

Task-based Augmented Reeb Graphs with Dynamic ST-Trees

C. Gueunet P. Fortin J. Jomier J. Tierny

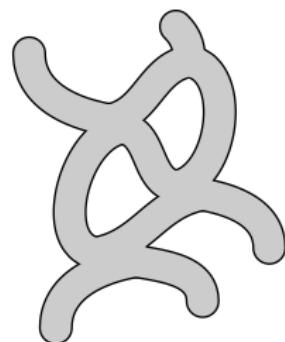
Eurographics 2019

Reeb graph

Definition

Given:

- ▶ a manifold \mathcal{M} ;

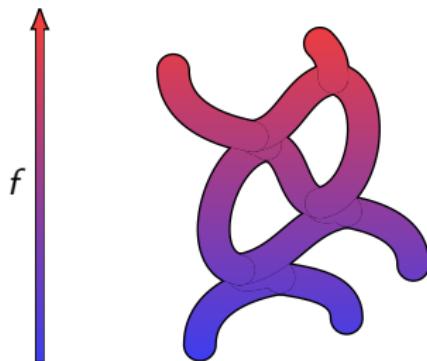


Reeb graph

Definition

Given:

- ▶ a manifold \mathcal{M} ;
- ▶ a Morse function $f: \mathcal{M} \rightarrow \mathbb{R}$ with distinct critical values;



Reeb graph

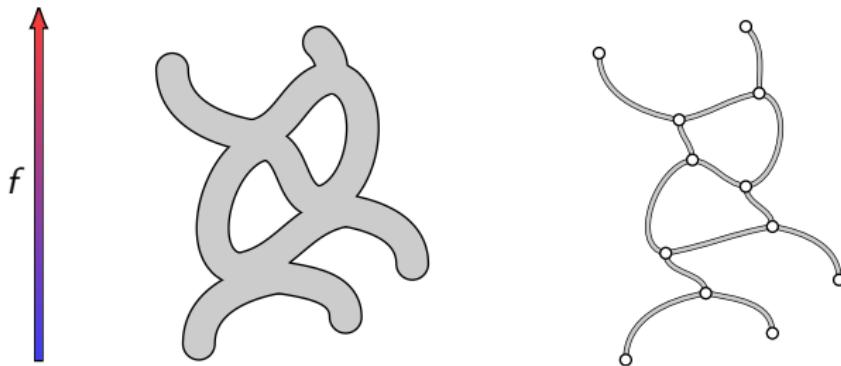
Definition

Given:

- ▶ a manifold \mathcal{M} ;
- ▶ a Morse function $f: \mathcal{M} \rightarrow \mathbb{R}$ with distinct critical values;

the **Reeb graph** of f is the 1-dimensional simplicial complex

$$\mathcal{R}(f) = \mathcal{M}/\sim.$$



Reeb graph

Definition

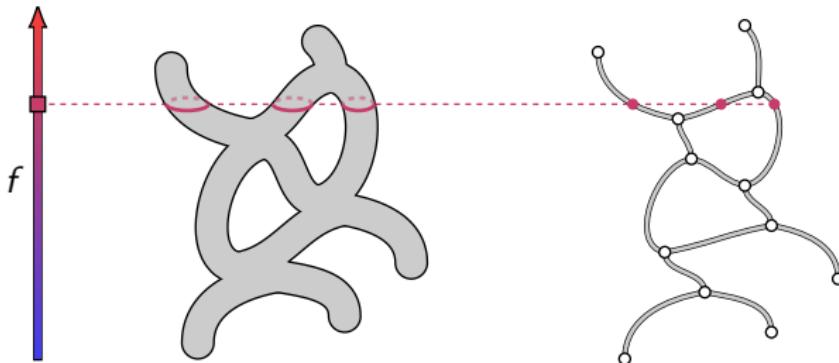
Given:

- ▶ a manifold \mathcal{M} ;
- ▶ a Morse function $f: \mathcal{M} \rightarrow \mathbb{R}$ with distinct critical values;

the **Reeb graph** of f is the 1-dimensional simplicial complex

$$\mathcal{R}(f) = \mathcal{M} / \sim,$$

$x \sim y$ if $f(x) = f(y)$ and they belong to the same connected component of $f^{-1}(f(x))$



Reeb graph

Definition

Given:

