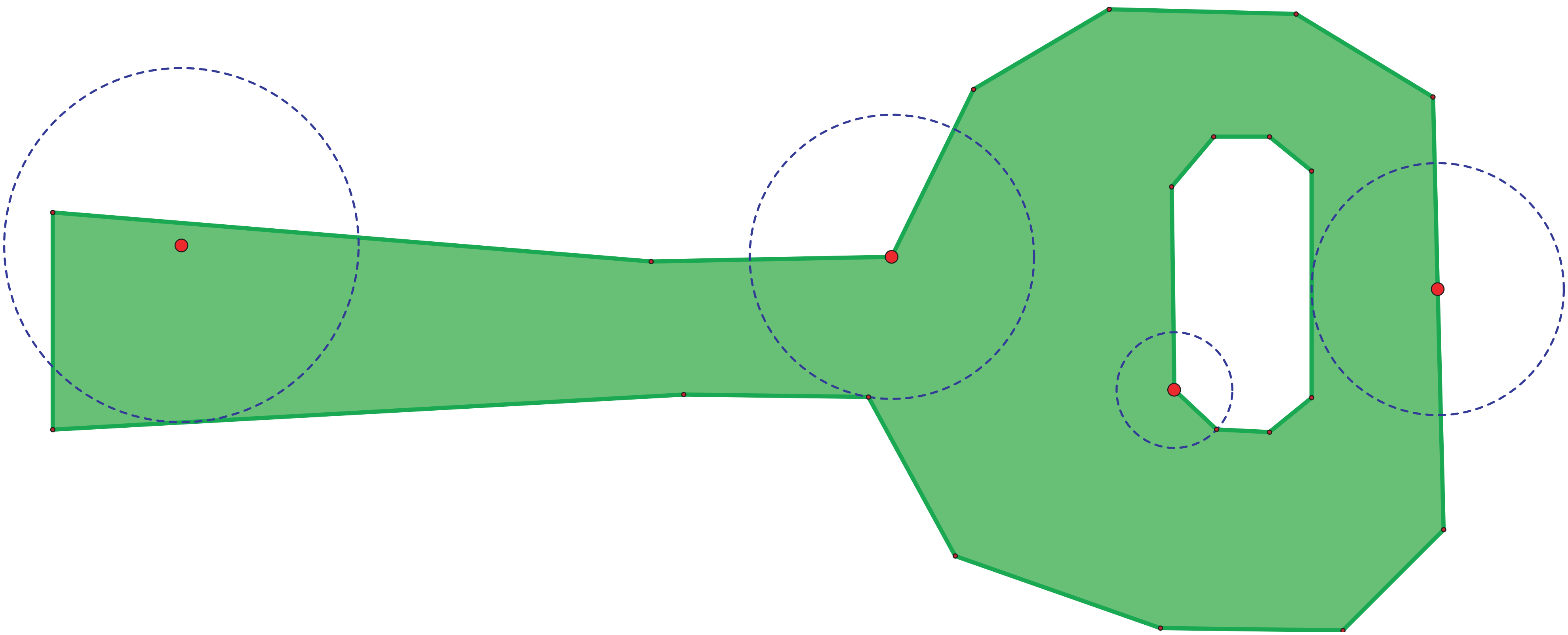
Ratio of diameter to height bounded by constant

Use "bounded aspect ratio" for all of these

Local feature size

of a polygonal domain [Ruppert, SODA 1993]

Other definitions possible e.g. in terms of medial axis [Amenta et al, GMIP 1998]



For any point *x* in domain, local feature size at *x*
is smallest distance from *x* to two nonadjacent edges
(measured along shortest paths inside domain)

Varies continuously over the domain and on its boundary
Nowhere zero

Well-spaced point sets

Placement of points is well-spaced if there exist constants ε, δ such that:

No two features are too close together

Any two placed points x, y are separated by a distance of at least $\varepsilon \max(\mathrm{lfs}(x), \mathrm{lfs}(y))$

Any point x placed in domain interior is separated from boundary by a distance of at least $\varepsilon\,\mathrm{lfs}(x)$

Whole domain is covered by placed points

Any point z in the domain is within distance $\delta\,\mathrm{lfs}(z)$ of some placed point

Any point z on the domain boundary is within distance $\delta\,\mathrm{lfs}(z)$ of some placed point on the domain boundary

Well spaced implies bounded aspect ratio

[Ruppert, SODA 1993; Mitchell, CCCG 1994]

Use constrained DT as the mesh, so each triangle has empty circumcircle

No two points are close together, so shortest edge of each triangle has length proportional to lfs

Circumradius of each triangle is also proportional to lfs

Proof by contradiction: suppose triangle has large circumcircle

Can't have large area inside domain else would violate well-covered property
so must be within lfs of domain boundary

Can't be closer than lfs to domain boundary due to well separation property

So, to be large, must have long intersection with domain boundary
but would violate well-covered boundary property

Bounded circumradius/shortest edge implies bounded aspect ratio

Well-spaced implies correct number of points

[Ruppert, SODA 1993; Mitchell, CCCG 1994]

Consider N = integral over domain of $1/\mathrm{lfs}(x)^2$

Any bounded aspect ratio triangulation has number of triangles at least proportional to N

Within any triangle, lfs ≤ distance to triangle boundary
So integral within triangle ≤ integral of $1/\mathrm{distance}^2$
But this is O(1) for bounded aspect ratio triangles
(yet another equivalent definition of bounded aspect ratio)

Any well spaced point set has number of points at most proportional to N

Draw disk of radius $\delta/2$ lfs(*x*) around each point *x*
Disks are disjoint, integral within each disk is at least a constant
so total number of disks must be small

Note N depends on geometry not just number of domain vertices

So bounded aspect ratio meshing is not polynomial in input size
But this argument shows non-polynomial dependence to be necessary

How to generate well-spaced points?

