One-Dimensional Computational Topology

III. Shortest nontrivial cycles

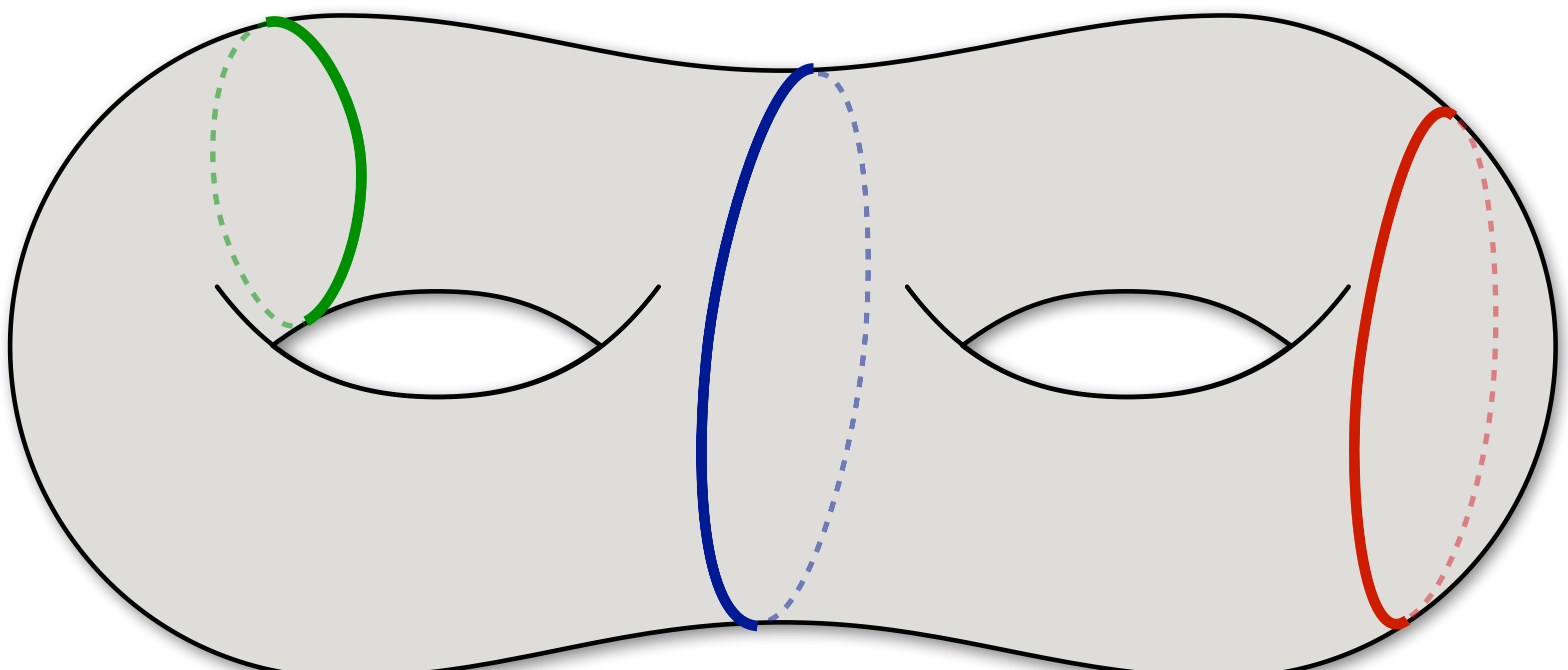
Jeff Erickson
University of Illinois, Urbana-Champaign

Today's Question

Given a surface Σ, find the shortest topologically nontrivial cycle in Σ.

Trivial cycles

- *contractible* = null-homotopic = boundary of a disk
- *separating* = null-homologous = boundary of a subsurface



Surface reconstruction

Surface reconstruction

scan object

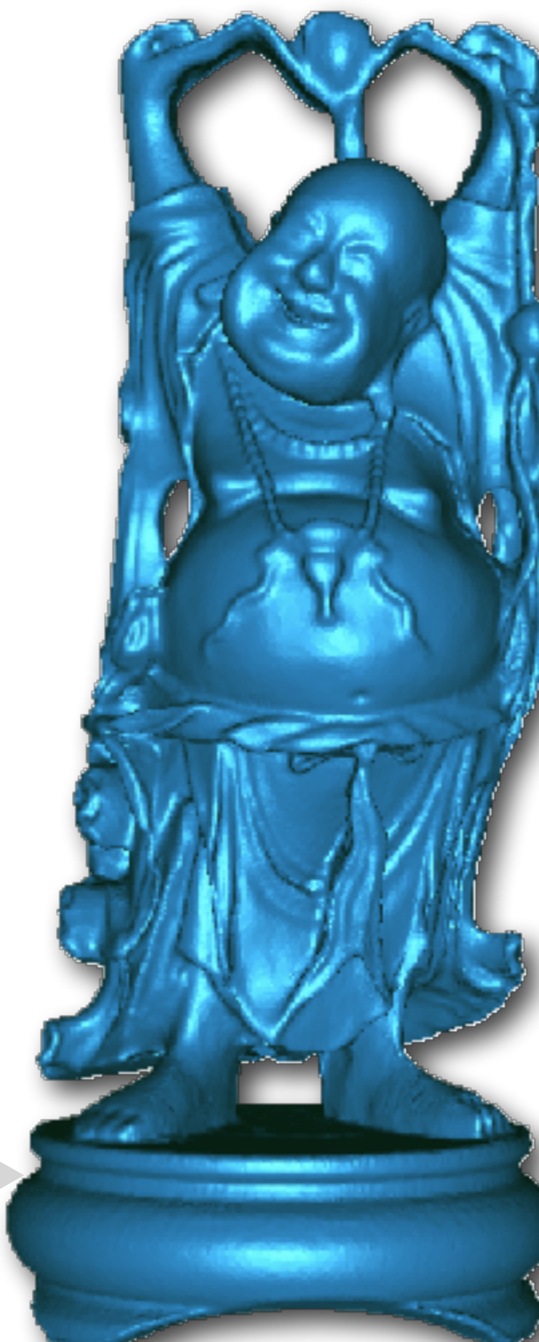
point cloud

Surface reconstruction

scan object

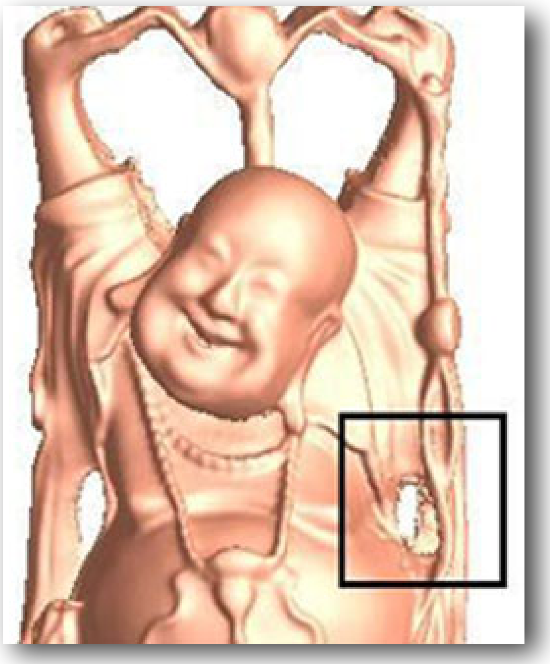
point cloud

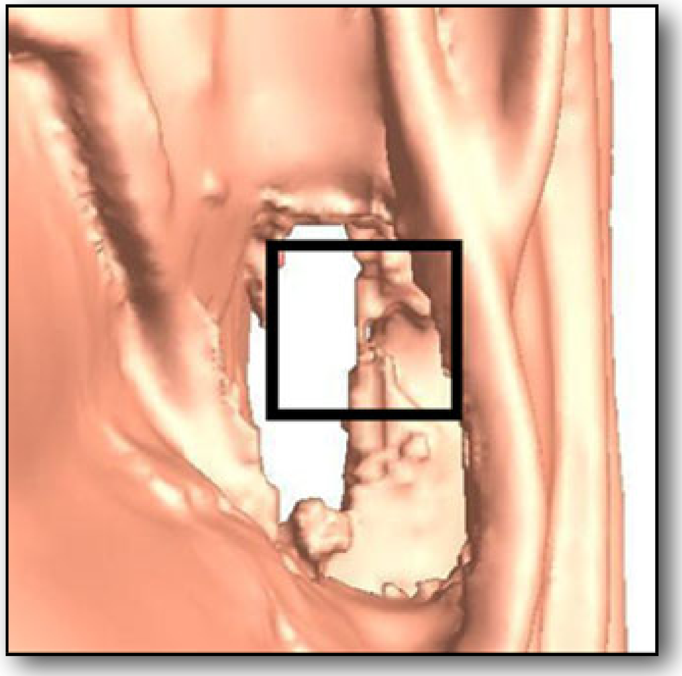
reconstruct surface

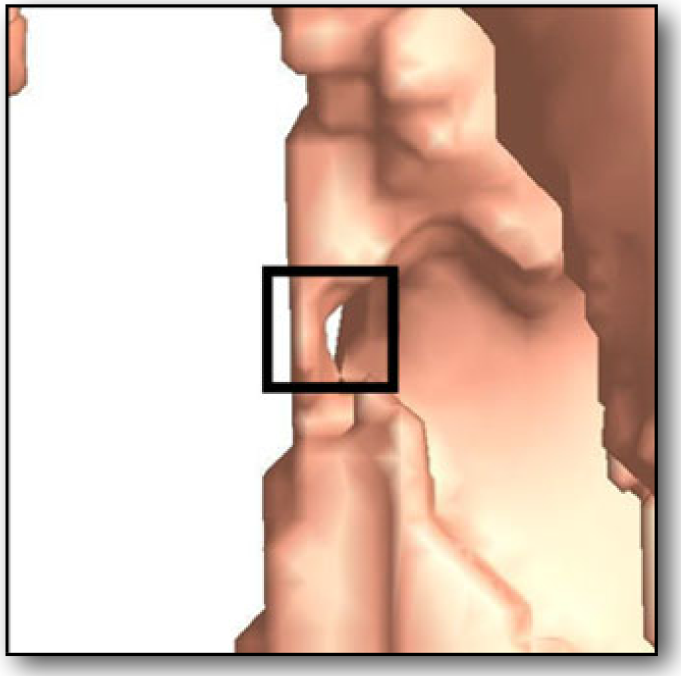


Topological noise

- Measurement errors from the scanning device add extra handles/tunnels to the reconstructed surface.





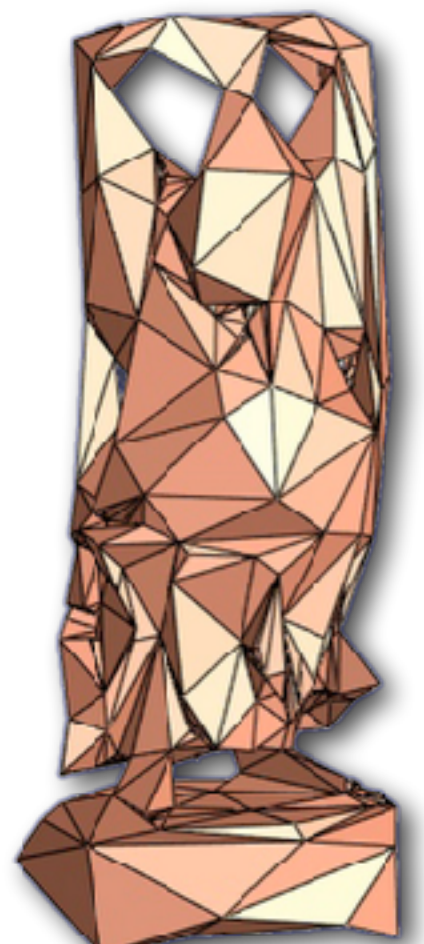


[Wood, Hoppe, Desbrun, Schröder '04]

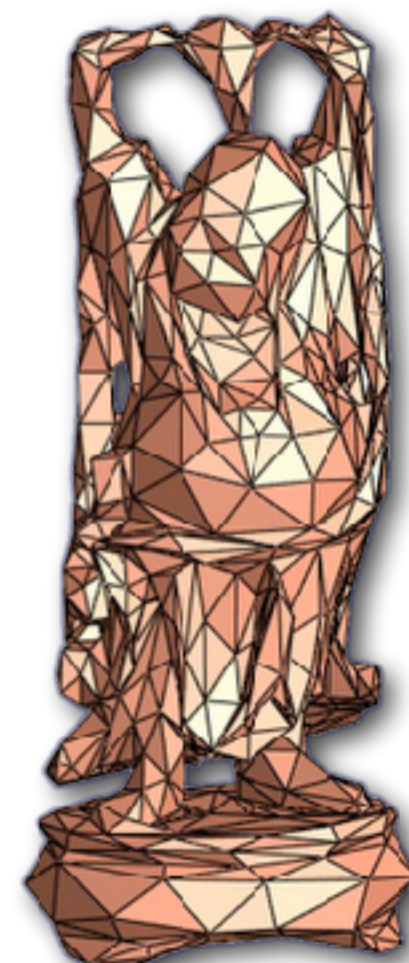
Topological noise

- These extra tunnels make compression difficult.

genus 104



genus 104
50K vertices



genus 6
50K vertices

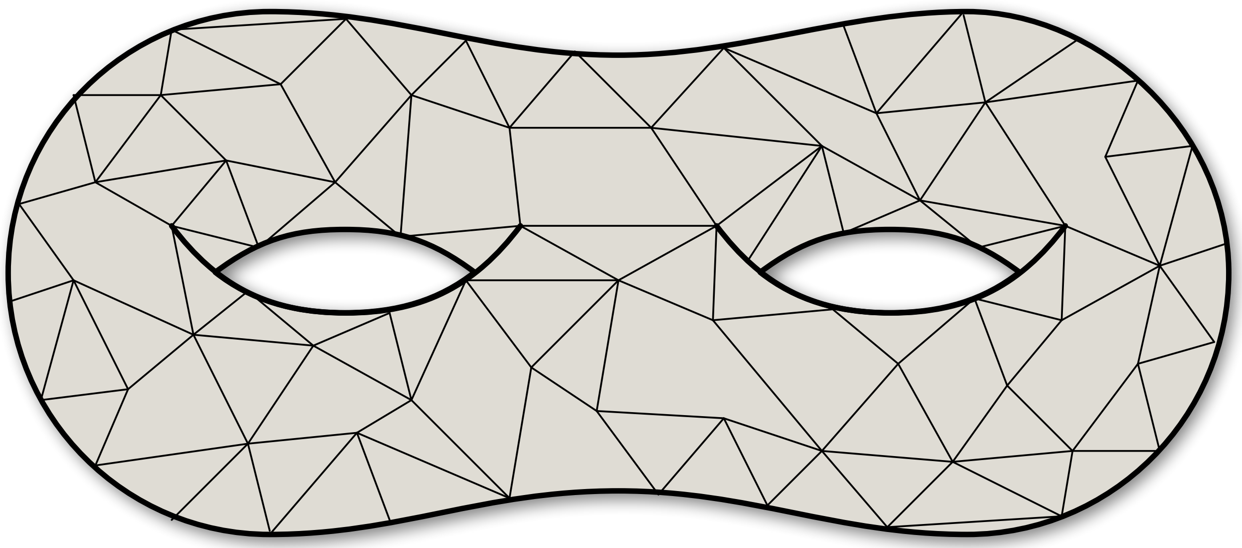
[Wood, Hoppe, Desbrun, Schröder ’04]

Connections

- Length of shortest noncontractible cycle
  - *systole [Loewner '49] [Pu '52] ... [Gromov 83] ...*
  - *representativity [Robertson, Seymour 87]*
  - *edge-width [Thomassen 90; Mohar, Thomassen 99]*
- First step of *many* other topological graph algorithms
- Related to broader problems in topological data analysis
  - Coverage analysis of ad-hoc/sensor networks
  - Identifying (un)important topological features in high-dimensional data sets

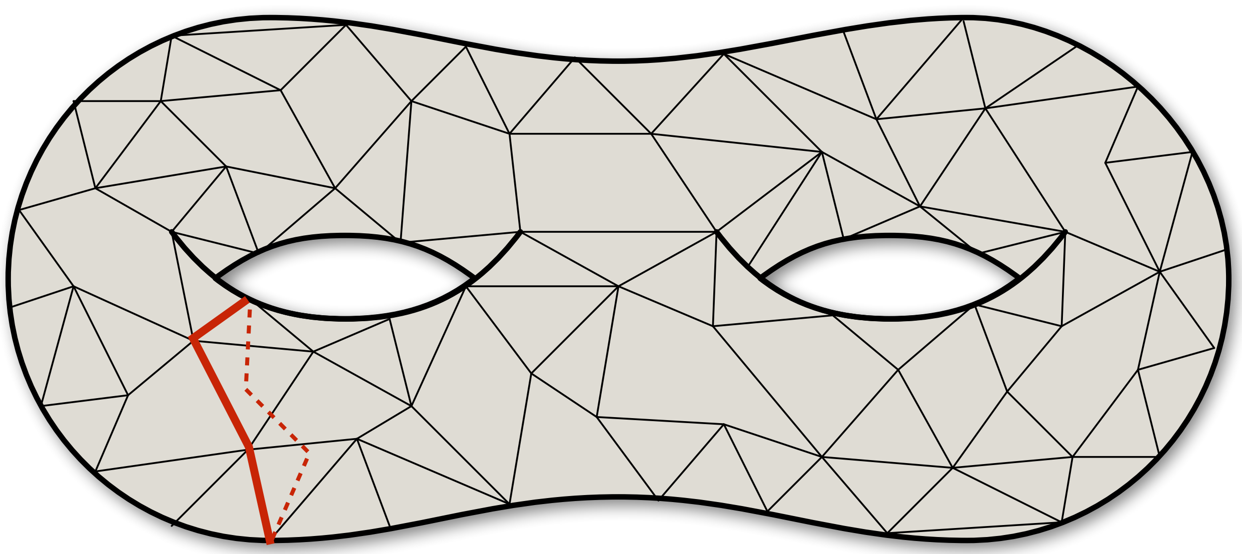
“Given”?

- Input:
  - Orientable surface map Σ with complexity n and genus g.
  - Length $\ell(e) \geq 0$ for every edge of Σ
    - No other assumptions. Not even the triangle inequality.



“Given”?

- Input:
  - Orientable surface map Σ with complexity n and genus g.
  - Length $\ell(e) \geq 0$ for every edge of Σ
    - No other assumptions. Not even the triangle inequality.



- Output:
  - Minimum-length cycle in the graph of Σ that is noncontractible or nonseparating in Σ.

Systolic inequalities

[Colin de Verdière, Hubard, de Mesmay 2013]

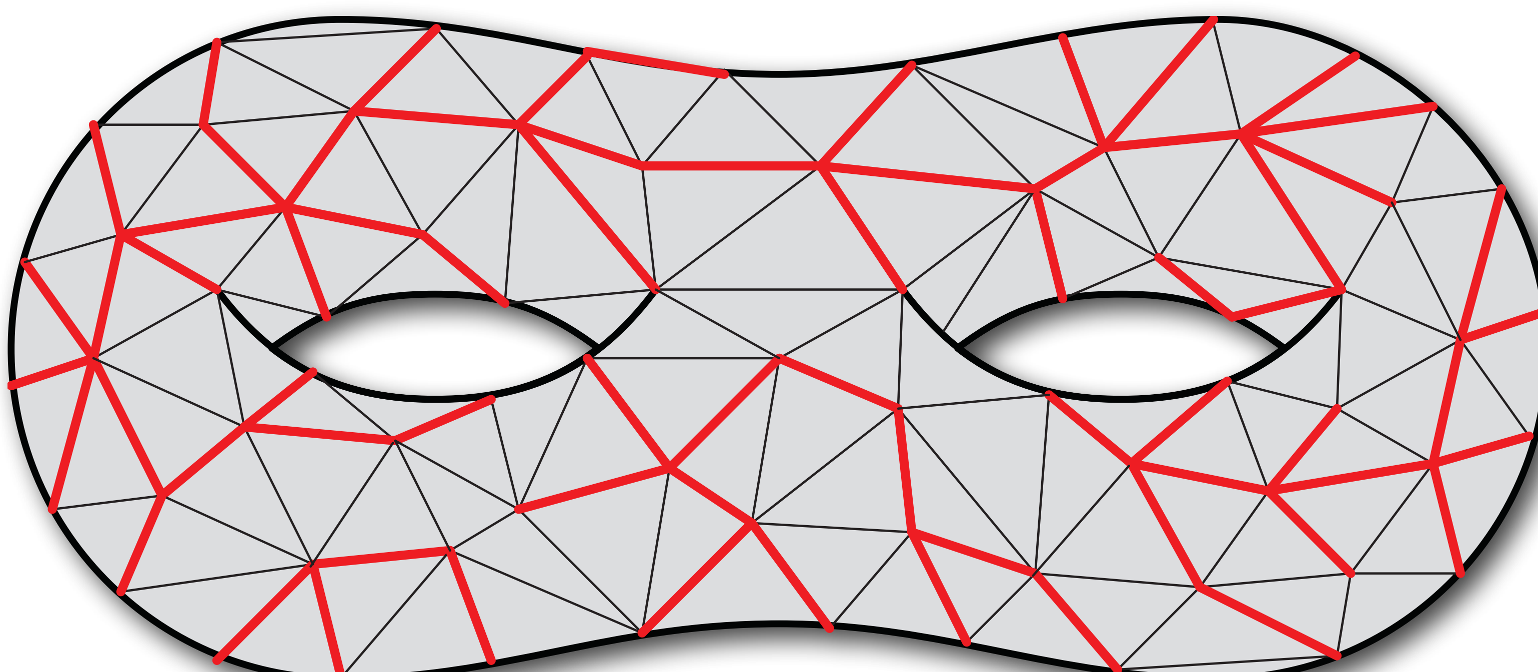
- Any Riemannian surface can be approximated (up to constant factors) by a combinatorial triangulation, and vice versa.
  - discrete→continuous: glue equilateral triangles, smooth vertices
  - continuous→discrete: intrinsic Voronoi diagram of ε-net
- Every Riemannian surface has systole ≤ $\frac{2}{3}\sqrt{A/g}\log g$ *[Gromov 1983, 1992]*
  - ⇒ Every triangulated surface map has edgewidth ≤ $2\sqrt{n/g}\log g$
    Improves [Hutchinson 1988]
- There are Riemannian surfaces with systole ≥ $\frac{1}{3}\sqrt{A/g}\log g$ *[Buser Sarnak 1994]*
  - ⇒ There are triangulated surface maps with edgewidth ≥ $\frac{1}{7}\sqrt{n/g}\log g$
    Conjectured by [Przytycka Przytycki 1993]

Tree-cotree structures

Tree-cotree decomposition

A *partition* of the edges into three disjoint subsets:

- A spanning tree ***T***
- A spanning cotree ***C*** — C^* is a spanning tree of G^*
- Leftover edges ***L*** $:= E \setminus (C \cup T)$ — Euler's formula implies $|L| = 2g$

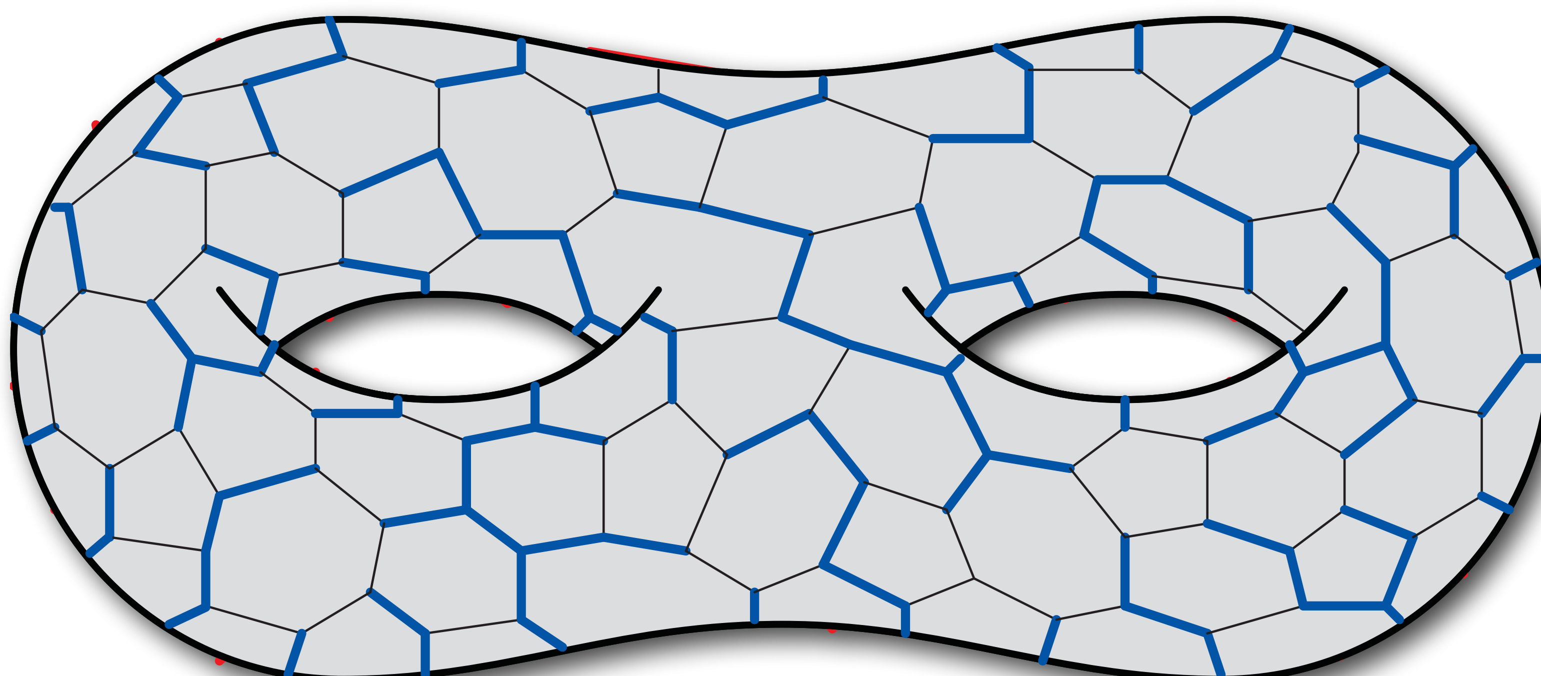


[von Staudt 1847; Dehn 1936; Biggs 1971; Eppstein 2003]

Tree-cotree decomposition

A *partition* of the edges into three disjoint subsets:

- A spanning tree ***T***
- A spanning cotree ***C*** — C^* is a spanning tree of G^*
- Leftover edges ***L*** $:= E \setminus (C \cup T)$ — Euler's formula implies $|L| = 2g$

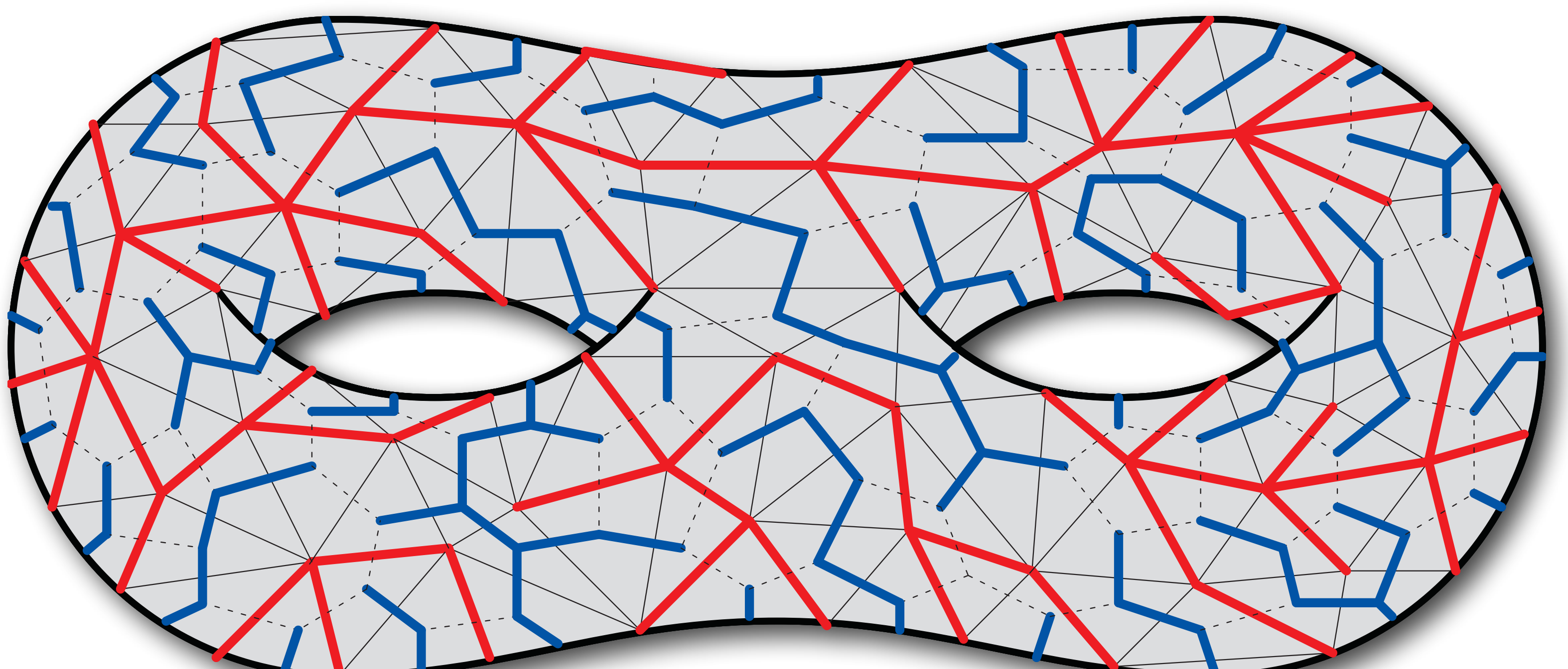


[von Staudt 1847; Dehn 1936; Biggs 1971; Eppstein 2003]

Tree-cotree decomposition

A *partition* of the edges into three disjoint subsets:

- A spanning tree ***T***
- A spanning cotree ***C*** — C^* is a spanning tree of G^*
- Leftover edges ***L*** $:= E \setminus (C \cup T)$ — Euler's formula implies $|L| = 2g$

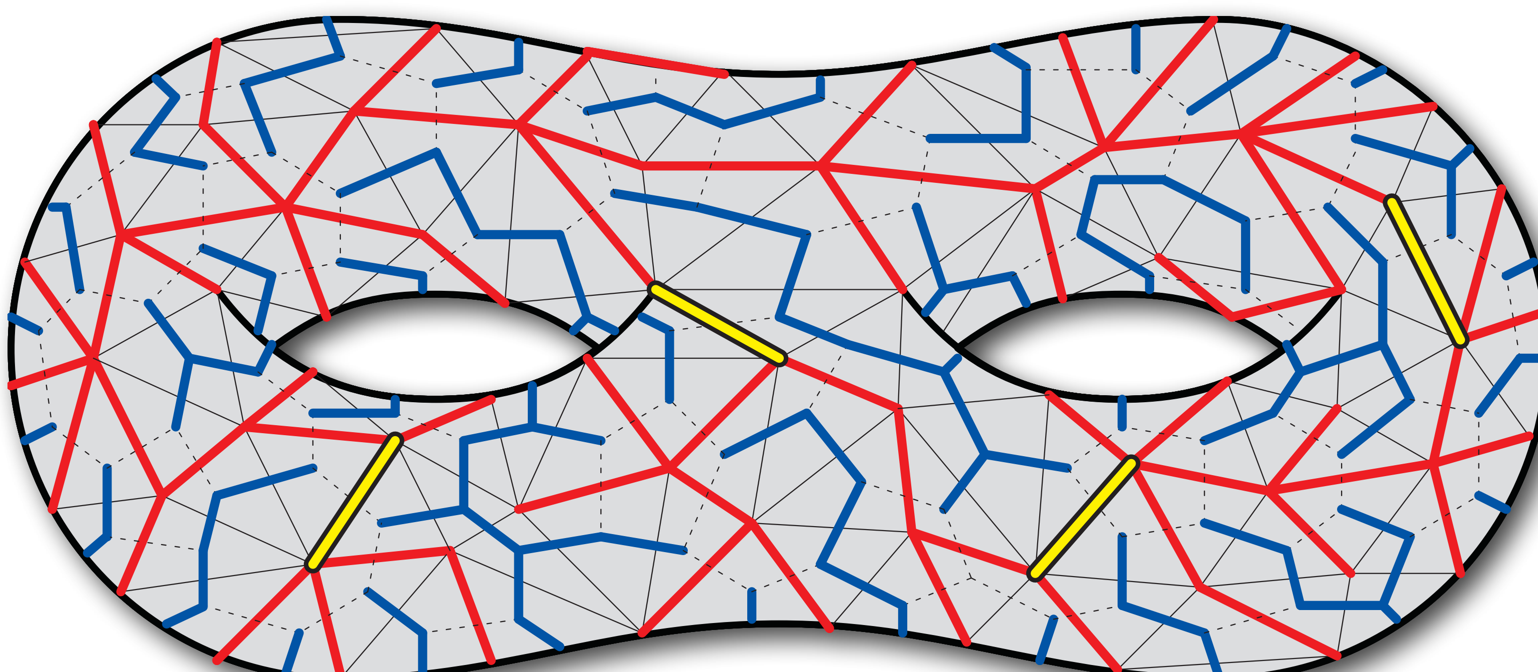


[von Staudt 1847; Dehn 1936; Biggs 1971; Eppstein 2003]

Tree-cotree decomposition

A *partition* of the edges into three disjoint subsets:

- A spanning tree ***T***
- A spanning cotree ***C*** — C^* is a spanning tree of G^*
- Leftover edges ***L*** := $E \setminus (C \cup T)$ — Euler's formula implies $|L| = 2g$



[von Staudt 1847; Dehn 1936; Biggs 1971; Eppstein 2003]

Fundamental loops and cycles

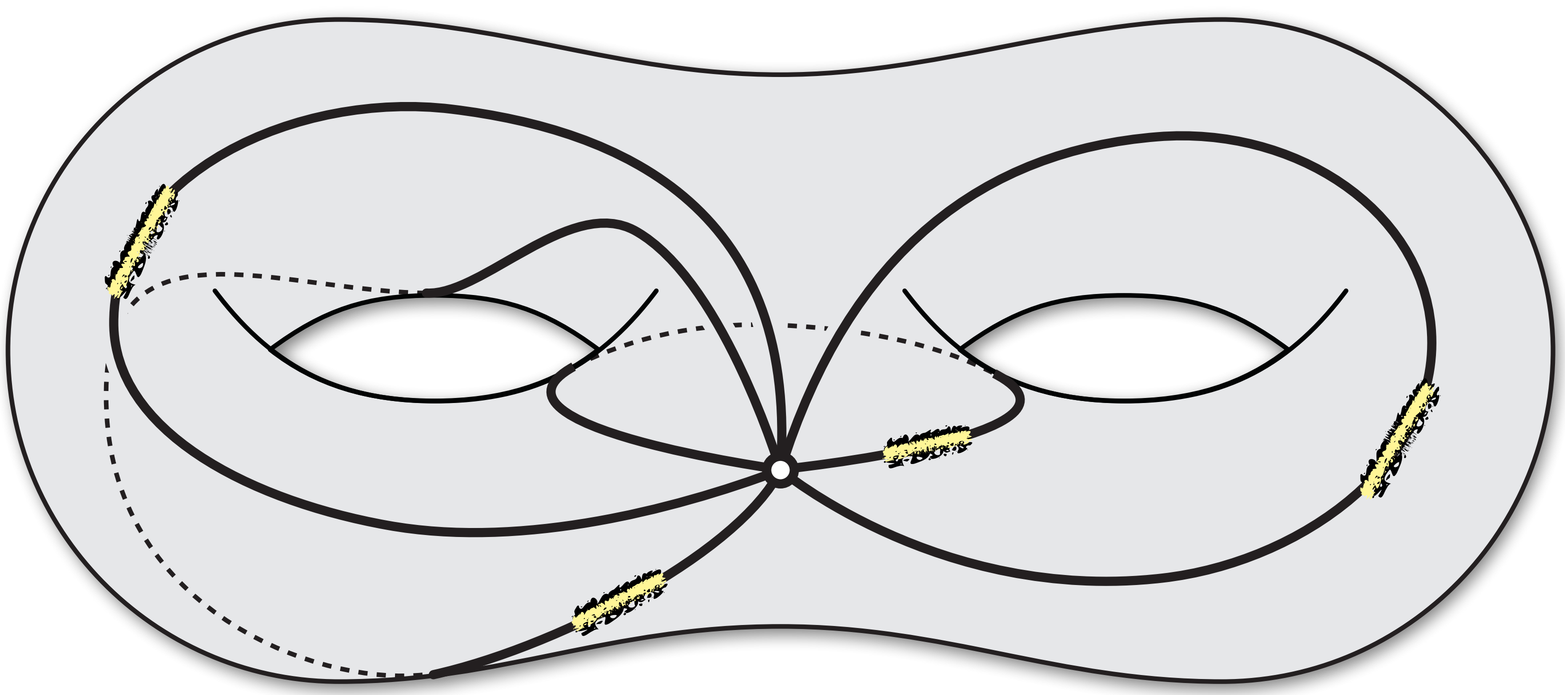
- Fix a tree-cotree decomposition (T, L, C) and a *basepoint* x.

