Surface embedded graphs

Erin Wolf Chambers

Why do computational geometers care about topology?

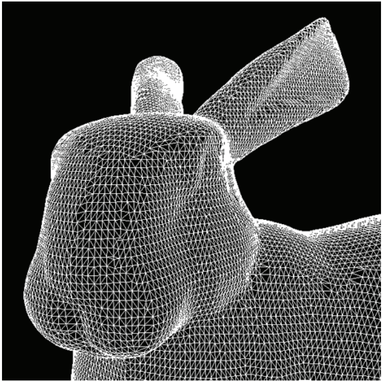
Many areas, such as graphics, biology, robotics, and networking, use algorithms which compute information about point sets.

Why do computational geometers care about topology?

Many areas, such as graphics, biology, robotics, and networking, use algorithms which compute information about point sets.

Surfaces

We've already seen algorithms that compute a mesh of these points in order to represent the original object.



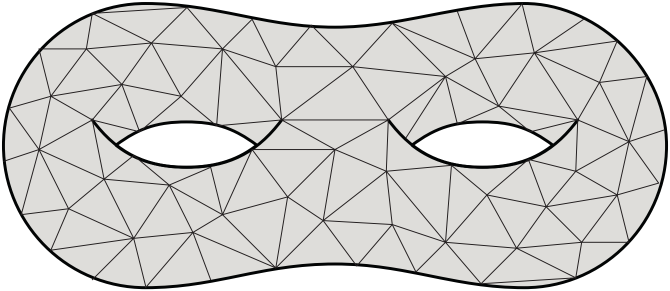
Figures courtesy of Stanford computer graphics laboratory.

Setting

This output mesh naturally leads to a setting that is at the boundaries of graph theory, topology, and geometry.

Setting

This output mesh naturally leads to a setting that is at the boundaries of graph theory, topology, and geometry.

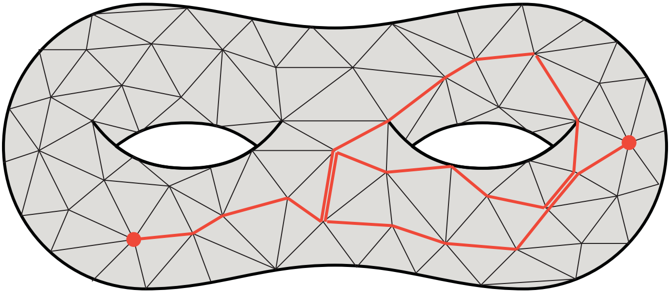


Definition

A combinatorial surface is a 2-manifold which has a weighted graph embedded on its surface so that every face of the graph is a disk.

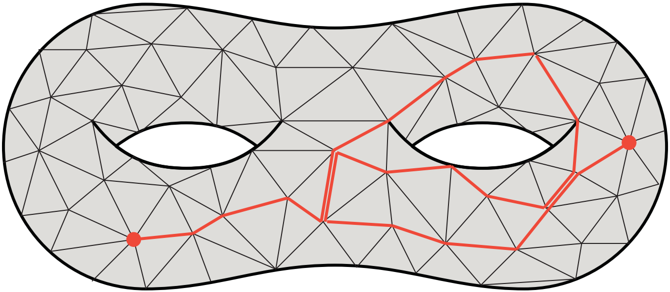
Setting

We will only consider paths and cycles which stay on the edges of the graph.



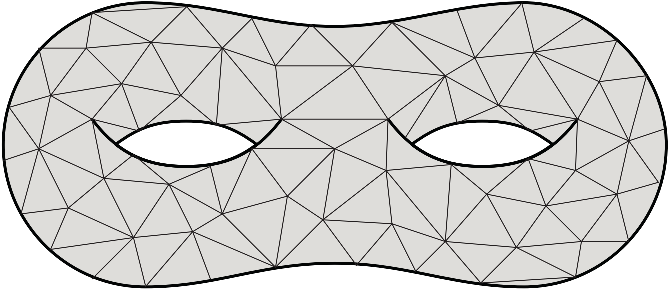
Setting

We will only consider paths and cycles which stay on the edges of the graph.



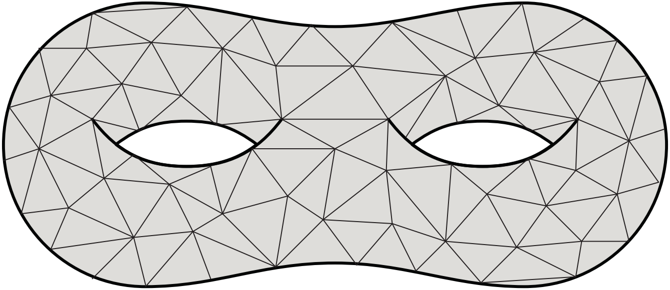
The underlying surface is actually unknown - all we have is the combinatorial structure of the graph (with weights), plus information about faces.