Method 1: Quadtree
[Bern et al., FOCS 1990 & JCSS 1992]

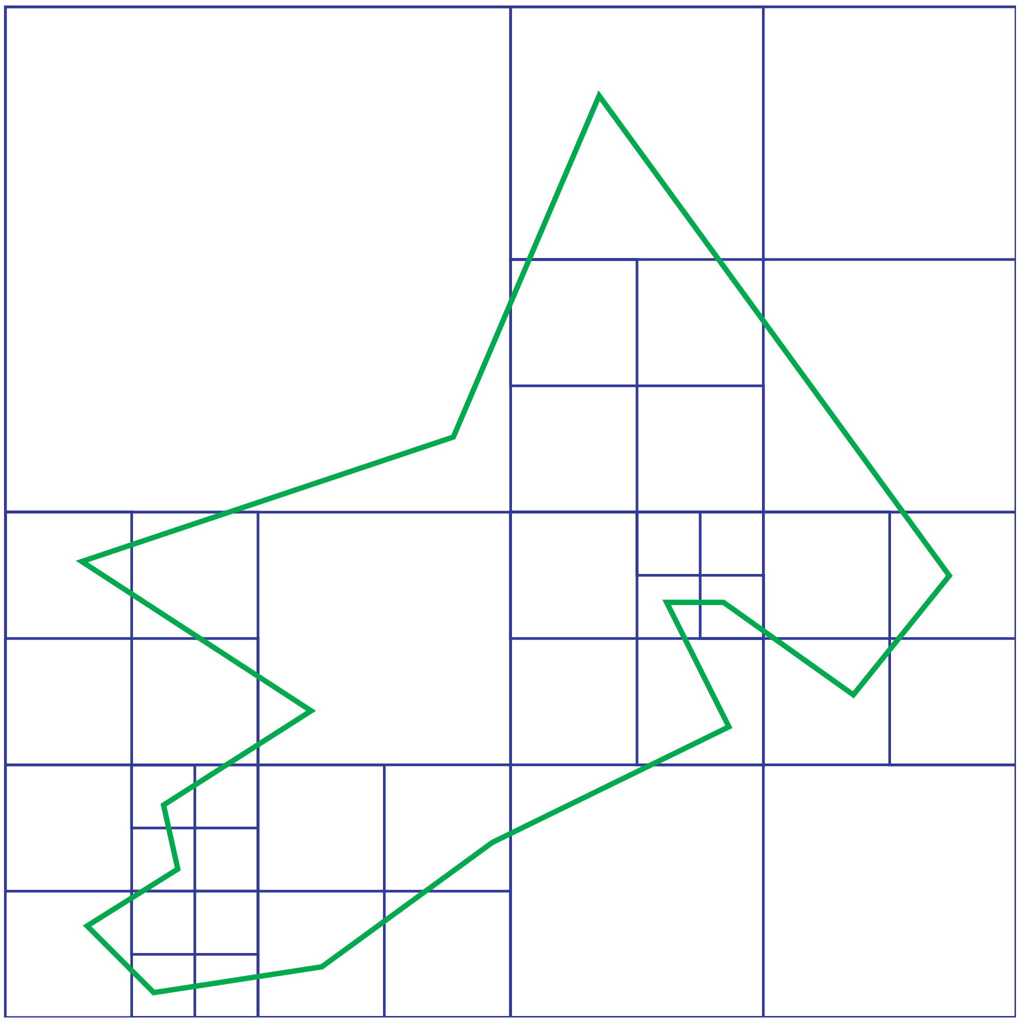
Start with square containing domain

Repeatedly subdivide squares into four smaller squares until each contains only a simple part of domain boundary

Use square corners as points

Complicated case analysis to warp points onto domain boundary, balance sizes of neighboring squares

Biased towards horizontal and vertical edges, large constant factors in analysis



How to generate well-spaced points?

Method 2: Delaunay refinement

[Chew, 1989; Ruppert, SODA 1993]

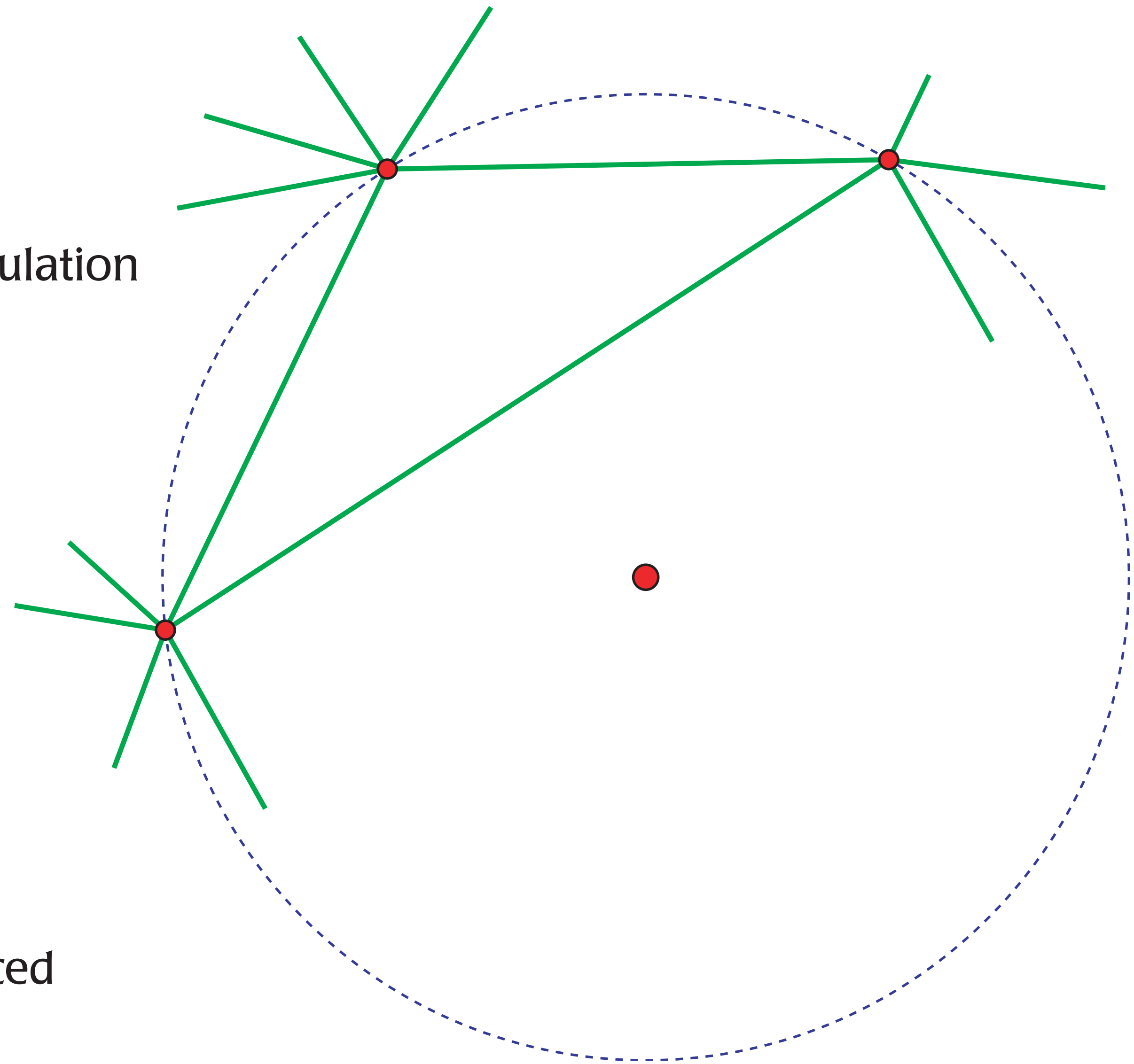
Start with constrained Delaunay triangulation