- Nontree edge *uv* defines a *fundamental loop* $loop(T,uv)$:
  - path from x to u + uv + path from v to x

- Nontree edge *uv* defines a *fundamental cycle* $cycle(T,uv)$:
  - unique cycle in $T \cup \{uv\}$
  - path from $lca(u,v)$ to u + uv + path from v to $lca(u,v)$

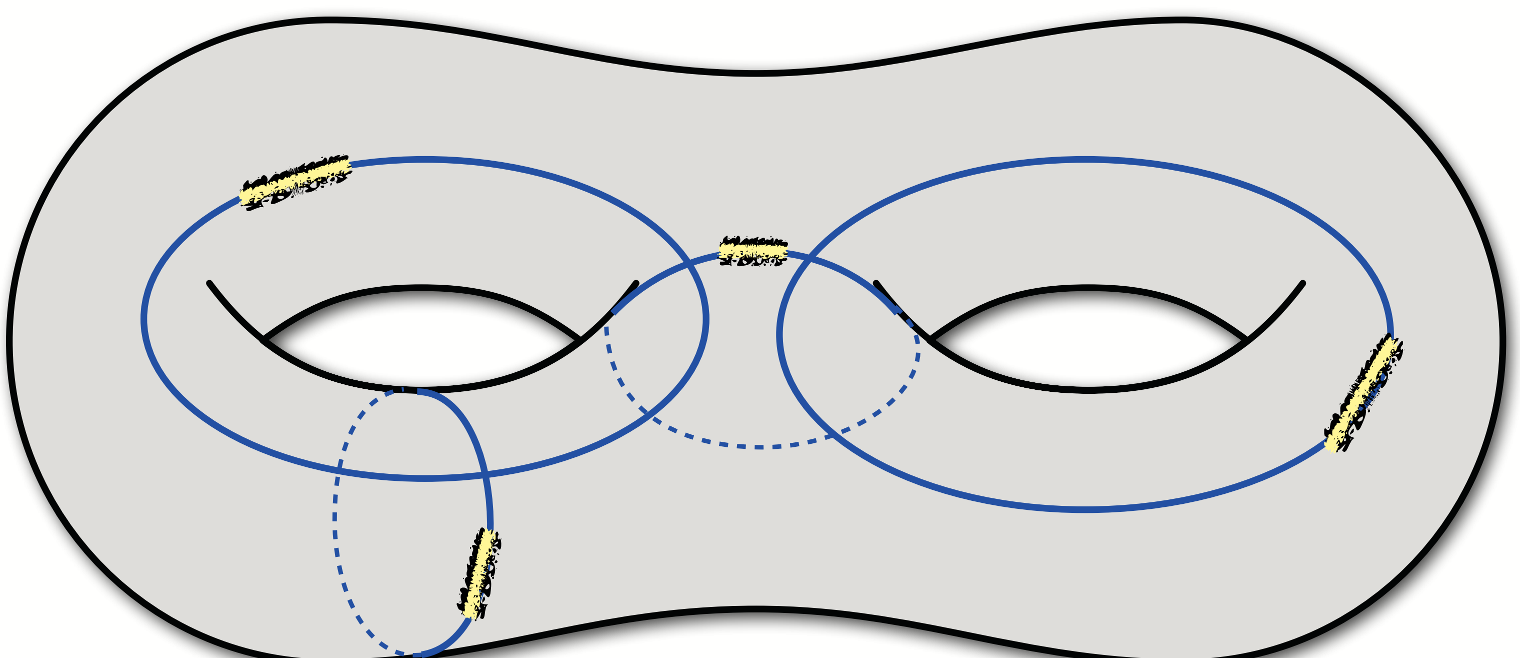
Tree-cotree structures

- *System of loops* $\{loop(T, e) \mid e \in L\}$
  - Cutting Σ along these loops leaves a disk
  - Basis for the fundamental group $\pi_1(\Sigma, x)$



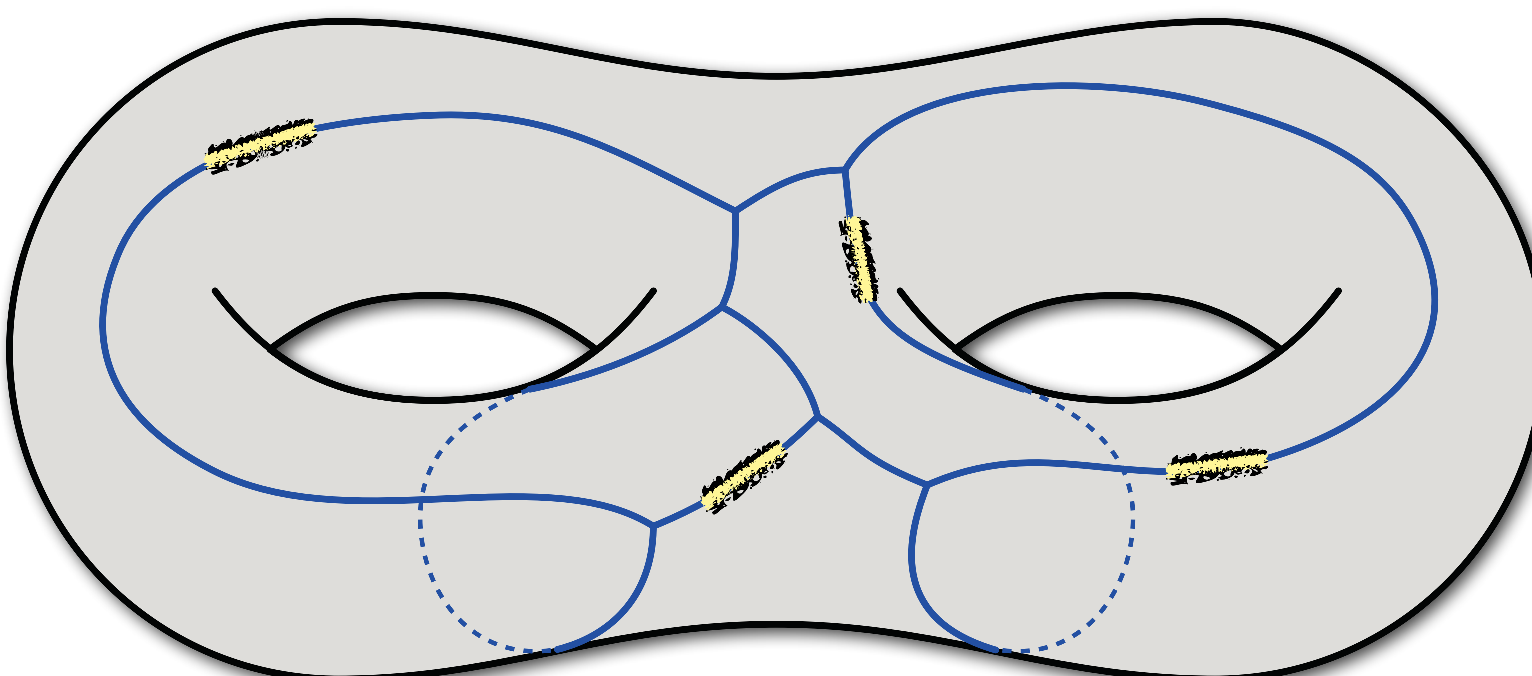
Tree-cotree structures

- *System of cycles* $\{cycle(T, e) \mid e \in L\}$
  - $2g$ simple cycles
  - Basis for the first homology group $H_1(\Sigma)$



Tree-cotree structures

- *Cut graph* $T \cup L = \Sigma \backslash C$
- Remove degree-1 vertices ⇒ *reduced cut graph*
  - Minimal subgraph with one face
  - Composed of at most $3g$ *cut paths* meeting at most $2g$ *branch points*

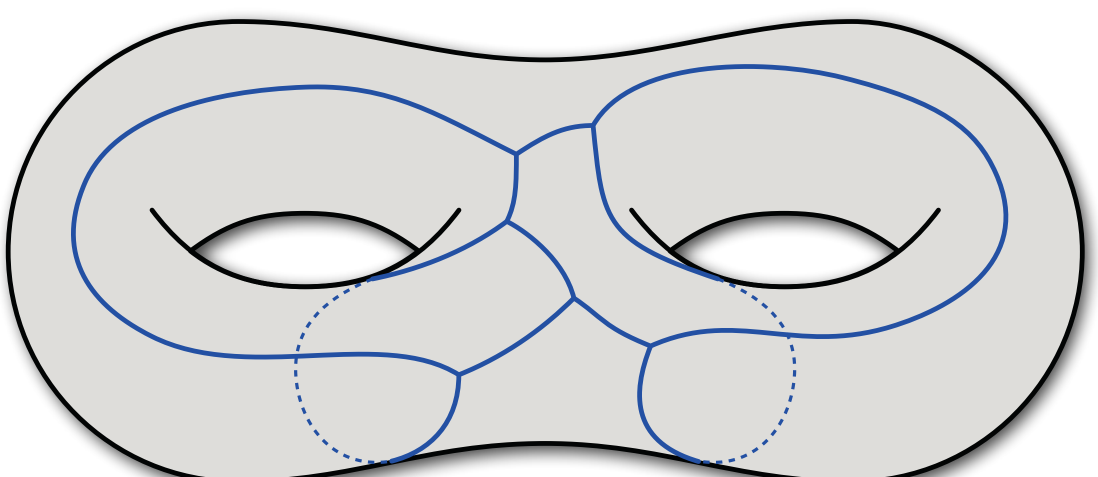
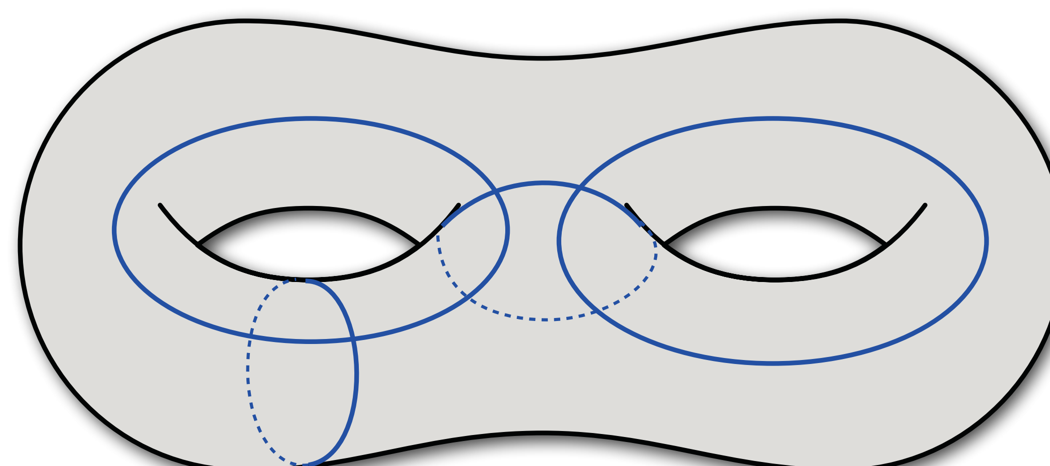


Tree-cotree structures

- Often useful to build these structures *in the dual map Σ^**.
  - dual system of loops
  - dual cut graph
  - dual system of *cocycles* = basis for first *co*homology group $H^1(\Sigma)$

Tree-cotree structures

- Every *noncontractible* cycle in Σ crosses every (dual) reduced cut graph.
- Every *nonseparating* cycle in Σ crosses at least one (co)cycle in every system of (co)cycles.

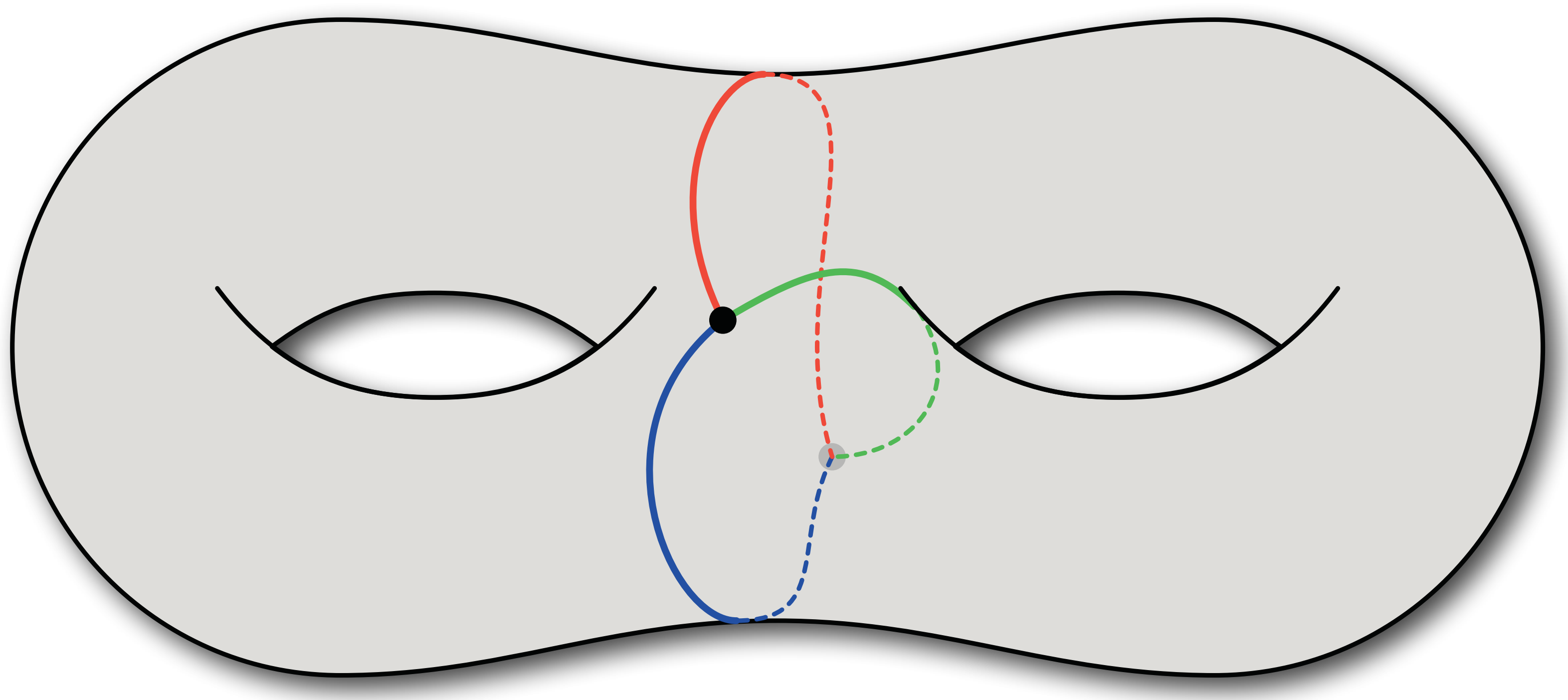



Shortest nontrivial cycles, take 1

Three-path condition

[Thomassen 1990]

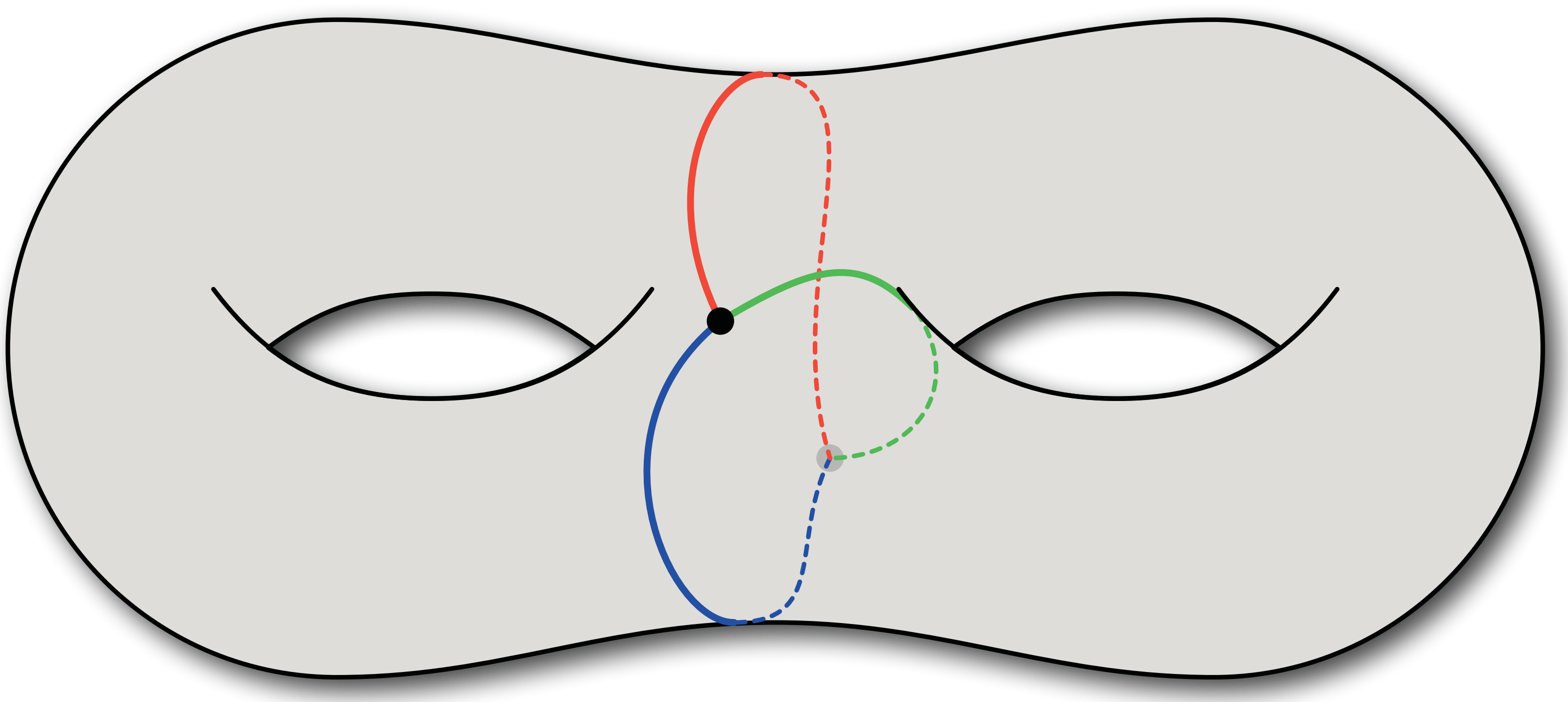
- Any three paths with the same endpoints define three cycles.
- If any two of these cycles are trivial, so is the third.



Three-path condition

[Thomassen 1990]

- The shortest nontrivial cycle consists of two shortest paths between any pair of antipodal points.
- Otherwise, the actual shortest path would create a shorter nontrivial cycle.



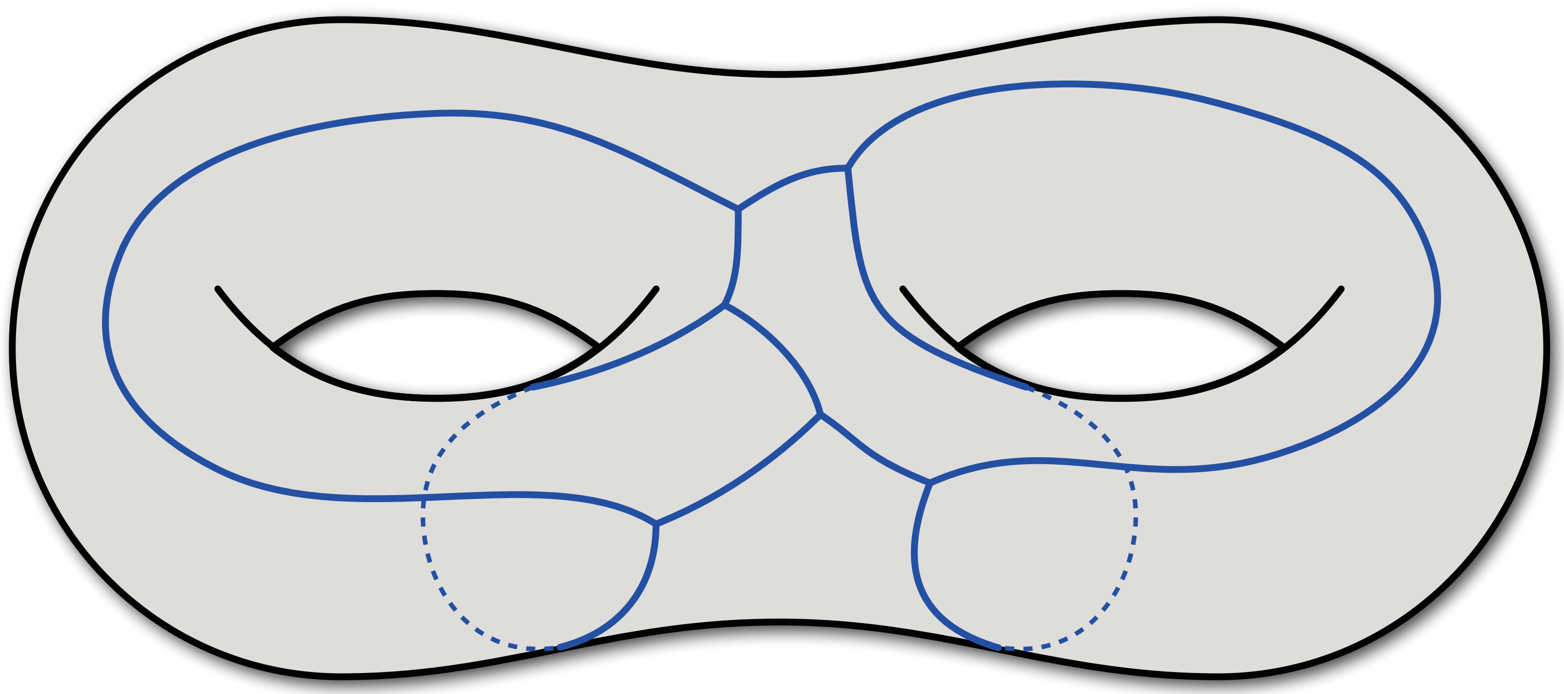
Greedy tree-cotree decomposition

- Assume edges have lengths $\ell(e) \geq 0$
- T = *shortest-path* tree in Σ with arbitrary source vertex x
  - = BFS tree if all lengths = 1
- C^* = *maximum* spanning tree of Σ^* where $w(e^*) = \ell(loop(T,e))$
- Computable in $O(n \log n)$ time using textbook algorithms.
  - $O(n)$ time if all lengths = 1
  - $O(n)$ time if $g=O(n^{1-\varepsilon})$ *[Henzinger et al. '97]*

[Eppstein 2003, Erickson Whittlesey 2005]

Shortest nontrivial loops

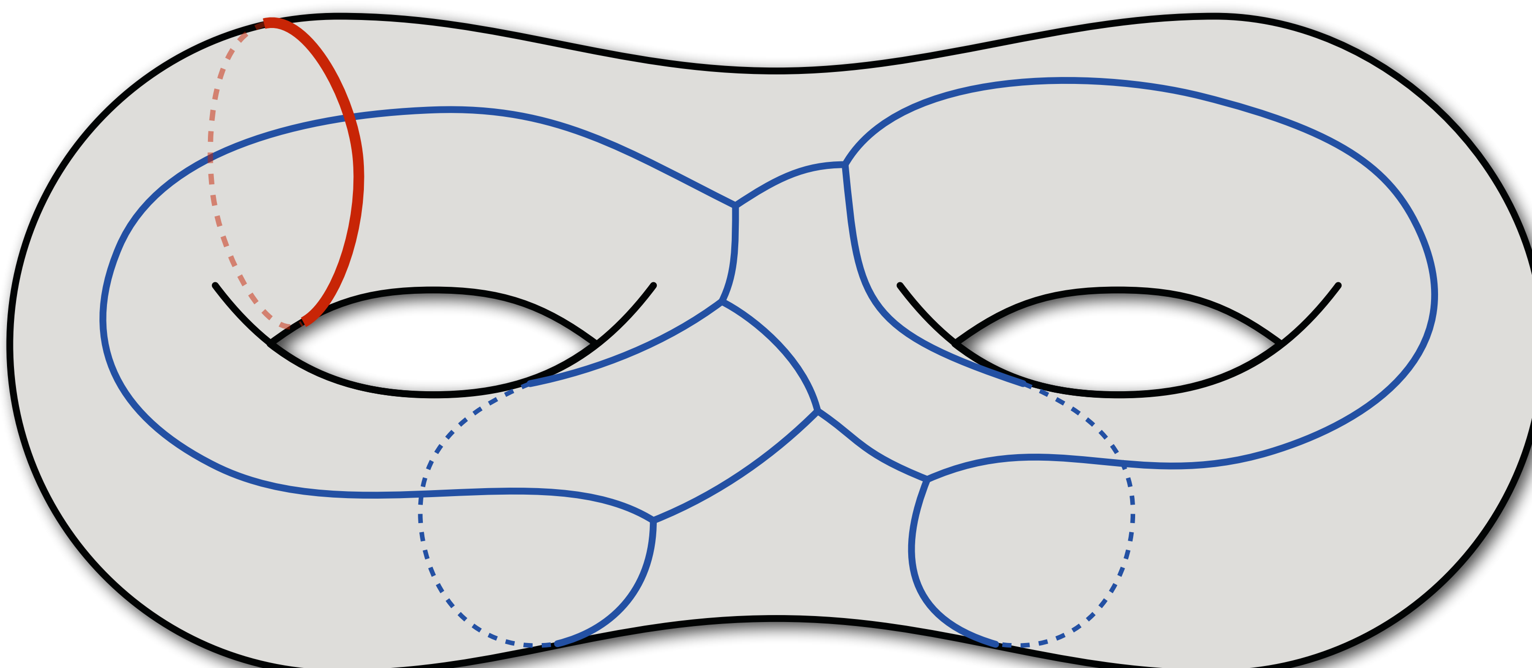
- Build greedy tree-cotree decomposition (T, L, C) based at x.
- Build dual cut graph $X^* = L^* \cup C^*$
- Reduce X^* to get R^*



[Erickson Har-Peled 2005]

Shortest nontrivial loops

- 3-path condition ⇒ We want *loop*(T, e) for some $e \notin T$
- *loop*(T, e) is *noncontractible* iff $e^* \in R^*$
- *loop*(T, e) is *nonseparating* iff $e^* \in R^*$ and $R^* \setminus e^*$ is connected

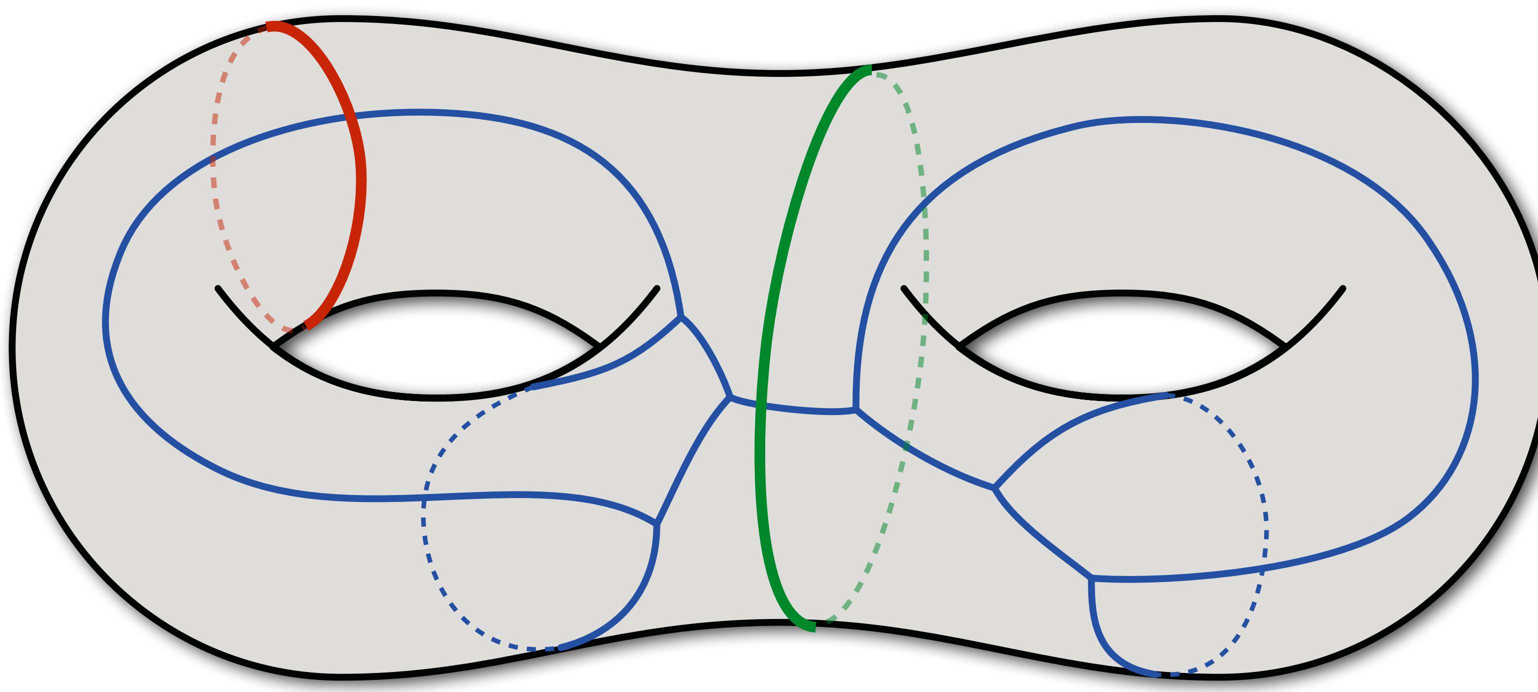


[Erickson Har-Peled 2005]

[Cabello, Colin de Verdière, Lazarus 2010]

Shortest nontrivial loops

- 3-path condition $\Rightarrow$ We want $loop(T, e)$ for some $e \notin T$
- $loop(T, e)$ is *noncontractible* iff $e^* \in R^*$
- $loop(T, e)$ is *nonseparating* iff $R^* \setminus e^*$ is connected



[Erickson Har-Peled 2005]

[Cabello, Colin de Verdière, Lazarus 2010]

Shortest non-trivial cycle

[Erickson Har-Peled 2005]

- For each basepoint: $O(n \log n)$ time.
- Try all possible basepoints: $O(n^2 \log n)$ time.

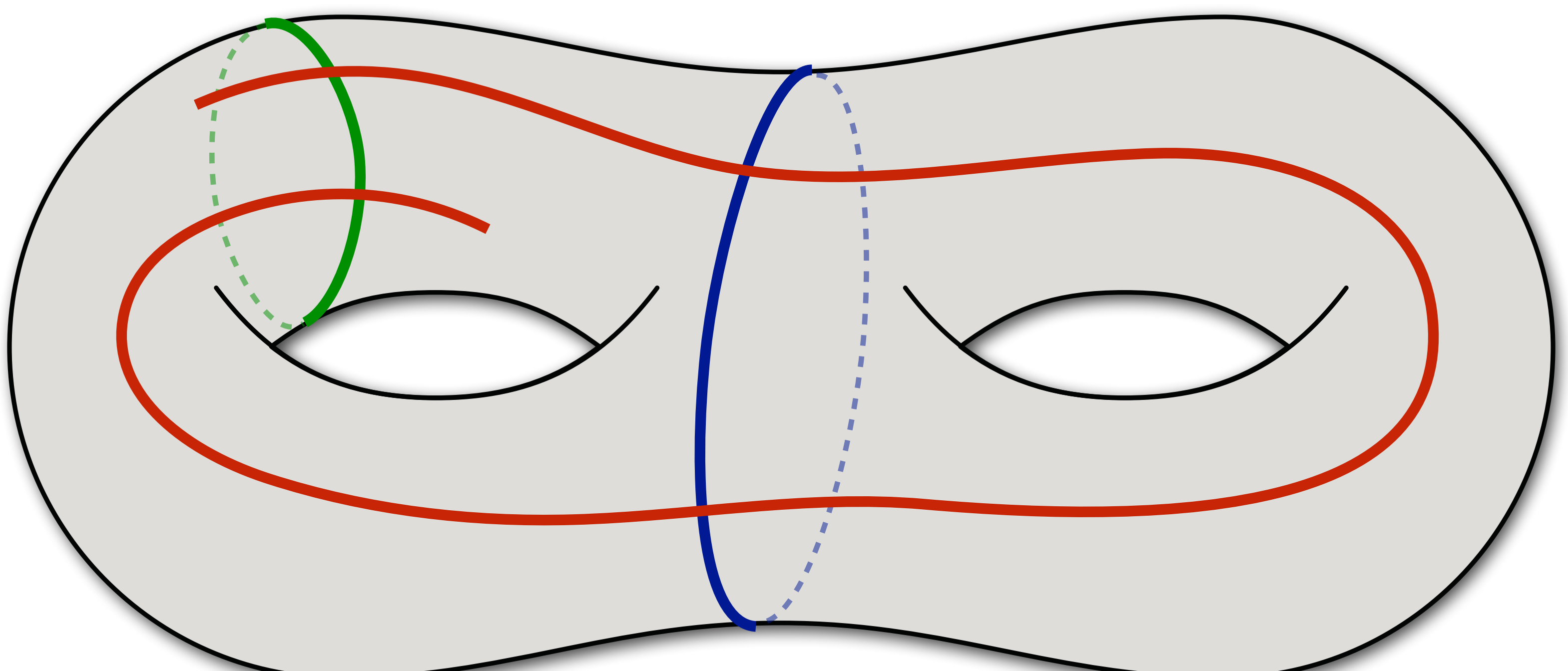
Shortest non-trivial cycle

[Erickson Har-Peled 2005]

- For each basepoint: $O(n \log n)$ time.
- Try all possible basepoints: $O(n^2 \log n)$ time.
- This is the fastest algorithm known.
  - Significant improvement would also improve the best time to compute the *girth* of a sparse graph: $O(n^2)$ = BFS at each vertex *[Itai Rodeh 1978]*
  - Computing the girth of a *dense* graph is at least as hard as all-pairs shortest paths and boolean matrix multiplication. *[Vassilevska Williams, Williams 2010]*

One-cross lemmas

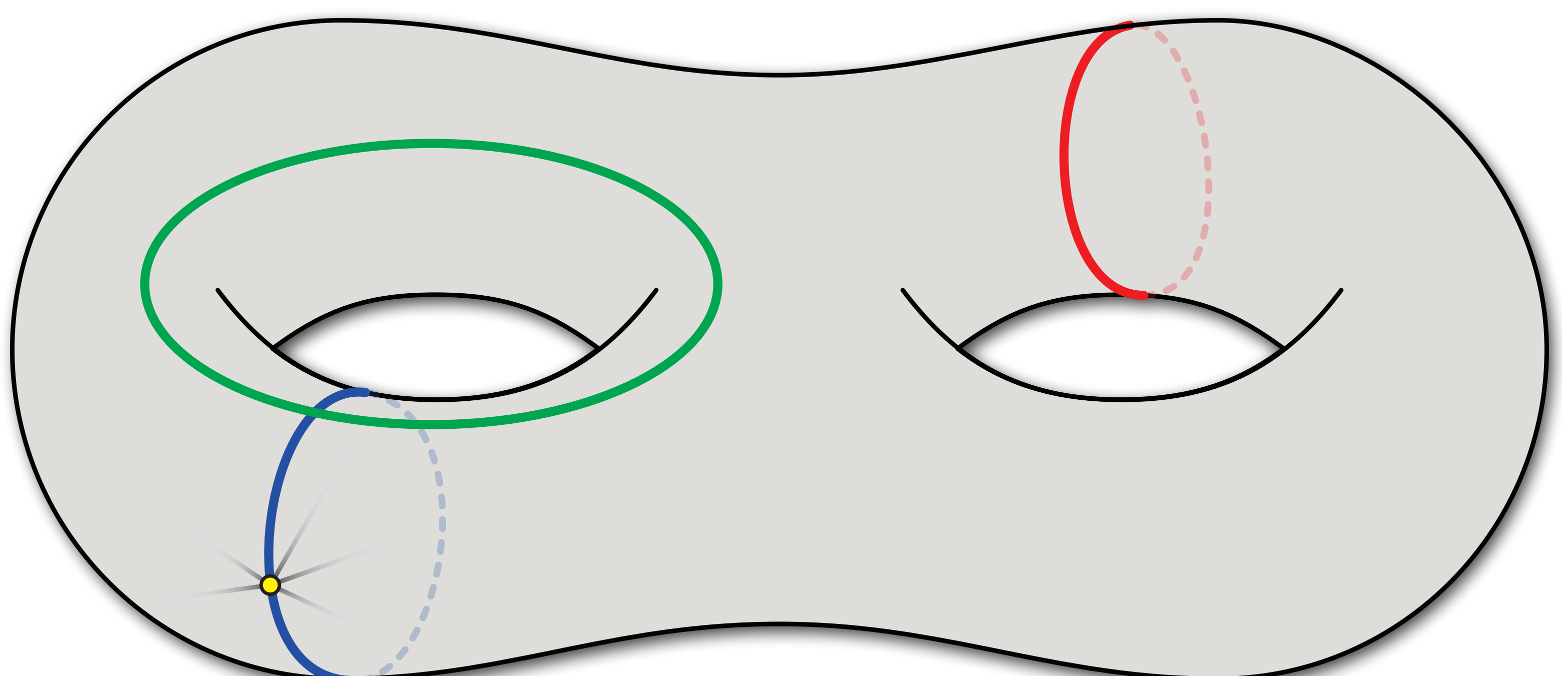
- The shortest nontrivial cycle crosses any shortest path at most once
- Otherwise, we could find a shorter nontrivial cycle!



One-cross lemmas

[Cabello Mojar 2005]

- Let γ^* be the shortest *nonseparating* cycle, and let γ be any cycle in a greedy system of cycles.
- Then γ^* and γ cross *at most* once.

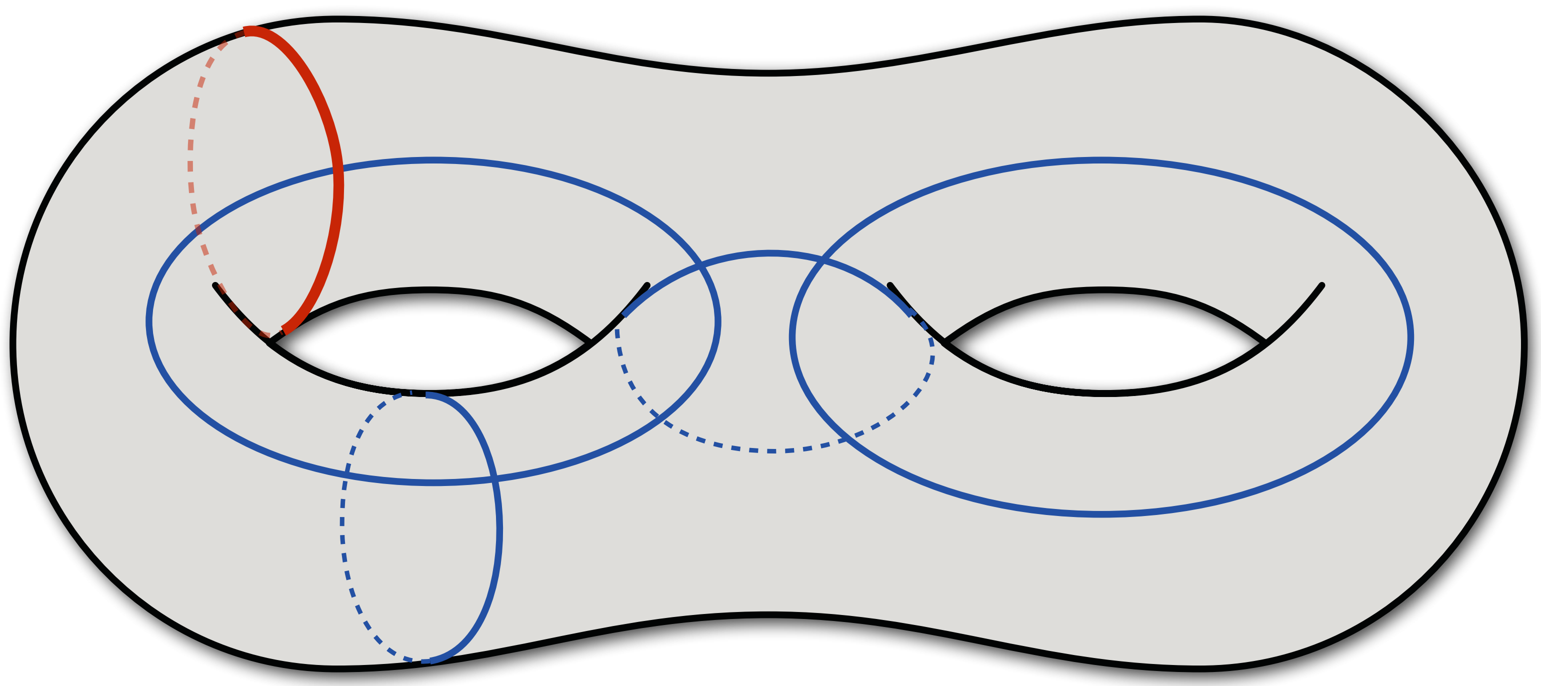


Faster algorithm

[Cabello Chambers 2007]

To compute the shortest *nonseparating* cycle:

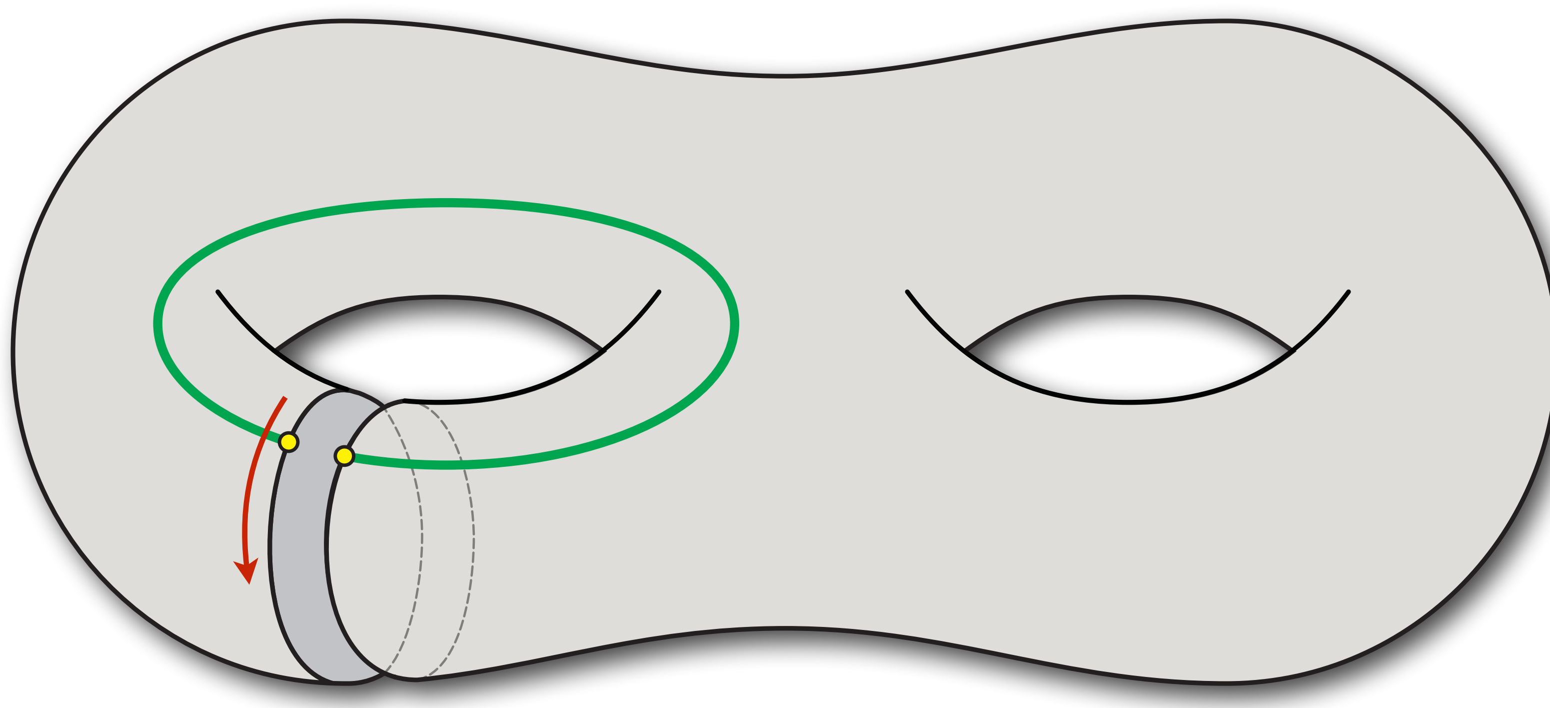
- Compute a greedy system of cycles $\gamma_1, \gamma_2, \ldots, \gamma_{2g}$
- Find the shortest cycle that crosses each greedy cycle γ_i once