We will let n be the total size of the graph, but this is not the only relevant parameter here.



Definitions

We will let n be the total size of the graph, but this is not the only relevant parameter here.



Any orientable surface is topologically equivalent to a sphere with some number of handles attached to it; this is the *genus* of the surface, g.

Euler characteristic

These parameters are connected:

- For any polyhedral manifold M, we know that $v - e + f = \chi(M)$, the Euler characteristic of M.

Euler characteristic

These parameters are connected:

- For any polyhedral manifold M, we know that $v - e + f = \chi(M)$, the Euler characteristic of M.
- This is independent of the triangulation on M: $\chi = 2 - 2g - k$ if the manifold is orientable, where g is the genus and k is the number of boundaries.

Euler characteristic

These parameters are connected:

- For any polyhedral manifold M, we know that $v - e + f = \chi(M)$, the Euler characteristic of M.
- This is independent of the triangulation on M: $\chi = 2 - 2g - k$ if the manifold is orientable, where g is the genus and k is the number of boundaries.
- This means that if the manifold has v vertices, then it has at most $3v - 6 + 6g$ edges and at most $2v - 4 + 4g - k$ faces. (Equality holds when every face and boundary is a triangle.)

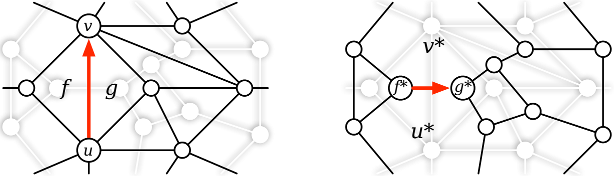
Euler characteristic

These parameters are connected:

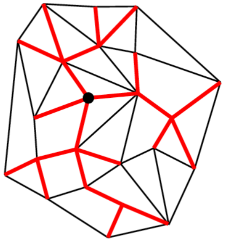
- For any polyhedral manifold M, we know that $v - e + f = \chi(M)$, the Euler characteristic of M.
- This is independent of the triangulation on M: $\chi = 2 - 2g - k$ if the manifold is orientable, where g is the genus and k is the number of boundaries.
- This means that if the manifold has v vertices, then it has at most $3v - 6 + 6g$ edges and at most $2v - 4 + 4g - k$ faces. (Equality holds when every face and boundary is a triangle.)
- Hence, we'll let $n \leq 6v - 10 + 10g - k$ be the total number of edges, faces, and vertices.

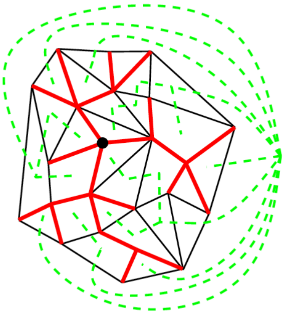
Dual graphs

Given an embedded graphs, we can form the *dual graph*:



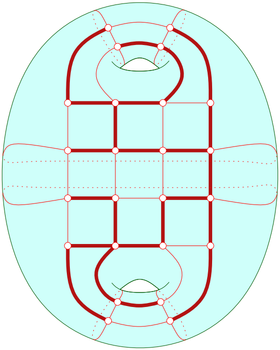
For a planar graph G with a spanning tree T, $G^* \setminus E(T)^*$ is a spanning tree of the dual graph G^*.

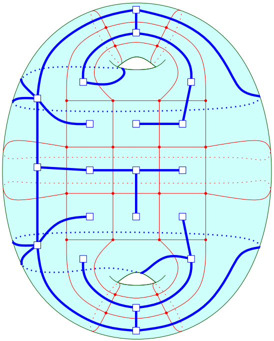




The genus g case

On a surface, we can still consider the dual of a tree, but $G^* \setminus E(T)^*$ is NOT a spanning tree of the dual graph G^*.





Instead, we can decompose into a tree, a co-tree, and $O(g)$ “extra” edges.

What can we compute?

There are many possible questions we can ask in this model, many of which are generalizations of questions on planar graphs.

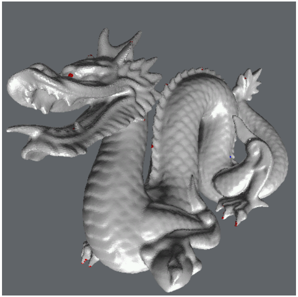
What can we compute?