If there exists a bad triangle that
encroaches on a domain edge
split that edge at its midpoint

Else if there is a bad triangle in the
domain interior, add its circumcenter
to the point set

Update Delaunay triangulation

New point is far from previous ones
Eventually point set becomes well-spaced



How to generate well-spaced points?

Method 3: Sink insertion

[Edelsbrunner and Guoy, SCG 2001]

Start with constrained Delaunay triangulation

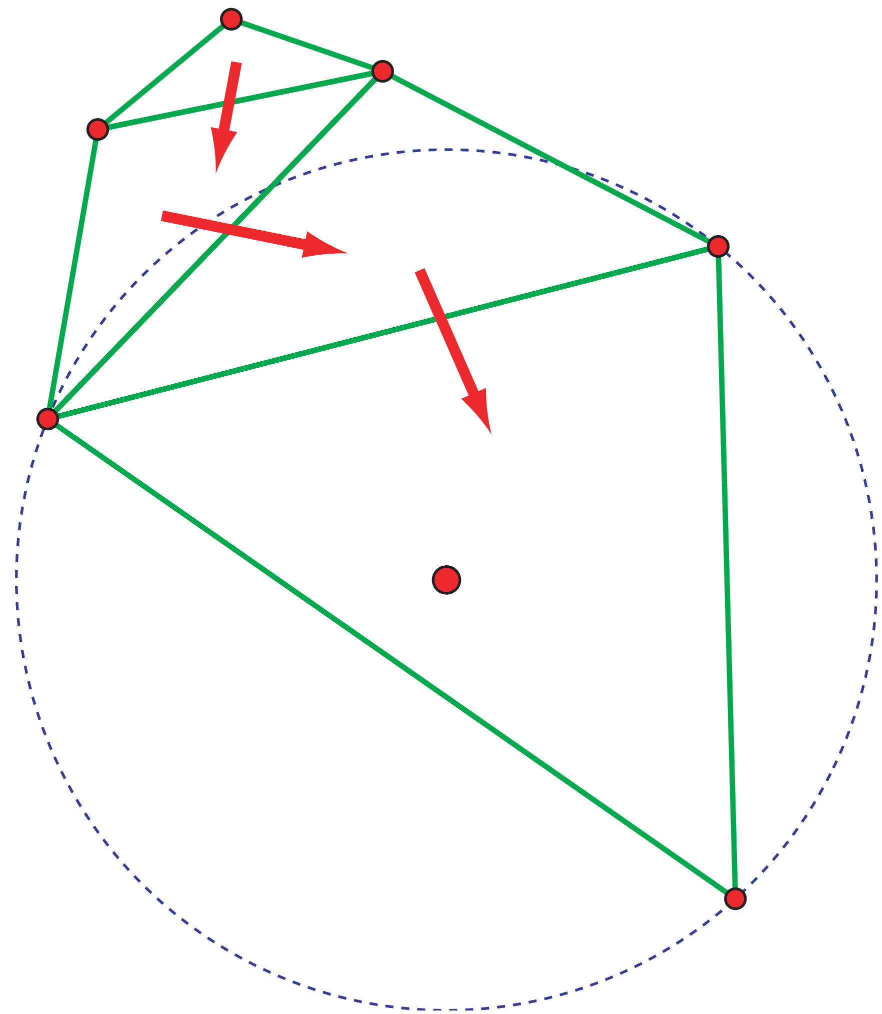
Follow path from bad triangle
to the triangle opposite its
obtuse angle, etc., until
either reaching an acute triangle
or the domain boundary

If acute triangle, add its circumcenter
If boundary edge, split it

Can insert many sink points simultaneously

Update Delaunay triangulation

Bad triangle will eventually be removed



How to generate well-spaced points?

Method 4: Disk packing

[Shimada, many papers]