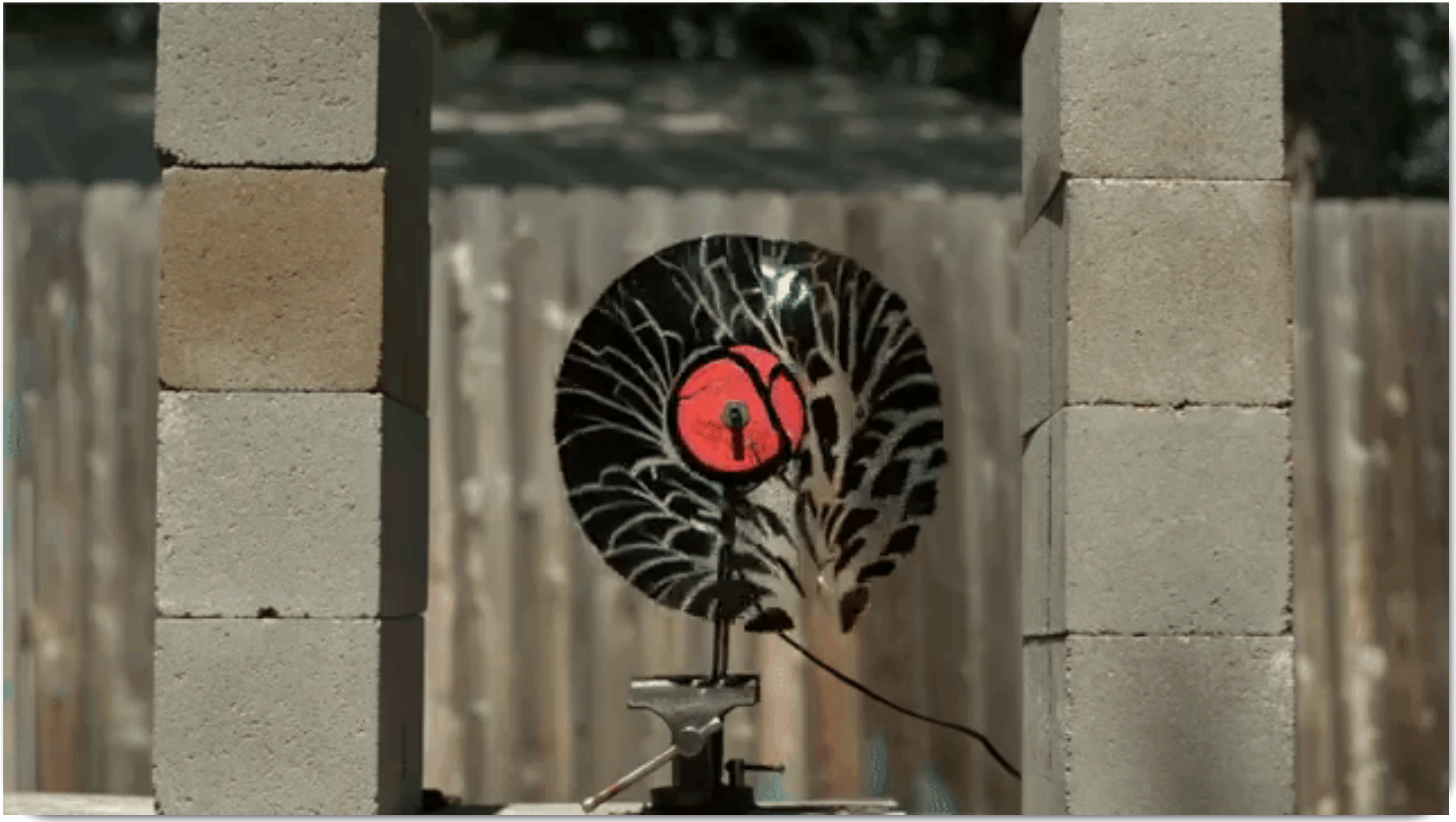
Algorithm

[Cabello Chambers 2007]

- To find the shortest cycle that crosses γ_i once:
  - Cut the surface open along γ_i. Resulting surface $\Sigma{\times}\gamma_i$ has two copies of γ on its boundary.
  - Find the *shortest path* in $\Sigma{\times}\gamma_i$ between the clones of each vertex of γ_i

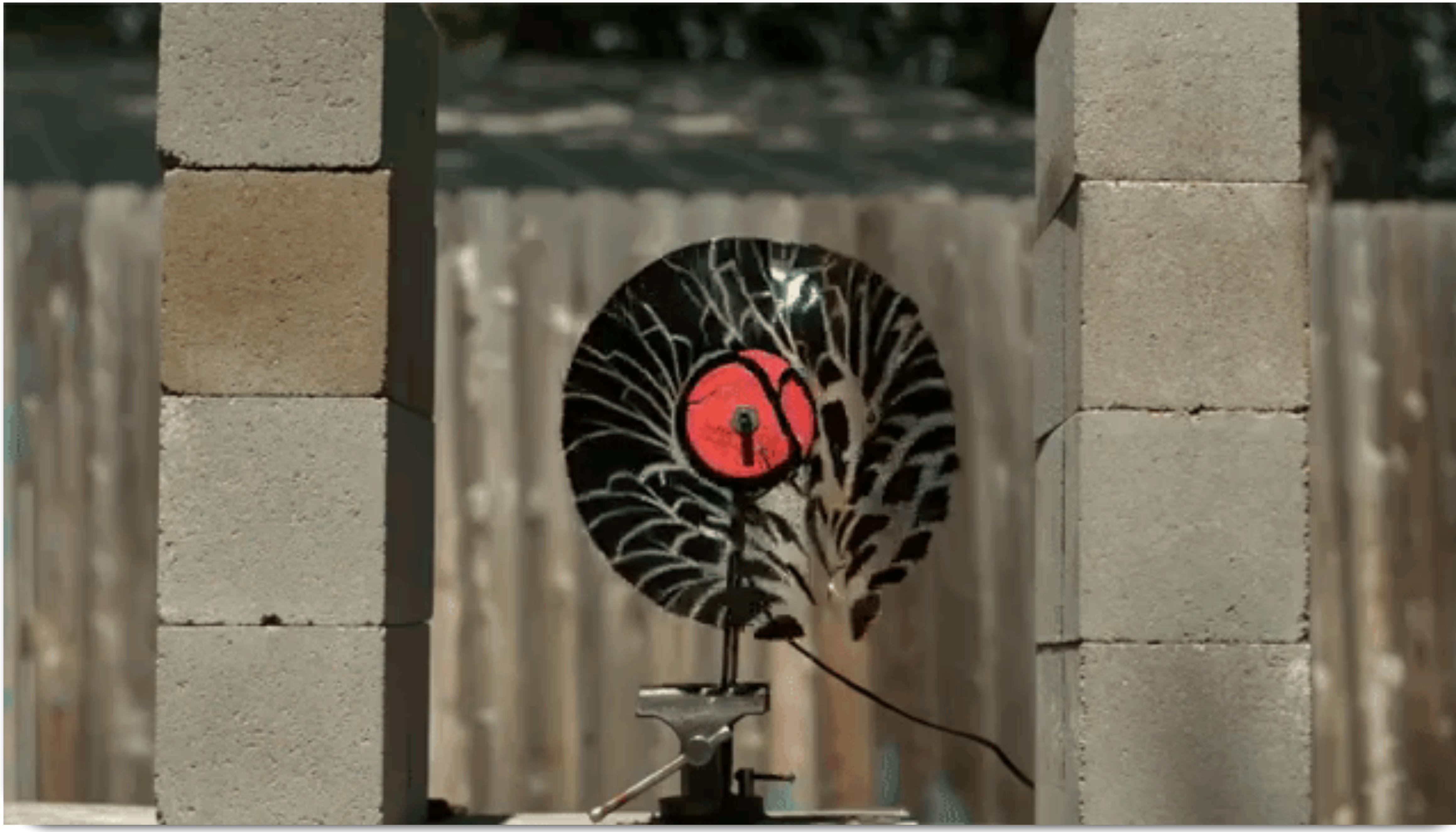


Multiple-Source Shortest Paths



[Free Gruchy ("Slow-Mo Guys") 2018]

Multiple-Source Shortest Paths

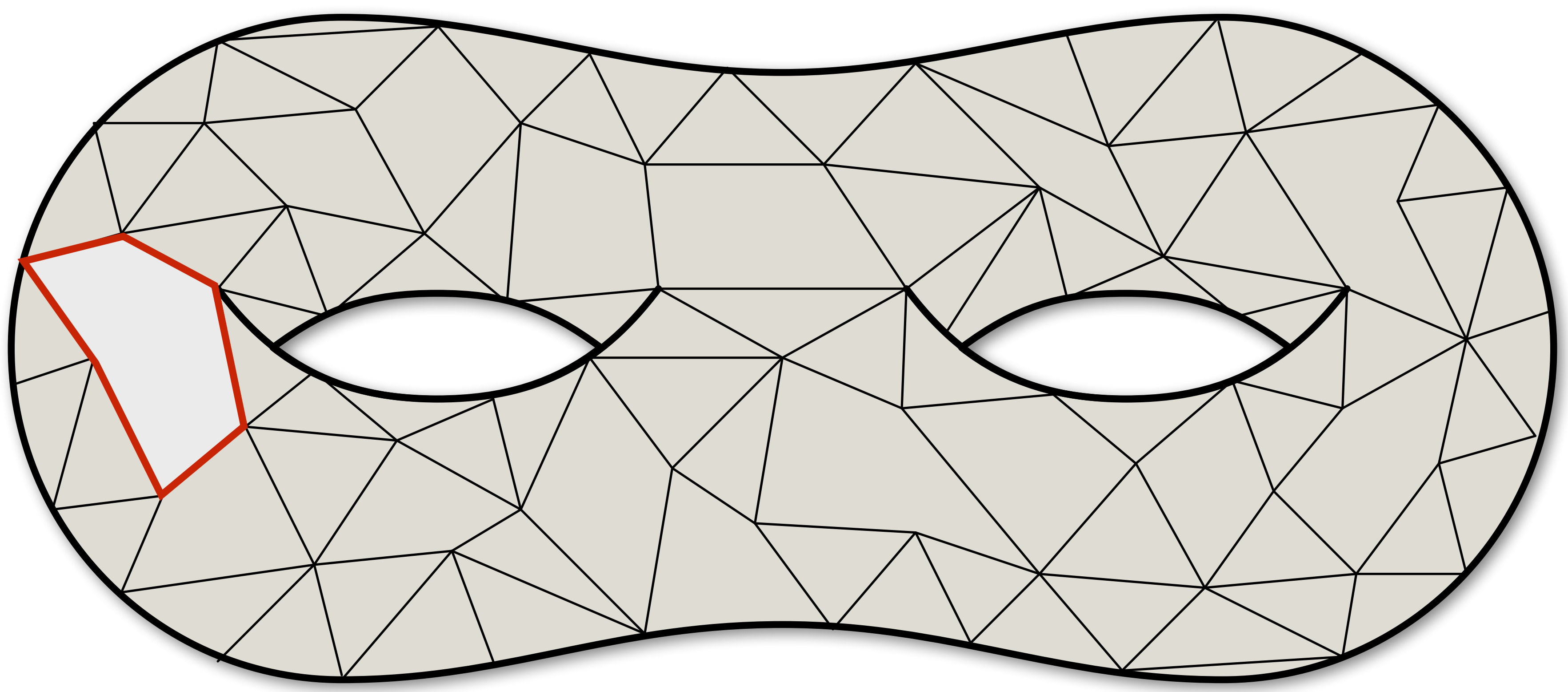


[Free Gruchy ("Slow-Mo Guys") 2018]

Multiple-Source Shortest Paths

[Klein 2005]

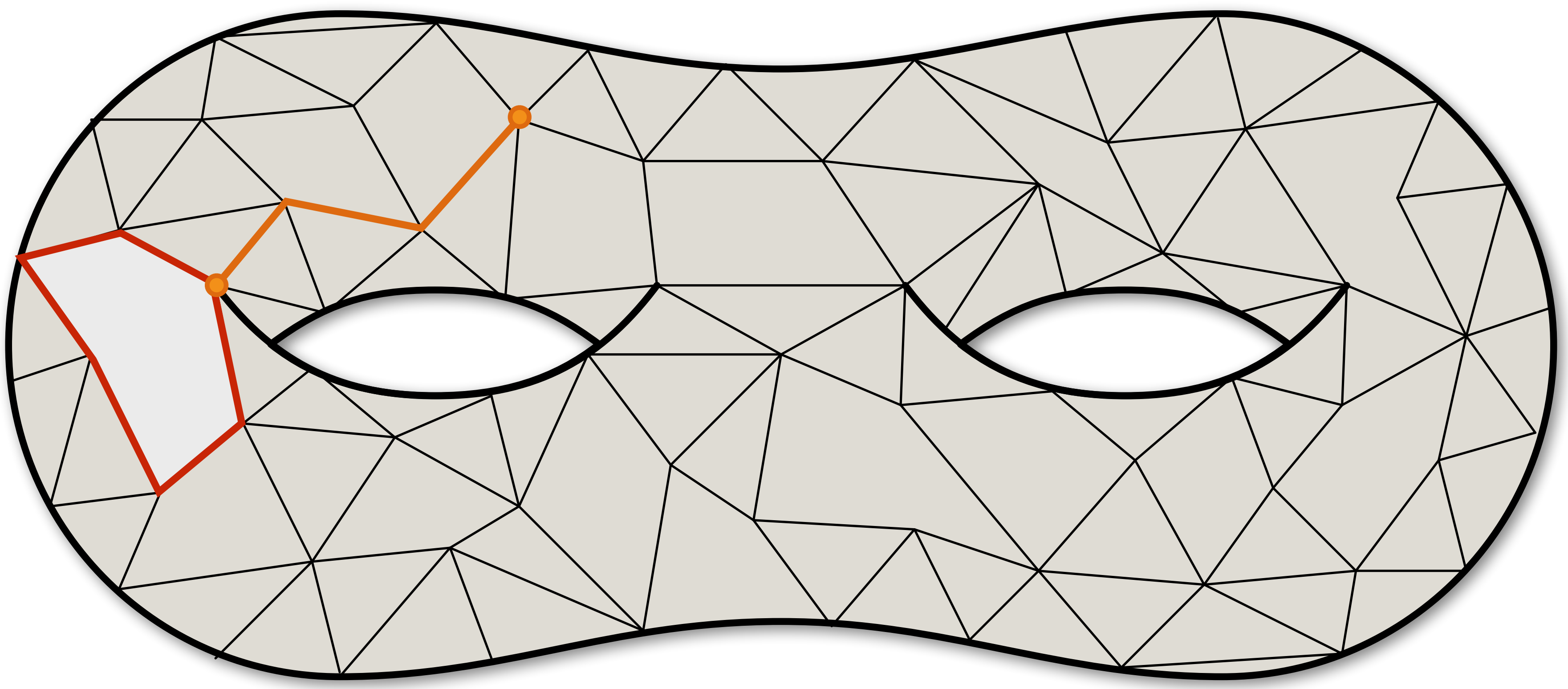
- Compute shortest paths between many pairs of vertices, with one vertex of each pair on a fixed “outer” face.



Multiple-Source Shortest Paths

[Klein 2005]

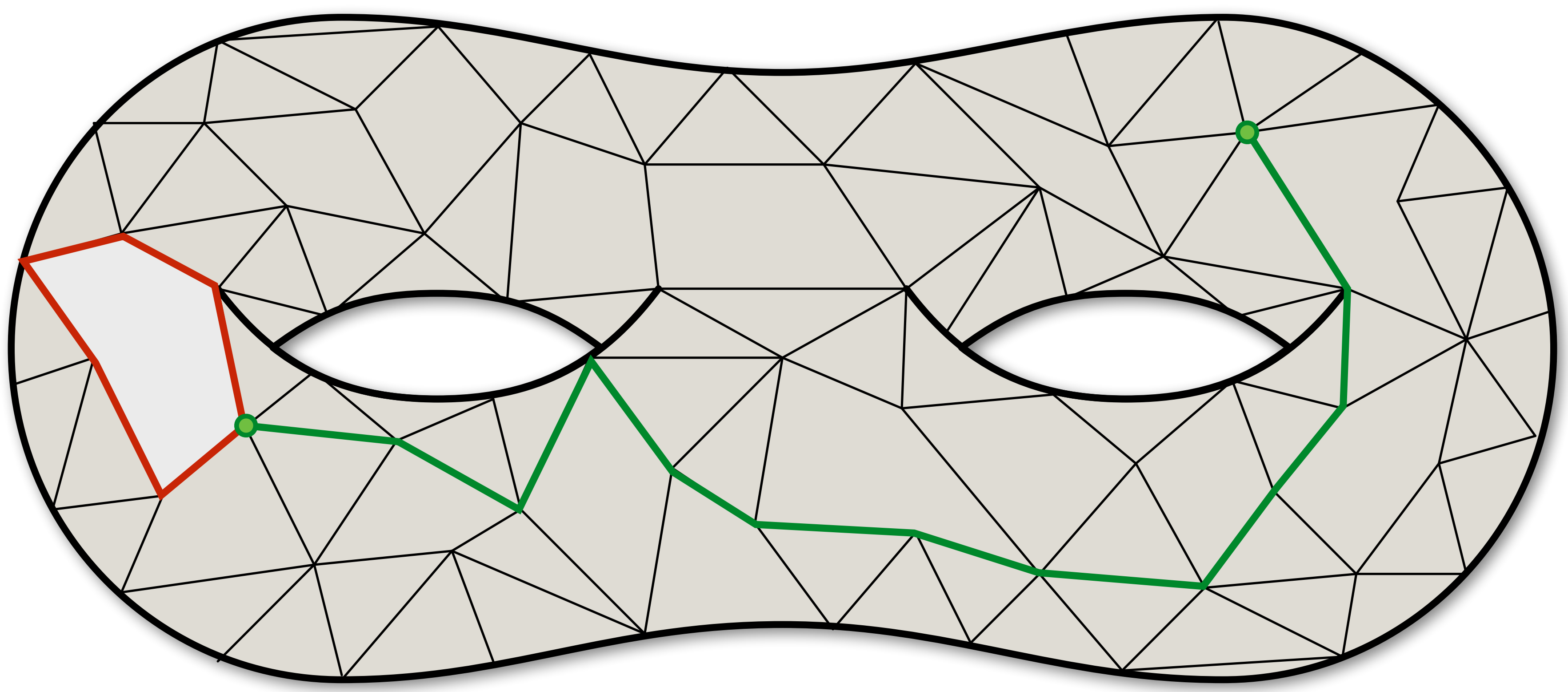
- Compute shortest paths between many pairs of vertices, with one vertex of each pair on a fixed “outer” face.



Multiple-Source Shortest Paths

[Klein 2005]

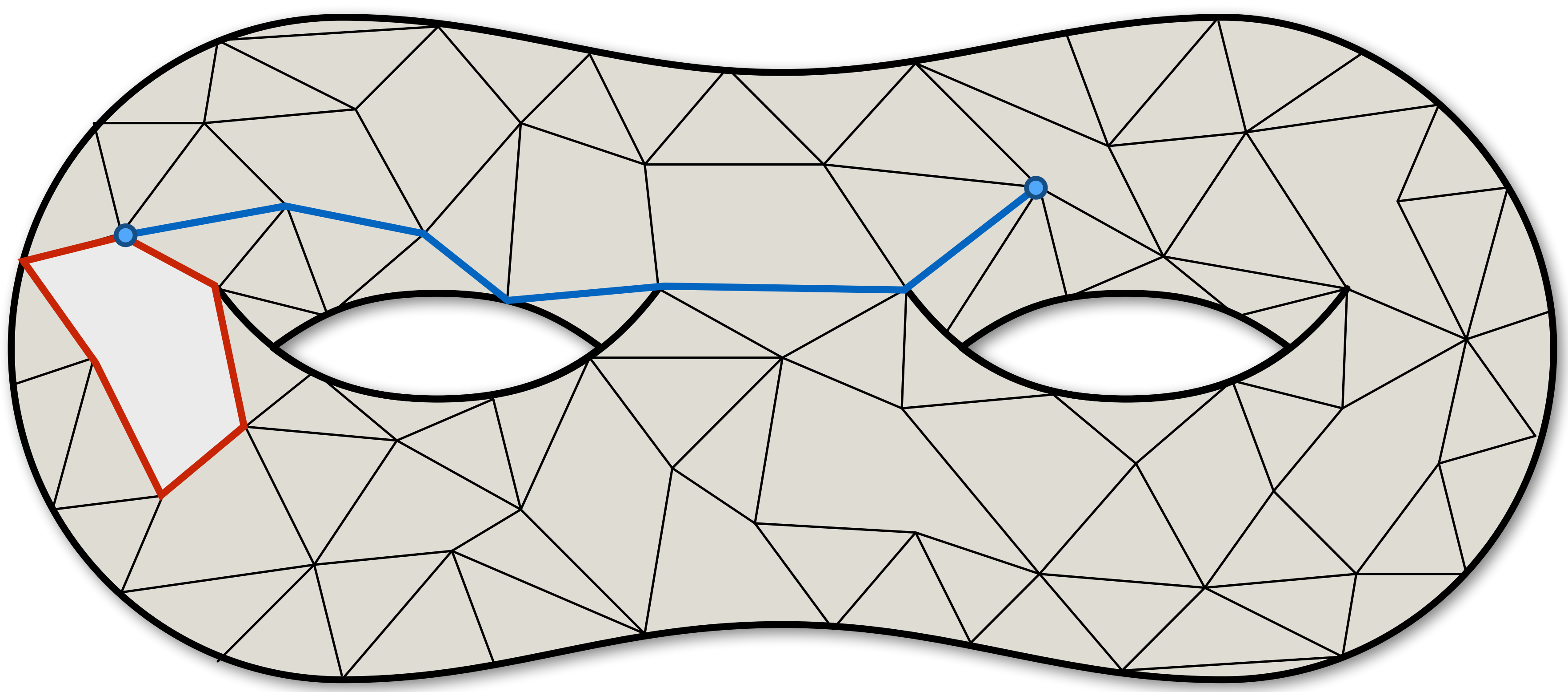
- Compute shortest paths between many pairs of vertices, with one vertex of each pair on a fixed "outer" face.



Multiple-Source Shortest Paths

[Klein 2005]

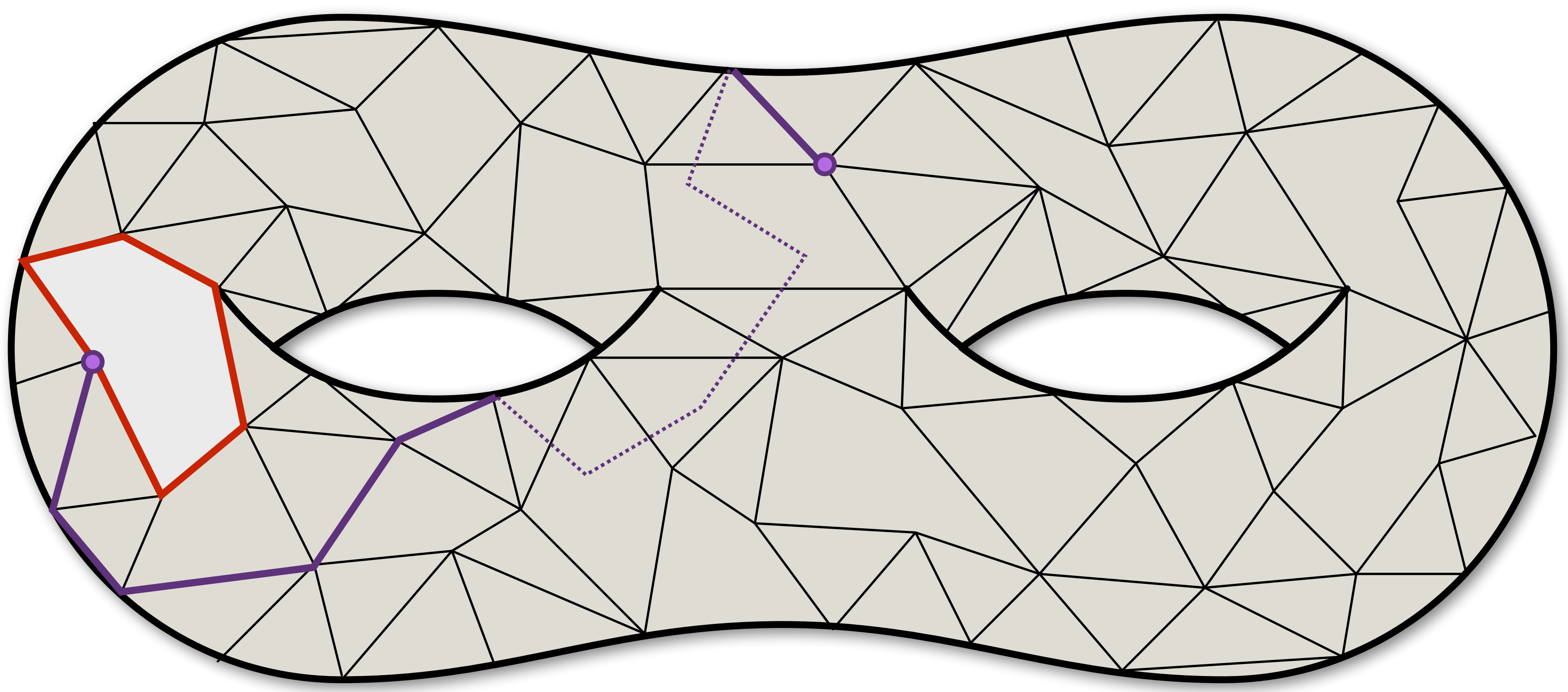
- Compute shortest paths between many pairs of vertices, with one vertex of each pair on a fixed “outer” face.



Multiple-Source Shortest Paths

[Klein 2005]

- Compute shortest paths between many pairs of vertices, with one vertex of each pair on a fixed “outer” face.



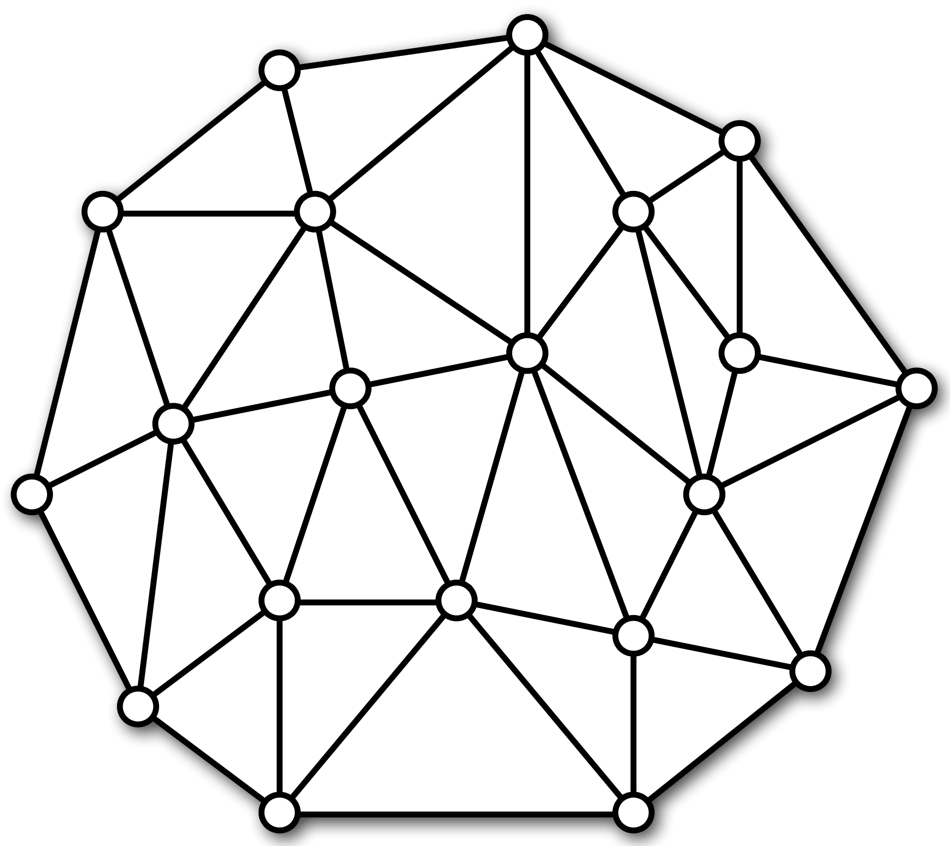
Naïve algorithm

- For each boundary vertex s, compute the shortest-path tree rooted at s in $O(n \log n)$ time. *[Dijkstra 1956]*
- The overall algorithm runs in $O(n^2 \log n)$ time.
- But in fact, we can (implicitly) compute ***all*** such distances in just $O(g^2 n \log n)$ time.

Planar MSSP

[Klein 2005]

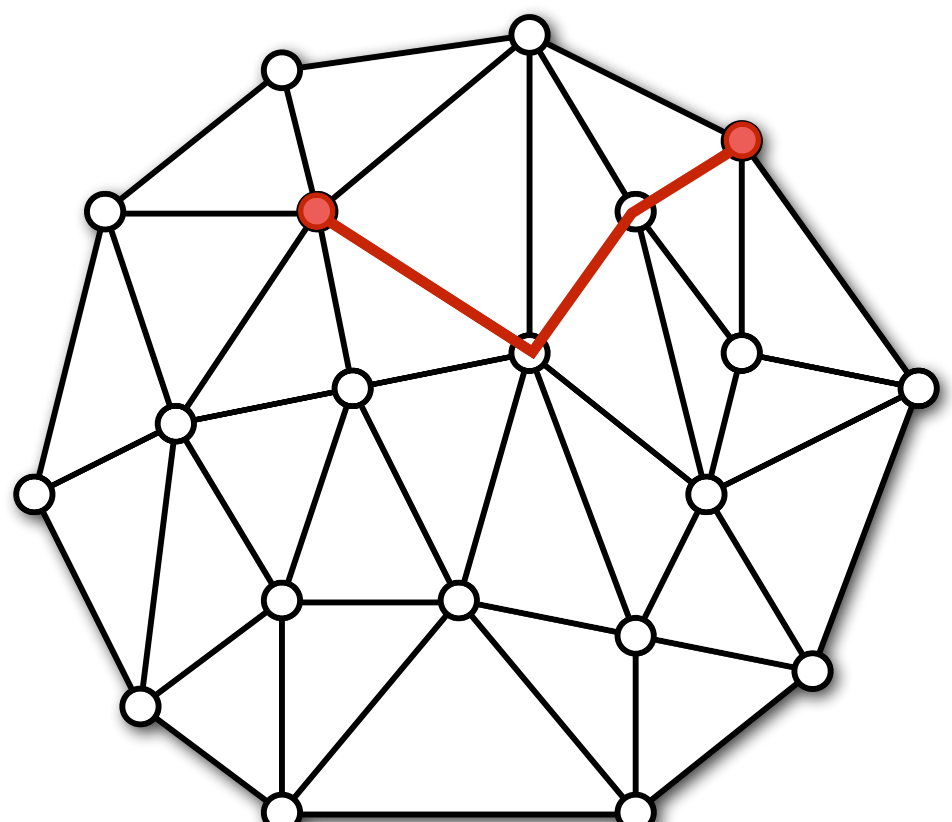
- Let's start with the simplest possible setting.
- Implicitly compute shortest paths in a *plane* graph G from every boundary vertex to every other vertex.



Planar MSSP

[Klein 2005]

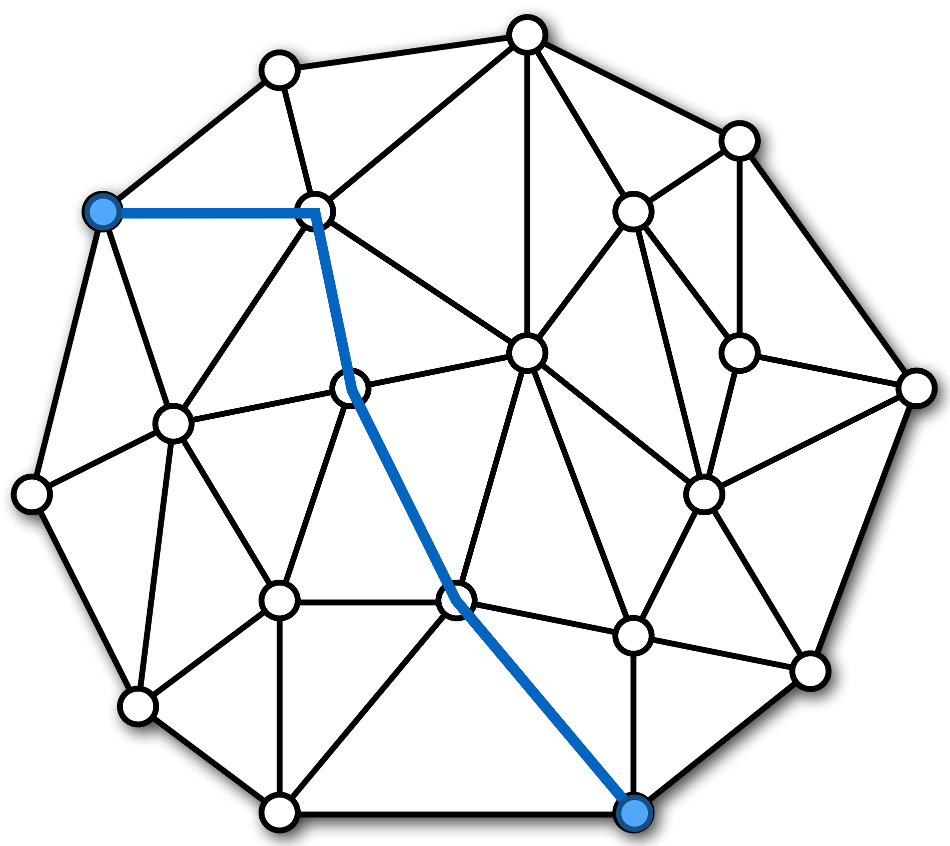
- Let's start with the simplest possible setting.
- Implicitly compute shortest paths in a *plane* graph *G* from every boundary vertex to every other vertex.



Planar MSSP

[Klein 2005]

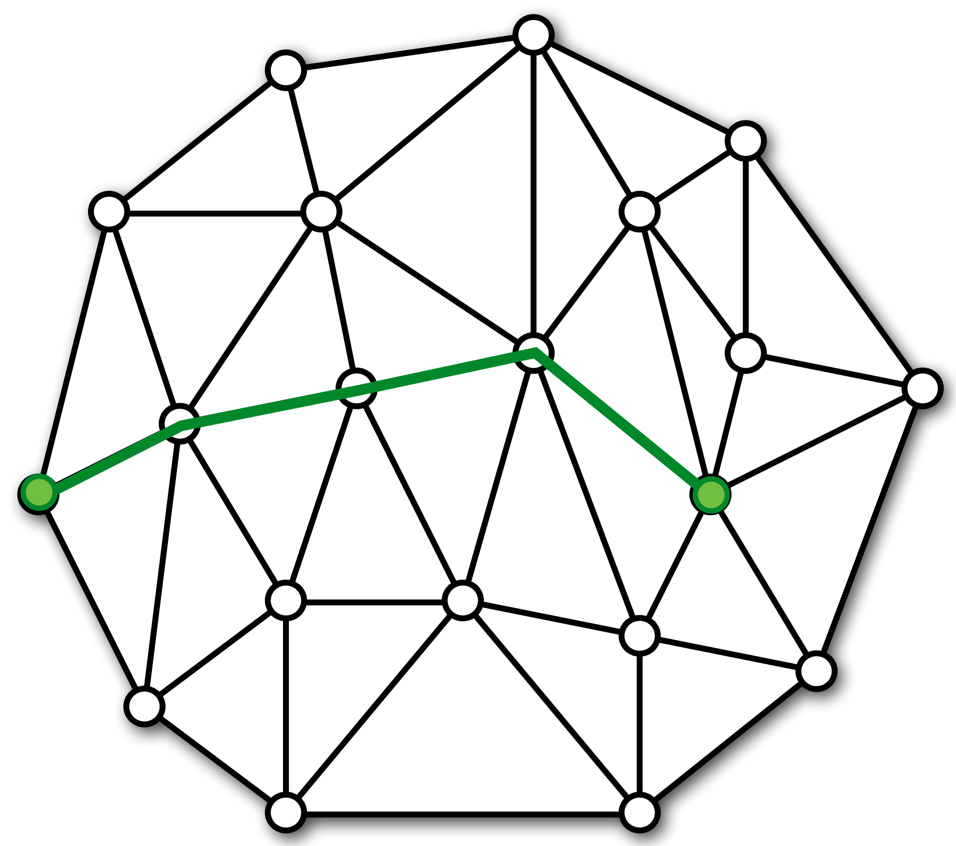
- Let's start with the simplest possible setting.
- Implicitly compute shortest paths in a *plane* graph G from every boundary vertex to every other vertex.



Planar MSSP

[Klein 2005]

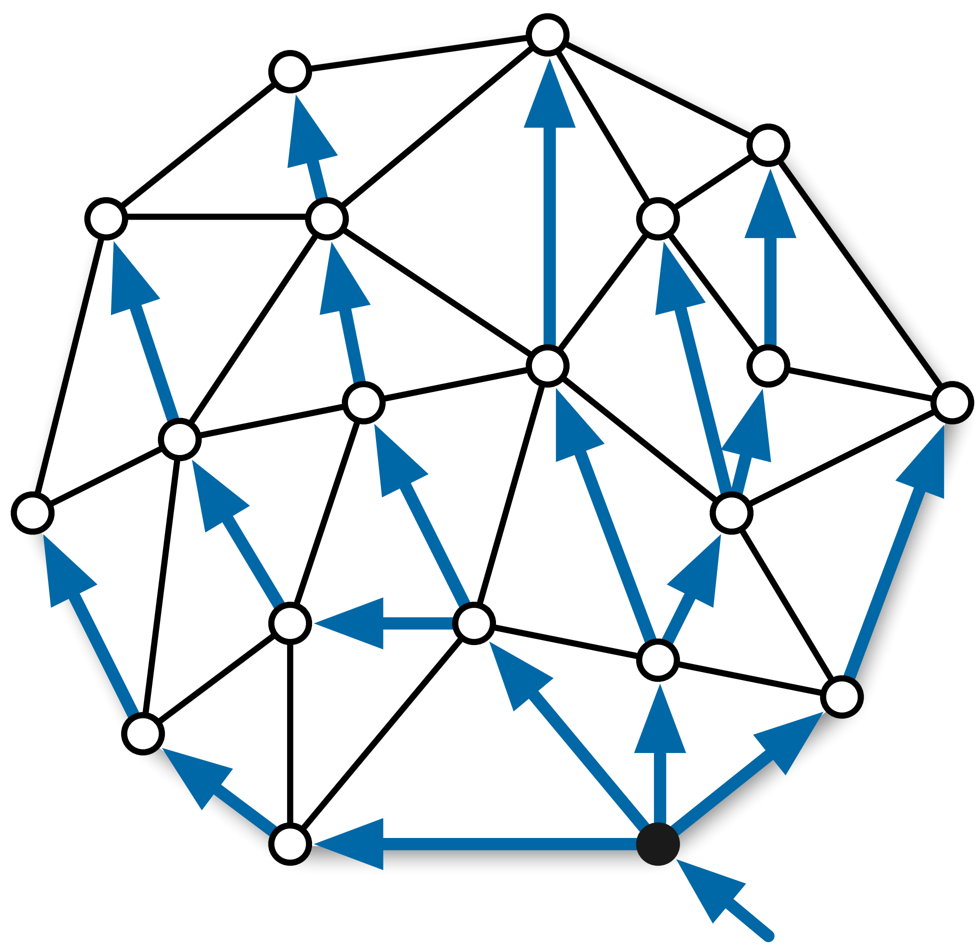
- Let’s start with the simplest possible setting.
- Implicitly compute shortest paths in a *plane* graph *G* from every boundary vertex to every other vertex.



Planar MSSP

[Klein 2005]

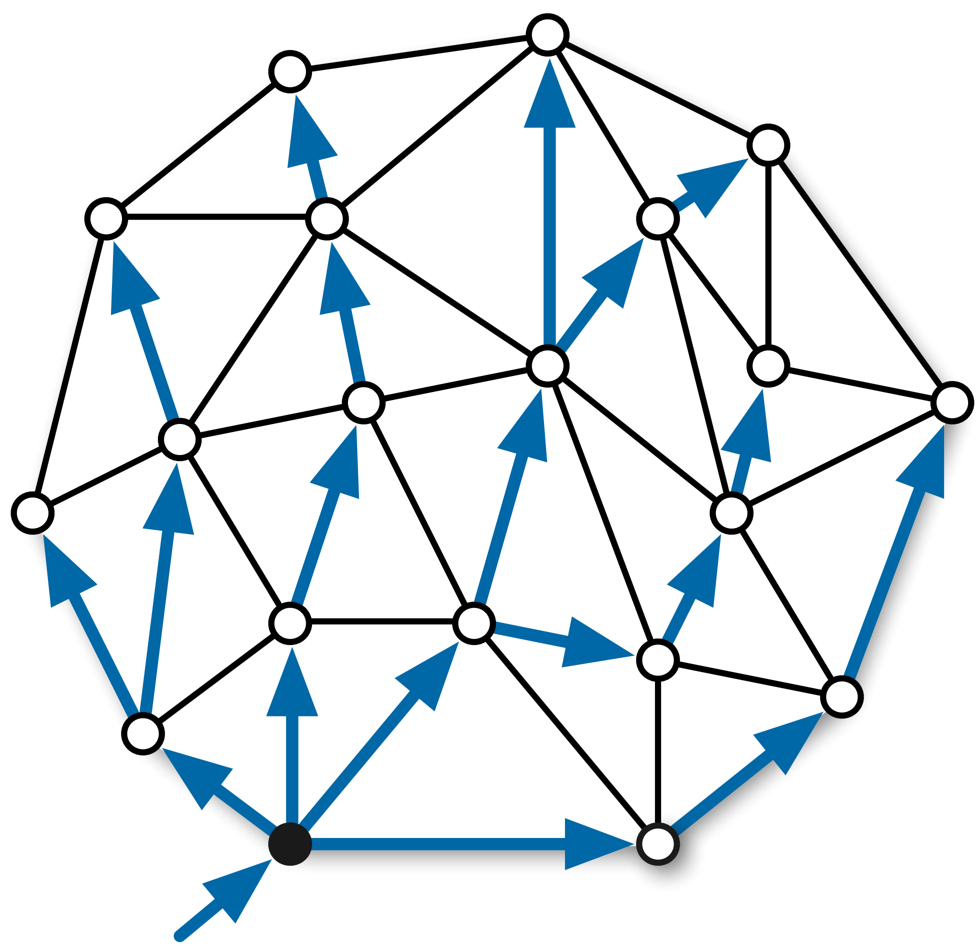
- Intuitively, we want the shortest-path tree rooted at every boundary vertex.