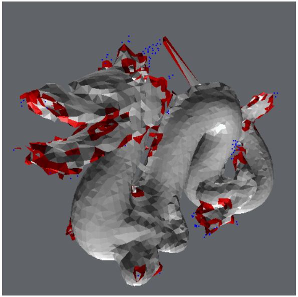
There are many possible questions we can ask in this model, many of which are generalizations of questions on planar graphs.

- How (fast) can we compute topologically interesting cycles?
- How can we tell if two curves are similar to each other?
- Can we tell if two such graphs are isomorphic?
- Can we given efficient ways to morph between two isomorphic graphs?
- Can we compute flows and cuts in these graphs?

Motivation: Finding simple cycles

For example, in graphics algorithms, we wish to make the mesh "look like" the original object, and yet be as compact as possible.

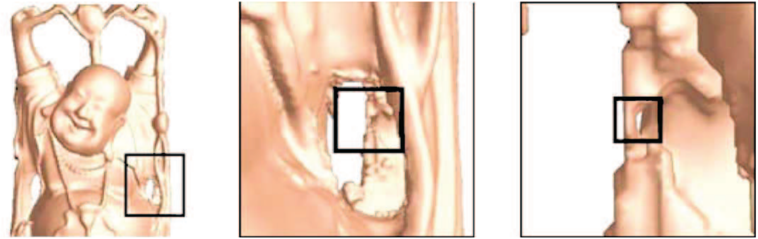




Figures courtesy of Joshua Levine, Univ. of Ohio.

Topological Noise

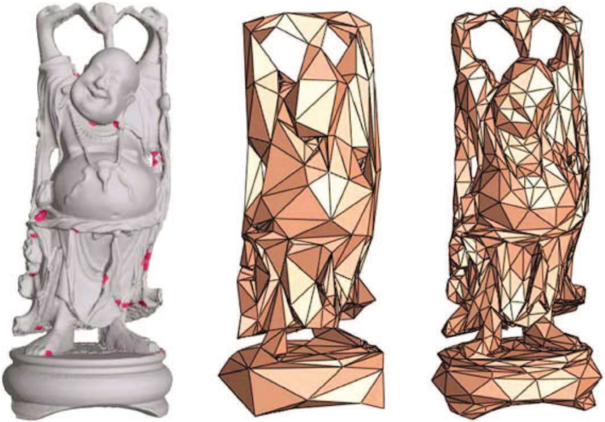
When creating a mesh, small errors can appear.



[Wood-Hoppe-Desbrun-Schroder 2004]

Topological Simplification

Simplification algorithms are hurt by this noise.



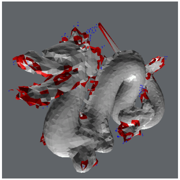
Genus 104 — Genus 104 (2K triangles) — Genus 6 (2K triangles)

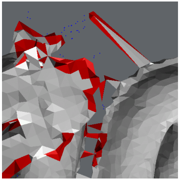
Original scan — Topologically simplified

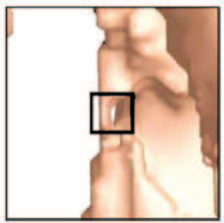
[Wood-Hoppe-Desbrun-Schroder 2004]

Why small cycles?

Extra noise in these examples corresponds to small handles in the mesh.







“Interesting” cycles

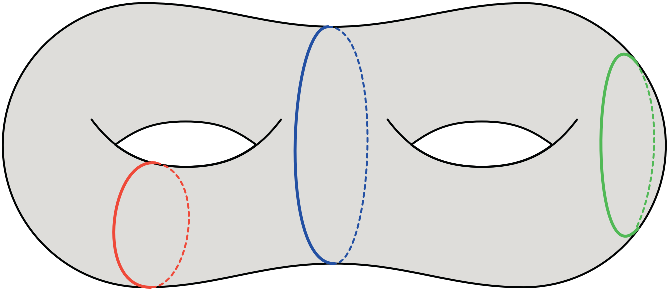
Definition

A homotopy is a continuous deformation of one path to another.
A cycle is contractible if it can be continuously deformed to a point.
A cycle γ is separating if $M - \gamma$ has 2 separate pieces.

"Interesting" cycles

Definition

A homotopy is a continuous deformation of one path to another.
A cycle is contractible if it can be continuously deformed to a point.
A cycle γ is separating if $M - \gamma$ has 2 separate pieces.



Computing interesting cycles

There is a simple algorithm to compute a non-contractible cycle, based on computing a breadth first search tree.

Computing interesting cycles

There is a simple algorithm to compute a non-contractible cycle, based on computing a breadth first search tree.