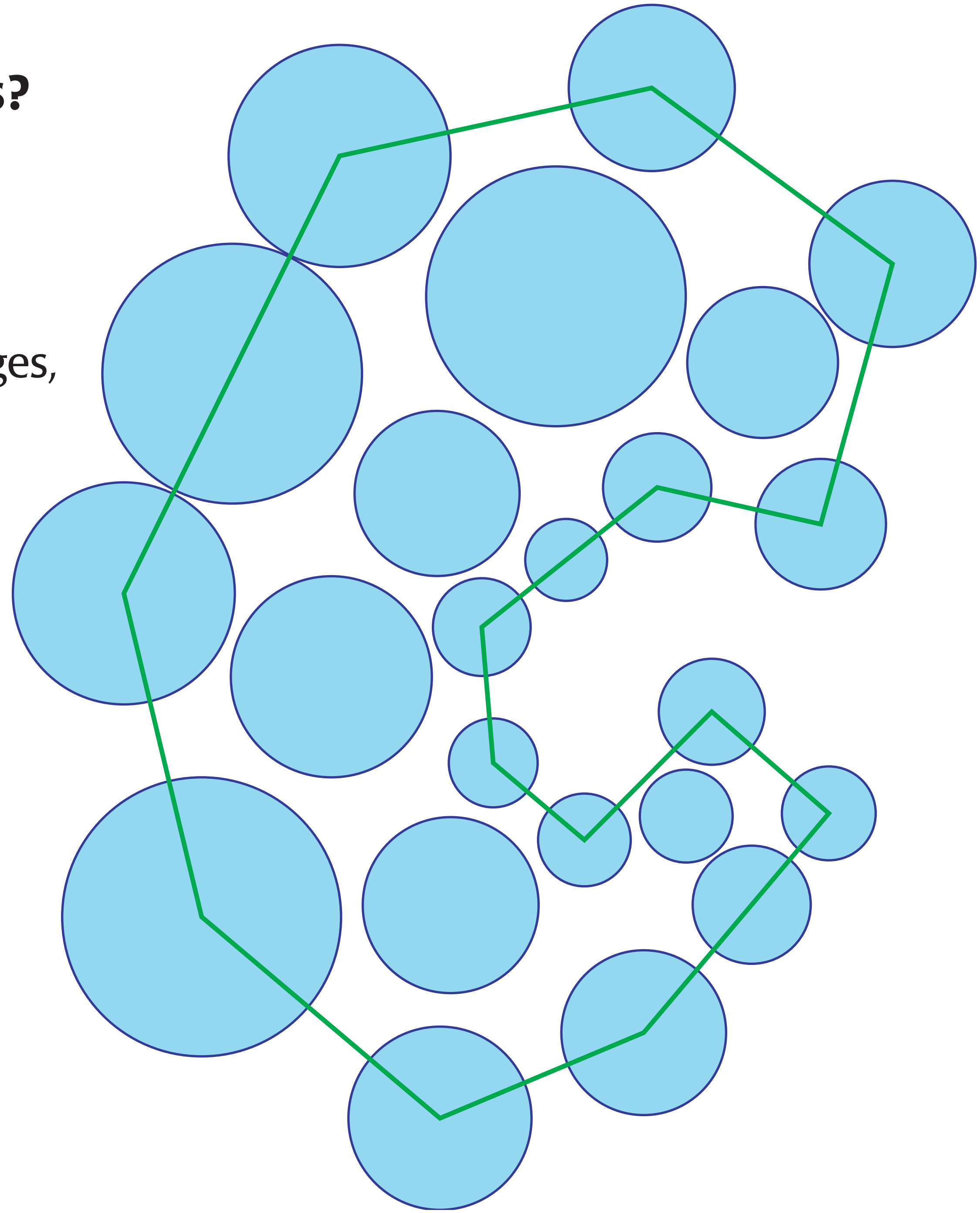
Add circles centered at vertices and on edges, radius proportional to local feature size, until no more room

Similarly add circles in interior

Optionally, move circles around to pack them tighter together

Use circle centers as point set, Delaunay triangulation or tangencies between packed circles for mesh

Not explicitly proven to yield well-spaced point placements, but seems straightforward



Can we reduce the number of points to polynomial (or even linear) by using weaker notions of element quality?

Nonobtuse triangulation

O(n) triangles for polygon input [Bern et al, SCG 1994]
More complicated when input may have internal boundaries

"Conforming Delaunay" triangulation

i.e., Delaunay triangulation of point set (not constrained DT)
such that all domain edges appear as unions of triangulation edges
$O(n^3)$ triangles [Edelsbrunner and Tan, SCG 1993]
Still a gap compared to the quadratic lower bound

Quadrilateral meshes

All quads kite shaped, no large angle, other possible guarantees
O(n) quadrilaterals [Bern and Eppstein, Roundtable 1997]

All use circle-packing approach + messy case analysis

Higher dimensions?

Well-spaced points can be defined in any dimension

Generating points is possible...

Quadtrees generalize to octrees etc., case analysis gets much worse
[Mitchell and Vavasis, SCG 1992]

Delaunay refinement extends to 3d [Shewchuk, SCG 1998]

Sink insertion applies in any dimension [Edelsbrunner and Guoy, SCG 2001]

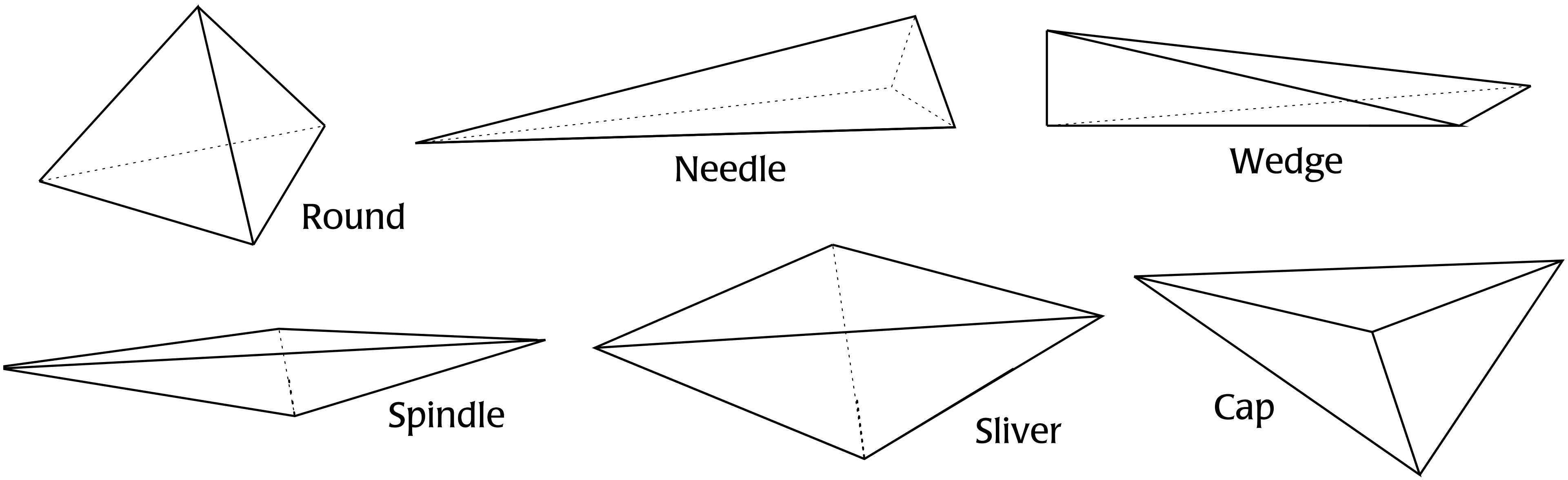
Sphere packing methods should also work

Bounds on numbers of points still work...