Planar MSSP

[Klein 2005]

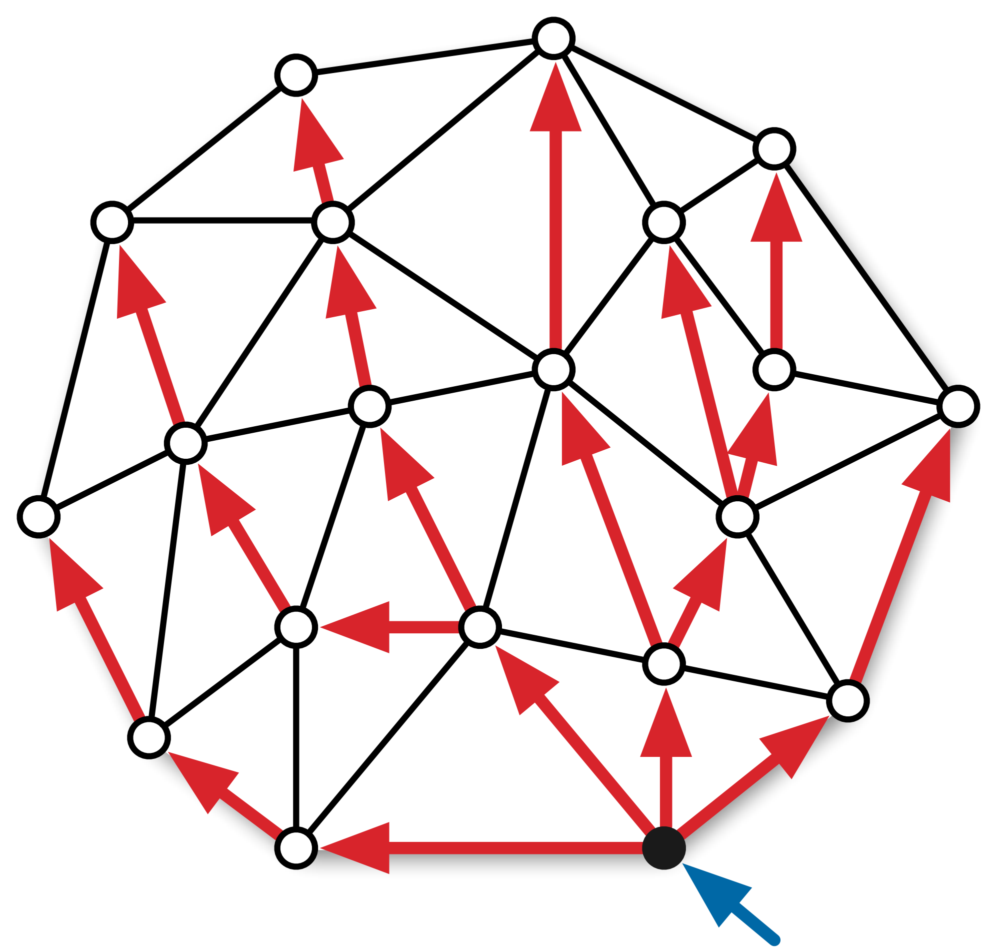
- Intuitively, we want the shortest-path tree rooted at every boundary vertex.



Planar MSSP

[Klein 2005]

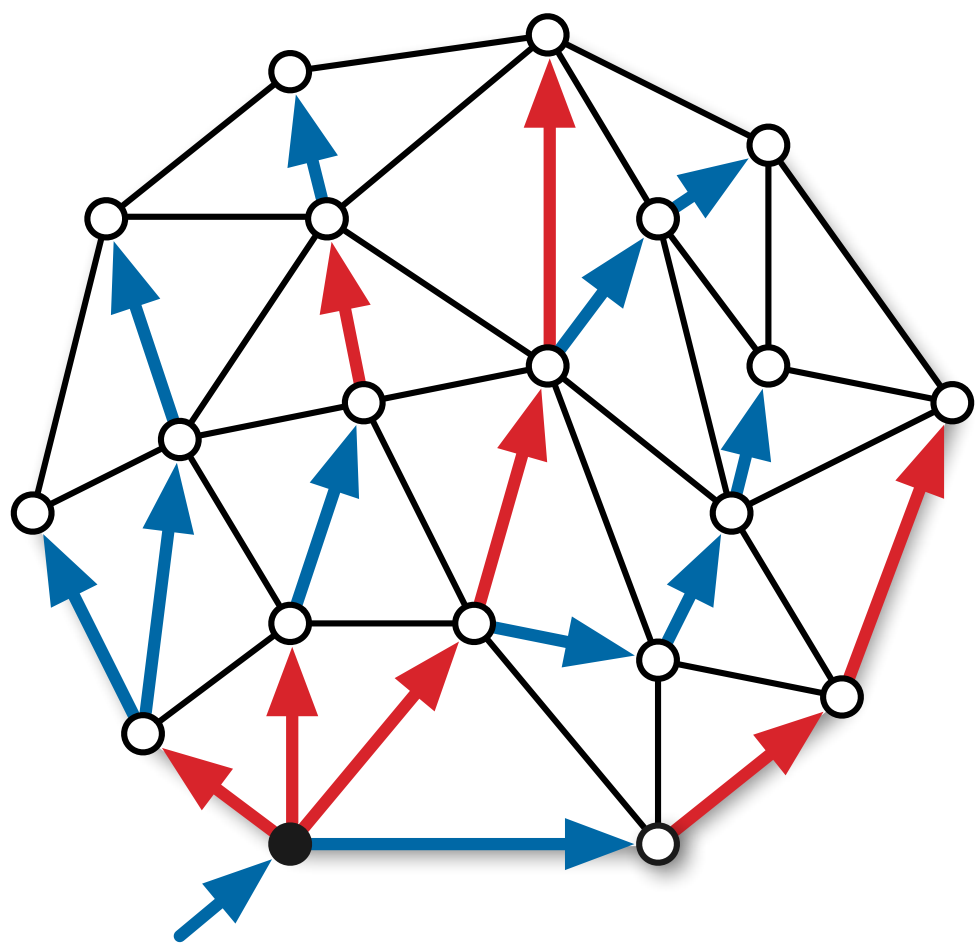
- In fact, we only need to compute the *first* shortest-path tree, followed by *changes* from each tree to the next.



Planar MSSP

[Klein 2005]

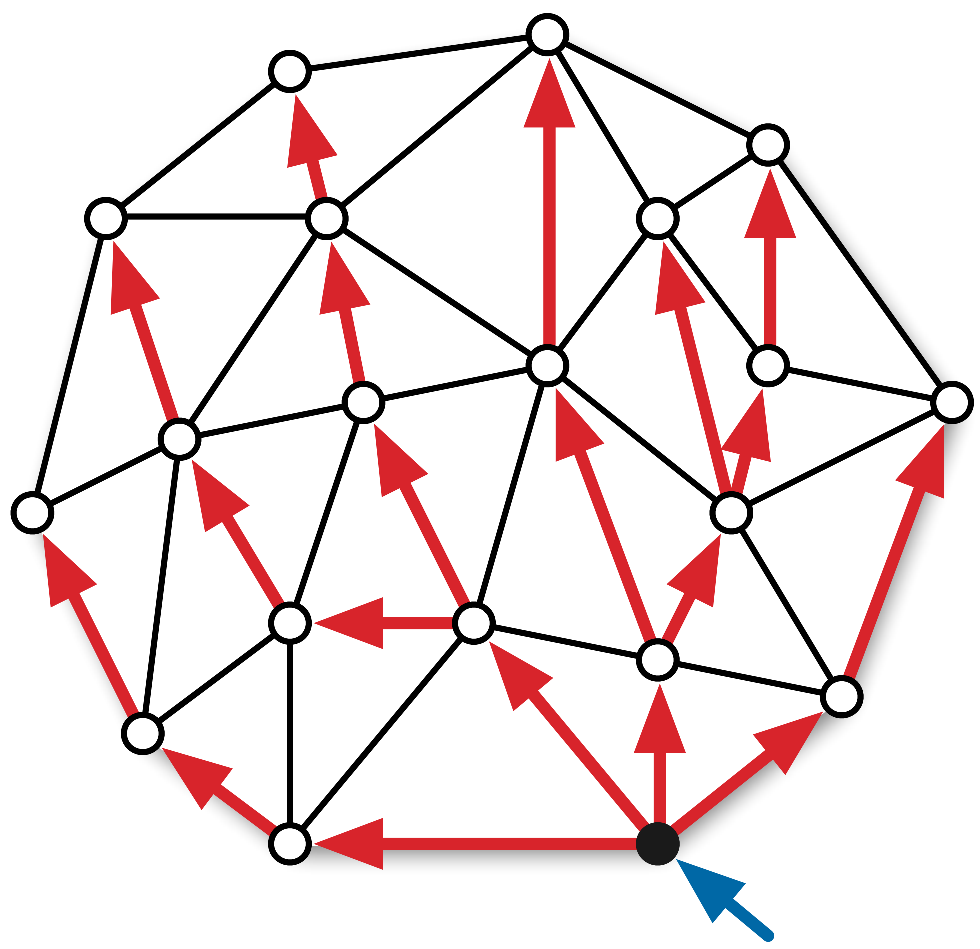
- In fact, we only need to compute the *first* shortest-path tree, followed by *changes* from each tree to the next.



Planar MSSP

[Klein 2005]

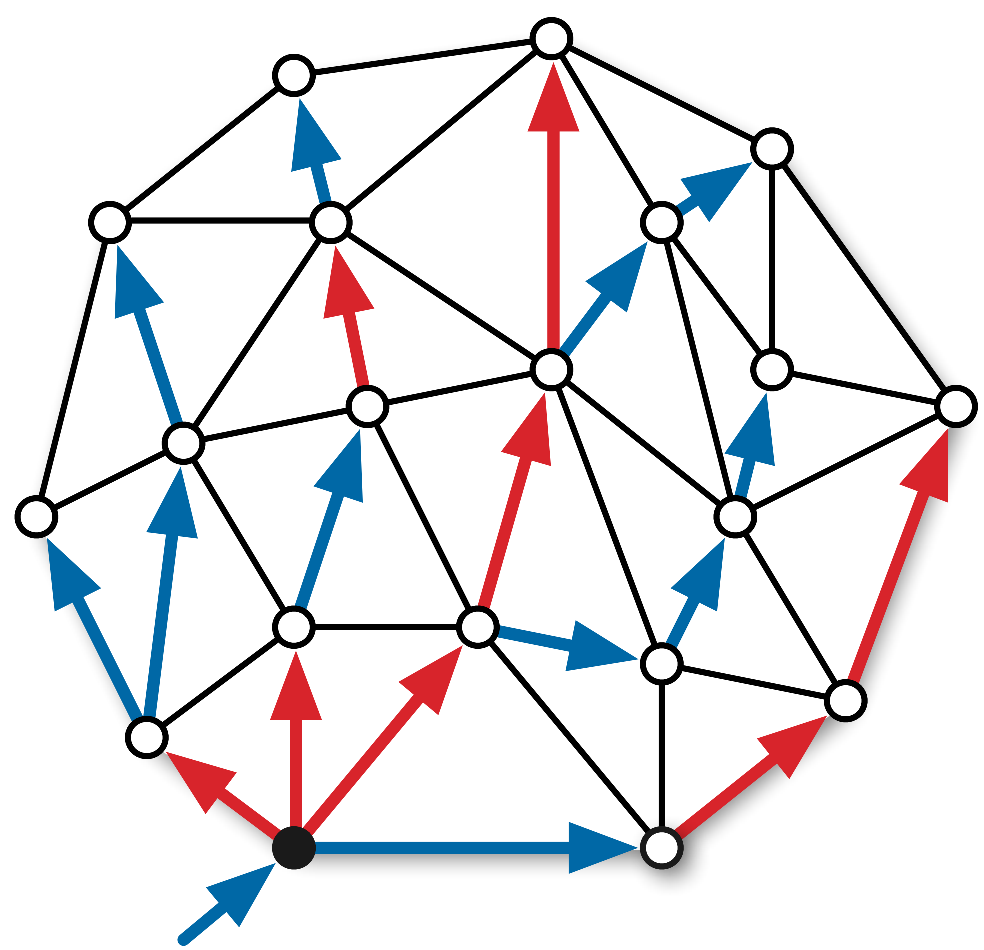
- In fact, we only need to compute the *first* shortest-path tree, followed by *changes* from each tree to the next.



Planar MSSP

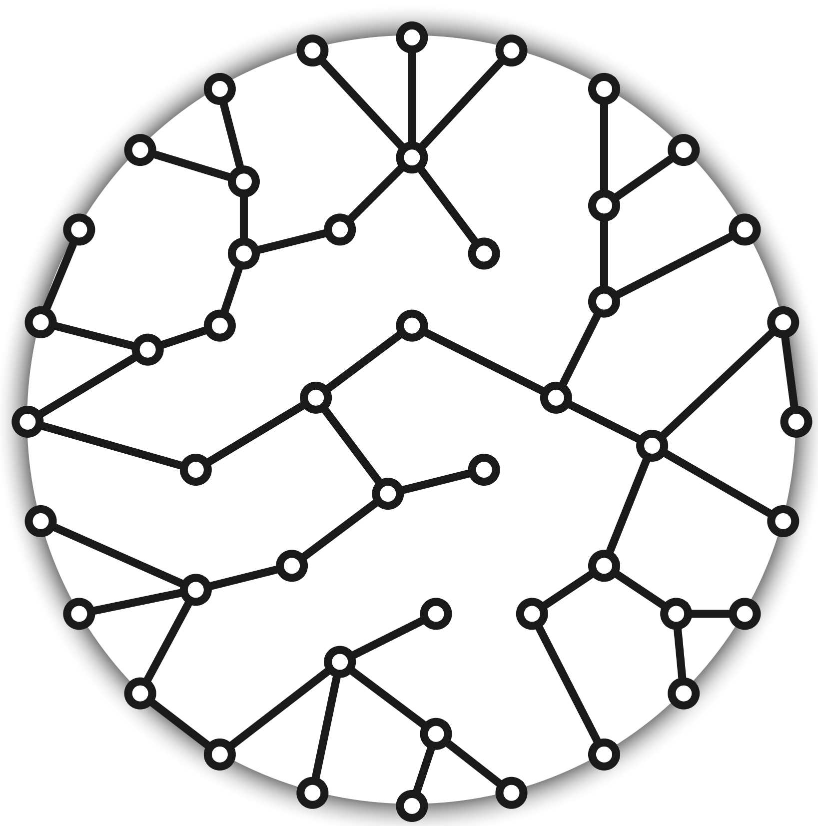
[Klein 2005]

- In fact, we only need to compute the *first* shortest-path tree, followed by *changes* from each tree to the next.



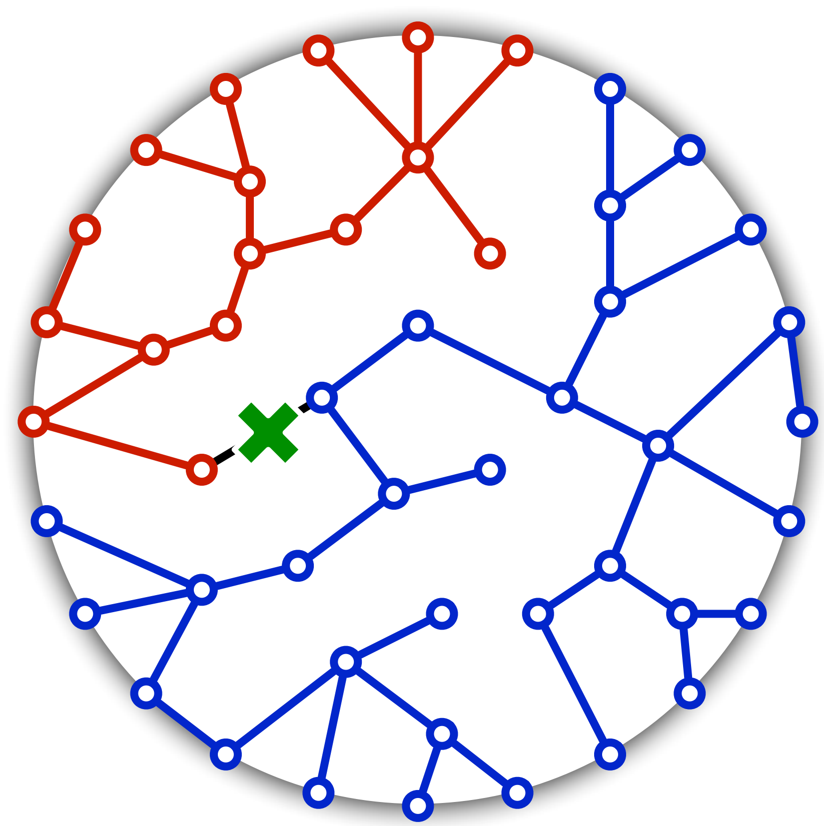
The disk-tree lemma

- Let T be any tree embedded on a closed disk. Vertices of T subdivide the boundary of the disk into intervals.
- Deleting any edge splits T into two subtrees R and B.
- At most two intervals have one end in R and the other in B.



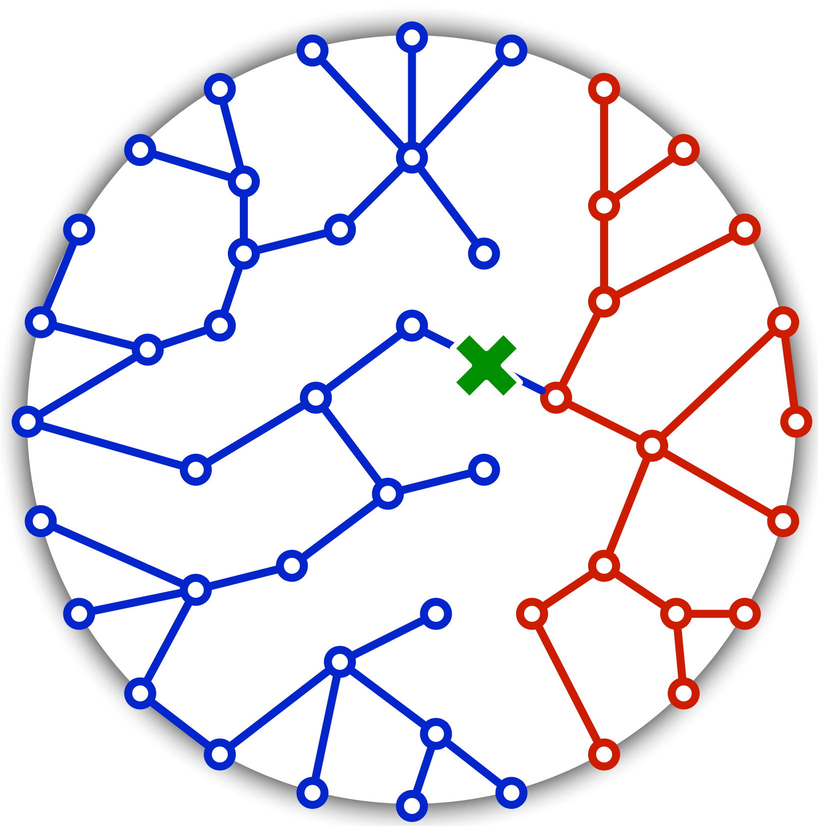
The disk-tree lemma

- Let *T* be any tree embedded on a closed disk. Vertices of *T* subdivide the boundary of the disk into intervals.
- Deleting any edge splits *T* into two subtrees *R* and *B*.
- At most two intervals have one end in *R* and the other in *B*.



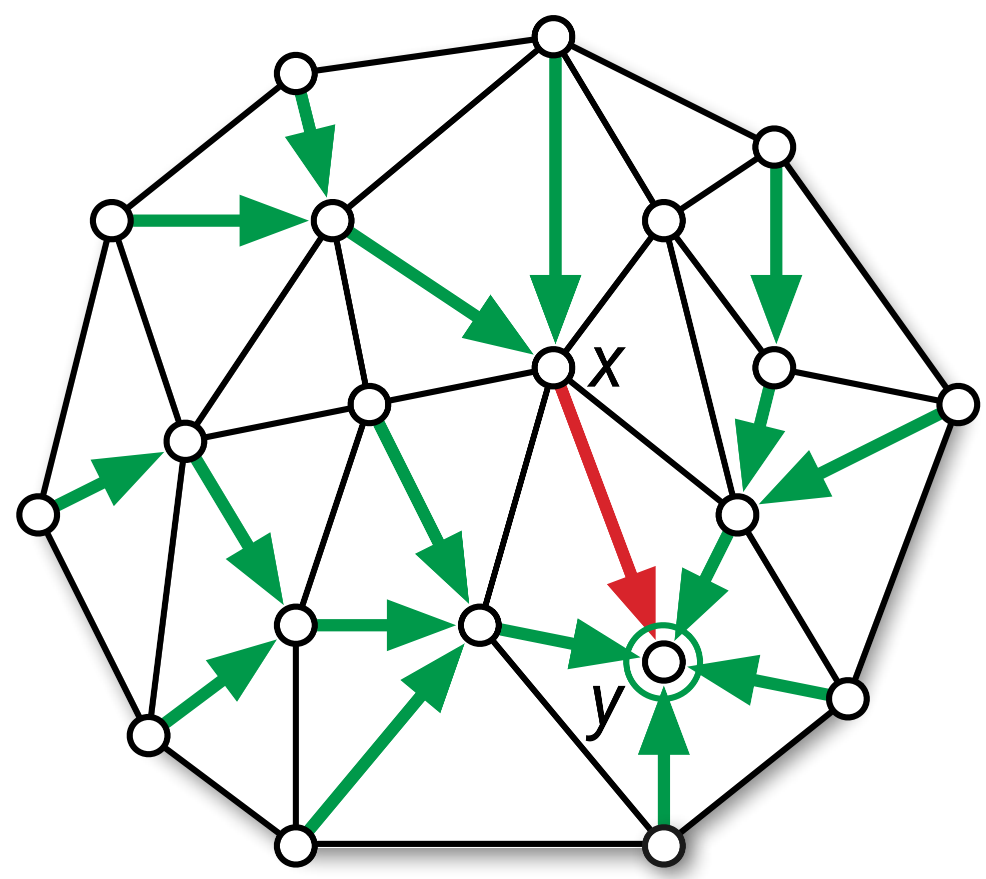
The disk-tree lemma

- Let T be any tree embedded on a closed disk. Vertices of T subdivide the boundary of the disk into intervals.
- Deleting any edge splits T into two subtrees R and B.
- At most two intervals have one end in R and the other in B.



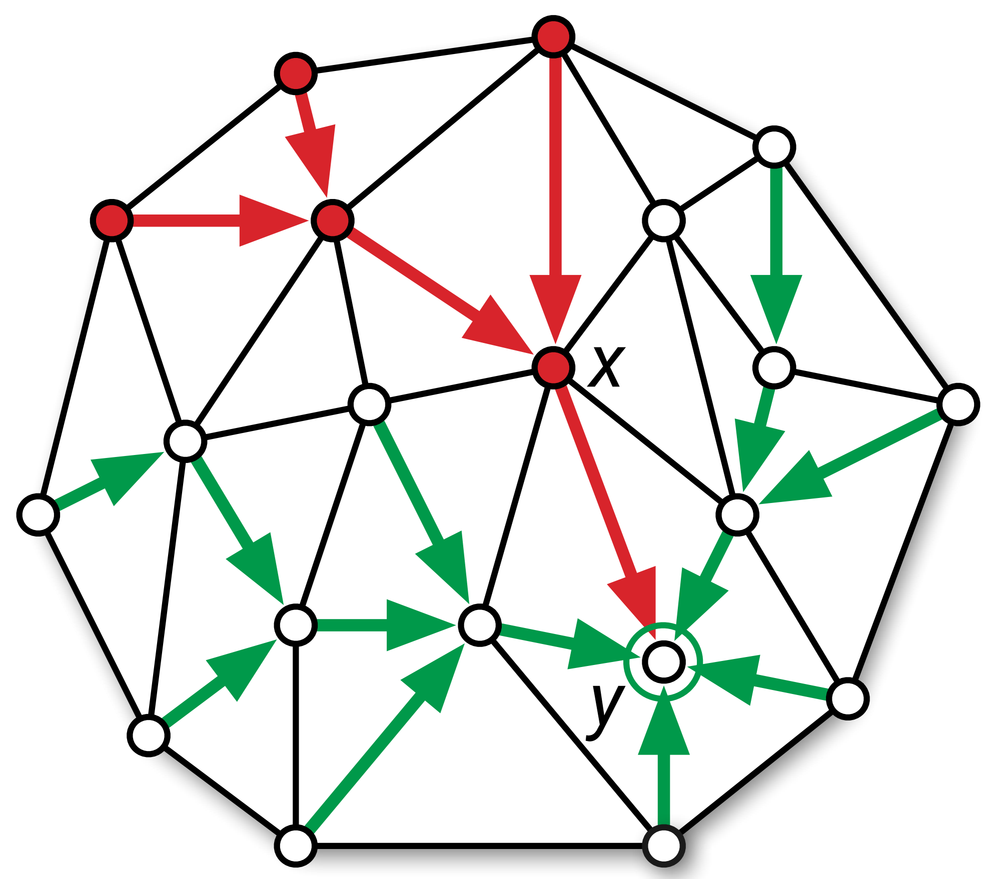
Number of pivots

- Each directed edge $x \to y$ pivots in *at most once*.
  - Consider the tree of shortest paths *ending at* y.



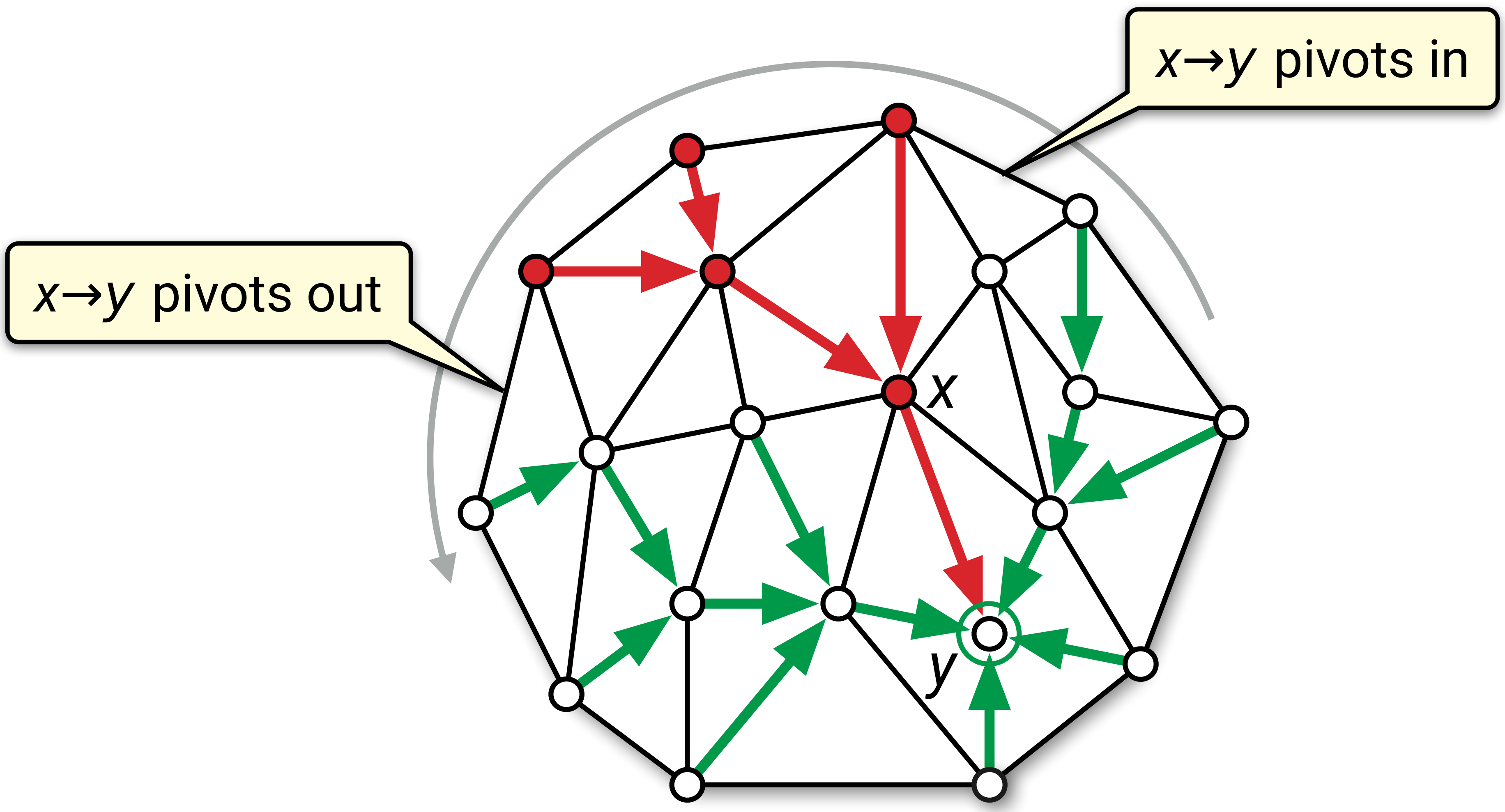
Number of pivots

- Each directed edge *x*→*y* pivots in *at most once*.
  - Consider the tree of shortest paths *ending at y*.



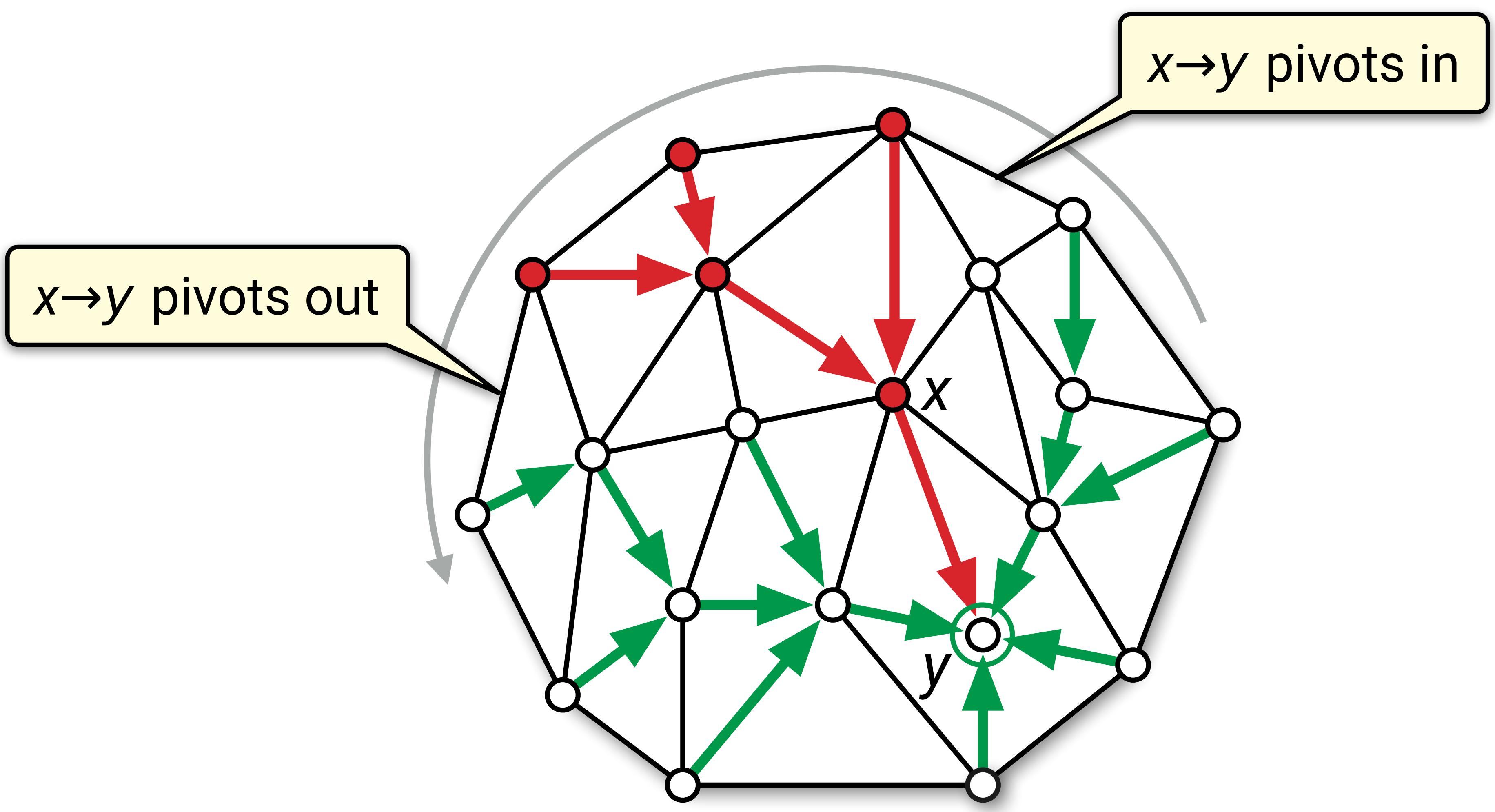
Number of pivots

- Each directed edge $x{\to}y$ pivots in *at most once*.
  - Consider the tree of shortest paths *ending at* y.



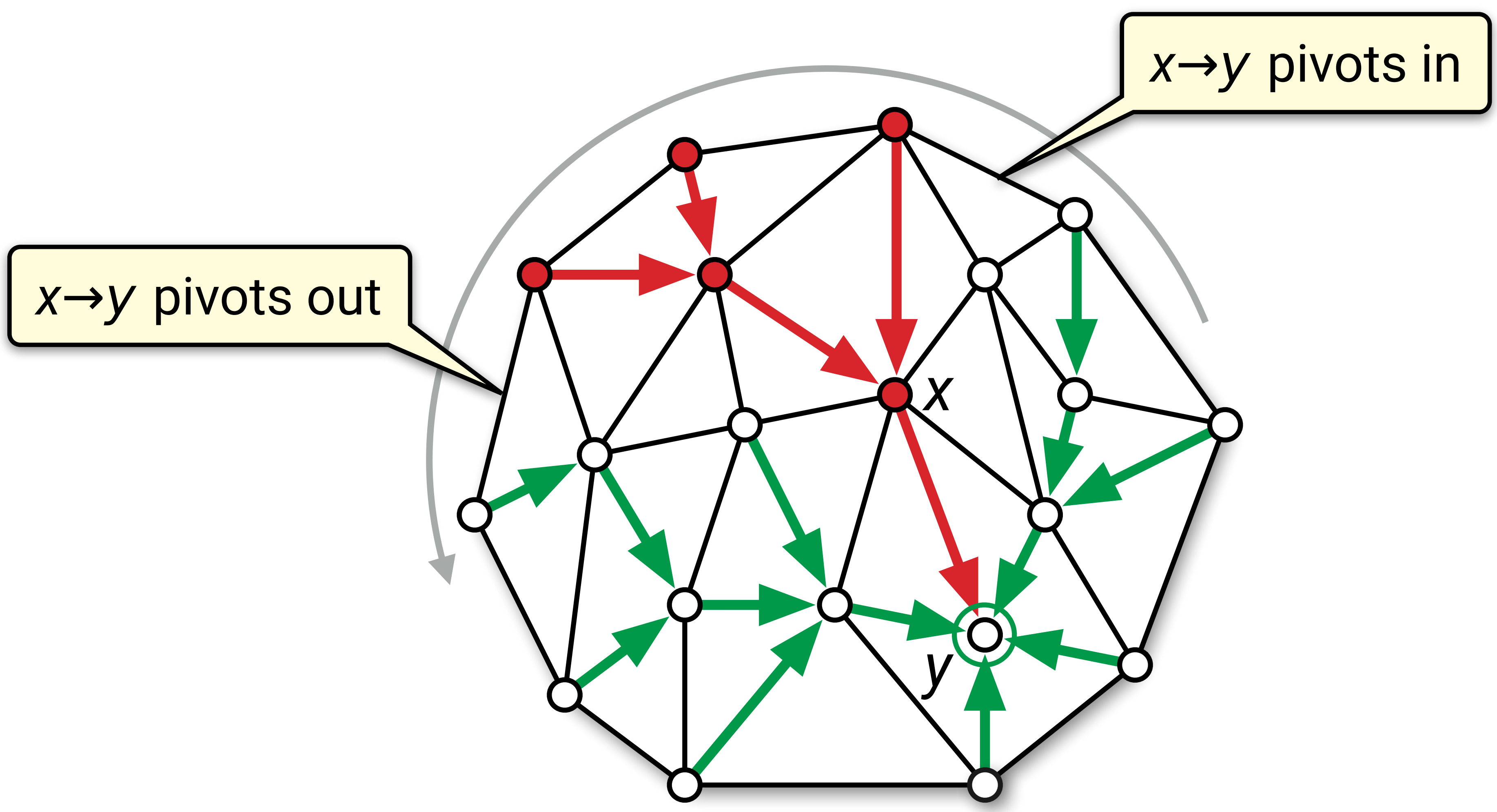
Number of pivots

- So the overall number of pivots is only O(n)!



Number of pivots

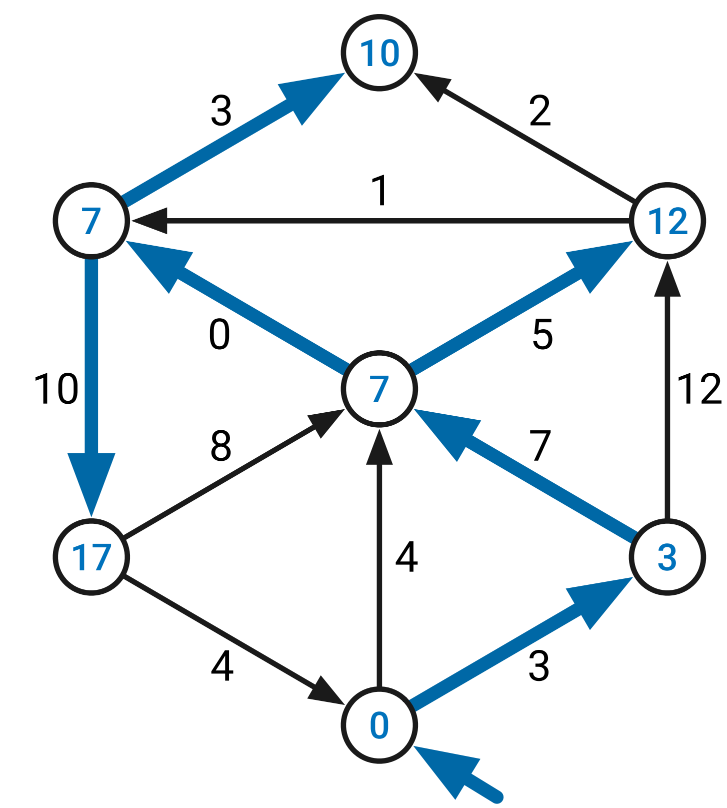
- So the overall number of pivots is only O(n)!
- But how do we find these pivots quickly?



How shortest paths work

[Ford 1956]

- Input:
  - Directed graph G = (V, E)
  - length $\ell(u{\rightarrow}v)$ for each edge $u{\rightarrow}v$
  - *A source* vertex *s*.

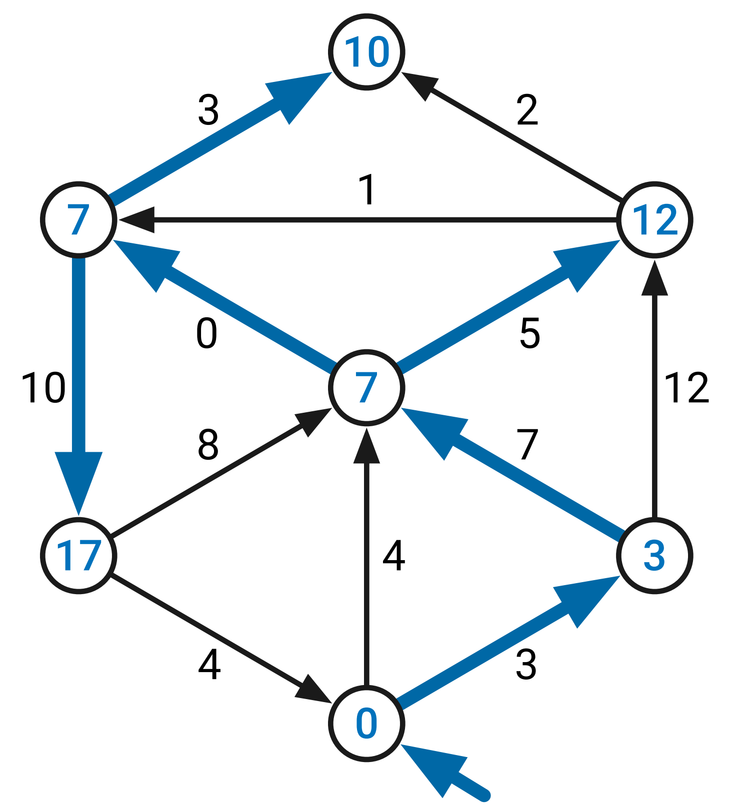


- Each vertex *v* maintains two values:
  - *dist*(*v*) is the length of some path from *s* to *v*
  - *pred*(*v*) is the next-to-last vertex of that path from *s* to *v*.