- Starting from a vertex v of the graph, explore all neighbors of v and add them to the tree. Continue exploring their neighbors, creating "level sets" of vertices at distance k from v (in terms of the number of edges, not actual distance).

Computing interesting cycles

There is a simple algorithm to compute a non-contractible cycle, based on computing a breadth first search tree.

- Starting from a vertex v of the graph, explore all neighbors of v and add them to the tree. Continue exploring their neighbors, creating "level sets" of vertices at distance k from v (in terms of the number of edges, not actual distance).
- If we come to a vertex already in the tree, then we have discovered a cycle. We can test if this is contractible in $O(n)$ time (using a straightforward computation of the Euler characteristic).

Computing interesting cycles

There is a simple algorithm to compute a non-contractible cycle, based on computing a breadth first search tree.

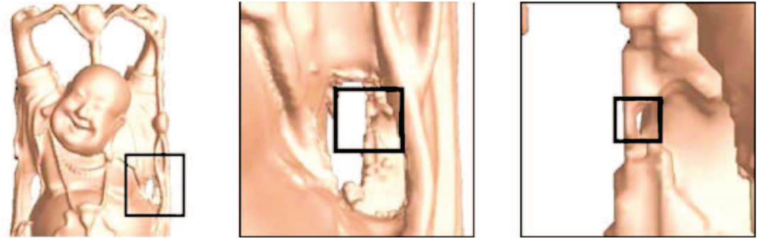
- Starting from a vertex v of the graph, explore all neighbors of v and add them to the tree. Continue exploring their neighbors, creating "level sets" of vertices at distance k from v (in terms of the number of edges, not actual distance).
- If we come to a vertex already in the tree, then we have discovered a cycle. We can test if this is contractible in $O(n)$ time (using a straightforward computation of the Euler characteristic).
- If the cycle is not interesting, then we can continue our search to one "side" of the cycle, since the other will be a disk (and so can be ignored).

Finding small cycles

However, while we can compute these cycles quickly, they are not exactly what we were looking for.

Finding small cycles

However, while we can compute these cycles quickly, they are not exactly what we were looking for.



Recall that *small* non-contractible or non-separating cycles may represent topological noise.

Shortest non-contractible or non-separating cycles

Computing the shortest non-contractible or non-separating cycle on a combinatorial surface has been of considerable interest of late.

Shortest non-contractible or non-separating cycles

Computing the shortest non-contractible or non-separating cycle on a combinatorial surface has been of considerable interest of late.

- $O(n^3)$ [Thomassen 1990]
- $O(n^2 \log n)$ [Erickson-Har-Peled 2002]
- $g^{O(g)} n^{3/2}$ for non-contractible, $g^{O(g)} n^{3/2} \log n$ for non-separating [Cabello-Mohar 2005]
- $g^{O(g)} n^{4/3}$ [Cabello 2006]
- $g^{O(g)} n \log n$ [Kutz 2006]
- $O(g^3 n \log n)$ [Cabello-Chambers 2007]
- $O(g^2 n \log n)$ [Cabello-Chambers-Erickson 2015]

Key Tool

Definition

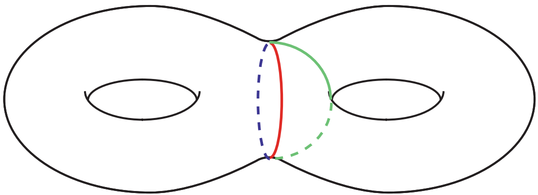
A property satisfies that *3-path condition* if for any three paths α, β, and γ between two points x and y where $\alpha \cdot \beta$ satisfies your property, than either $\alpha \cdot \gamma$ or $\beta \cdot \gamma$ will also satisfy your property.

Key Tool

Definition

A property satisfies that *3-path condition* if for any three paths α, β, and γ between two points x and y where $\alpha \cdot \beta$ satisfies your property, than either $\alpha \cdot \gamma$ or $\beta \cdot \gamma$ will also satisfy your property.

Thomassen proved that the set of non-contractible cycles satisfies the 3-path property.

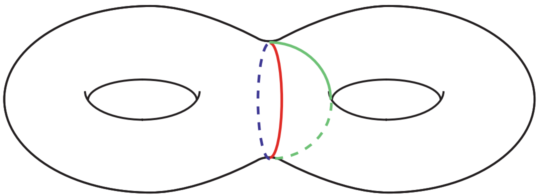


Key Tool

Definition

A property satisfies that *3-path condition* if for any three paths α, β, and γ between two points x and y where $\alpha \cdot \beta$ satisfies your property, than either $\alpha \cdot \gamma$ or $\beta \cdot \gamma$ will also satisfy your property.