But Delaunay triangulation may not have well-shaped elements!

Tetrahedra classified by their bad angles

[Bern et al, SODA 1995; Baker, ICFEMFP 1989]



Round = good aspect ratio

Spindle = small solid angles, wide dihedrals

Needle = good dihedrals, small solid angle

Sliver = sharp and wide dihedrals

Wedge = small dihedrals and solid angles

Cap = wide solid angles

Similar classification possible based on other criteria e.g. circumradius
(sliver has circumradius similar to edge length, small volume)
[Cheng et al., SCG 1999 & STOC 2000]

Well-spaced point sets form only round and sliver tetrahedra

Outline:

Introduction

Mesh quality issues, meshing steps

Connectivity optimization

Delaunay triangulation, edge insertion

Global point placement

Quadtrees, incremental Delaunay refinement

Individual point placement

Quasiconvex programming, sliver exudation

Mesh improvement

Well-spaced points and Delaunay triangulation give provably good 2d meshes

But... may still be able to make further improvements
Won't augment guarantees, may help in practice

In 3d, methods described so far still leave slivers
Can we modify mesh to avoid all slivers?

Two types of mesh improvement

Connectivity updates e.g. flipping

Mostly similar to original mesh connectivity construction, not described here
But see later weighted Delaunay techniques for sliver exudation

Modified point locations

What we want: global best set of point positions for given mesh connectivity
But: too difficult to solve exactly

Early alternative: globally change points with very simple new position calculation
each point moves to average of neighbors′ positions
But: not well-motivated, can make quality worse or even violate mesh validity

Optimization based smoothing

Move points one by one to optimal location re unmoving neighbors
Each step improves quality, maintains mesh validity

Possibly make multiple passes, combine with connectivity updates

[Amenta et al., SODA 1997 and J. Algorithms 1999]
[Bank & Smith, SIAM J. Numer. Anal.]
[Canann et al., Roundtable 1998]
[Freitag et al., Roundtable 1995; McNU 1997; IJNME 1997; SIAM J. Sci. Comp. 1999]

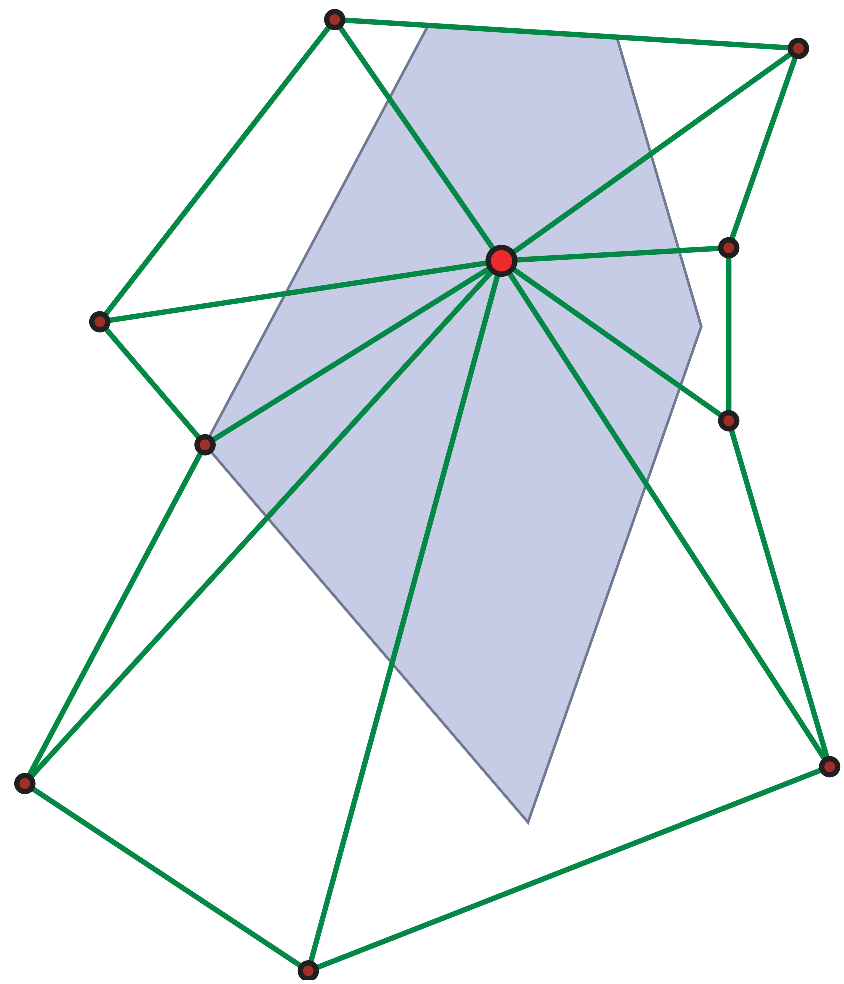
Constraints on smoothed point's location

Smoothed point must stay within kernel of polygon formed by triangles around it

(kernel = points that can see entire polygon boundary)