How shortest paths work

[Ford 1956]

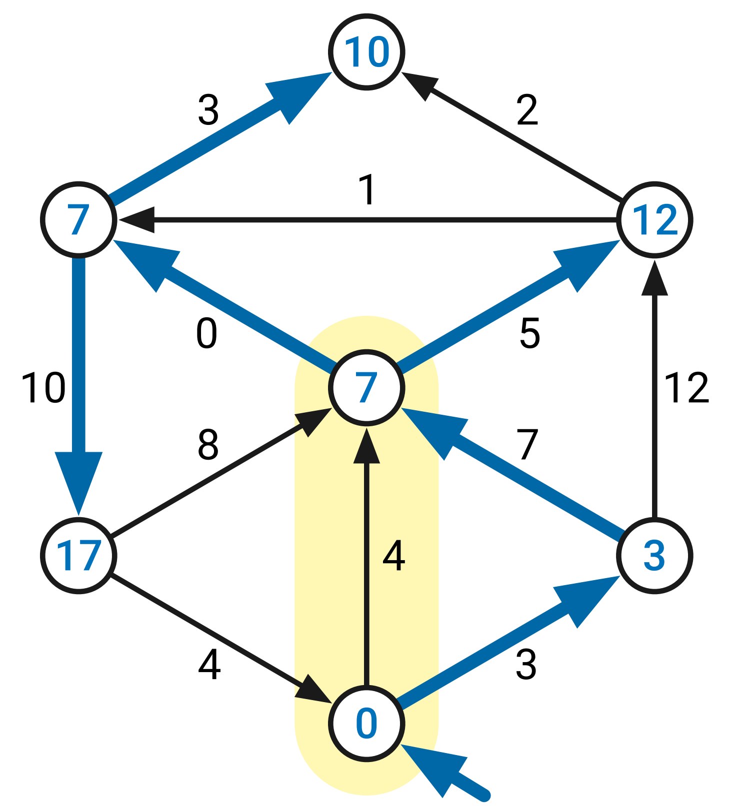
- Edge *u*→*v* is *tense* iff *dist*(*v*) ≥ *dist*(*u*) + *ℓ*(*u*→*v*).



How shortest paths work

[Ford 1956]

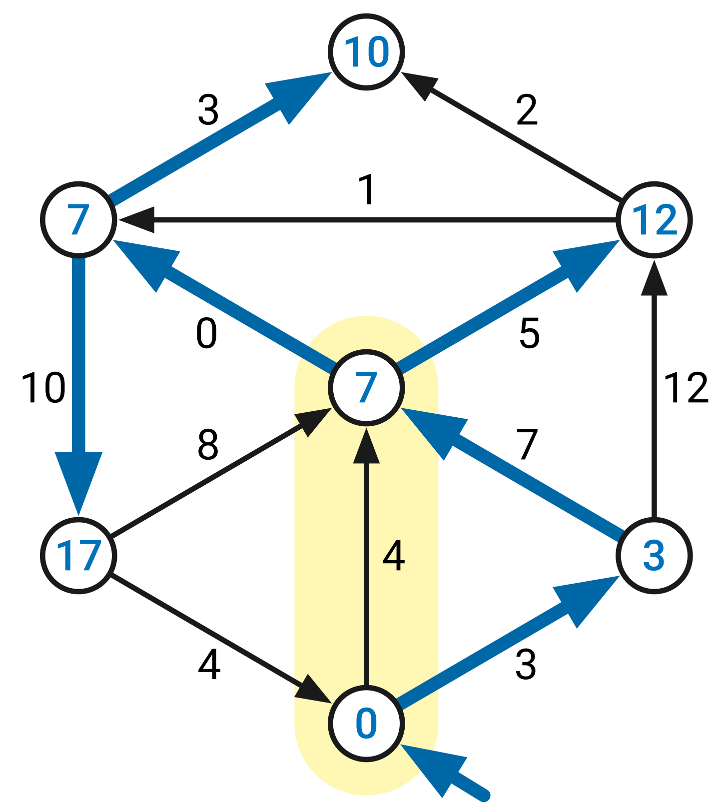
- Edge $u{\to}v$ is *tense* iff $dist(v) \geq dist(u) + \ell(u{\to}v)$.



How shortest paths work

[Ford 1956]

- Edge $u \to v$ is *tense* iff $dist(v) \geq dist(u) + \ell(u \to v)$.

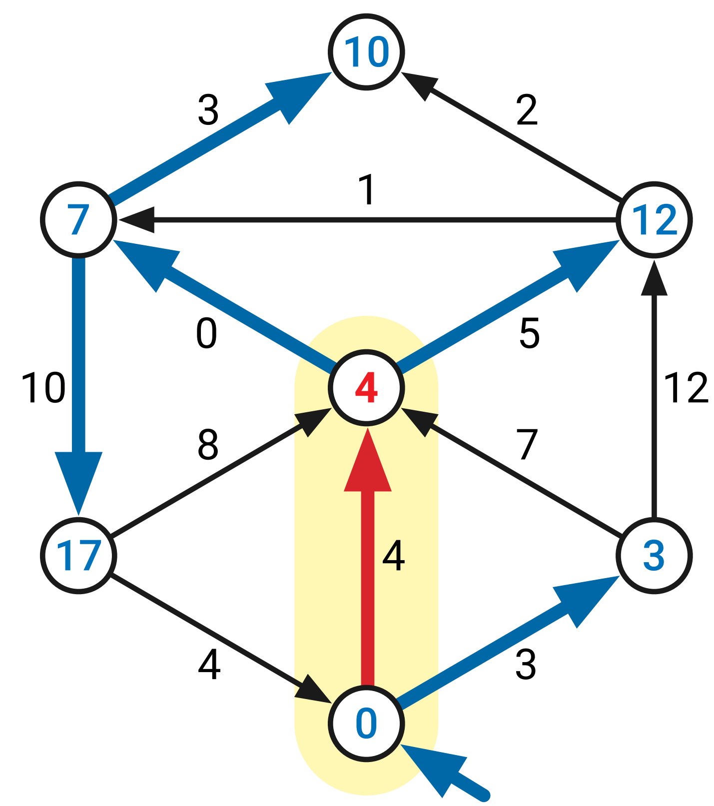


- To *relax* $u \to v$, set $dist(v) = dist(u) + \ell(u \to v)$ and $pred(v) = u$

How shortest paths work

[Ford 1956]

- Edge $u \to v$ is *tense* iff $dist(v) \geq dist(u) + \ell(u \to v)$.

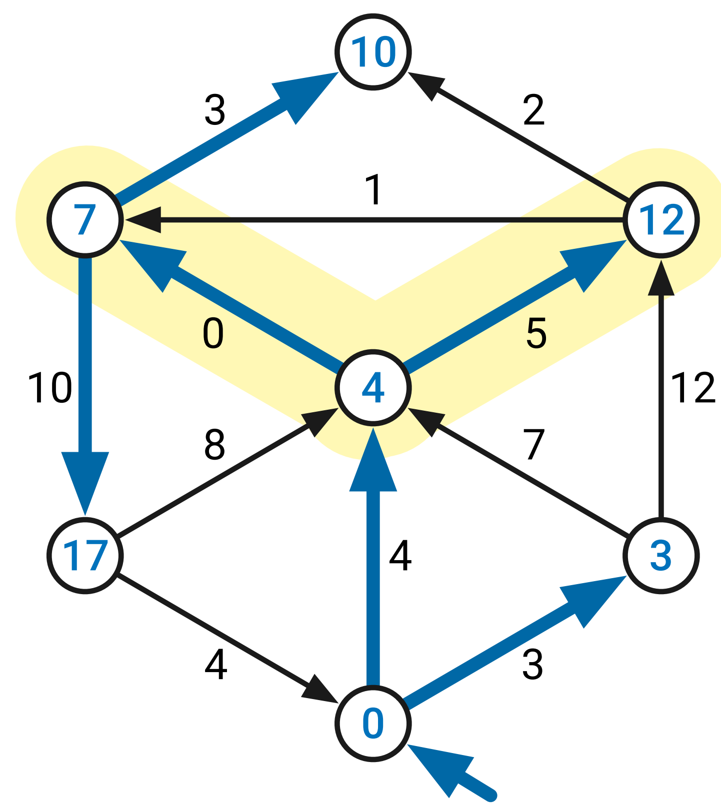


- To *relax* $u \to v$, set $dist(v) = dist(u) + \ell(u \to v)$ and $pred(v) = u$

How shortest paths work

[Ford 1956]

- Edge *u*→*v* is *tense* iff *dist*(*v*) ≥ *dist*(*u*) + *ℓ*(*u*→*v*).

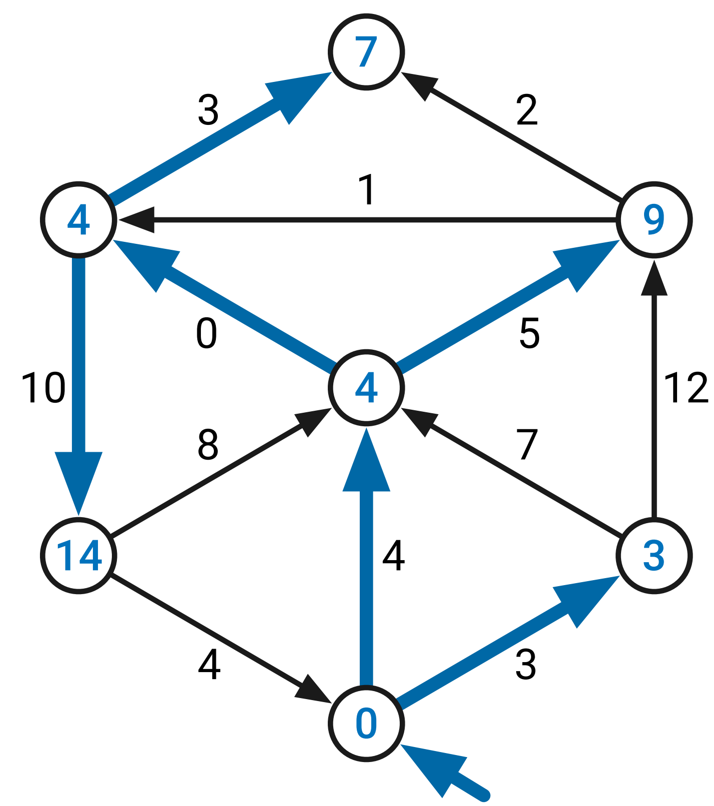


- To *relax* *u*→*v*, set *dist*(*v*) = *dist*(*u*) + *ℓ*(*u*→*v*) and *pred*(*v*) = *u*

How shortest paths work

[Ford 1956]

- Edge $u \rightarrow v$ is *tense* iff $dist(v) \geq dist(u) + \ell(u \rightarrow v)$.



- If *no* edges are tense, then $dist(v)$ is the length of the *shortest* path from s to v, for every vertex v.

Back to MSSP

[Cabello Chambers Erickson 2013]

- Maintain the shortest path tree rooted at a point s that is moving *continuously* around the outer face.
- Also maintain the *slack* of each edge $u \to v$:

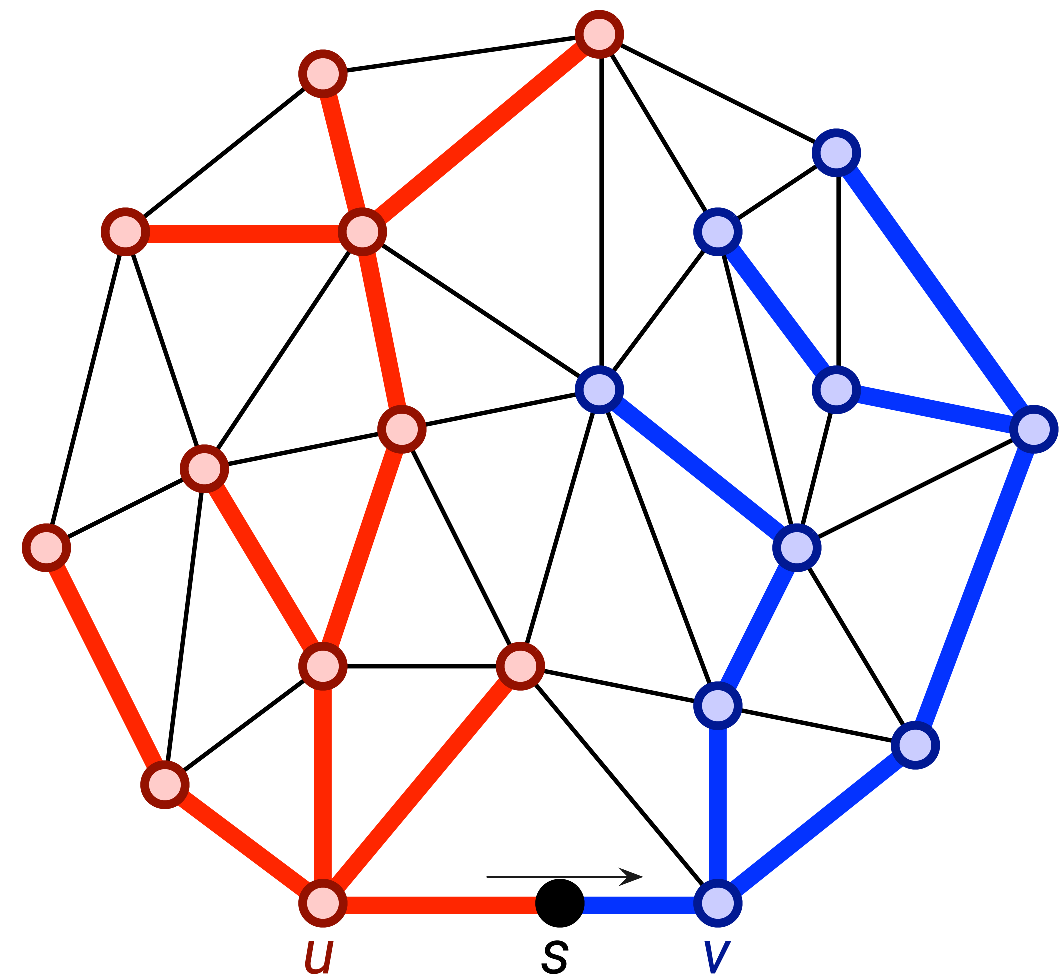
$$slack(u \to v) := dist(u) + \ell(u \to v) - dist(v)$$

- Distances and slacks change continuously with s, but in a controlled manner.
- The shortest path tree is correct as long as $slack(u \to v) > 0$ for every edge $u \to v$.

Distance and slack changes

[Doppler 1842]
[Fizeau 1848]

- Red: dist growing
- Blue: dist shrinking

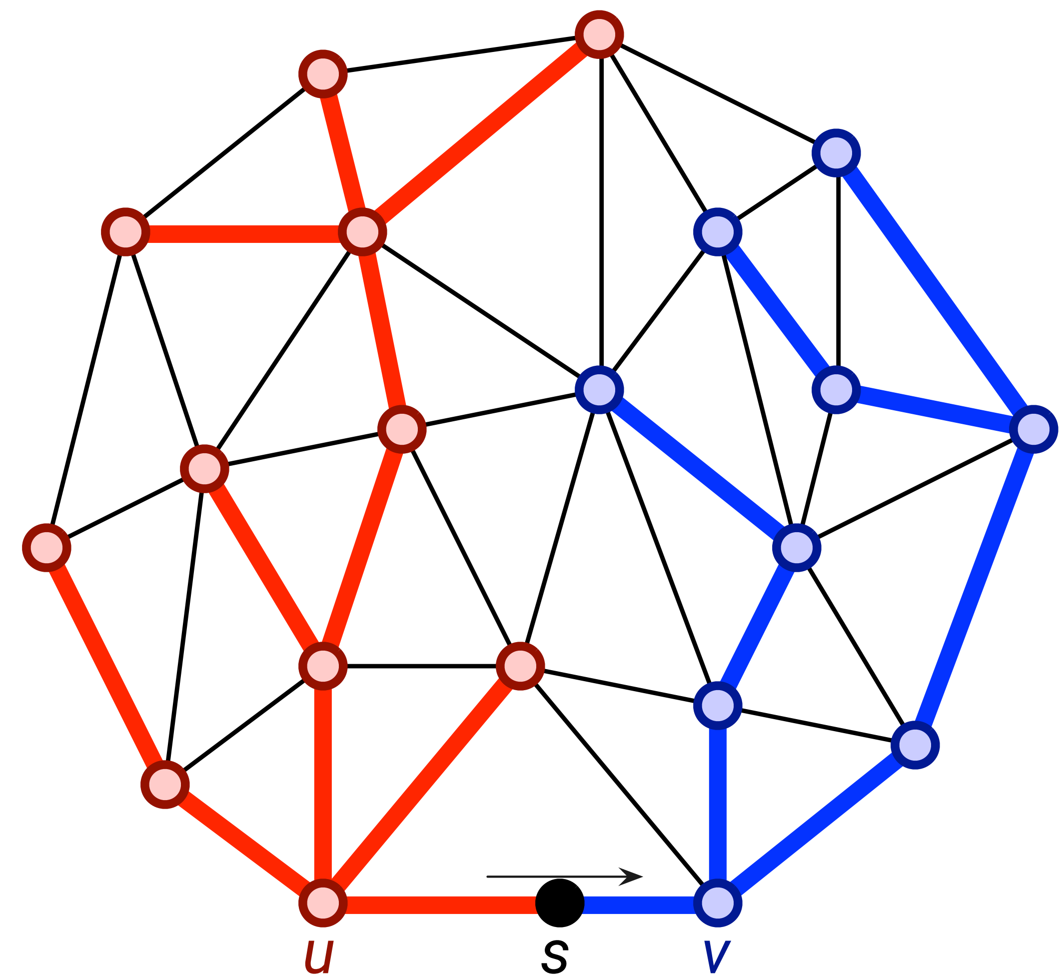


Distance and slack changes

[Doppler 1842]
[Fizeau 1848]

- Red: dist growing
- Blue: dist shrinking

- Red→red: slack constant
- Blue→blue: slack constant
- Red→blue: slack growing
- Blue→red: slack shrinking

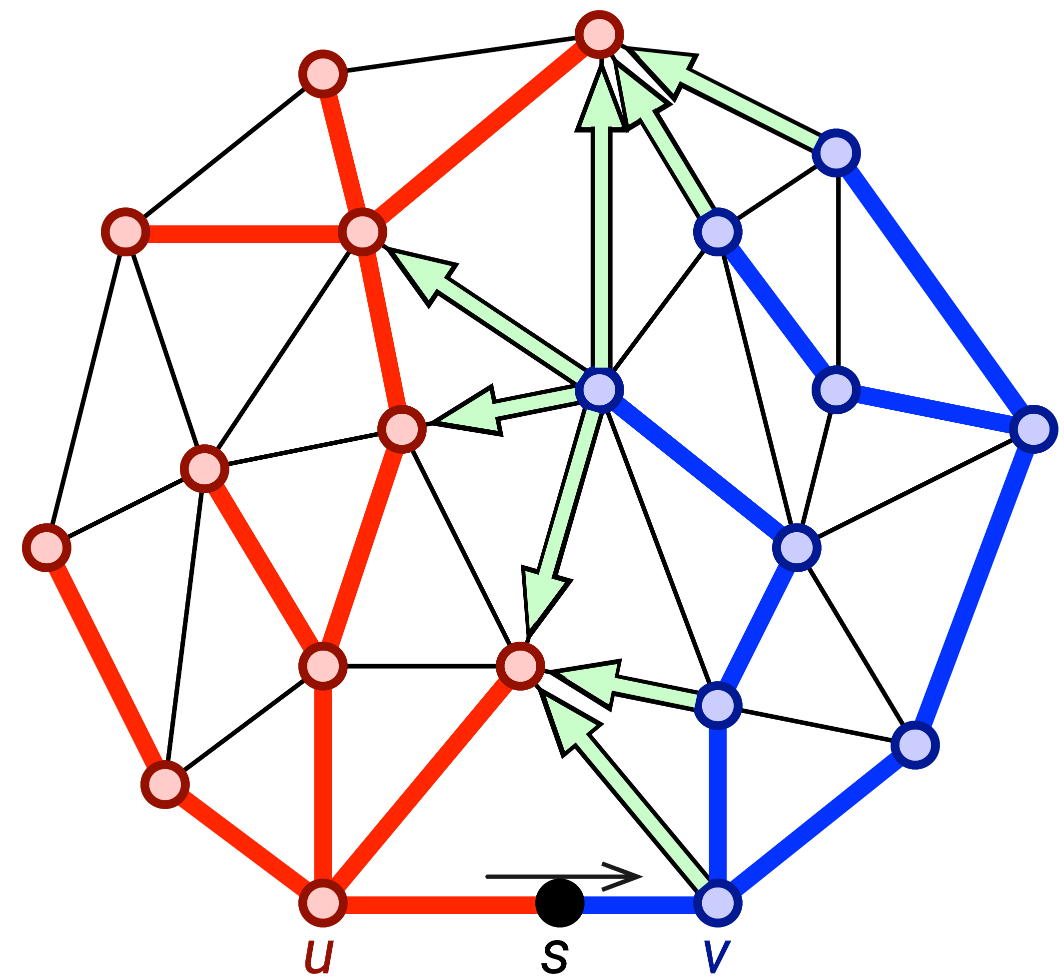


Distance and slack changes

[Doppler 1842]
[Fizeau 1848]

- Red: dist growing
- Blue: dist shrinking

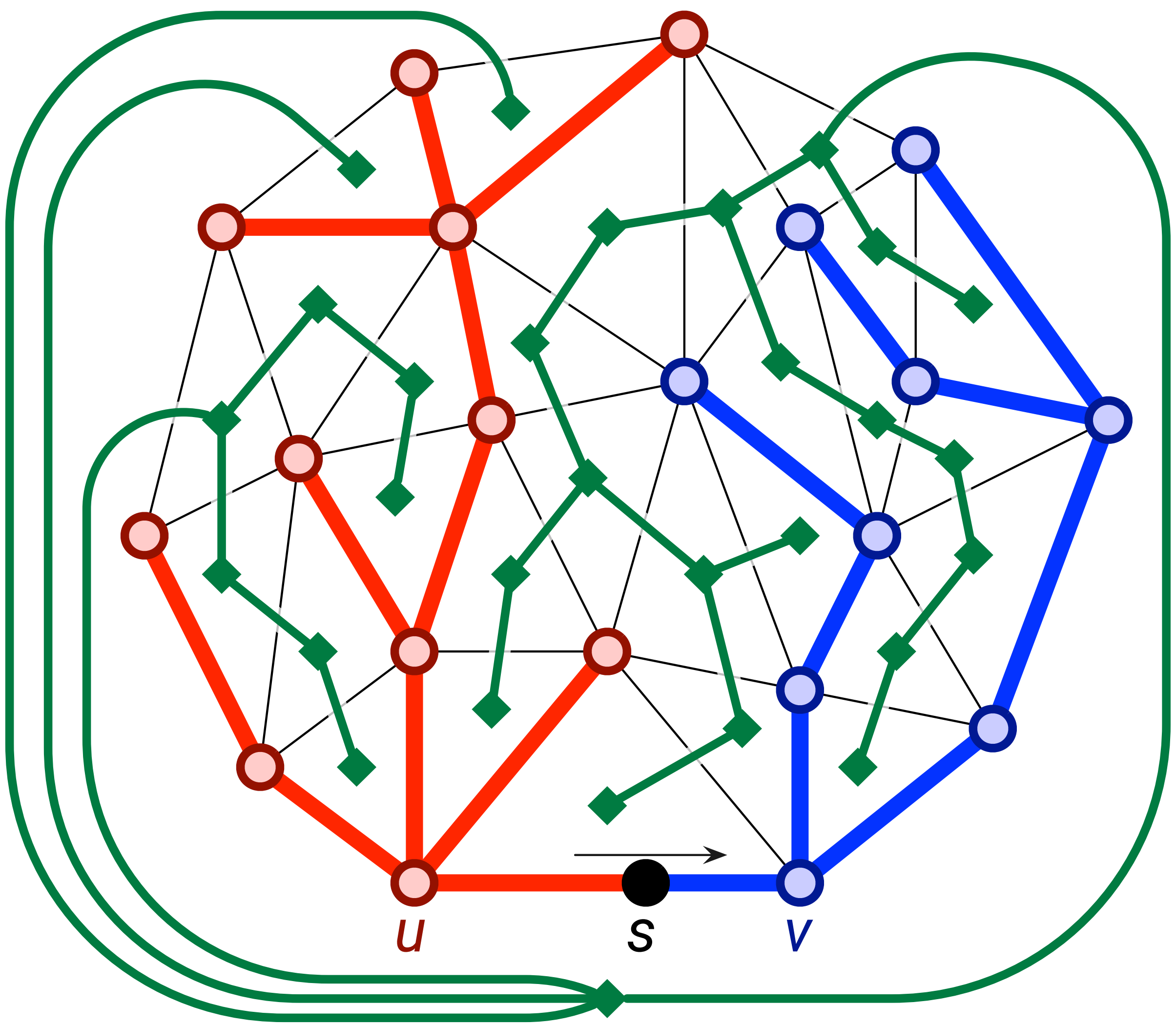
- Red→red: slack constant
- Blue→blue: slack constant
- Red→blue: slack growing
- Blue→red: slack shrinking
  - *active* edges



Tree-cotree decomposition

[von Staudt 1847]
[Whitney 1932]
[Dehn 1936]

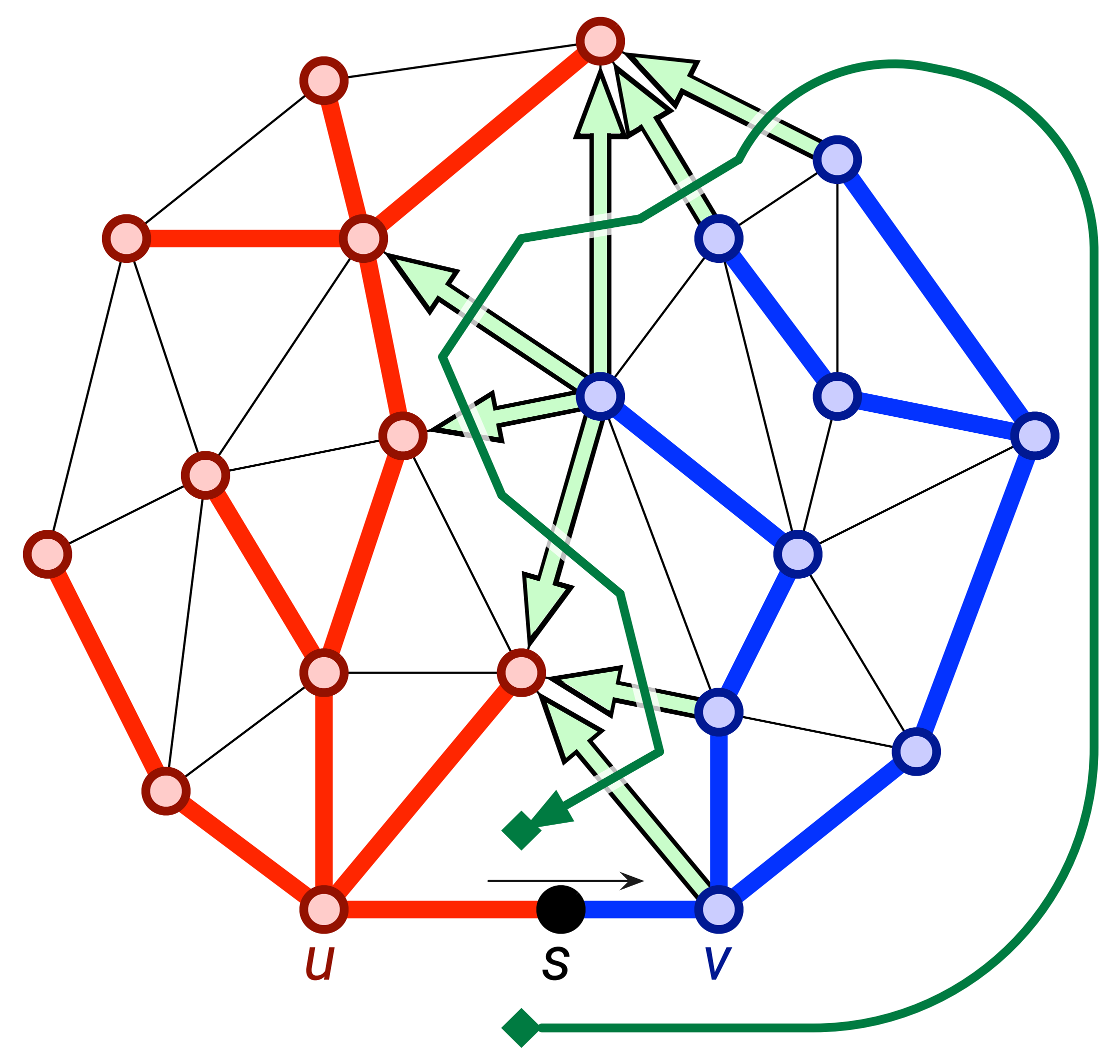
- Complementary dual spanning tree $C^* = (G \backslash T)^*$
- Red and blue subtrees are separated by a path in C^*
- Active edges are dual to edges in this path.



Tree-cotree decomposition

[von Staudt 1847]
[Whitney 1932]
[Dehn 1936]

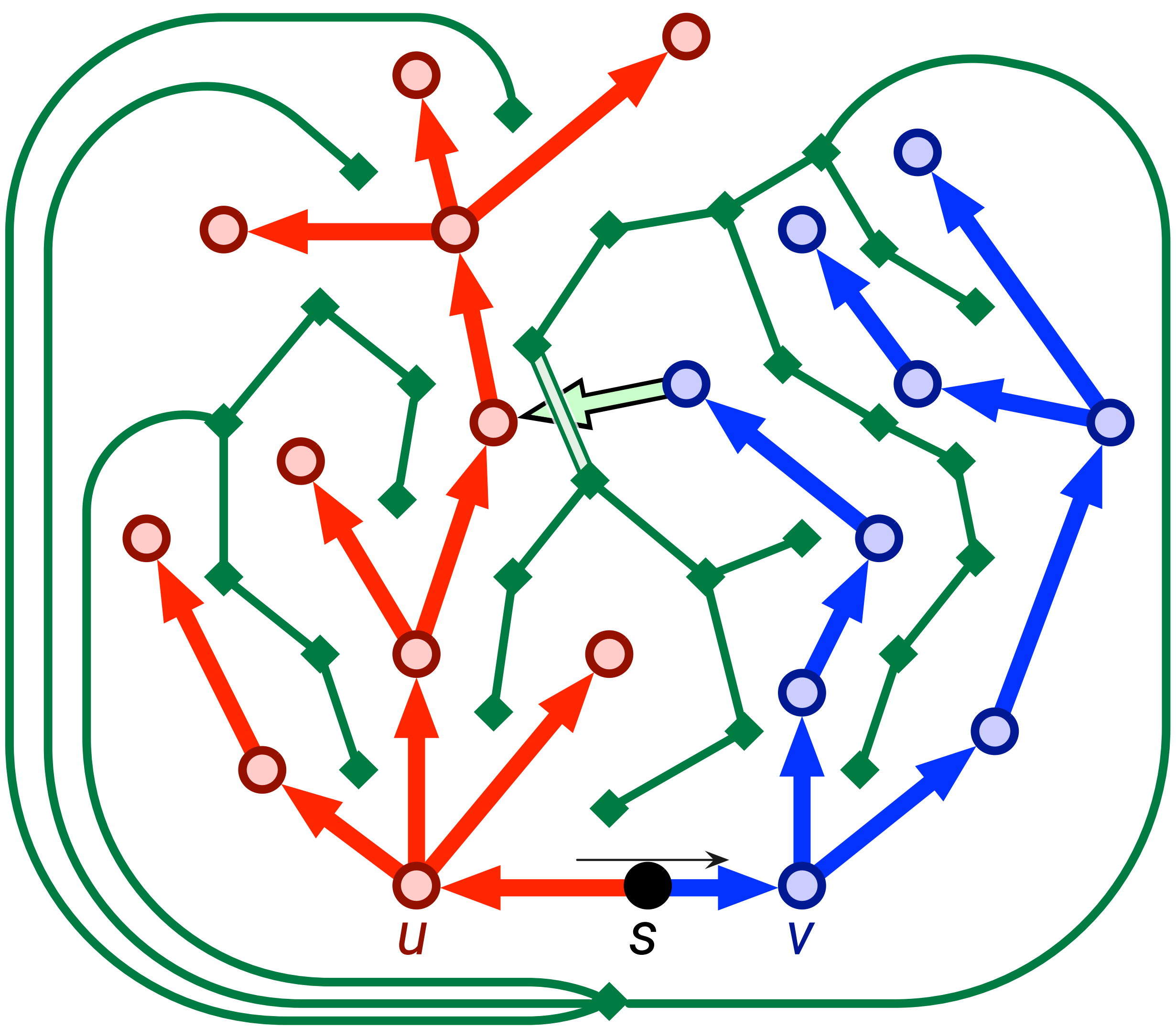
- Complementary dual spanning tree $C^* = (G\backslash T)^*$
- Red and blue subtrees are separated by a path in C^*
- Active edges are dual to edges in this path.



Pivot

[Ford 1956]

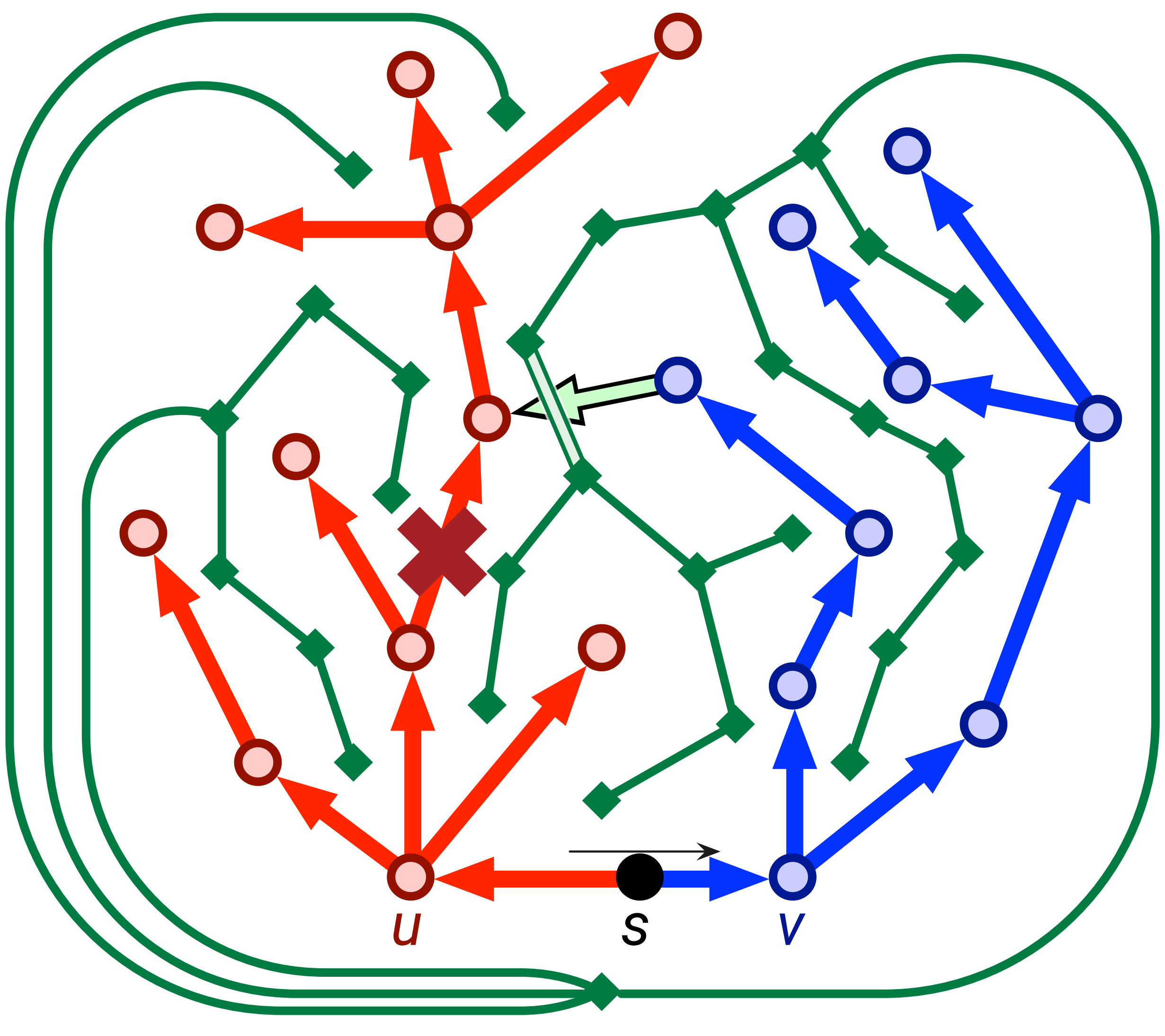
- When *slack*(*u*→*v*) becomes 0, relax *u*→*v*
  - Delete *pred*(*v*)→*v* from *T*
  - Insert *u*→*v* into *T*.
  - Delete (*u*→*v*)* from *C**.
  - Insert (*pred*(*v*)→*v*)* into *C**
  - Set *pred*(*u*) := *v*



Pivot

[Ford 1956]

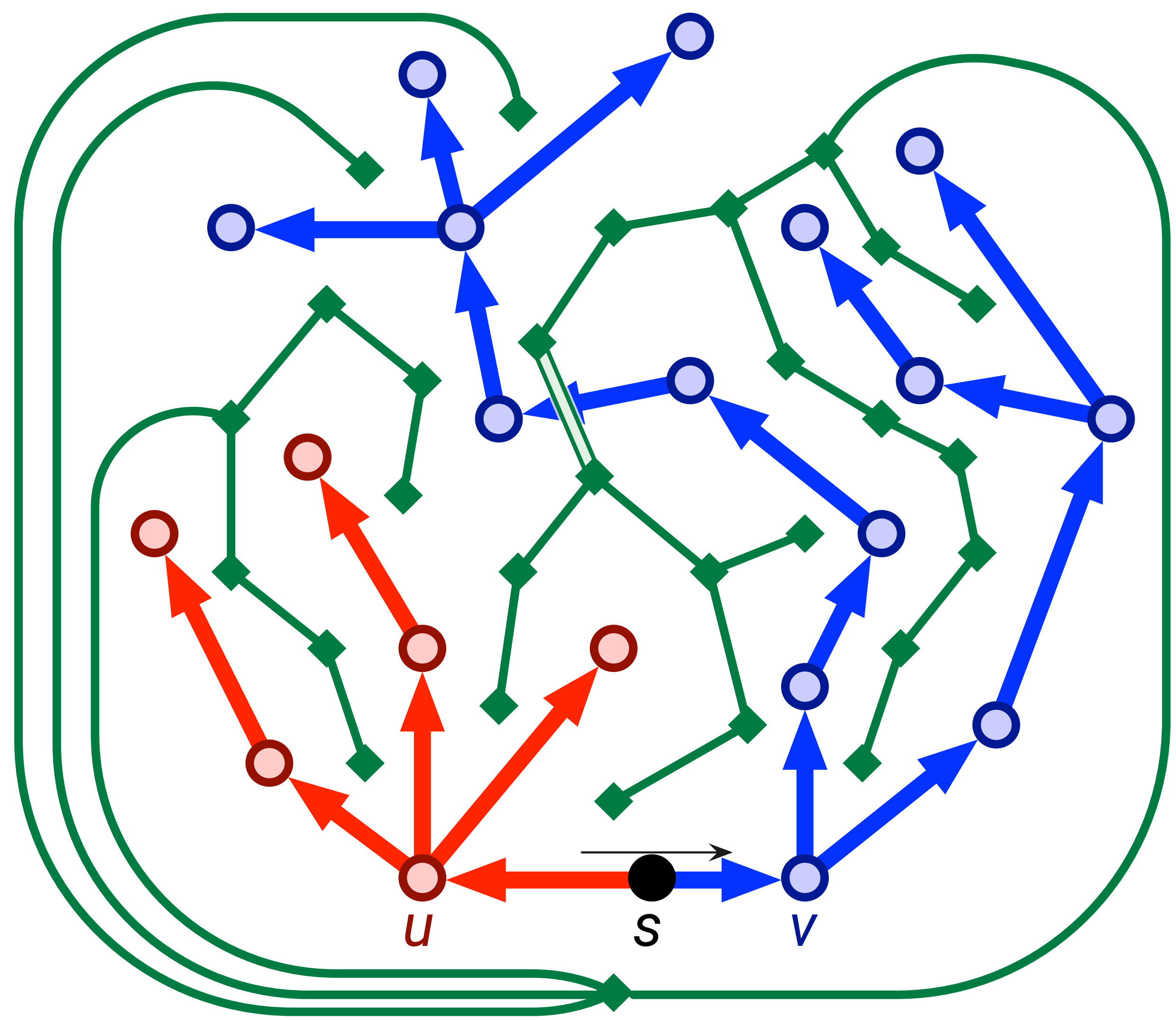
- When *slack*(*u*→*v*) becomes 0, relax *u*→*v*
  - Delete *pred*(*v*)→*v* from *T*
  - Insert *u*→*v* into *T*.
  - Delete (*u*→*v*)* from *C**.
  - Insert (*pred*(*v*)→*v*)* into *C**
  - Set *pred*(*u*) := *v*