Thomassen proved that the set of non-contractible cycles satisfies the 3-path property.



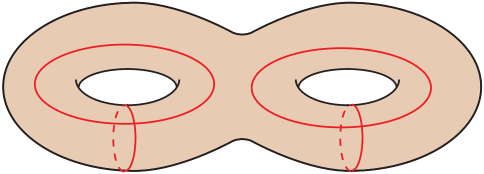
This meant that the shortest non-contractible cycle was composed of 2 shortest paths, which are well studied in the graph theory and algorithms literature.

Faster algorithm for non-separating cycles

Let M be an surface of complexity n and genus g. We want to find a shortest non-separating cycle on M.

Faster algorithm for non-separating cycles

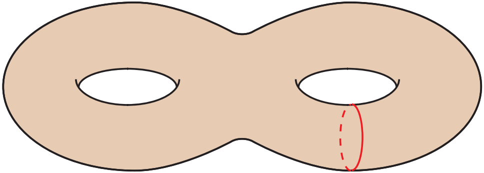
Let M be an surface of complexity n and genus g. We want to find a shortest non-separating cycle on M.



Cabello and Mohar gave an algorithm to compute a particular set of cycles (a homology basis) in $O(gn \log n)$ time.
Key fact - this is a set of $O(g)$ simple loops such that the shortest non-separating cycle must cross one of these loops exactly once.

Faster algorithm for non-separating cycles

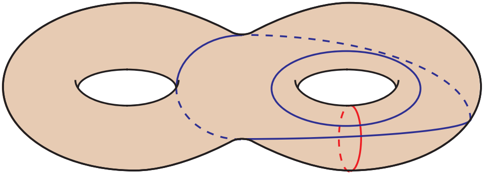
Let M be an surface of complexity n and genus g. We want to find a shortest non-separating cycle on M.



Consider any one of these loops.

Faster algorithm for non-separating cycles

Let M be an surface of complexity n and genus g. We want to find a shortest non-separating cycle on M.



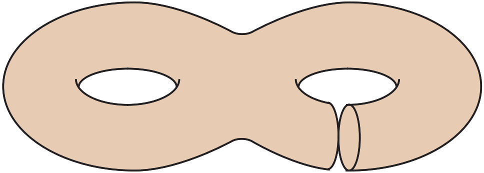
Find the shortest cycle which crosses the loop exactly once.

Crossing cycles

Let α be a simple cycle in M. We want to find a shortest cycle which crosses α exactly once.

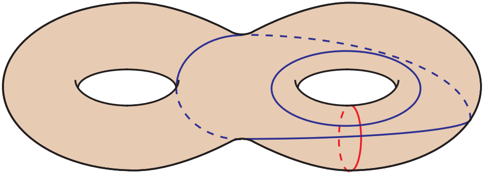
Crossing cycles

Let α be a simple cycle in M. We want to find a shortest cycle which crosses α exactly once.



Consider the surface N obtained by cutting M along α and gluing disks to each of the copies of α.

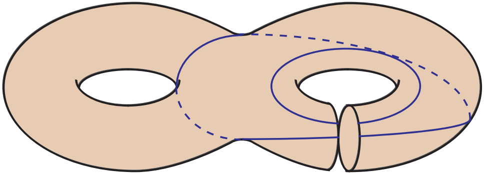
Let α be a simple cycle in M. We want to find a shortest cycle which crosses α exactly once.



Consider the set of cycles crossing α exactly once on M.

Crossing cycles

Let α be a simple cycle in M. We want to find a shortest cycle which crosses α exactly once.



A shortest cycle that crosses α once is a shortest path in N.

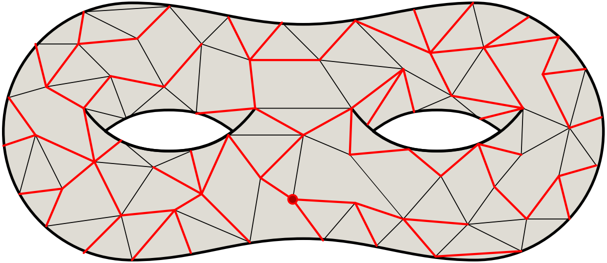
Computing shortest paths

So we now need to be able to compute shortest paths for all the vertices on this face quickly.

Computing shortest paths

So we now need to be able to compute shortest paths for all the vertices on this face quickly.

So the goal is now to compute a shortest path tree, which allows us to find all the relevant shortest paths to consider.