Within this region, new triangulation can have same connectivity as old

Outside this region, connectivity must be changed to preserve mesh validity

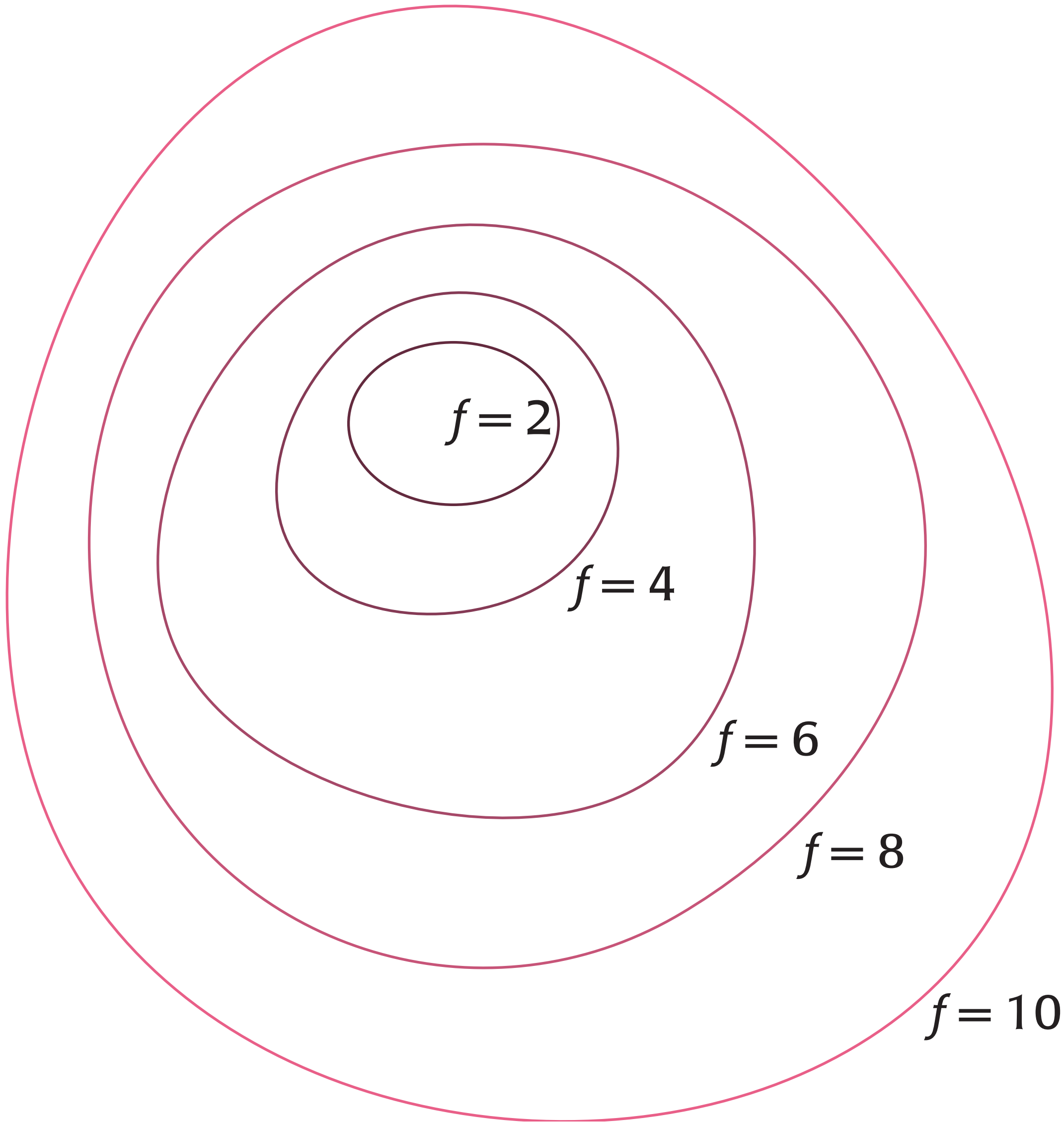


Quasiconvex function:

Level sets (points of equal function value) form nested convex sets

Example:
$f(x, y) = (x^2 + y^2)^{1/100}$
level sets are just circles

Maximum of quasiconvex functions remains quasiconvex
(just take intersection of level sets)



Quasiconvex program:

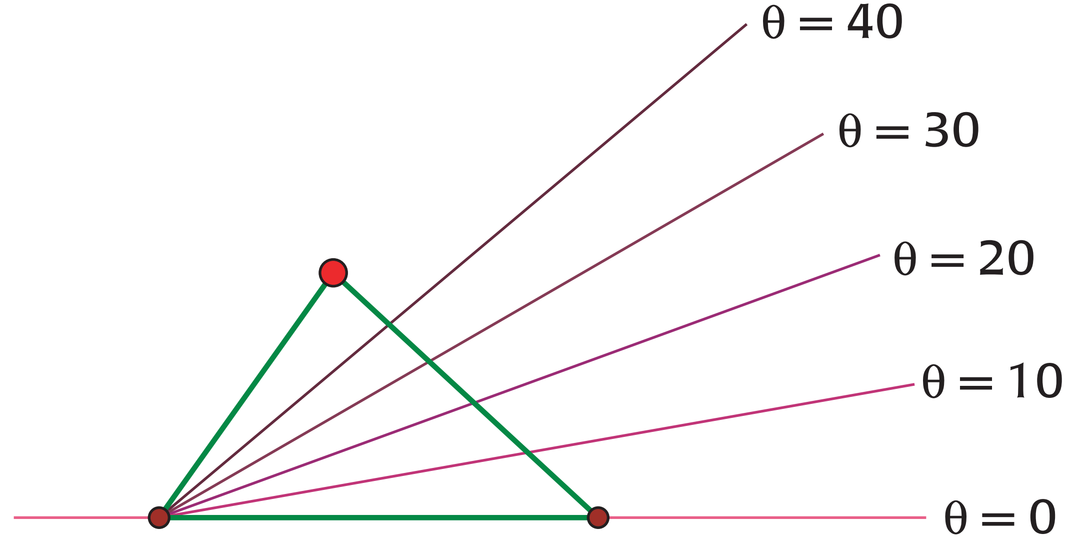
Find point minimizing the maximum of n given quasiconvex functions

Many mesh smoothing problems are quasiconvex programs!