Pivot

[Ford 1956]

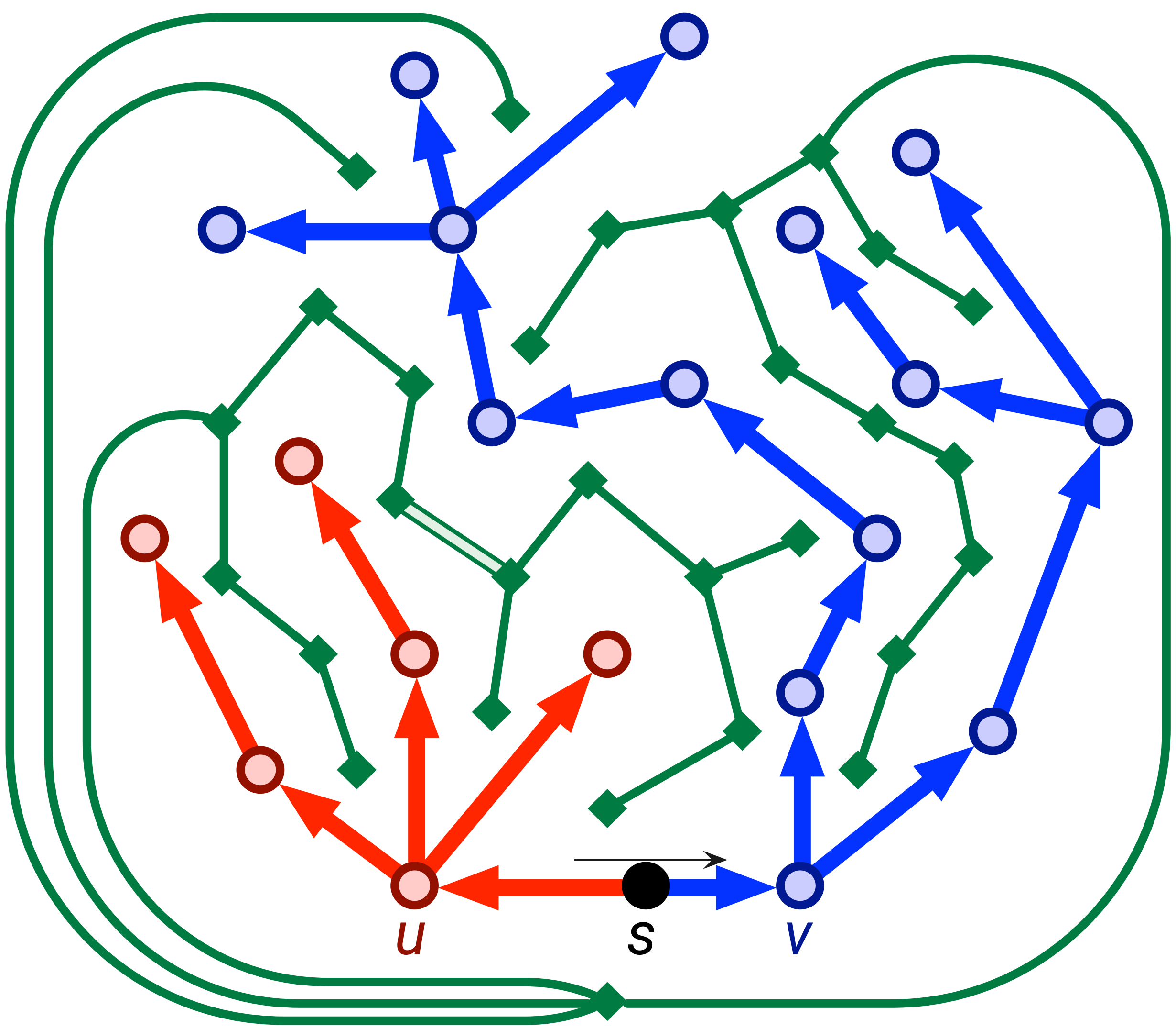
- When *slack*(*u*→*v*) becomes 0, relax *u*→*v*
  - Delete *pred*(*v*)→*v* from *T*
  - Insert *u*→*v* into *T*.
  - Delete (*u*→*v*)* from *C**.
  - Insert (*pred*(*v*)→*v*)* into *C**
  - Set *pred*(*u*) := *v*



Pivot

[Ford 1956]

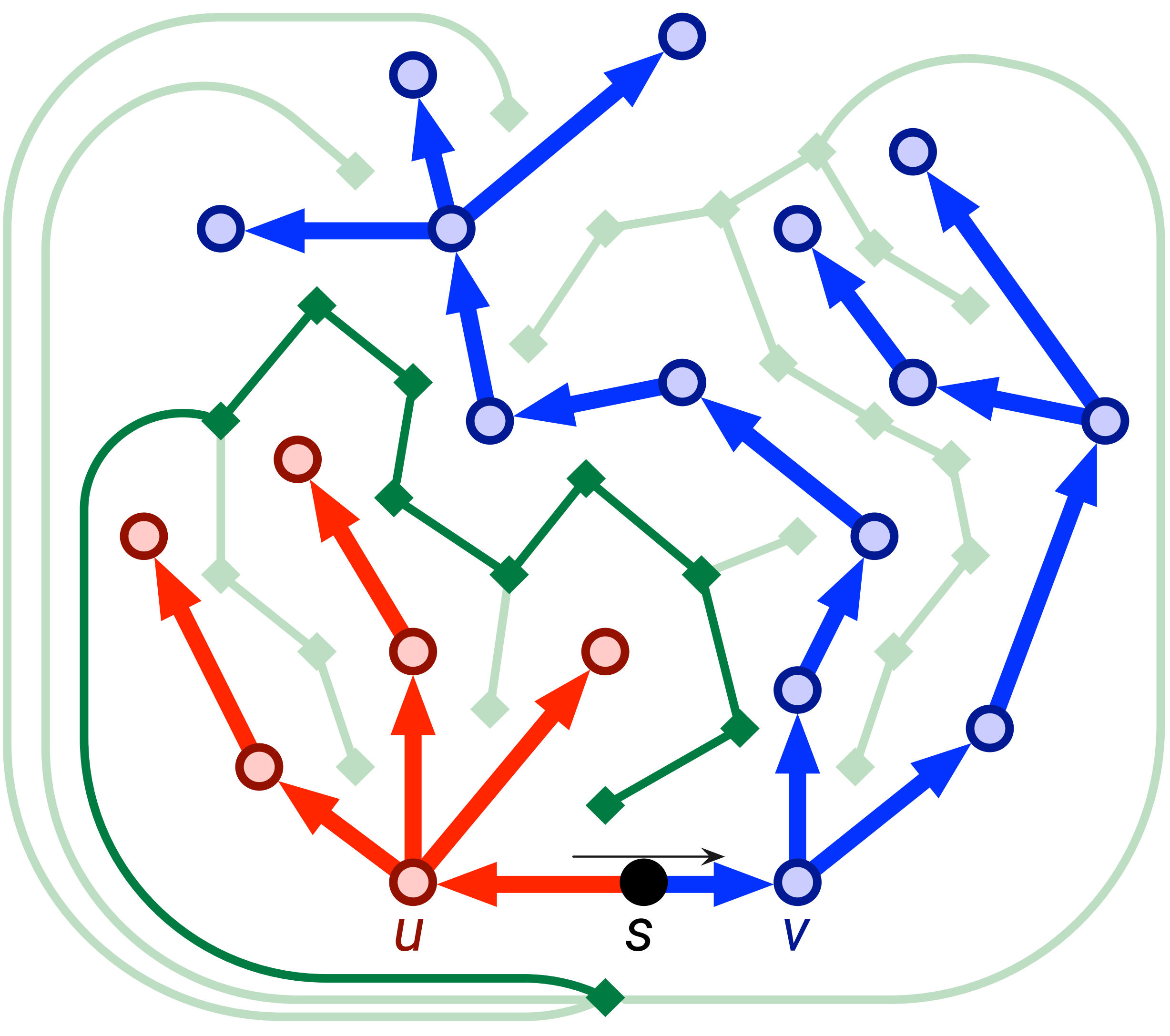
- When *slack*(*u*→*v*) becomes 0, relax *u*→*v*
  - Delete *pred*(*v*)→*v* from *T*
  - Insert *u*→*v* into *T*.
  - Delete (*u*→*v*)* from *C**.
  - Insert (*pred*(*v*)→*v*)* into *C**
  - Set *pred*(*u*) := *v*



Pivot

[Ford 1956]

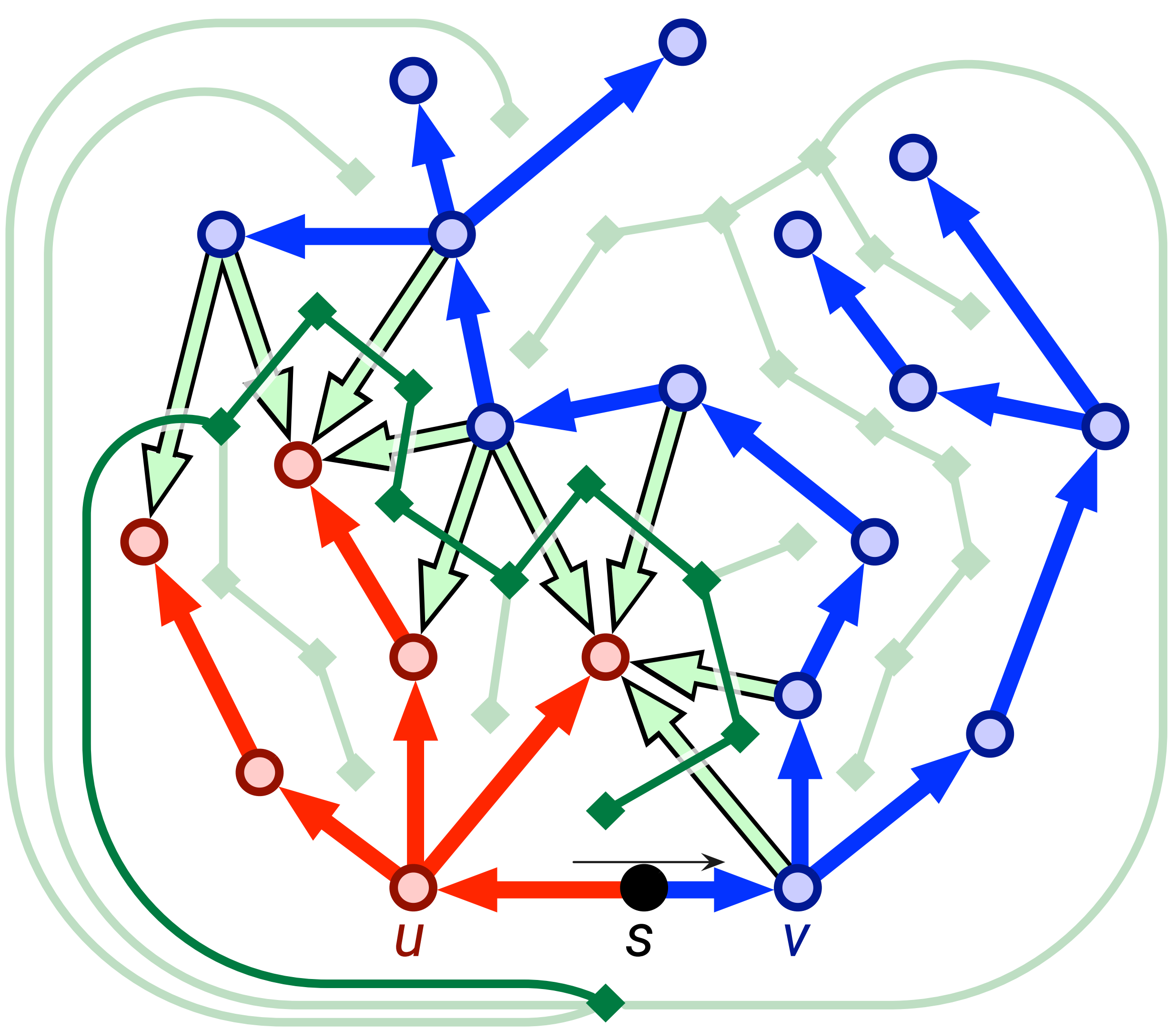
- When *slack*(*u*→*v*) becomes 0, relax *u*→*v*
  - Delete *pred*(*v*)→*v* from *T*
  - Insert *u*→*v* into *T*.
  - Delete (*u*→*v*)* from *C**.
  - Insert (*pred*(*v*)→*v*)* into *C**
  - Set *pred*(*u*) := *v*



Pivot

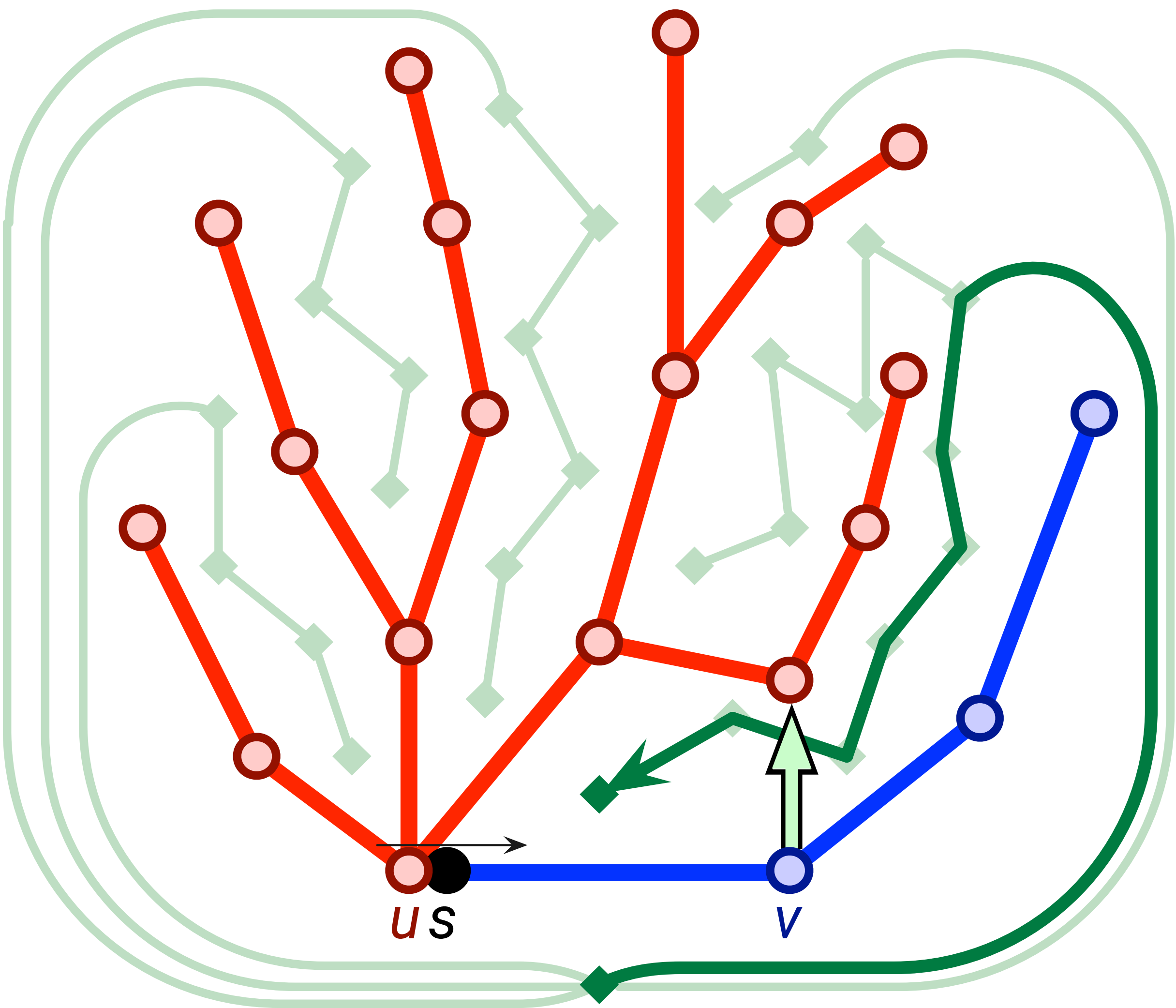
[Ford 1956]

- When *slack*(*u*→*v*) becomes 0, relax *u*→*v*
  - Delete *pred*(*v*)→*v* from *T*
  - Insert *u*→*v* into *T*.
  - Delete (*u*→*v*)* from *C**.
  - Insert (*pred*(*v*)→*v*)* into *C**
  - Set *pred*(*u*) := *v*



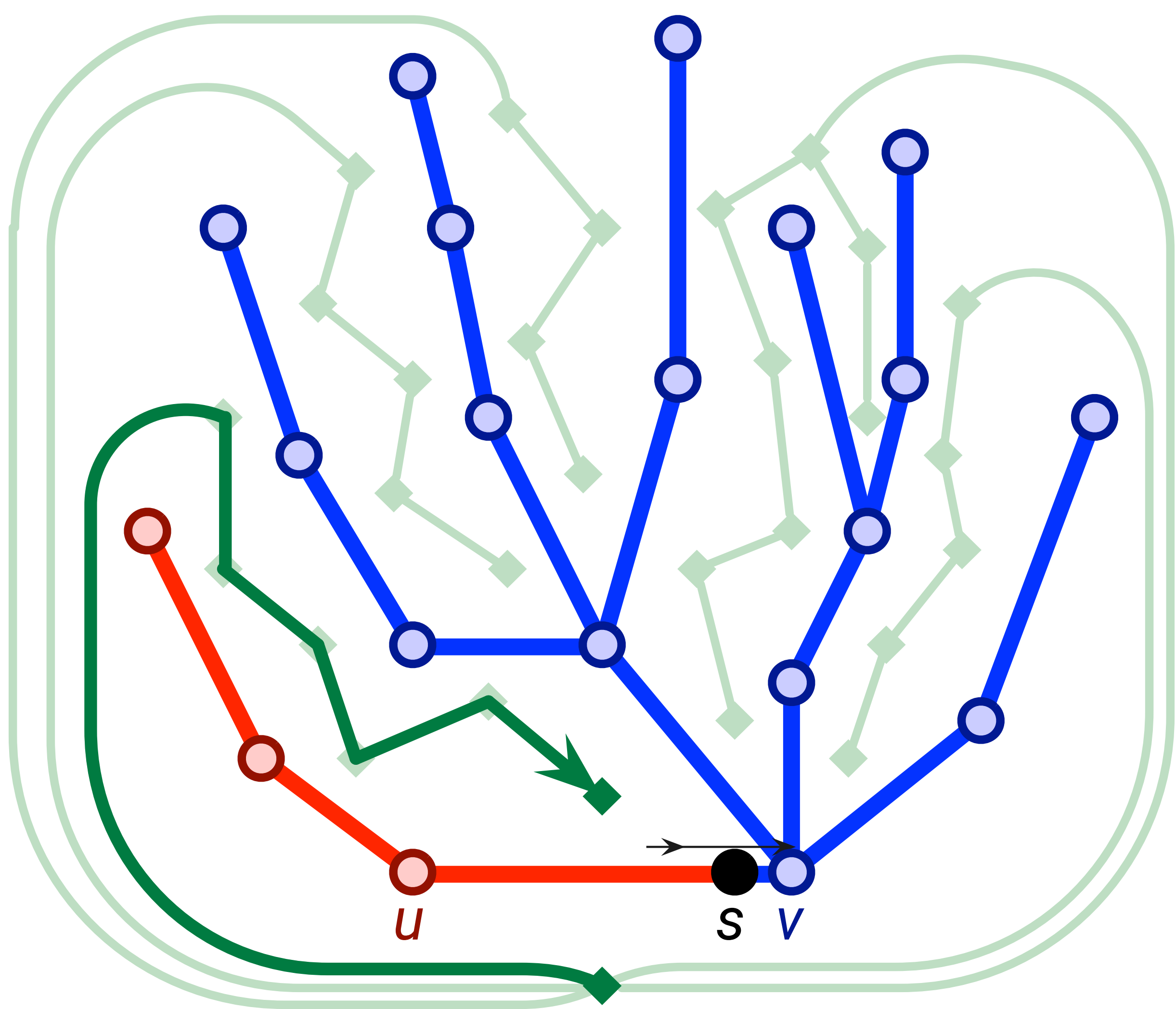
Pivots

- Vertices can only change from red to blue.
- So any edge that pivots into T stays in T.



Pivots

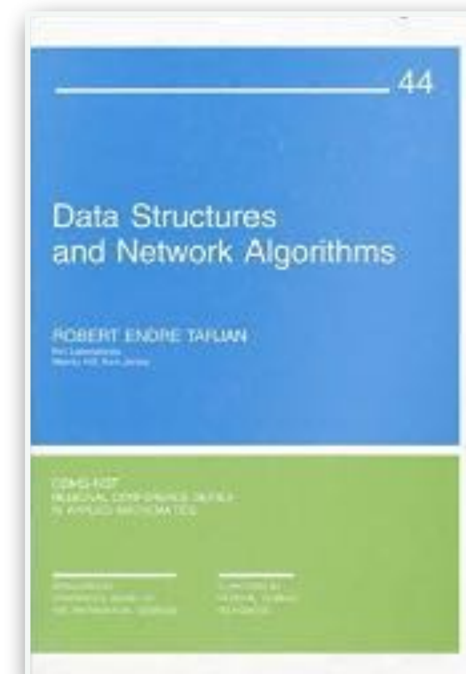
- Vertices can only change from red to blue.
- So any edge that pivots into T stays in T.



Fast implementation

[Sleator Tarjan 1983]
⋮
[Tarjan Werneck 2005]

- We maintain T and C^* in *dynamic forest* data structures that support the following operations in $O(\log n)$ amortized time:
  - Remove and insert edges:
    - Cut(uv), Link(u,v)
  - Maintain distances at vertices of T:
    - gETnODEvALUE(v), aDDsUBTREE(Δ, v)
  - Maintain slacks at edges of C^*:
    - gETdARTvALUE($u \rightarrow v$), aDDpATH(Δ, u, v), mINpATH(u, v)



- So we can identify and execute each pivot in $O(\log n)$ amortized time.

Planar MSSP summary

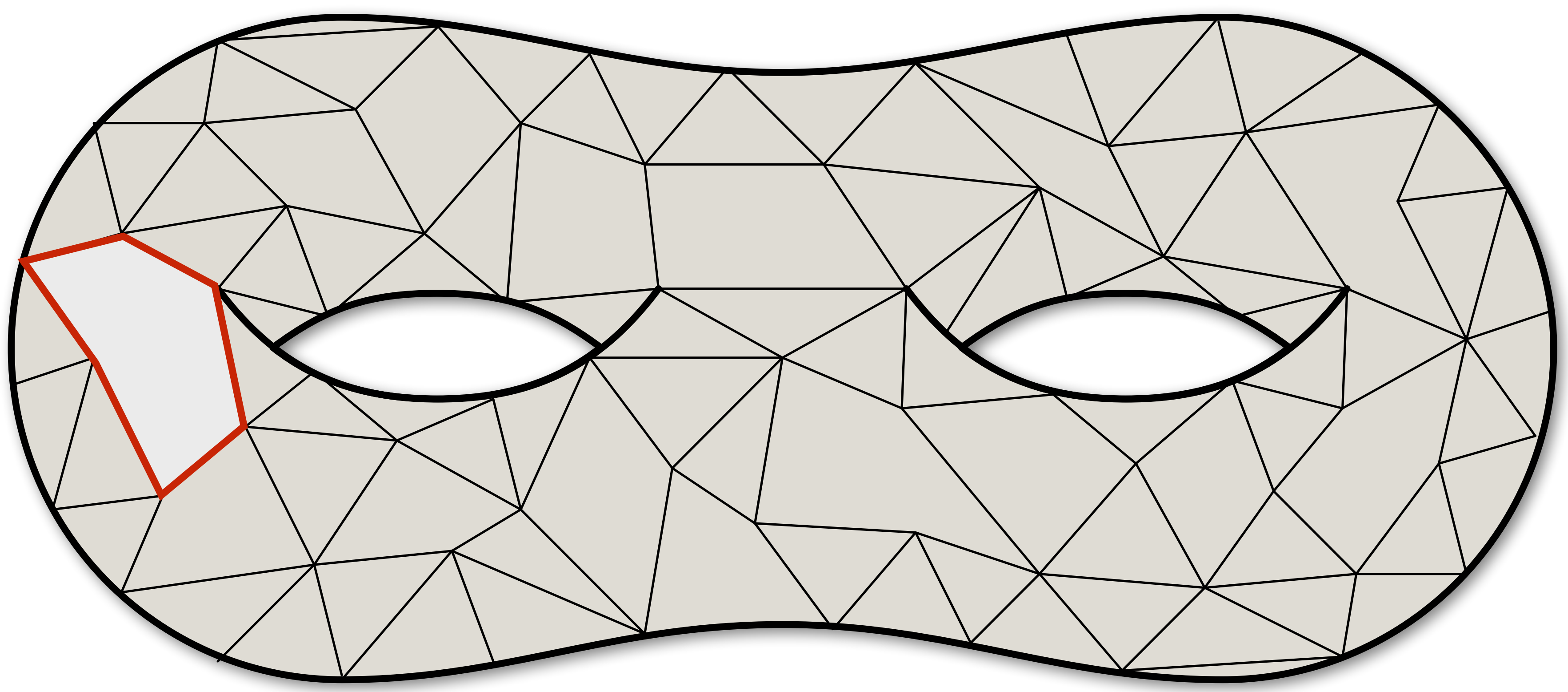
[Klein 2005]

- We can (implicitly) compute distances from every boundary vertex to every vertex in any planar map in $O(n \log n)$ time!

- More accurately: Given k vertex pairs, where one vertex of each pair is on the boundary, we can compute those k shortest-path distances in $O(n \log n + k \log n)$ time.

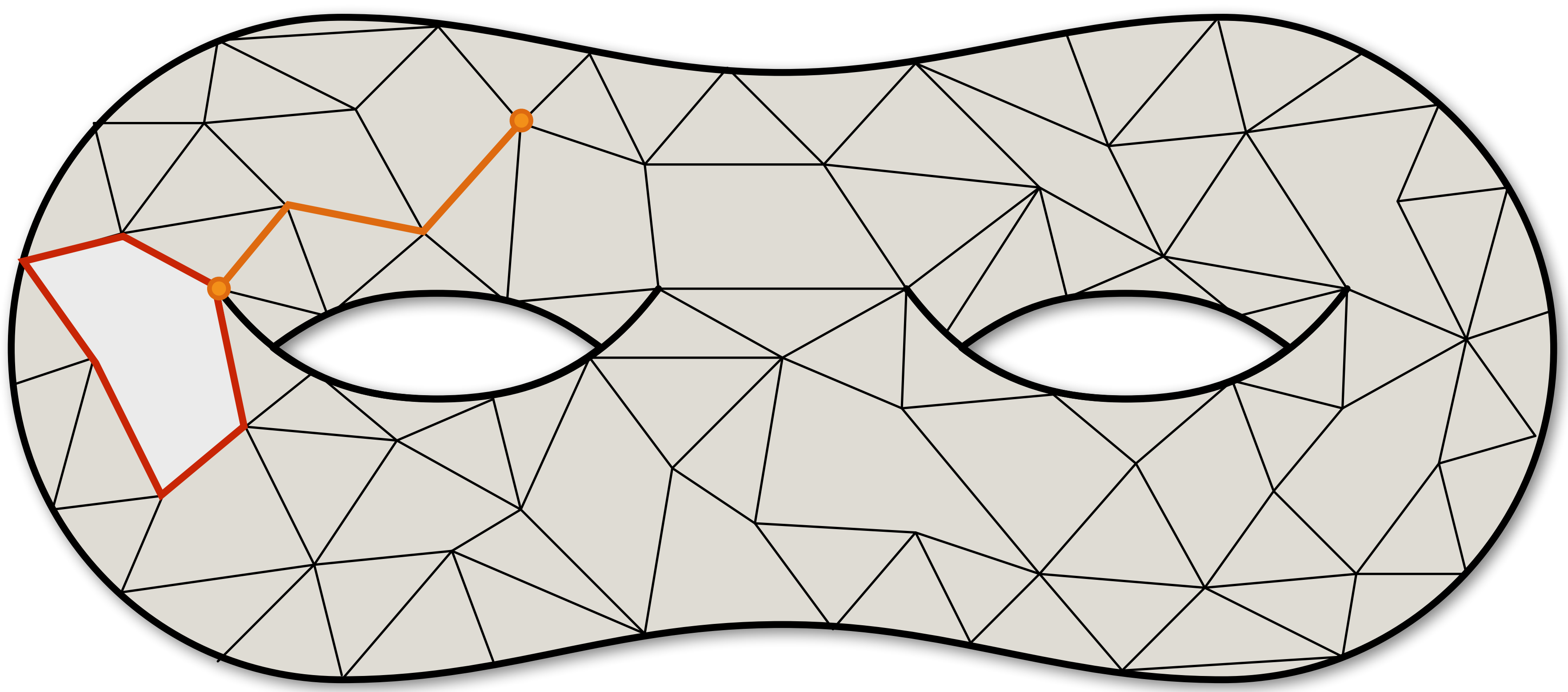
Higher-genus MSSP

- Let Σ be any surface map with genus g. Fix a face f of Σ.
- We want to compute the shortest path trees rooted at every vertex of some "outer" face f.



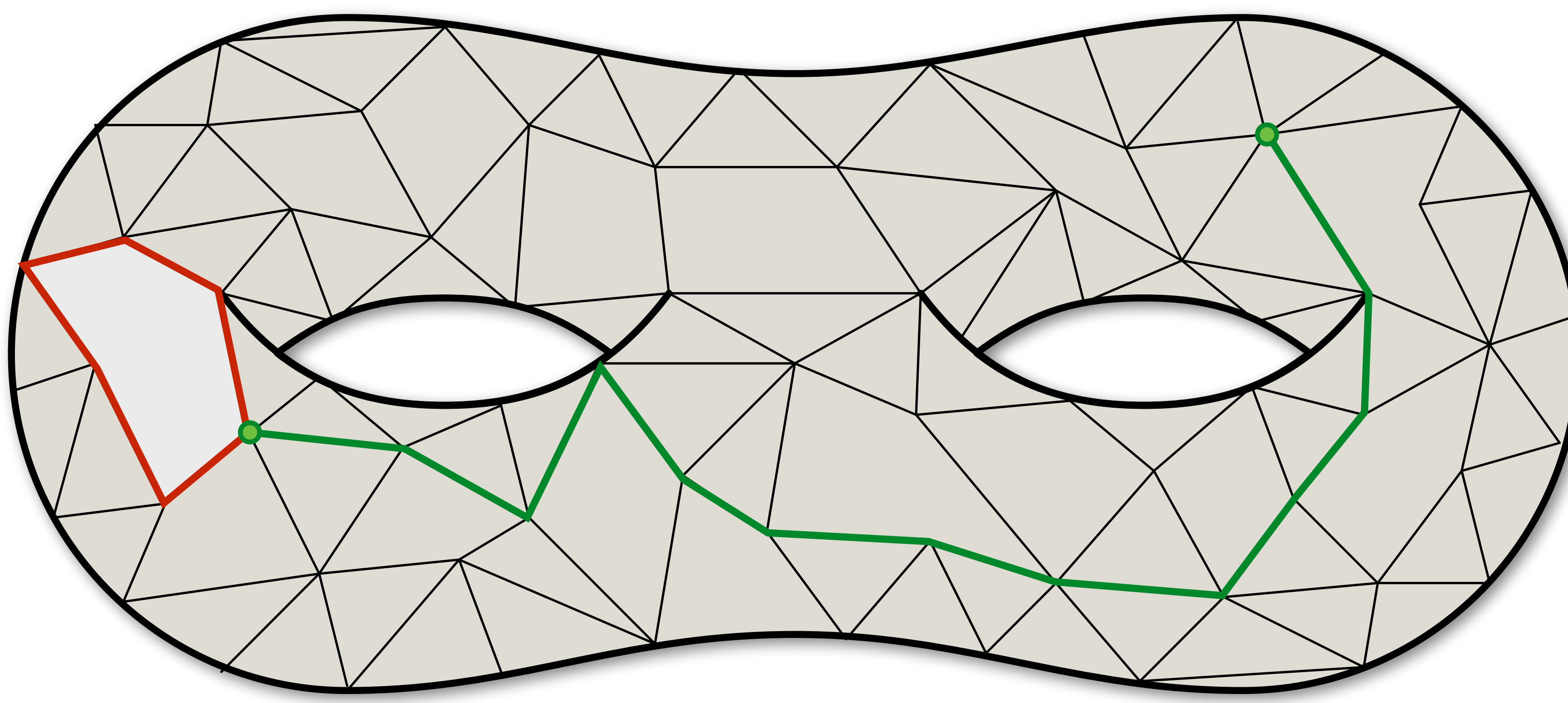
Higher-genus MSSP

- Let Σ be any surface map with genus *g*. Fix a face *f* of Σ.
- We want to compute the shortest path trees rooted at every vertex of some “outer” face *f*.



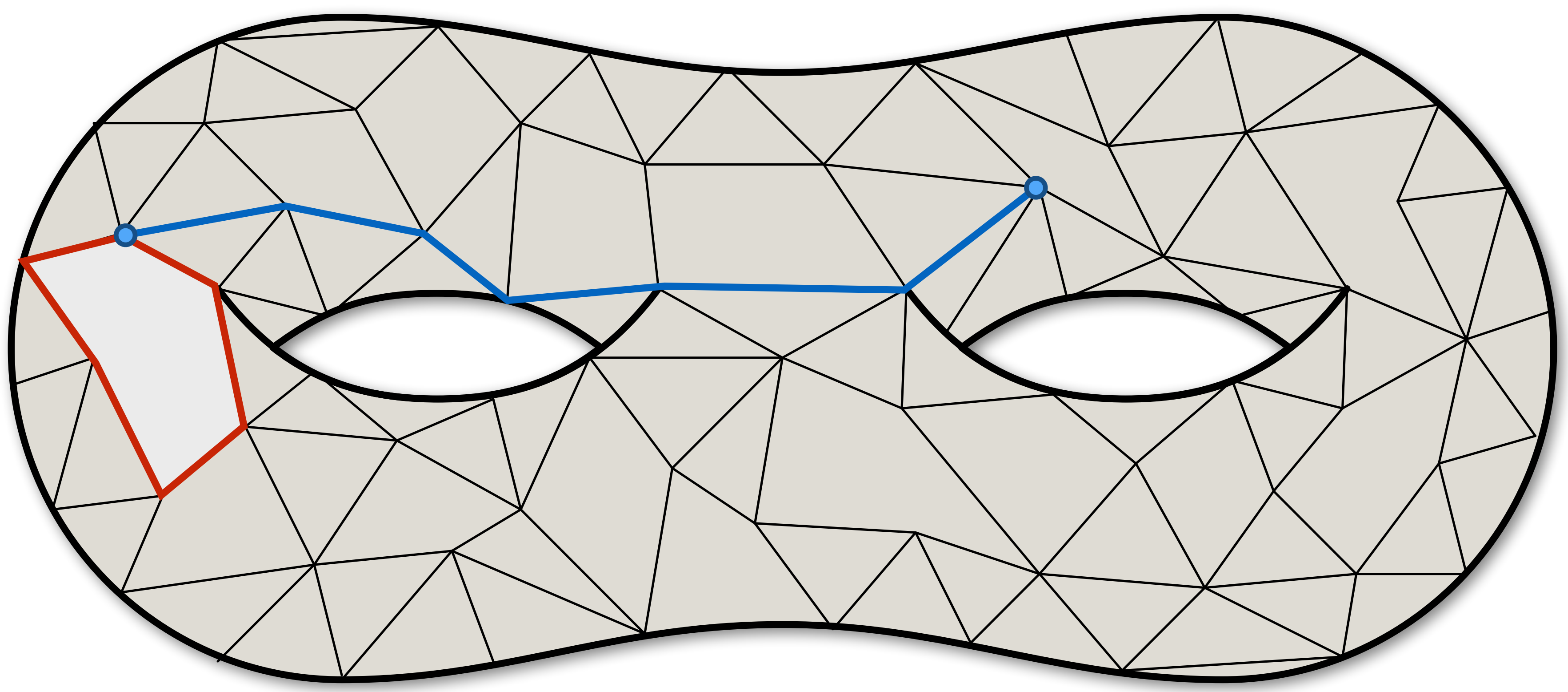
Higher-genus MSSP

- Let Σ be any surface map with genus *g*. Fix a face *f* of Σ.
- We want to compute the shortest path trees rooted at every vertex of some “outer” face *f*.



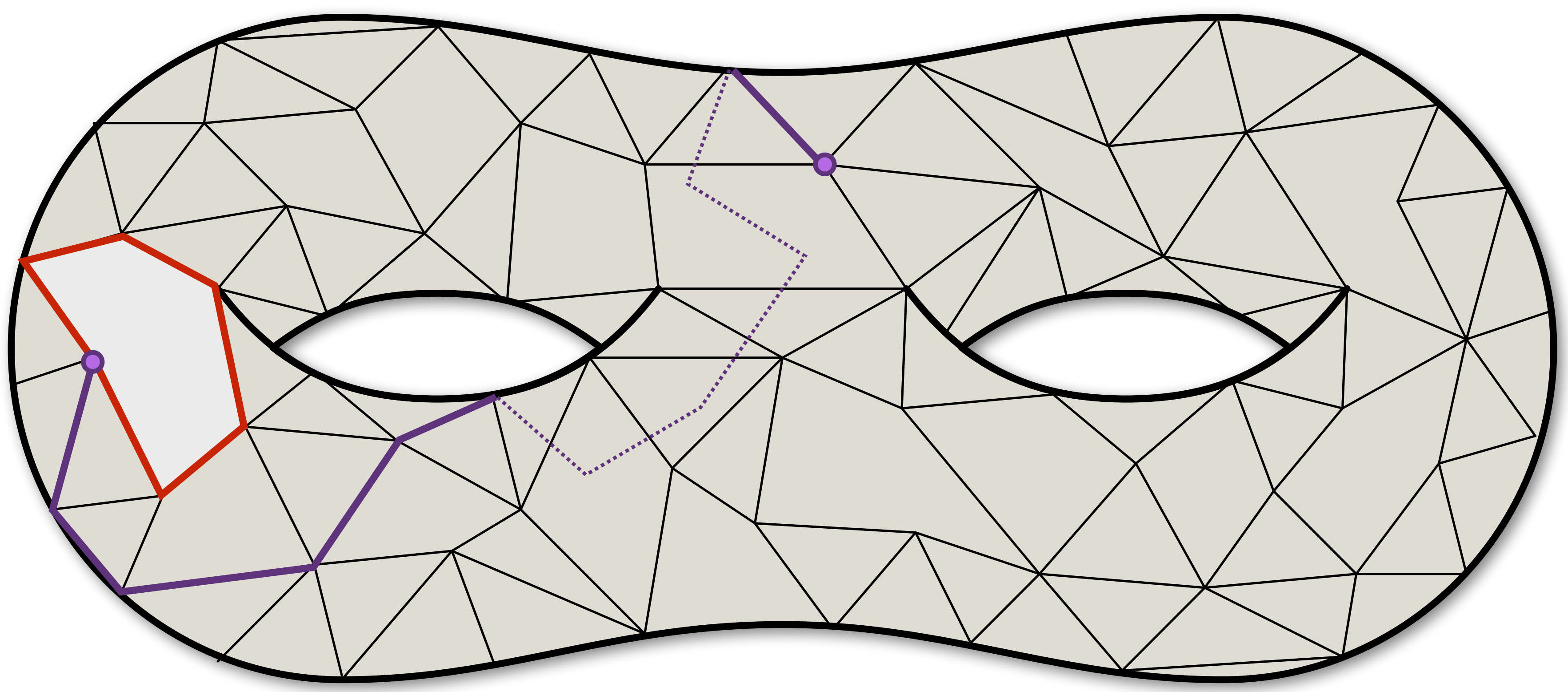
Higher-genus MSSP

- Let Σ be any surface map with genus g. Fix a face f of Σ.
- We want to compute the shortest path trees rooted at every vertex of some “outer” face f.