Shortest Path Trees

Definition

A shortest path tree (or sp-tree) in a graph is a tree consisting of all shortest paths from some vertex to all other vertices.

Shortest Path Trees

Definition

A shortest path tree (or sp-tree) in a graph is a tree consisting of all shortest paths from some vertex to all other vertices.

In SODA 2005, Klein showed how to compute sp-trees in a planar graph for all vertices on a single face of the graph. The running time is $O(n \log n)$.

Shortest Path Trees

Definition

A shortest path tree (or sp-tree) in a graph is a tree consisting of all shortest paths from some vertex to all other vertices.

In SODA 2005, Klein showed how to compute sp-trees in a planar graph for all vertices on a single face of the graph. The running time is $O(n \log n)$.
In SODA 2007, we give an algorithm to compute sp-trees for all vertices on a single face of a genus g graph in $O(g^2 n \log n)$ time.

Tense edges

Let $d_T : V \to \mathbb{R}$ be the distance in a rooted directed tree T from the source of the tree to the specified vertex.

Tense edges

Let $d_T : V \to \mathbb{R}$ be the distance in a rooted directed tree T from the source of the tree to the specified vertex.

Definition

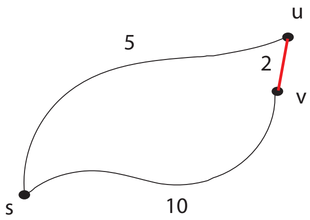
Define the tension of an edge $\overrightarrow{uv}$ as $t(\overrightarrow{uv}) = d(v) - w(\overrightarrow{uv}) - d(u)$.

Tense edges

Let $d_T : V \to \mathbb{R}$ be the distance in a rooted directed tree T from the source of the tree to the specified vertex.

Definition

Define the tension of an edge $\overrightarrow{uv}$ as $t(\overrightarrow{uv}) = d(v) - w(\overrightarrow{uv}) - d(u)$.
We say an edge $\overrightarrow{uv}$ is tense if $t(\overrightarrow{uv}) > 0$.

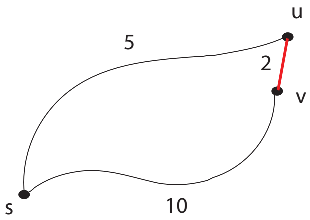


Tense edges

Let $d_T : V \to \mathbb{R}$ be the distance in a rooted directed tree T from the source of the tree to the specified vertex.

Definition

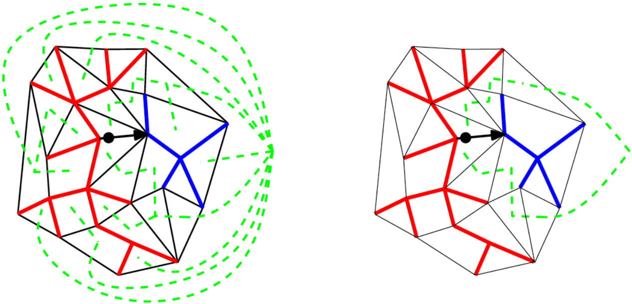
Define the tension of an edge $\overrightarrow{uv}$ as $t(\overrightarrow{uv}) = d(v) - w(\overrightarrow{uv}) - d(u)$.
We say an edge $\overrightarrow{uv}$ is tense if $t(\overrightarrow{uv}) > 0$.



Fact: T is an shortest path tree if and only if no edges are tense.

Maintaining shortest path trees

We consider maintaining a shortest path tree kinetically - while the root is moving from one vertex to another.



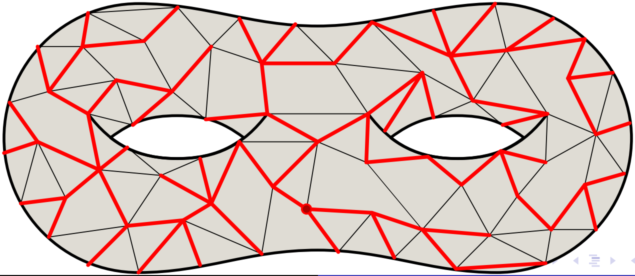
In the dual, the first edge to be tense is on a specific path in the dual tree

Key Lemma

Lemma

We give a data structure to represent shortest path trees in G such that:

- *a distance from the root to a query vertex can be answered in $O(\log n)$ time;*
- *the shortest path tree rooted at u can be changed to the tree rooted at a neighbor of u in $O(k \log n)$ time, where k is the number of edges entering or leaving the trees.*