Example:
Smoothing to maximize min angle

Function = one of the angles
of the triangles changed by the motion

So if moving point is adjacent to *k* triangles, there are 3*k* functions

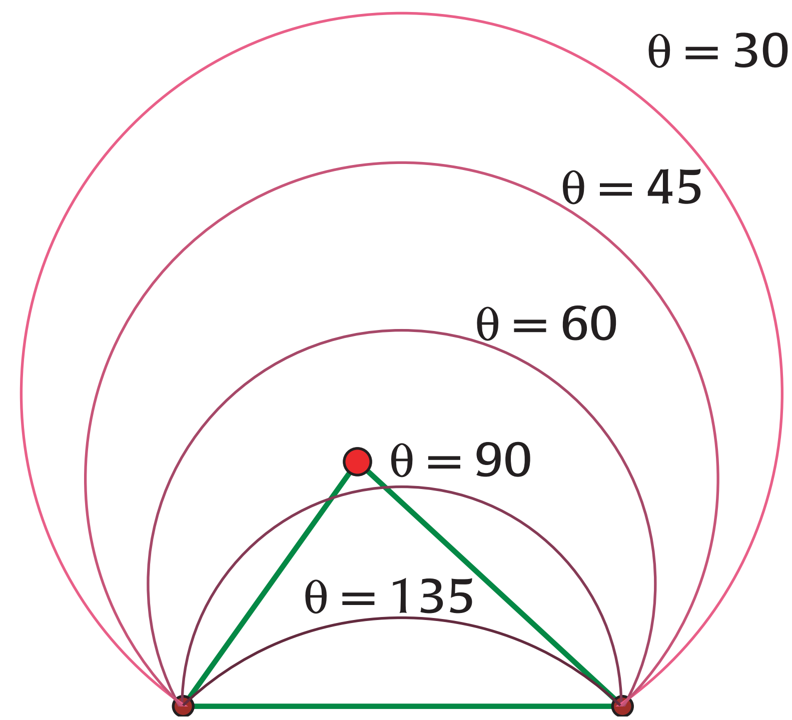


Angles at fixed points

Level sets are convex wedges
defined by line through base of triangle
and line at given angle to base

Angles at moving point

Level sets are circular disks
intersected with halfplane above base



Point maximizing min angle

= maximizer of min(3*k* quasiconcave funcs)
= solution to (negated) quasiconvex program

How to solve QC programs?

LP-type algorithms

Based on low-dimensional linear programming techniques
from computational geometry

In linear time, reduce problem to constant-sized subproblems

But in mesh smoothing applications, initial problem is already small,
exact solution of subproblems may be difficult

Numerical improvement methods

Hill-climbing, gradient descent, etc.

Quasiconvexity implies **no local optima to get stuck at**

Numerical approximation of optimum
is irrelevant for mesh smoothing

Quality criteria for quasiconvex mesh smoothing

Triangulations

maximize minimum triangle area, minimize maximum area
maximize minimum height, angle
minimize maximum edge length or element diameter
minimize maximum diameter/height ratio (aspect ratio)
minimize maximum perimeter, containing circle radius
maximize minimum inscribed circle radius
maximize minimum area/(sum squared edge lengths) [Bank & Smith]
mixtures of criteria

Quadrilateral meshes

area, angle, edge length, perimeter, containing circle, diameter – same as triangles
inscribed circle radius? seems to be quasiconvex, not proven

Tetrahedral meshes

maximize minimum volume, minimize maximum volume
minimize maximum edge length, triangle area, tetrahedron surface area
minimize maximum containing sphere radius
maximize minimum solid angle

Main difficulty for 3d meshes: slivers
Can smoothing remove them?

Sliver exudation

[Chew, SCG 1997]
[Cheng et al., SCG 1999 & JACM 2000]
[Edelsbrunner et al., STOC 2000]
[Li & Teng, SODA 2001]
[Cheng & Dey, SODA 2002]

Key ideas:

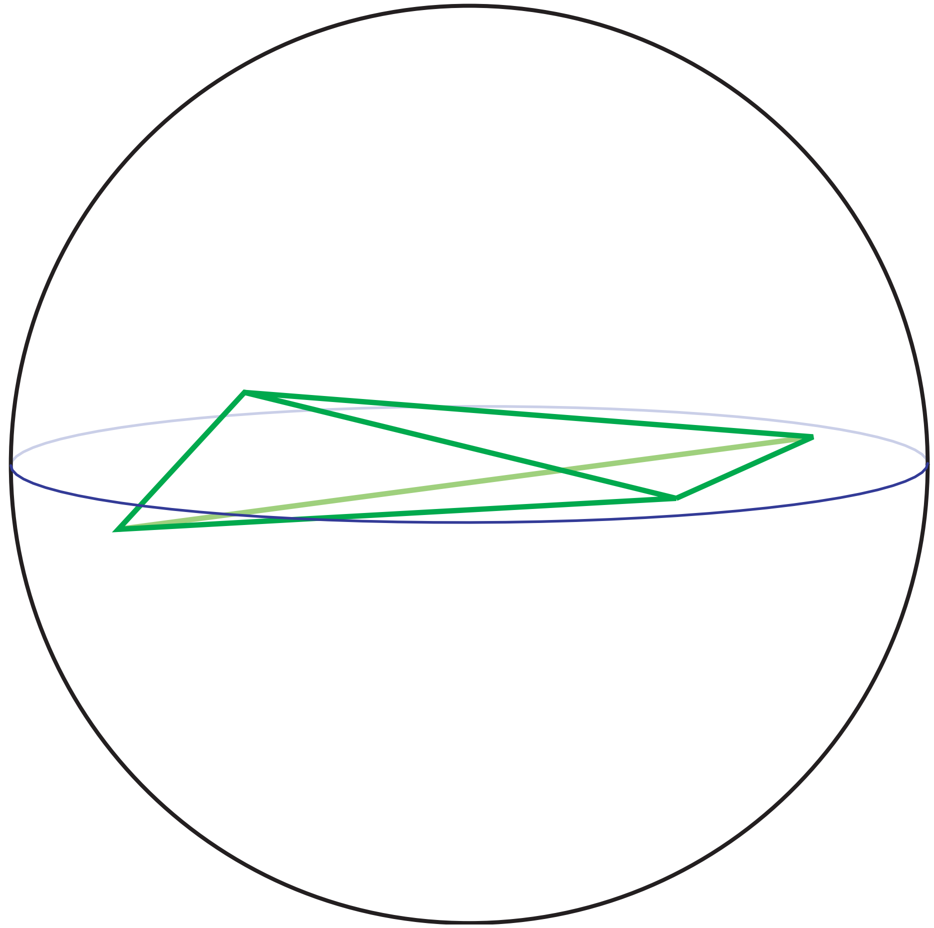
For well-spaced points, each vertex incident to few tetrahedra

If we perturb the vertex, the region in which some tetrahedron is a sliver is small

So, plenty of room to perturb it to avoid all slivers!

Non-quasiconvex

What is a sliver?