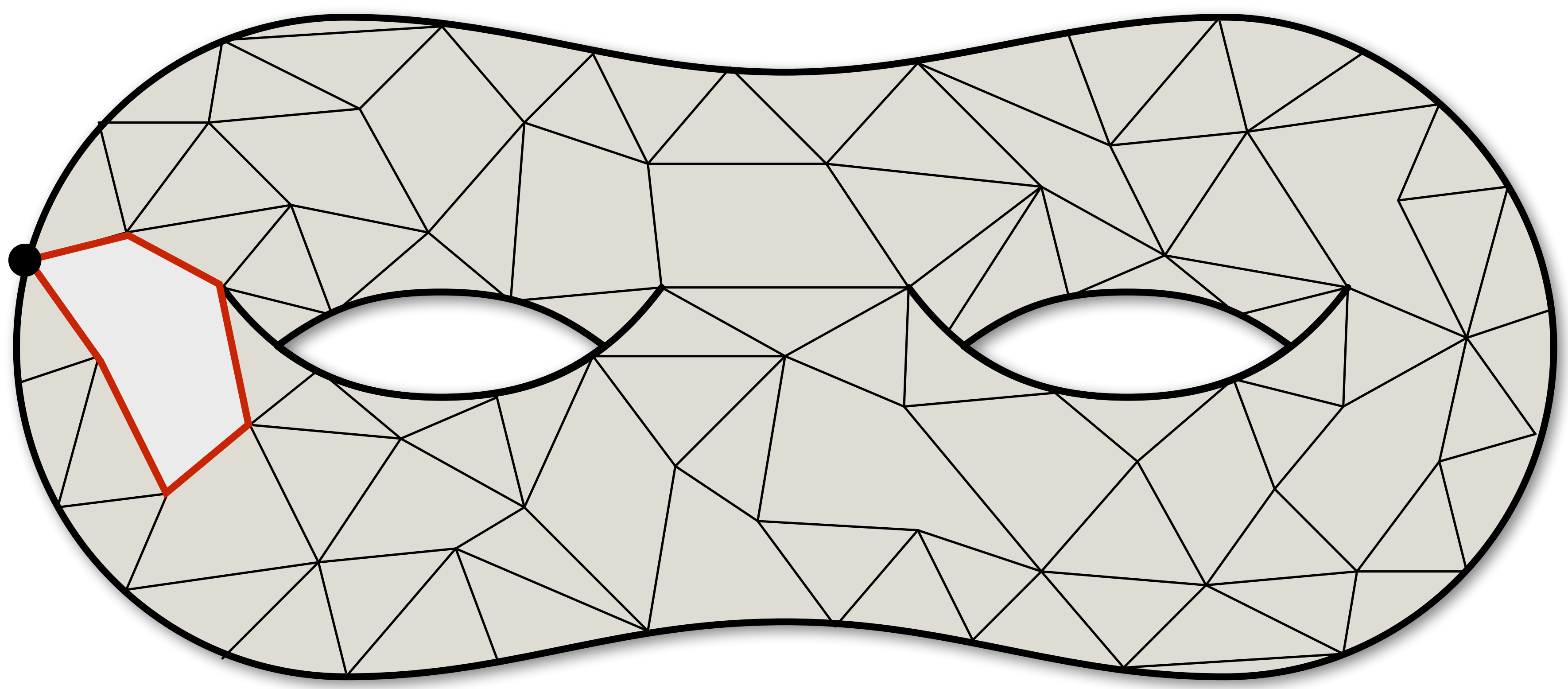
Higher-genus MSSP

- Let Σ be any surface map with genus g. Fix a face f of Σ.
- We want to compute the shortest path trees rooted at every vertex of some “outer” face f.



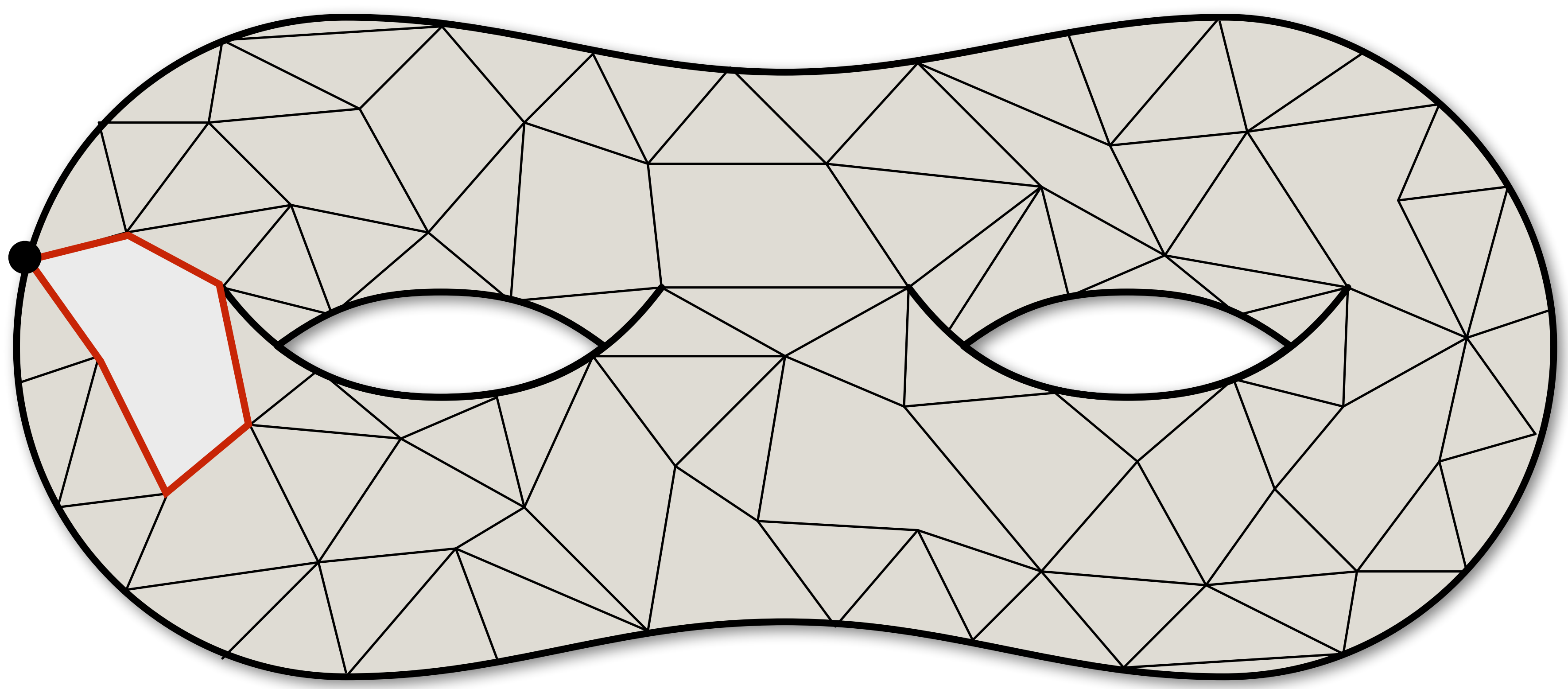
Same strategy!

- Move a point *s continously* around *f*, maintaining both the shortest-path tree rooted at s and the complementary slacks. Whenever a non-tree edge becomes tense, relax it.



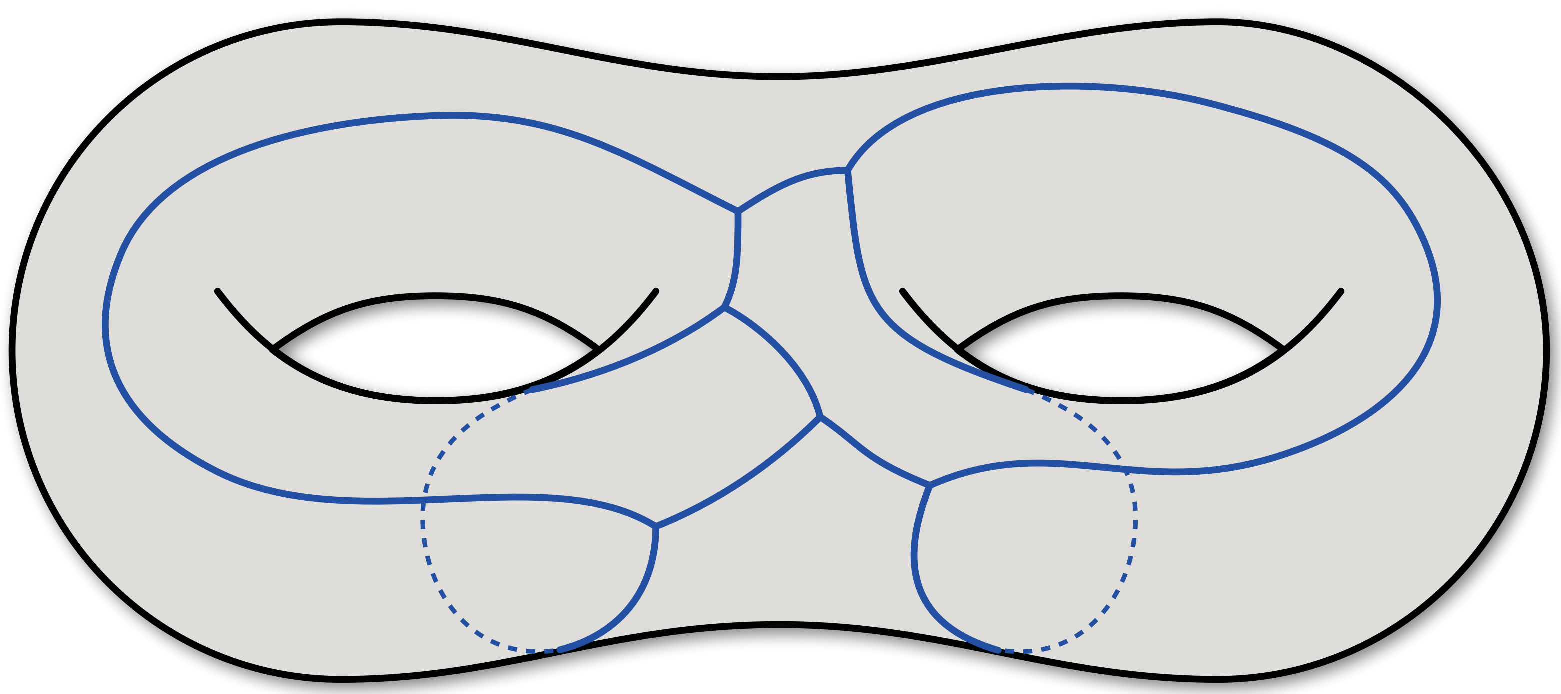
Same strategy!

- Move a point *s continously* around *f*, maintaining both the shortest-path tree rooted at s and the complementary slacks. Whenever a non-tree edge becomes tense, relax it.



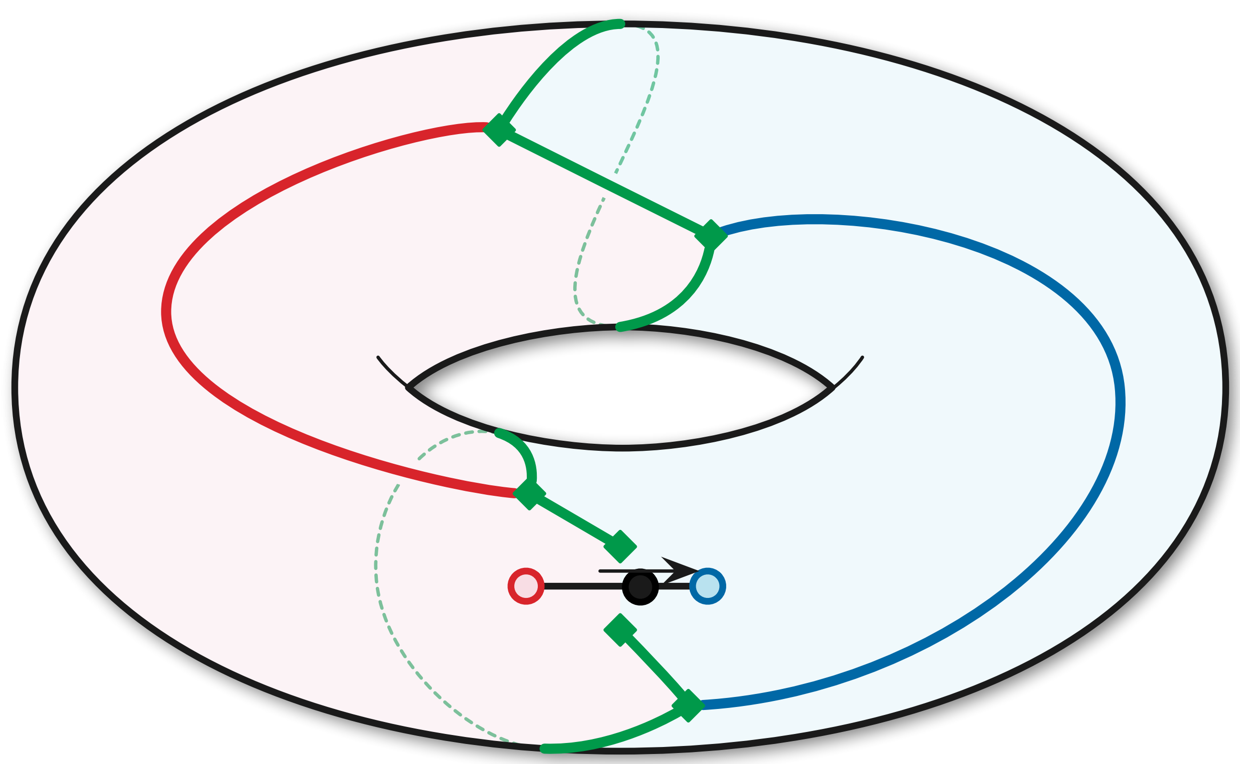
Complementary grove

- The dual *cut graph* $X^* = (G \backslash T)^*$ is no longer a spanning tree!
- *Grove decomposition*: partition X^* into $6g$ subtrees of G^*.
  - Each subtree contains one dual cut path and all attached "hair"
  - Maintain each subtree in its own dynamic forest data structure



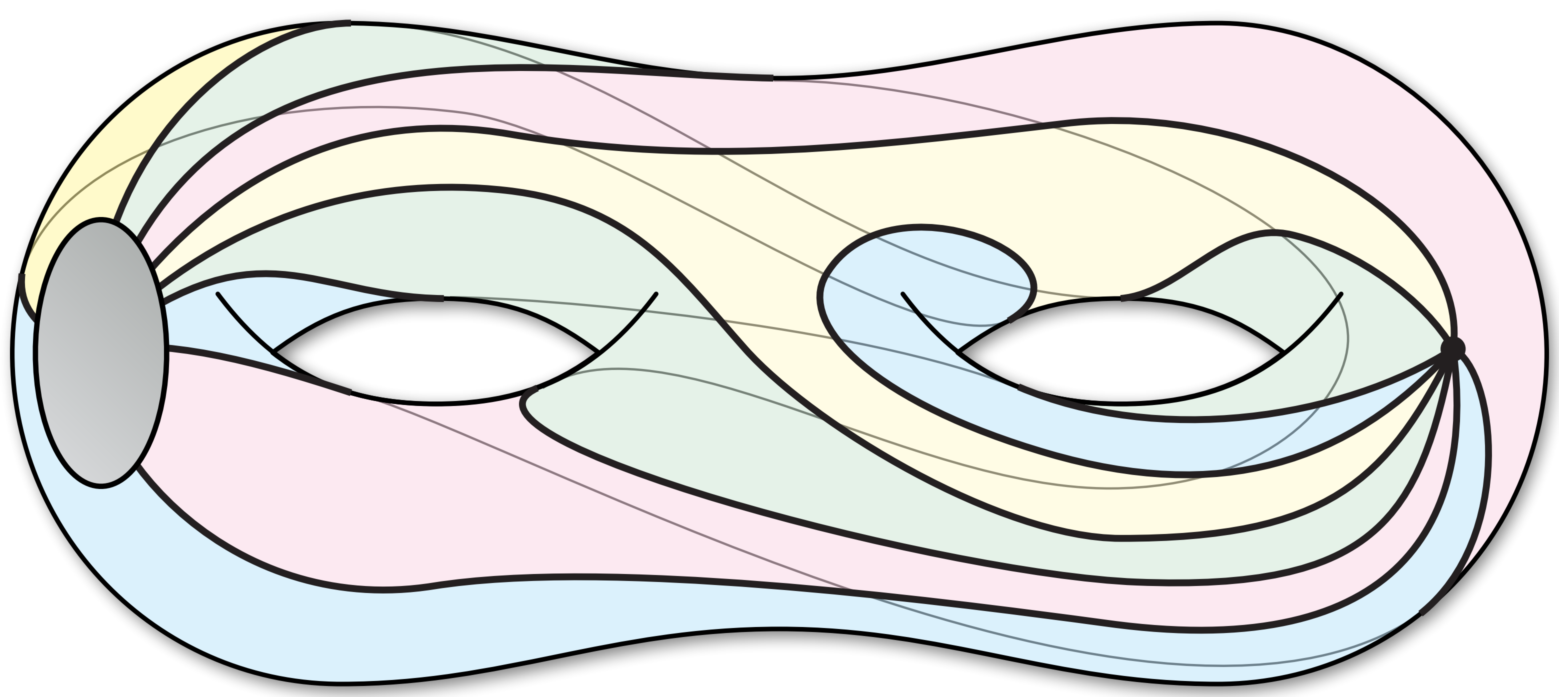
Where are the pivots?

- All *active* edges are dual to edges in some dual cut path.
- We can find and execute each pivot using $O(g)$ dynamic forest operations = $O(g \log n)$ amortized time.



How many pivots?

- Each directed edge pivots into T at most $4g$ times.
  - $4g$ = max # disjoint non-homotopic paths between two points in Σ
  - = # edges in a system of quads!
- So the total number of pivots is $O(gn)$



Summary

[Cabello Chambers Erickson 2013]
[Fox Erickson Lkhamsuren 2018]

- Given any surface map Σ with complexity n and genus g, with non-negatively weighted edges, and a face f.
- We can (implicitly) compute shortest-path distances from every vertex of f to every vertex of Σ...
  - in $O(gn \log n)$ time with high probability
  - or in $O(\min\{g, \log n\} \cdot gn \log n)$ worst-case deterministic time

Picky details

[Cabello Chambers Erickson 2013]

- Everything so far assumes that shortest paths are unique, and that at most one edge becomes tense at a time.
- We can enforce this assumption by perturbing the edge weights.
  - Randomized perturbation: $O(1)$ time penalty, but succeeds only with high probability
    [Mulmuley Vazirani Vazirani 1987]
  - Lexicographic perturbation: $O(\log n)$ time penalty
    [Charnes 1952] [Dantzig Orden Wolfe 1955]
  - Homologically-least leftmost (“holiest”) perturbation: $O(g)$ time penalty
    [Fox Erickson Lkhamsuren 2018]

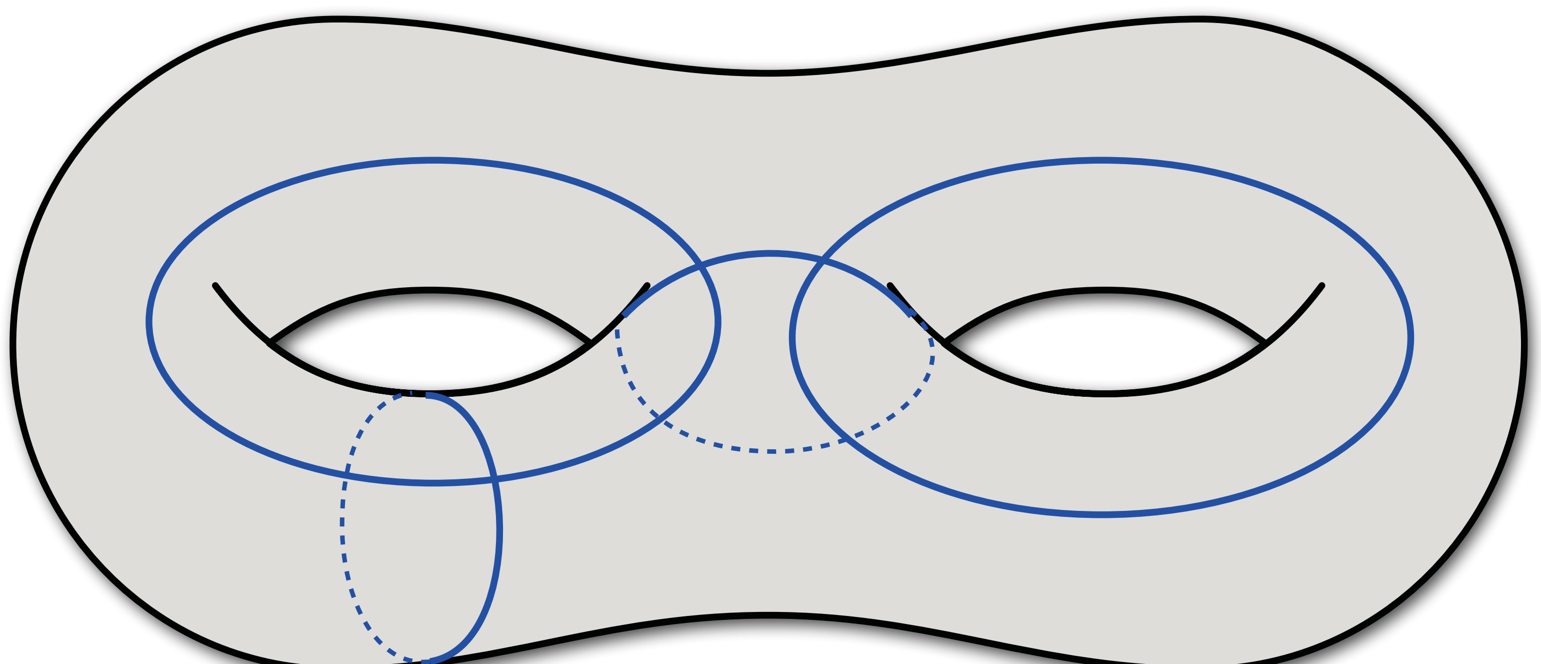
Shortest nontrivial cycles, take 2

Faster algorithm

[Cabello Chambers 2007]

To compute the shortest *nonseparating* cycle:

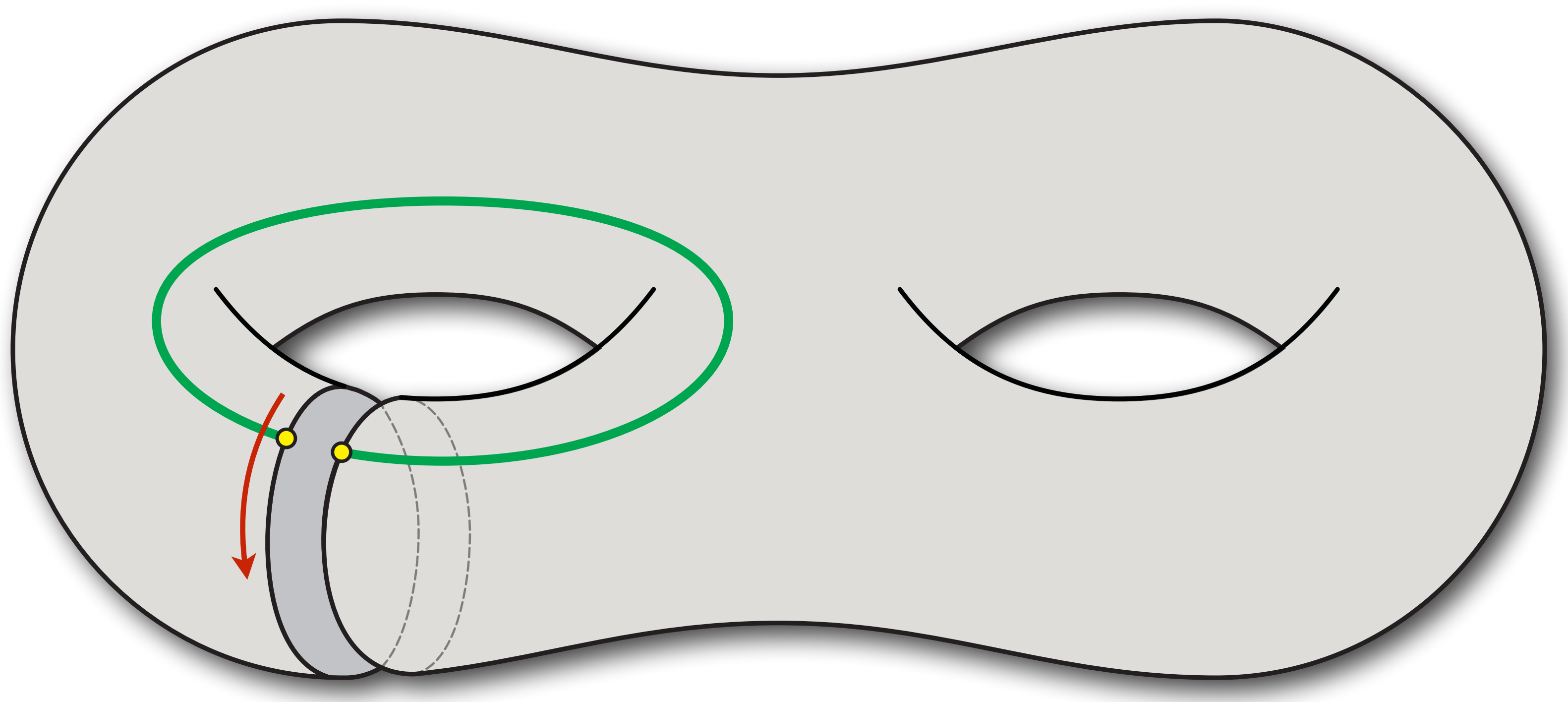
- Compute a greedy system of cycles $\gamma_1, \gamma_2, \ldots, \gamma_{2g}$
- Find the shortest cycle that crosses each greedy cycle γ_i once



Algorithm

[Cabello Chambers 2007]

- To find the shortest cycle that crosses γ_i once:
  - Cut the surface open along γ_i. Resulting surface $\Sigma ✂ \gamma_i$ has two copies of γ on its boundary.
  - Find *shortest path* in $\Sigma ✂ \gamma_i$ between two copies of each vertex of γ_i
  - **MSSP:** $O(gn \log n)$ time *with high probability*



Algorithm 1

[Cabello Chambers Erickson 2013]

To compute the shortest nonseparating cycle:

- Compute a greedy tree-cotree decomposition
- Compute a greedy system of cycles $\gamma_1, \gamma_2, \ldots, \gamma_{2g}$
- Find the shortest cycle that crosses each greedy cycle γ_i once

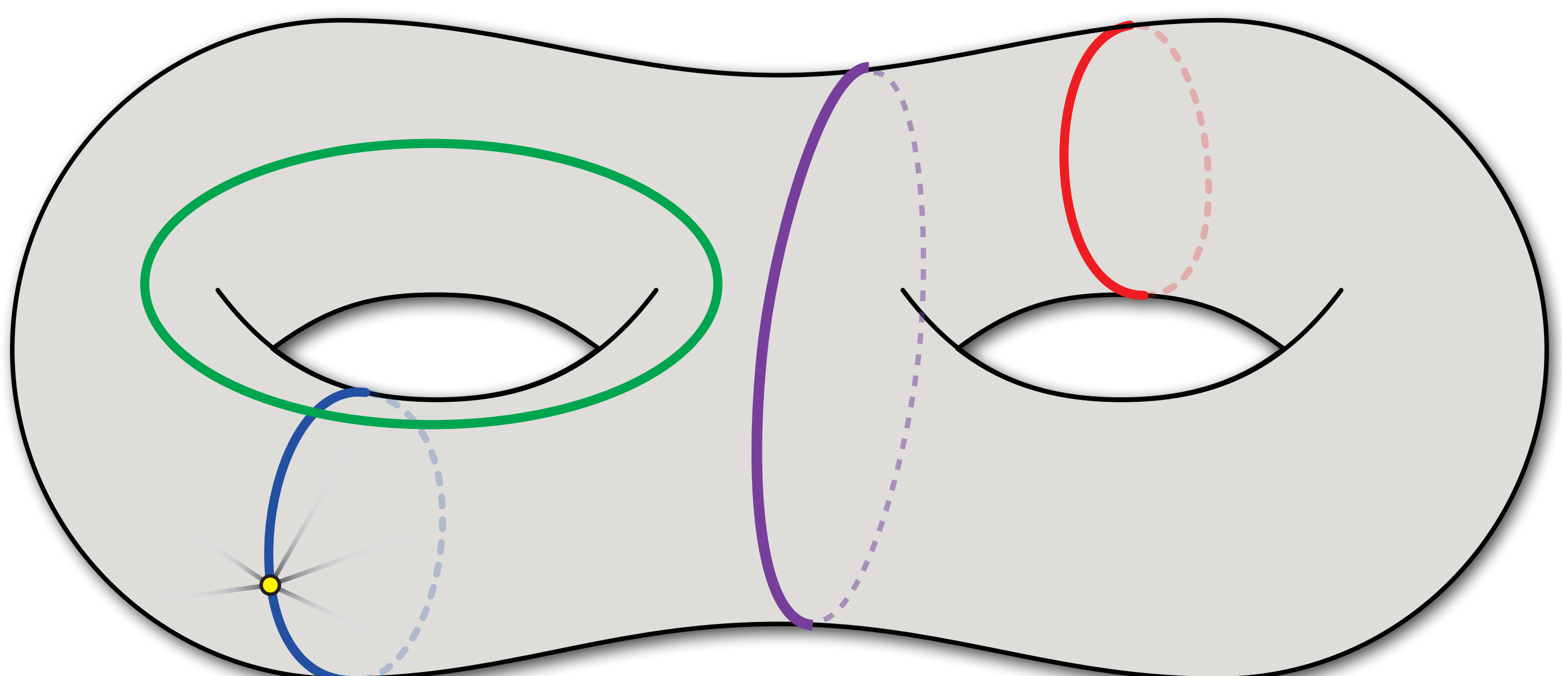
- $O(g^2 n \log n)$ time *with high probability*

- This is the fastest algorithm known in terms of both n and g.

One-cross lemmas

[Cabello Chambers 2007]

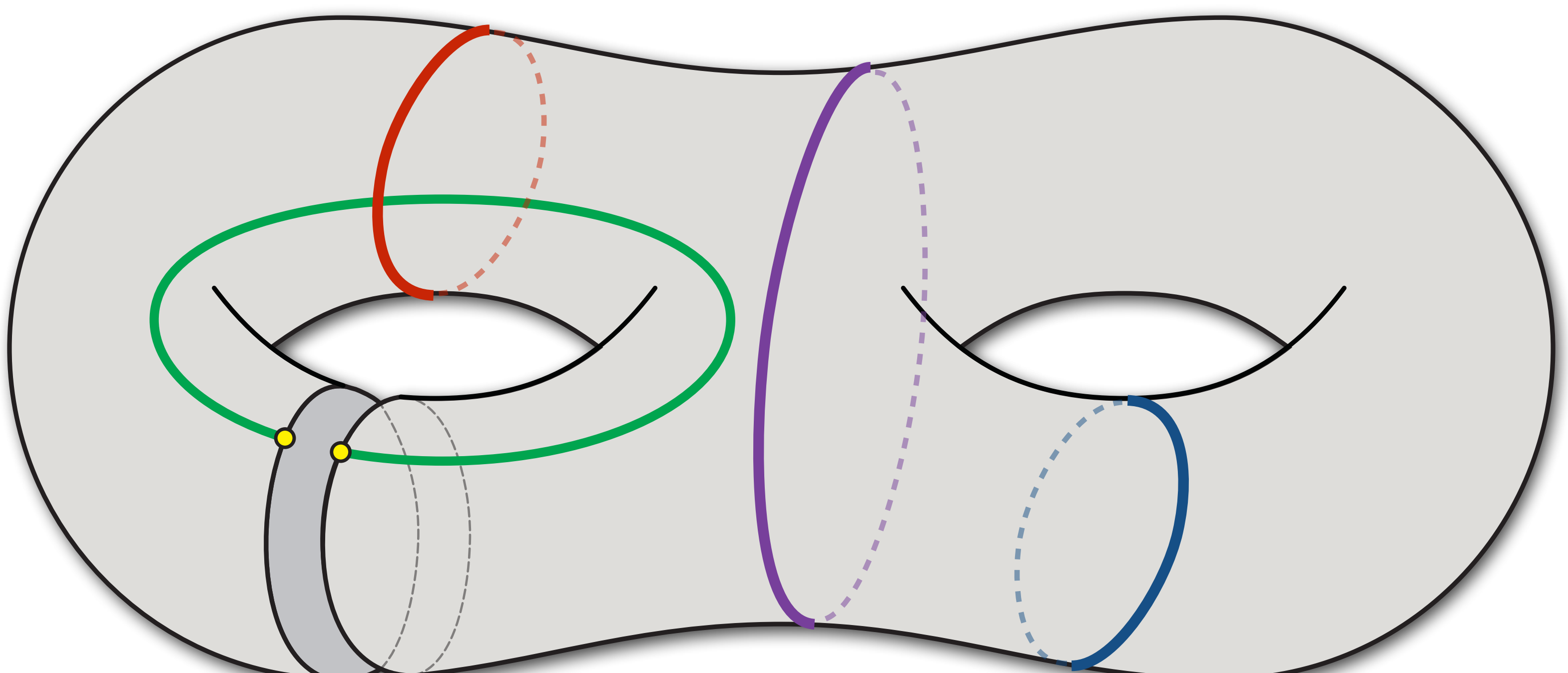
- Let γ^* be the shortest *noncontractible* cycle, and let ℓ be the shortest noncontractible loop at an arbitrary basepoint.
- Then γ^* and ℓ cross at most once.



One-cross lemmas

[Cabello Chambers 2007]

- Let *γ** be the shortest *noncontractible* cycle, and let *π* be a shortest *nonseparating* path between two boundary points.
- Then *γ** and π cross at most once.



Algorithm 2

[Cabello Chambers Erickson 2013]

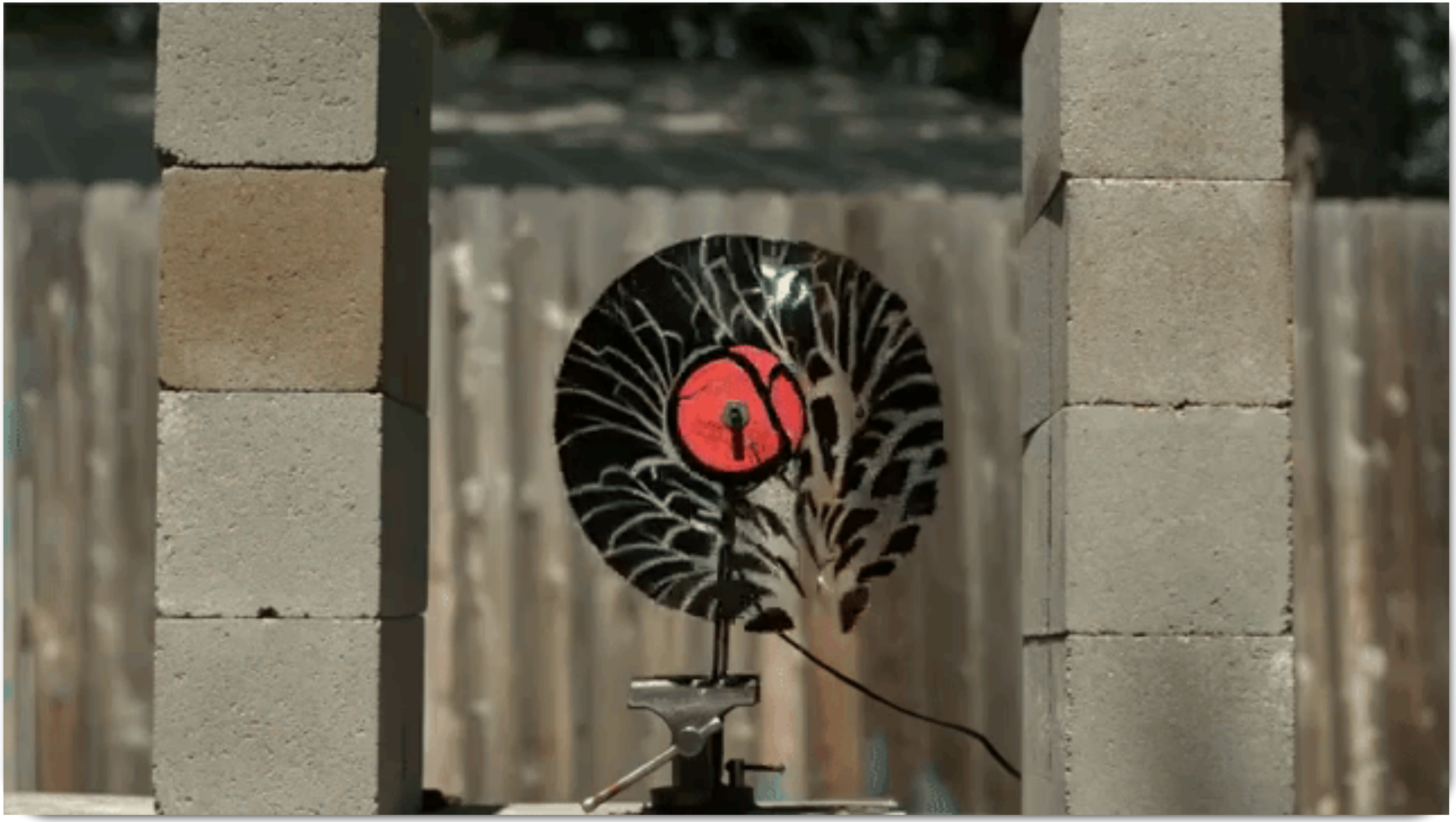
To compute the shortest noncontractible cycle:

- Find shortest non-contractible loop ℓ at some basepoint
- Find shortest cycle crossing ℓ once
- Cut the surface along ℓ
- While the surface is not a disk:
  - Find shortest non-separating boundary to boundary path π
  - Find shortest cycle crossing π once
  - Cut the surface along π

- $O(g^2 n \log n)$ time *with high probability*

- This is the fastest algorithm known in terms of both n and g.

Thank you!



[Free Gruchy (“Slow-Mo Guys”) 2018]

Thank you!

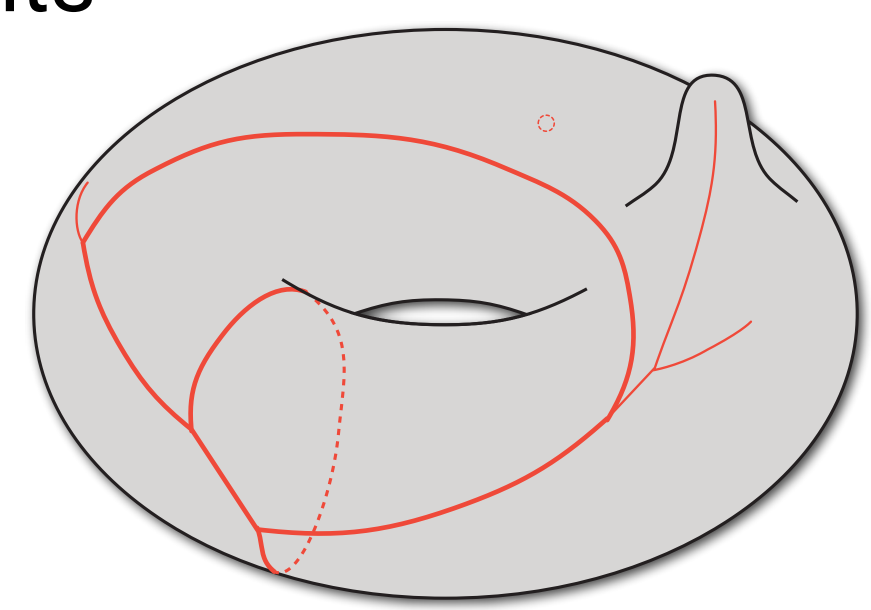
[Free Gruchy ("Slow-Mo Guys") 2018]

Continuous surfaces

or “Why not solve the *real* problem?”

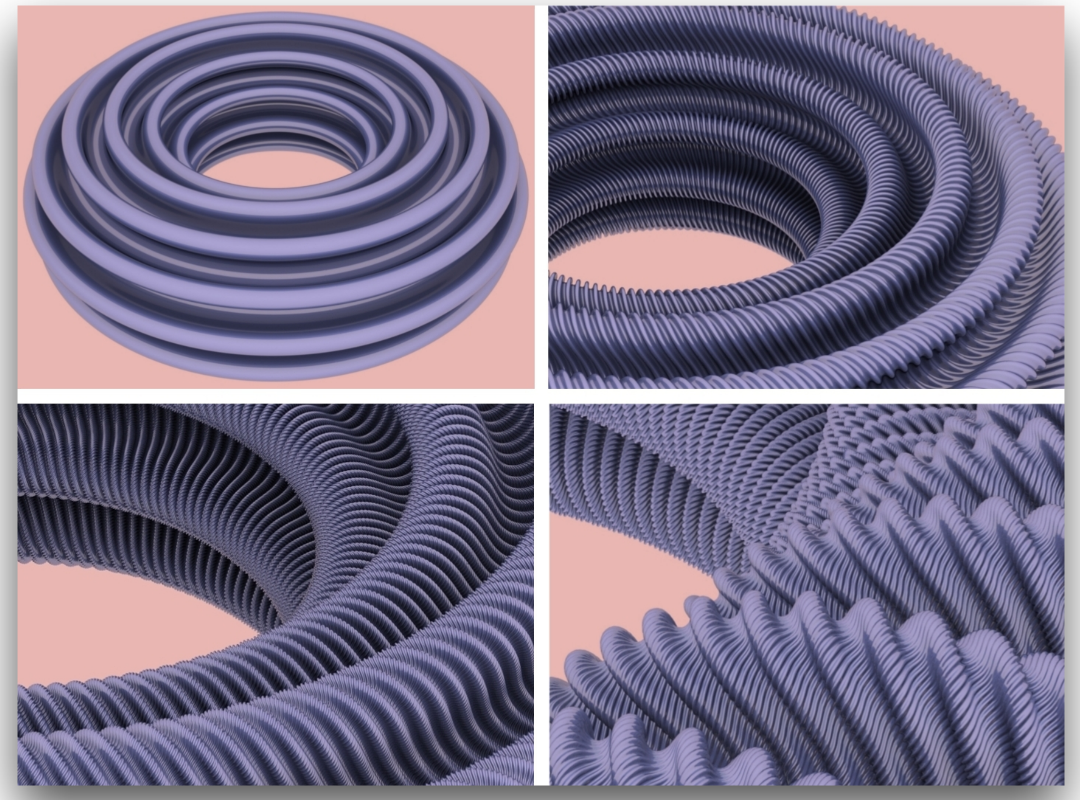
Structural results generalize...