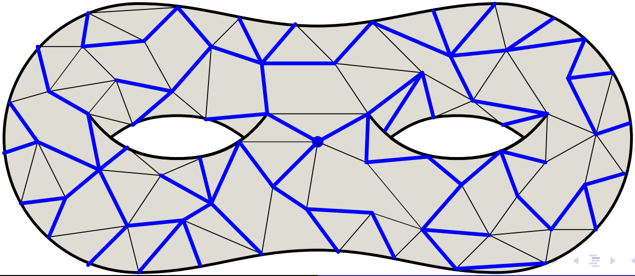


Key Lemma

Lemma

We give a data structure to represent shortest path trees in G such that:

- *a distance from the root to a query vertex can be answered in $O(\log n)$ time;*
- *the shortest path tree rooted at u can be changed to the tree rooted at a neighbor of u in $O(k \log n)$ time, where k is the number of edges entering or leaving the trees.*

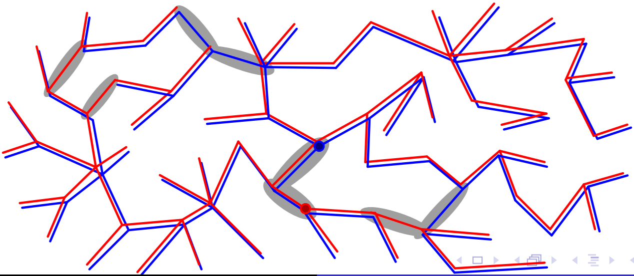


Key Lemma

Lemma

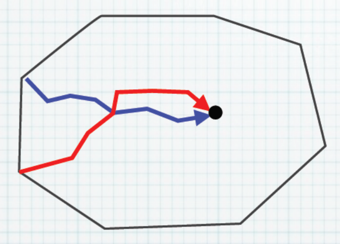
We give a data structure to represent shortest path trees in G such that:

- *a distance from the root to a query vertex can be answered in $O(\log n)$ time;*
- *the shortest path tree rooted at u can be changed to the tree rooted at a neighbor of u in $O(k \log n)$ time, where k is the number of edges entering or leaving the trees.*



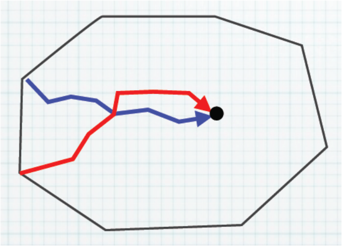
The planar data structure

In a planar graph, an edge can only swap in or out of the shortest path tree a constant number of times when the root of the tree moves around a face.



[Klein 2005]

In a planar graph, an edge can only swap in or out of the shortest path tree a constant number of times when the root of the tree moves around a face.

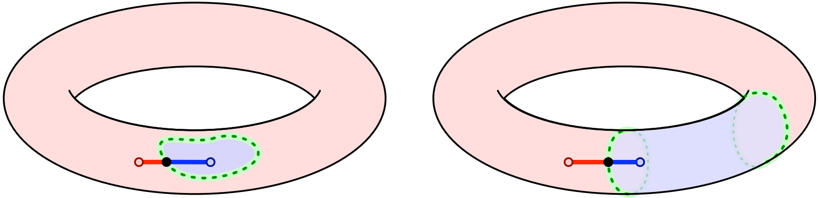


[Klein 2005]

Using this fact, we can in $O(n \log n)$ time construct our data structure which supports shortest path queries for any vertex on a common face in $O(\log n)$ time.

Our decomposition

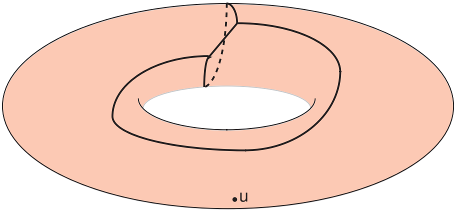
- Red vertices: the root is getting further from them.
- Blue vertices: the root is getting closer to them.
- Green (dual) edges: dual to an edge with both red and blue endpoints.



The green edges

Green edges are still the potential tense edges.

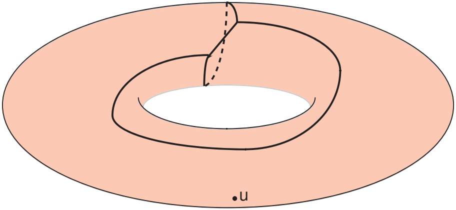
Here, they form $O(g)$ paths (called a cut graph) in $G^* \setminus E(T)^*$.



The green edges

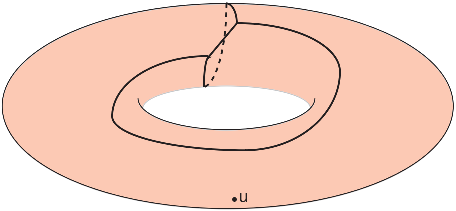
Green edges are still the potential tense edges.