Four well-spaced vertices near the equator of their circumsphere

Circumradius / minimum edge length is bounded
Circumsphere volume / tetrahedron volume is unbounded

All faces have high surface area
Vertices are nearly coplanar

Sliver exudation by weighted Delaunay triangulation

[Cheng et al., SCG 1999 & JACM 2000; Cheng & Dey, SODA 2002]

Use lifting transformation to view Delaunay triangulation as 4d convex hull

Perturb points by raising or lowering above or below paraboloid

$(x, y, z) \rightarrow (x, y, z, x^2 + y^2 + z^2 + \text{perturbation})$

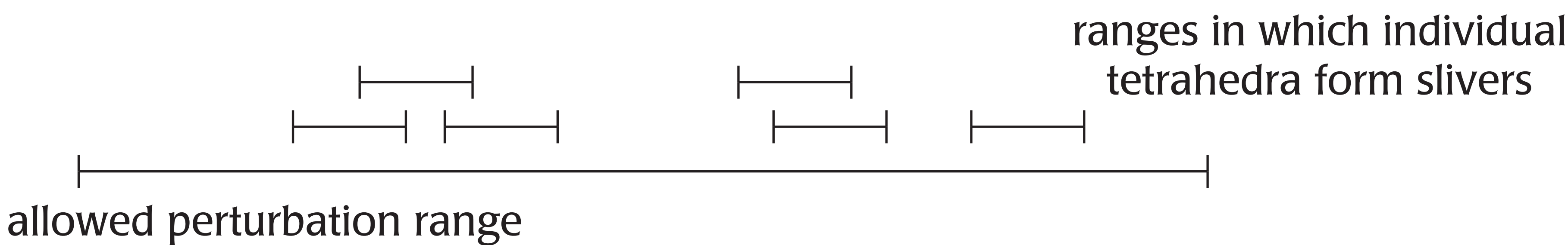
Compute convex hull of perturbed points

Project back down to 3d to form weighted DT

(can also be interpreted as dual of power diagram of spheres in 3d)

Allow perturbation size to be proportional to local feature size

With appropriate tuning of constants, each potential sliver obstructs small interval



Advantages: all points stay in place, no new points

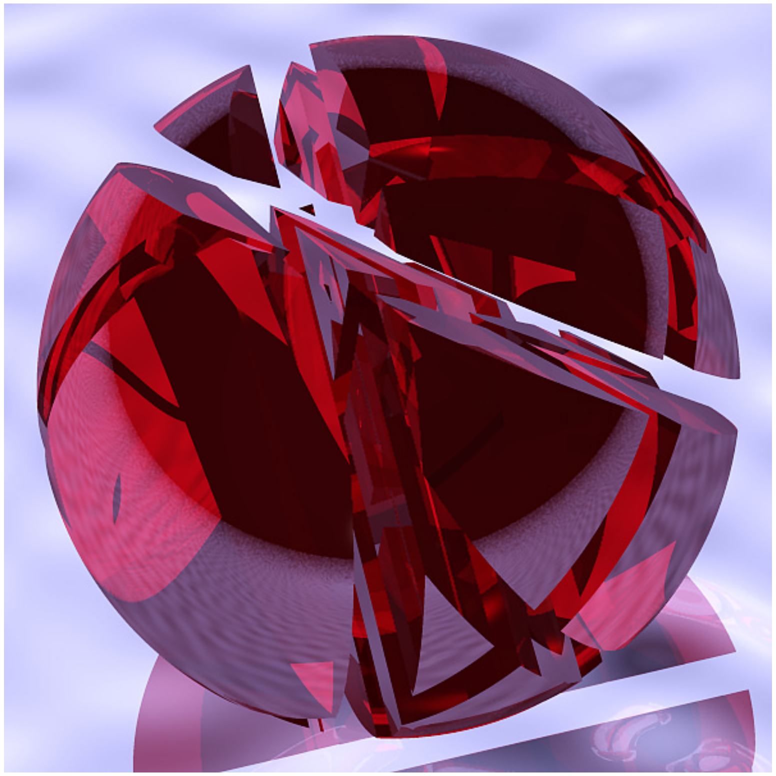
Disadvantages: not true Delaunay triangulation, unable to handle boundaries

Sliver exudation by Delaunay perturbation

[Edelsbrunner et al., STOC 2000; Li & Teng, SODA 2001]

Perturb points within a small sphere proportional to local feature size
Compute Delaunay triangulation of perturbed points

Volume of potential sliver is small only in thin slab near plane of other three points
so total obstructed volume is small, some unobstructed point can be found



Advantages: true Delaunay triangulation, no new points
Disadvantages: points move, unable to handle boundaries

Sliver exudation by Delaunay refinement

Combine circumcenter-insertion or sink-insertion technique with weighted or unweighted Delaunay perturbation
[Li & Teng, SODA 2001; Cheng & Dey, SODA 2002]

As each point is added, give it perturbed weight or perturb its position from circumcenter

Same sort of argument shows that sliver-avoiding perturbation always possible

Advantages: no separate placement method needed, handles boundaries
Disadvantages: adds additional points

Conclusions

Delaunay triangulation

Solves many optimal triangulation problems in 2d, fast and practical
Edge insertion optimizes some criteria not optimized by DT, still mainly theoretical

3d Delaunay not as satisfactory but no good alternatives, analysis of practicality ongoing

Well-spaced point placement

Solves most 2d guaranteed-quality meshing problems with optimal # triangles
Several practical placement methods available
Useful as an initial placement in 3d, but further mesh improvement necessary

Circle packing methods can solve some other placement problems with fewer triangles

Mesh optimization

Can make good meshes even better in 2d
Many criteria can be optimized via quasiconvex programming

Necessary to eliminate slivers in 3d
several perturbation or refinement based methods available
constant factors still too high, needs practical refinements

General References:

Franz Aurenhammer and Yinfeng Xu.
Optimal triangulations.
Encyclopedia of Optimization, Kluwer, 1999.
http://citeseer.nj.nec.com/aurenhammer99optimal.html

Marshall Bern and David Eppstein.
Mesh generation and optimal triangulation.
Computing in Euclidean Geometry, World Scientific, 1995.
http://www.ics.uci.edu/~eppstein/pubs/BerEpp-CEG-95.pdf

Marshall Bern and Paul Plassmann.
Mesh generation.
Handbook of Computational Geometry, Elsevier, 2000.
http://citeseer.nj.nec.com/bern00mesh.html