- The 3-path and 1-crossing conditions still hold
- The shortest non-trivial cycle still contains shortest paths between any pair of antipodal points
- The greedy system of loops is still optimal
- Every cycle in a greedy system of cycles contains shortest paths between any two antipodal points
- The continuous analogue of the greedy cut graph is a *cut locus*

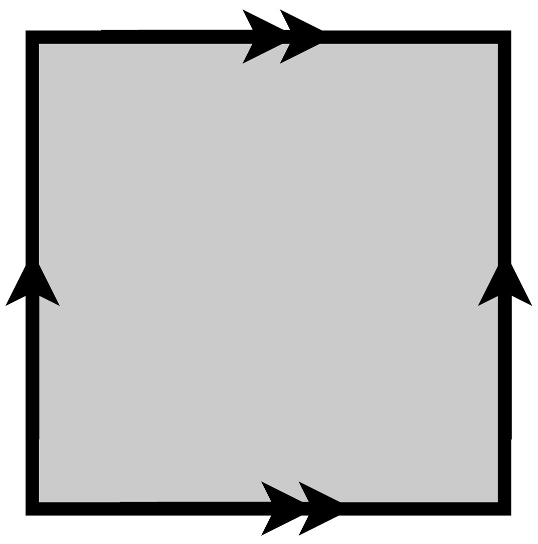


...but what about algorithms?

- All algorithms ultimately rely on *computing shortest paths*.
- So we must be *given* a surface representation that supports computing shortest paths!

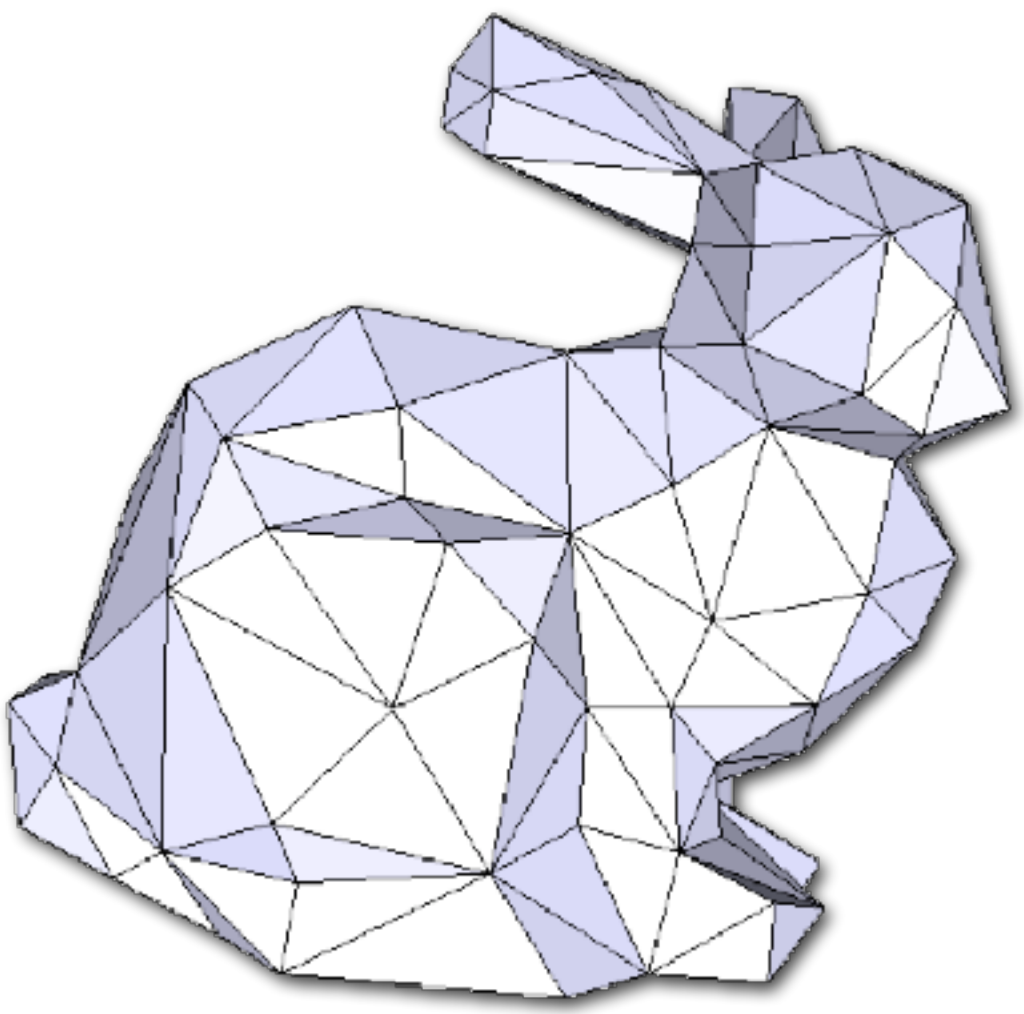


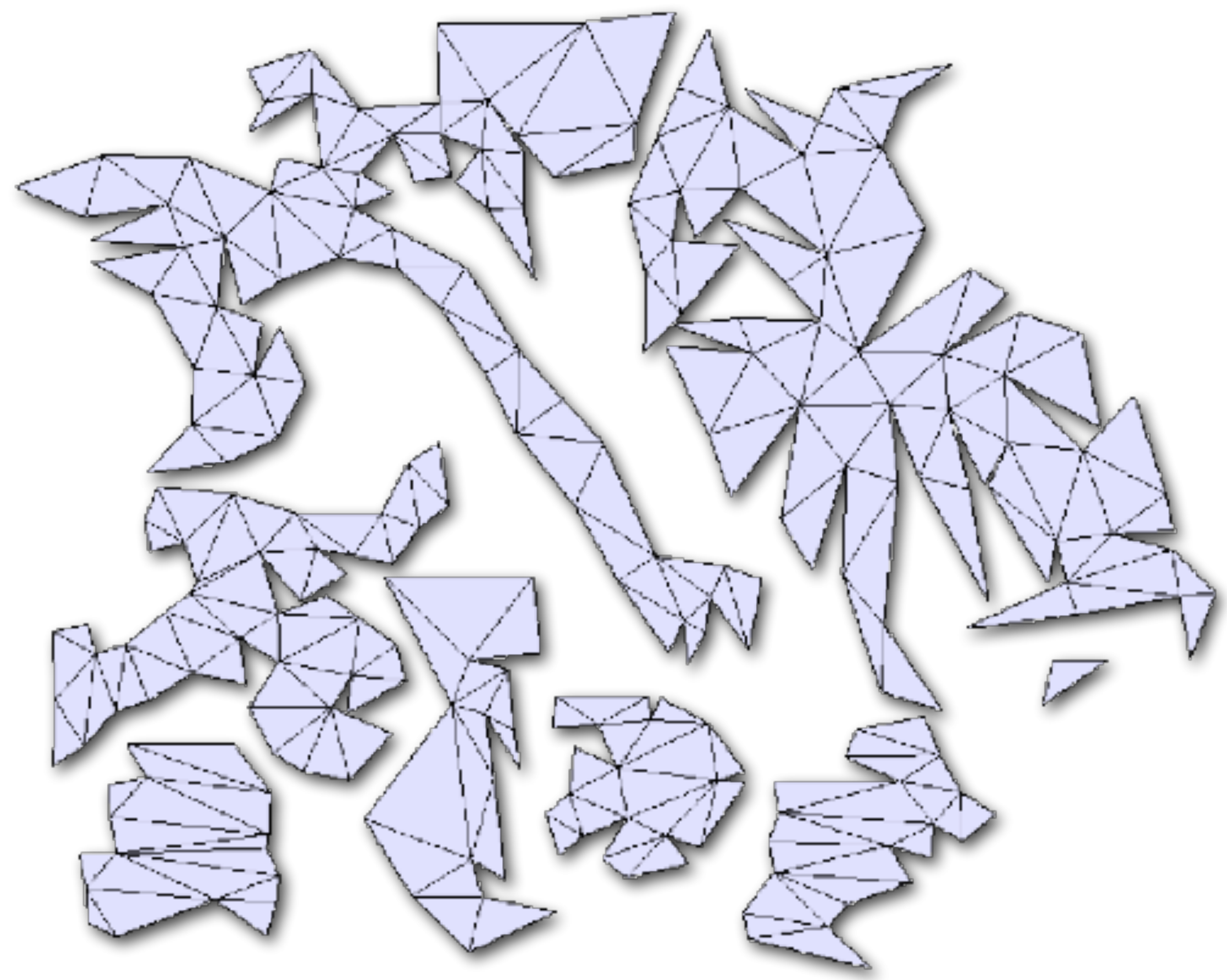
[Borelli Jabrane Lazarus Rohmer Thibert 2012]



Piecewise-linear surfaces

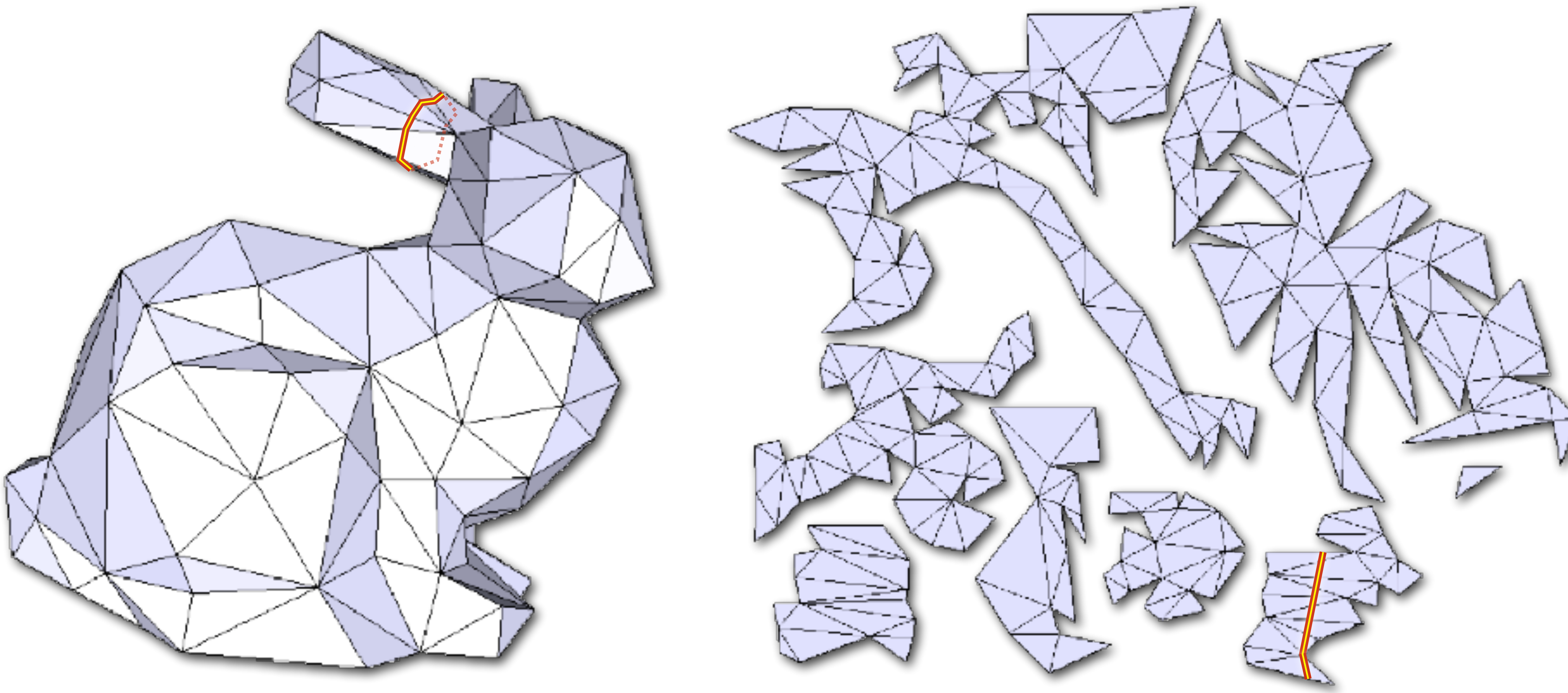
- Complex of Euclidean polygons with pairs of equal-length edges identified (glued)





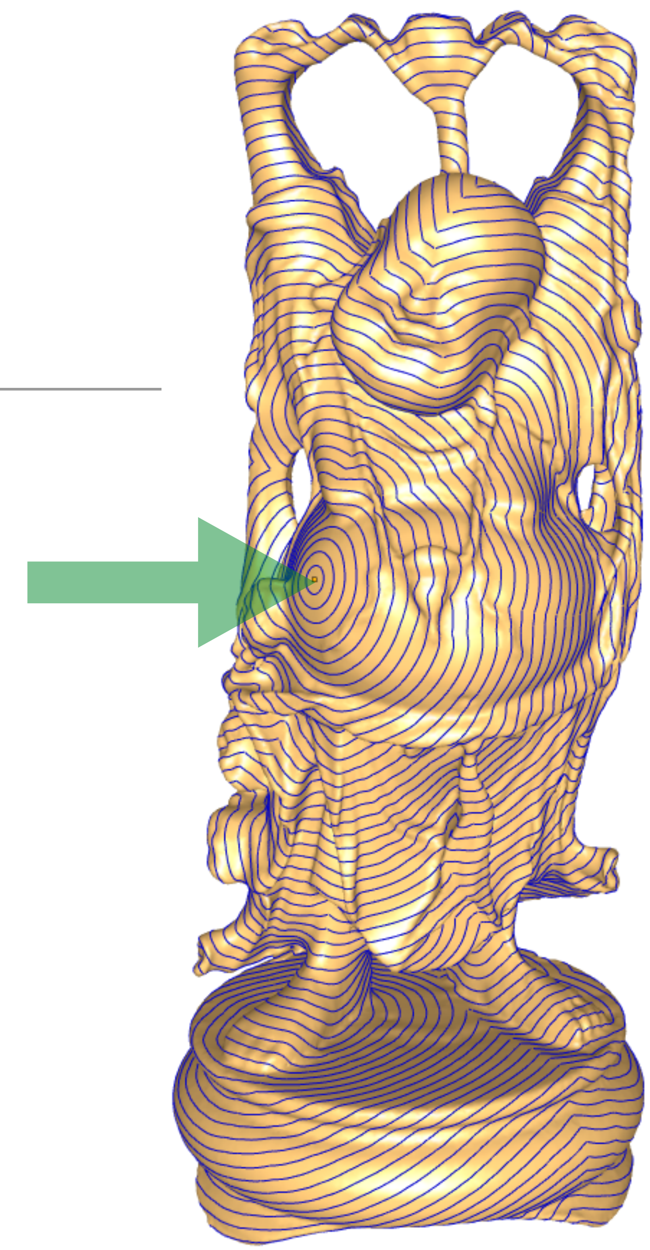
Piecewise-linear surfaces

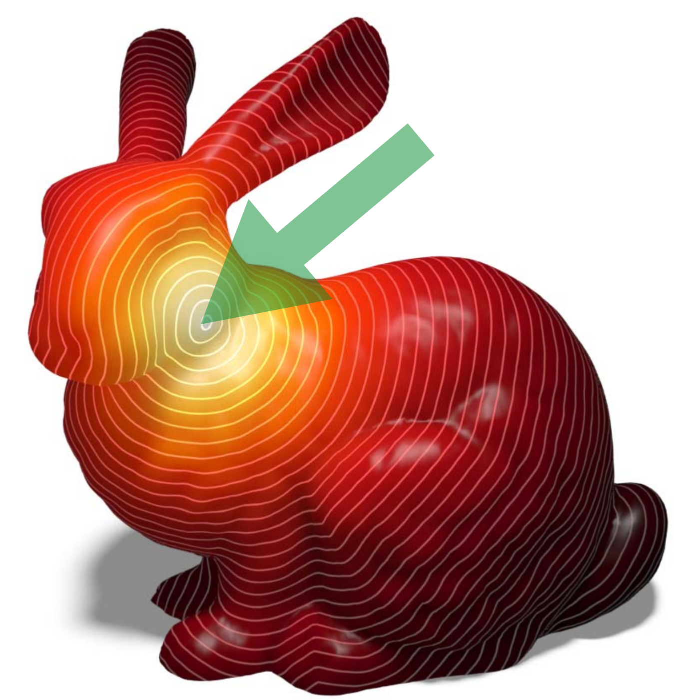
- Metric is Euclidean everywhere except at vertices
- Paths and cycles can be *anywhere* on the surface



PL shortest paths

- "Continuous Dijkstra"
  - $O(n^2 \log n)$ time *[Mitchell Mount Papadimitriou 1987]*
  - $O(n^2)$ *[Chen Han 1990]*
- This lets us compute shortest nontrivial cycles in $O(n^3)$ time.
- **Lots** of approximation algorithms, faster special cases, practical heuristics, and false starts
  - Practical implementation *[Surazhsky Surazhsky Kirsanov Gortler Hoppe 2005]*
  - Heat equation *[Crane Weischedel Wardetzky 2013]*





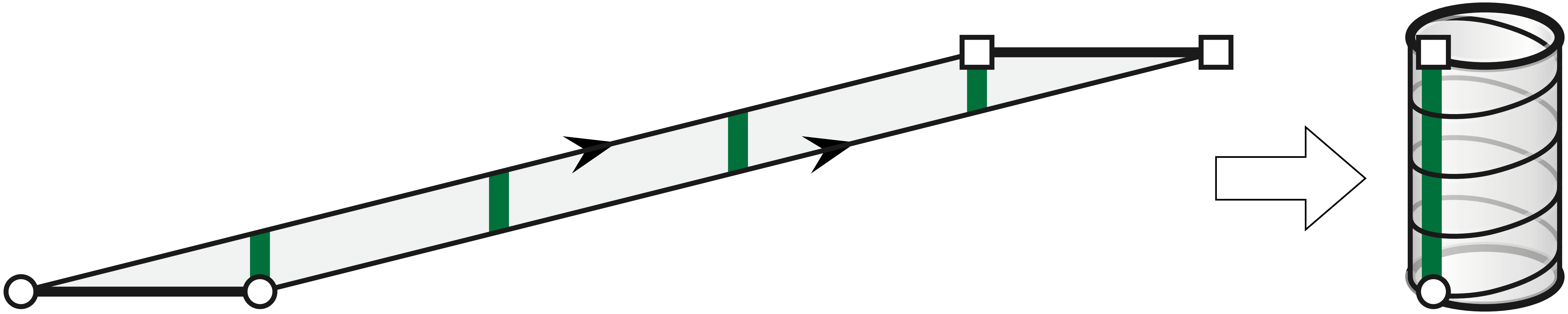
Hidden assumptions

- Exact algorithms require exact real arithmetic
  - Ugly theoretical quagmire, but not a significant issue "in practice"

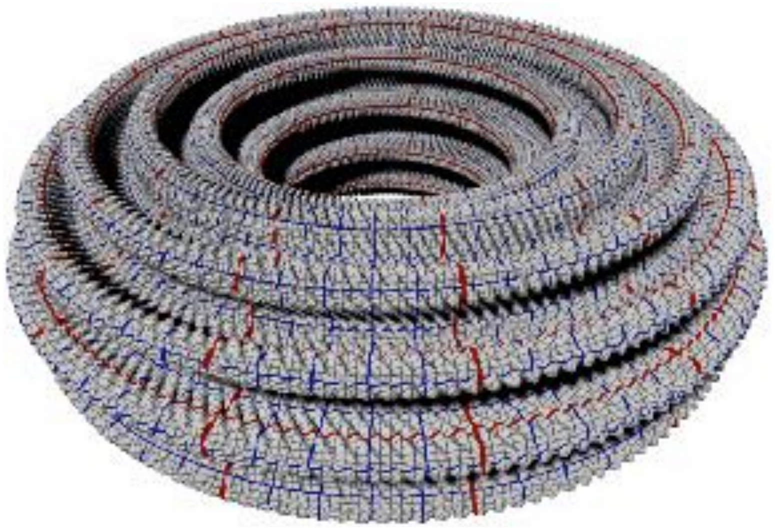
- Analysis assumes that every shortest path crosses each edge of the given PL structure *at most once*.
  - True for piecewise-flat maps into any $\mathbf{R}^d$.
  - True (or close enough) for PL triangulations with fat triangles
  - True for *some* PL structure of *every* PL surface.
    [Zalgaller 1958] [Burago Zalgaller 1995] [Bern Hayes 2011]
  - But ***not*** true for arbitrary PL structures!

Toilet paper tube

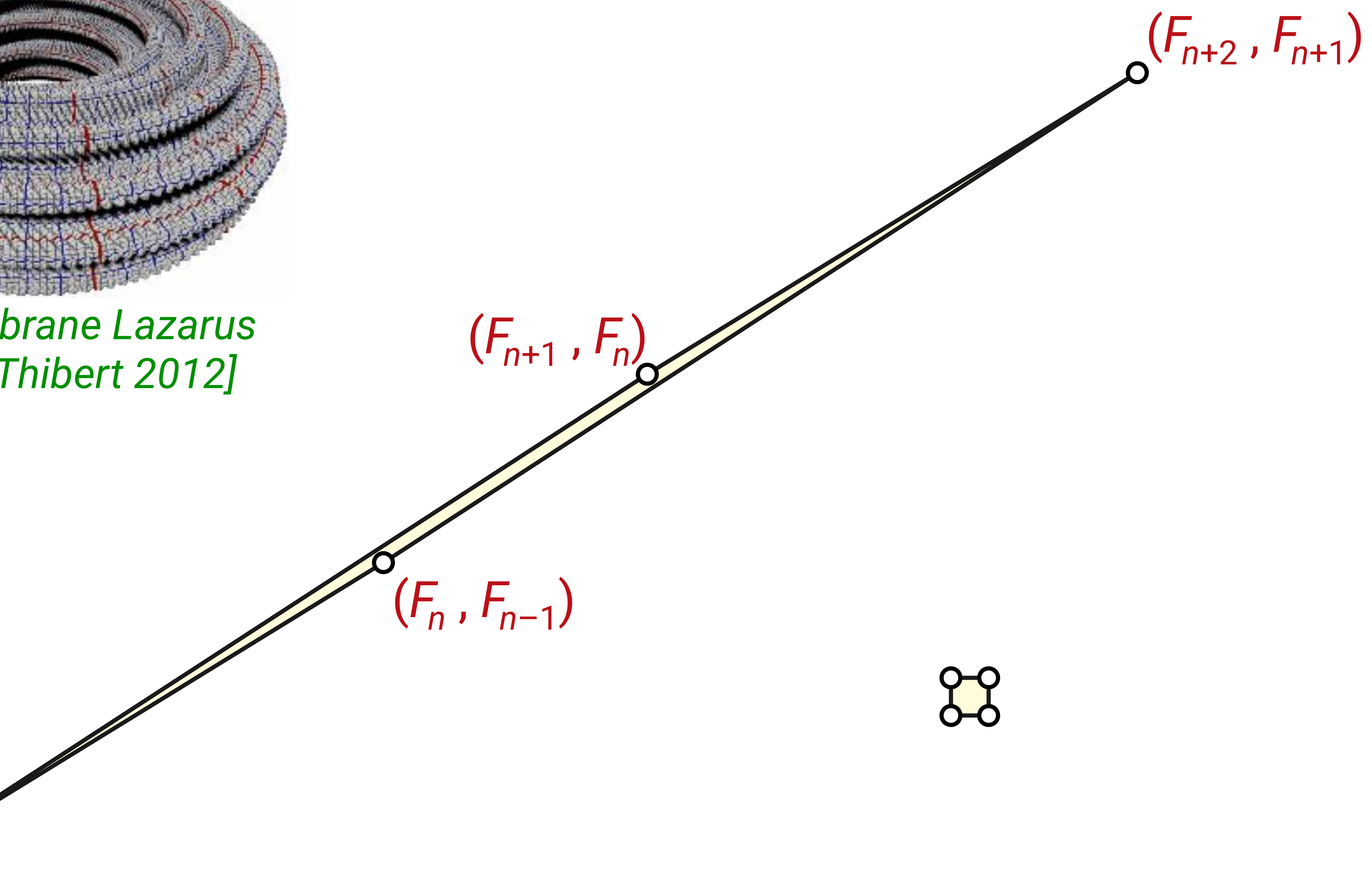
[Alexandrov 1942]
[Zalgaller 1997]



Square (sic) flat torus



[Borelli Jabrane Lazarus Rohmer Thibert 2012]



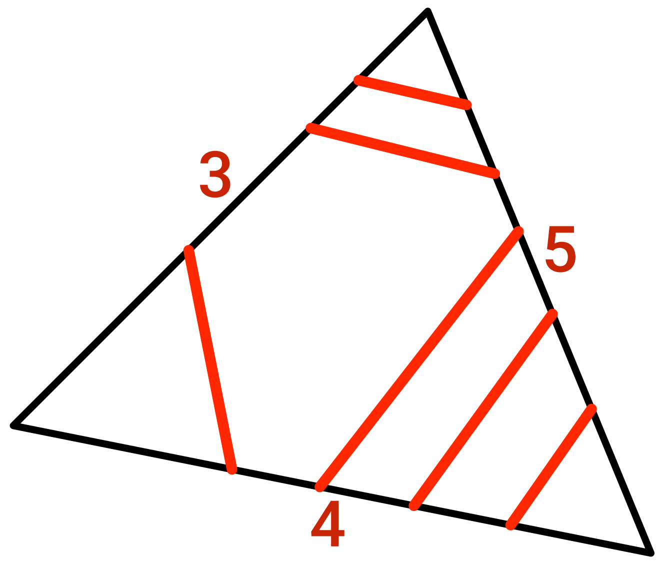
Unbounded time

- Let α = maximum *aspect ratio* of any triangular facet.
- **Good news:**
  - Any shortest path crosses each edge $O(\alpha)$ times (and this is tight).
  - So we can find the shortest nontrivial cycles in $O(poly(n, \alpha))$ time!
- **Bad news:**
  - If edge lengths or local coordinates are integers, then α can be *exponential* in the input size (# vertices + # edges + # bits).
  - If edge lengths or local coordinates are real numbers, then α is *not bounded by **any** function* of the input size (# vertices + # edges).

Normal coordinates to the rescue?

- We can *implicitly* represent any simple cycle or arc using $O(n \log X)$ bits, where X = # crossings.
  [Kneser 1930]

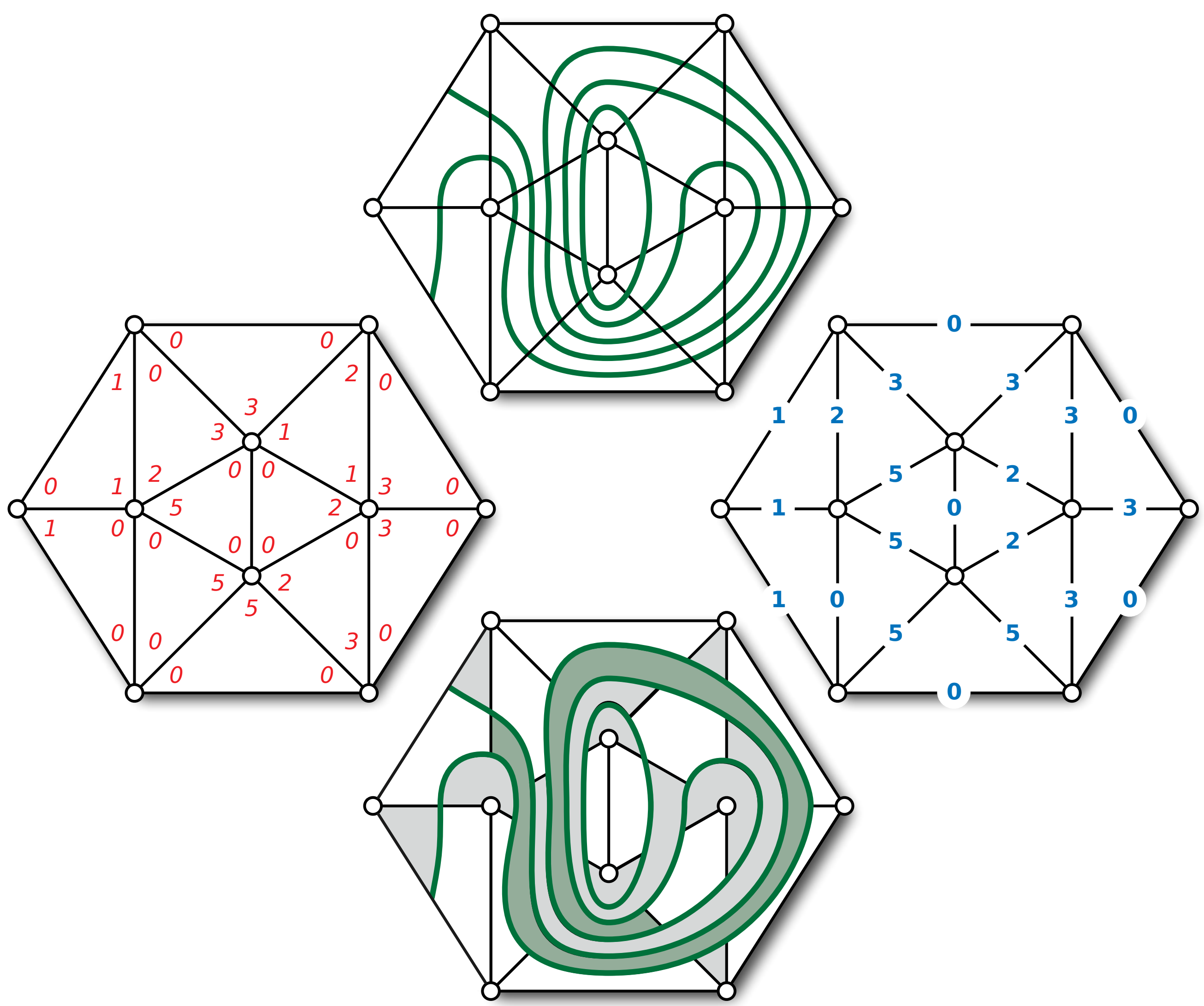
- Several algorithms for normal curves:
  - Counting and isolating components
  - Counting isotopy classes
  - Intersection numbers
  - Image of one curve under a mapping class
  - Distance between two curves in the curve complex
  - Classifying mapping classes

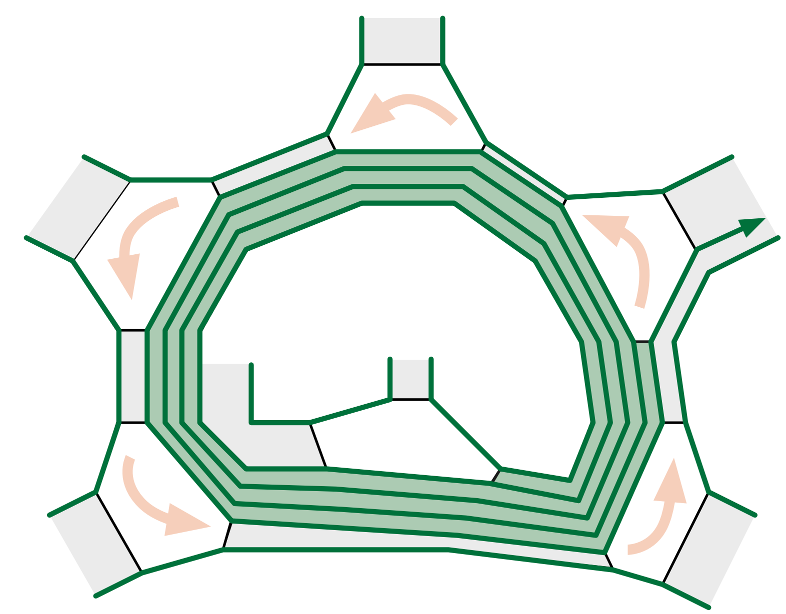


[Schaefer, Sedgwick, Štefankovic 2003] [Agol Hass Thurston 2006]
[Erickson Nayyeri 2013] [Bell Webb 2016]

Normal coordinates to the rescue?

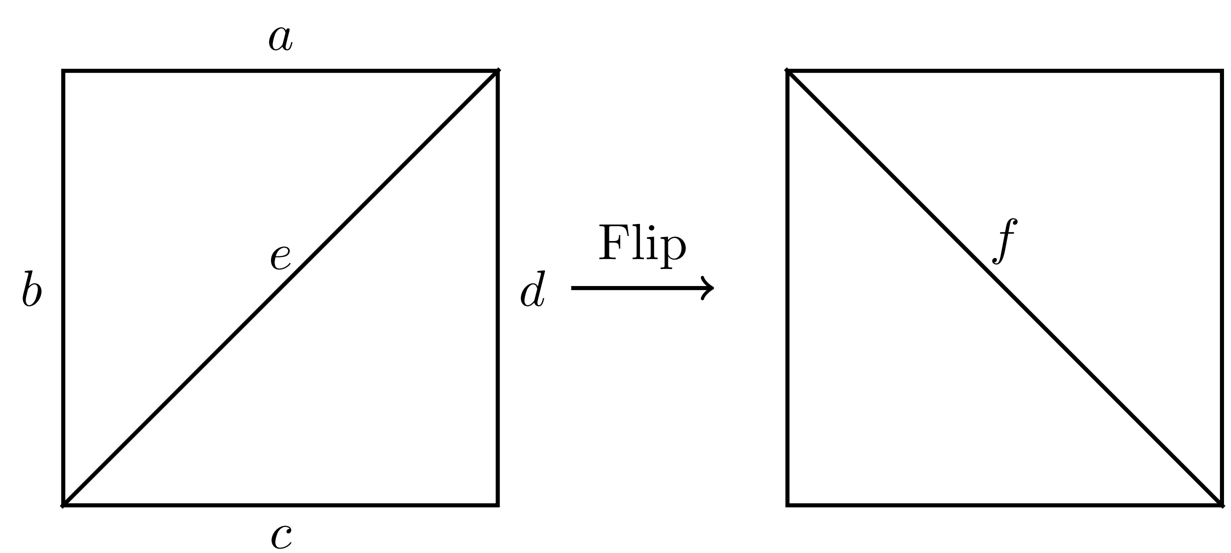
- We can “trace” any simple geodesic through a PL triangulation in $O(n^2 \log X)$ time.
  [Erickson Nayyeri 2013]

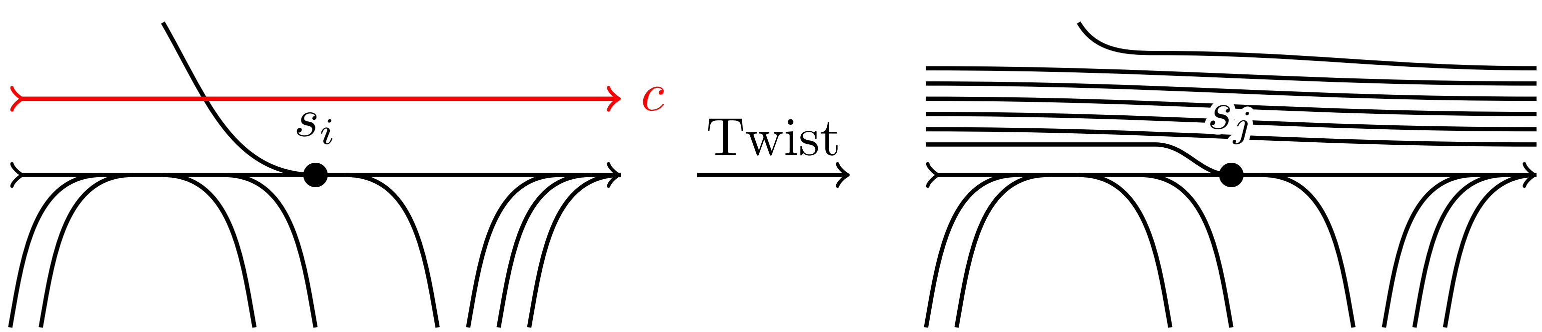


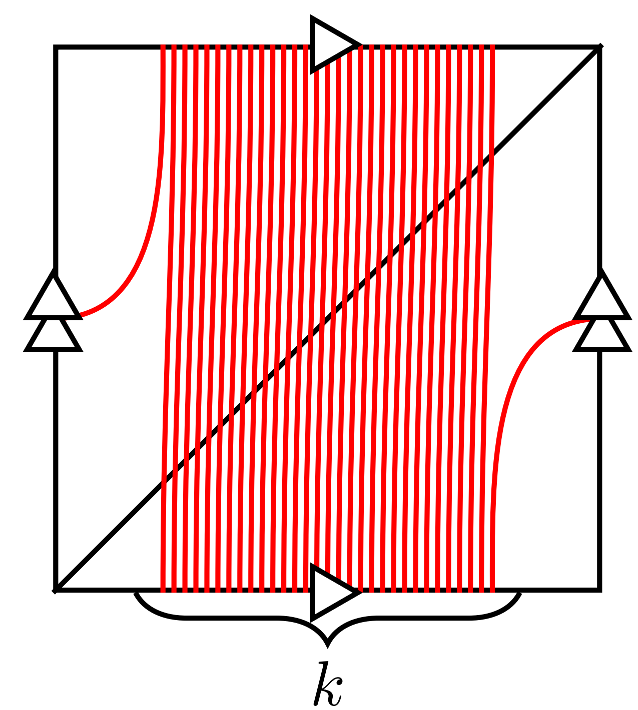


Normal coordinates to the rescue?

- We can compute a minimal (abstract) triangulation for a given normal curve in $O(poly(n \log X))$ time.
  [Bell 2016] [Bell Webb 2016]







Open problems

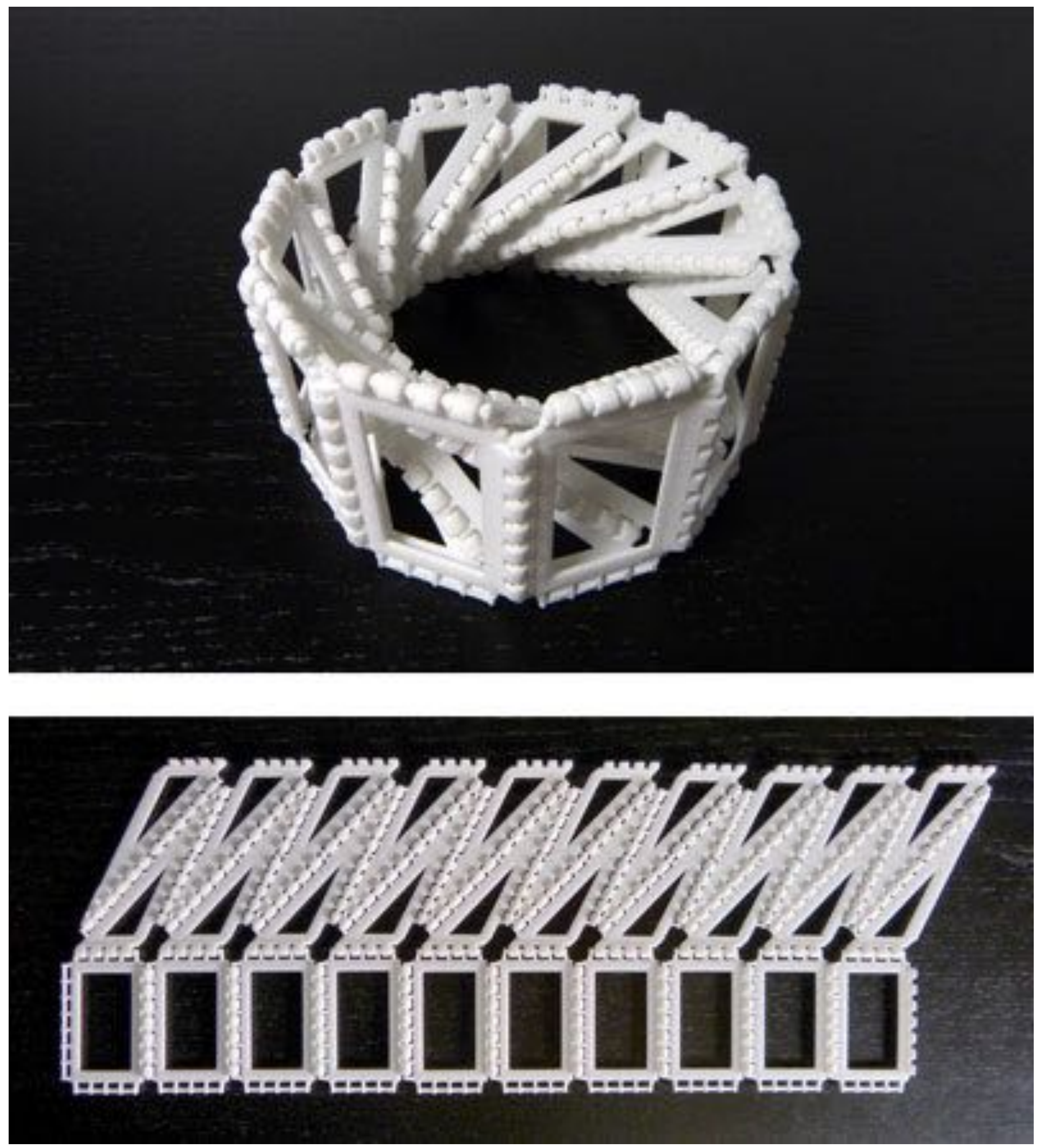
- Can we compute (the normal coordinates of) the shortest nontrivial cycle in an arbitrary triangulated PL surface in $O(poly(n \log \alpha))$ time?

Open problems

- Can we compute (the normal coordinates of) the shortest nontrivial cycle in an arbitrary triangulated PL surface in $O(poly(n \log \alpha))$ time?

- More generally, can we compute a *useful* PL triangulation (for example, the intrinsic Delaunay triangulation) of an arbitrary triangulated PL surface in $O(poly(n \log \alpha))$ time?

Thank you!



[Segerman 2015]