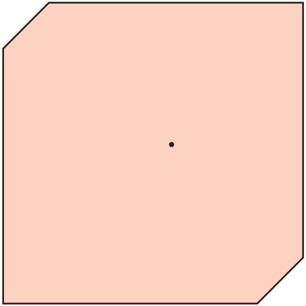
Here, they form $O(g)$ paths (called a cut graph) in $G^* \setminus E(T)^*$.



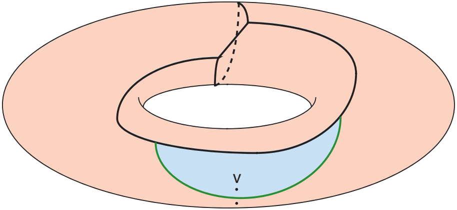
Key idea: Use $O(g)$ different trees to track the tensions of $G^* \setminus E(T)^*$; we call this a grove.

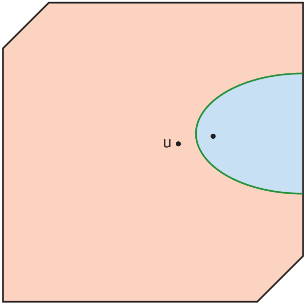
A Genus 1 Example



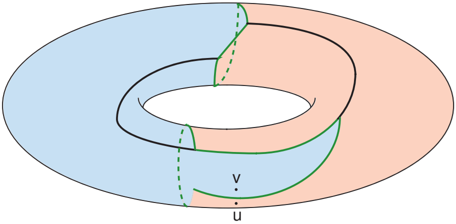


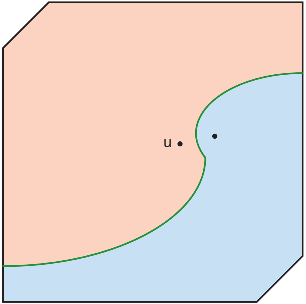
A Genus 1 Example





A Genus 1 Example





How many times to relax?

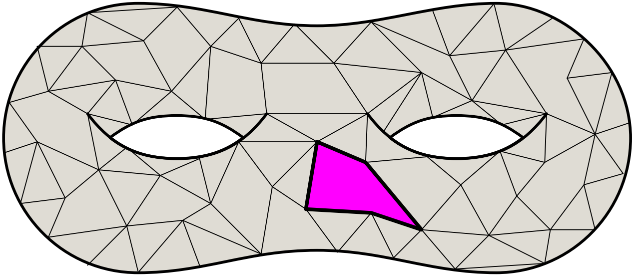
So we can now relax tense edges quickly. It remains to bound the number of times an edge can enter or leave the shortest path tree.

How many times to relax?

So we can now relax tense edges quickly. It remains to bound the number of times an edge can enter or leave the shortest path tree.

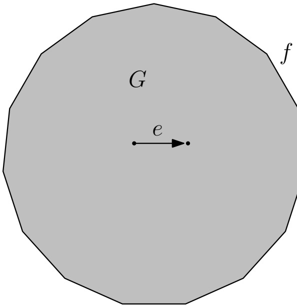
Lemma

As the root of the shortest path tree moves along a face, each edge enters or leaves the tree $O(g)$ times.



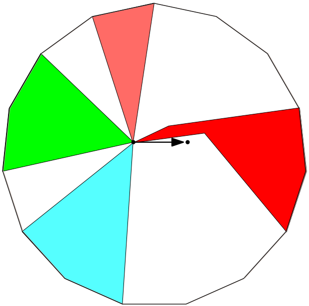
Sketch of the proof

Consider a directed edge e in the graph



Sketch of the proof

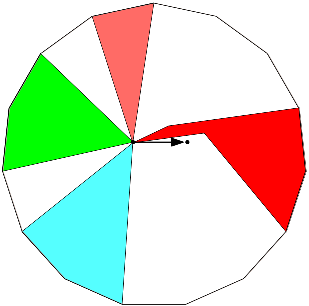
Consider a directed edge e in the graph, and the roots where it appears at the tree.



In the planar case, there was a single contiguous region on the boundary where this edge appeared, since it could swap in and out at most once.

Sketch of the proof

Consider a directed edge e in the graph, and the roots where it appears at the tree.



In the planar case, there was a single contiguous region on the boundary where this edge appeared, since it could swap in and out at most once.

Here, each contiguous region describes a relative homotopy class of curves, so instead we get at most $O(g)$ pieces.

The End

Thanks for your attention!

Questions?