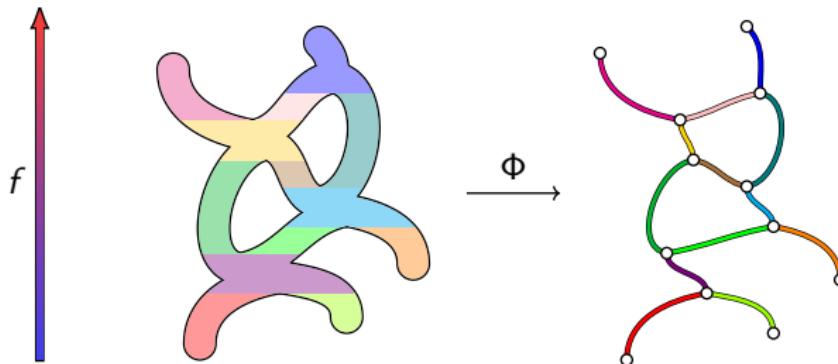
- ▶ a manifold \mathcal{M} ;
- ▶ a Morse function $f: \mathcal{M} \rightarrow \mathbb{R}$ with distinct critical values;

the **Reeb graph** of f is the 1-dimensional simplicial complex

$$\mathcal{R}(f) = \mathcal{M}/\sim.$$

The **segmentation map** is the quotient map

$$\Phi: \mathcal{M} \longrightarrow \mathcal{R}(f).$$

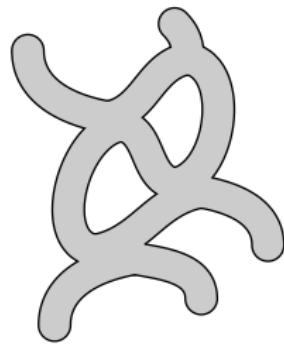


Reeb graph

Desired algorithm

Input:

- ▶ a PL manifold \mathcal{M}

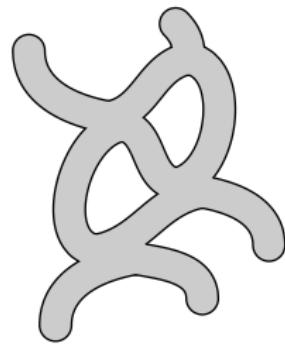


Reeb graph

Desired algorithm

Input:

- ▶ a PL manifold \mathcal{M}
 \rightsquigarrow a triangulated mesh \mathcal{M} ;

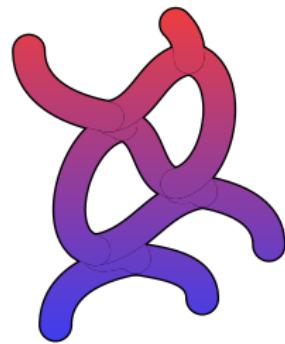


Reeb graph

Desired algorithm

Input:

- ▶ a PL manifold \mathcal{M}
~~ a triangulated mesh \mathcal{M} ;
- ▶ a non-degenerate PL scalar field f on \mathcal{M}



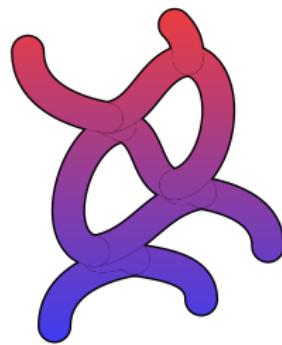
Reeb graph

Desired algorithm

Input:

- ▶ a PL manifold \mathcal{M}
~~ a triangulated mesh \mathcal{M} ;
- ▶ a non-degenerate PL scalar field f on \mathcal{M}
~~ a **scalar value** $f(v)$ for each vertex v of \mathcal{M} .

→ pairwise different, in order to ensure non-degeneracy; this can be achieved by random perturbations



Reeb graph

Desired algorithm

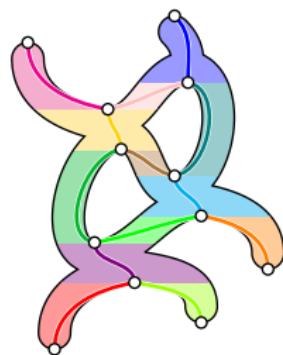
Input:

- ▶ a PL manifold \mathcal{M}
~~ a triangulated mesh \mathcal{M} ;
- ▶ a non-degenerate PL scalar field f on \mathcal{M}
~~ a scalar value $f(v)$ for each vertex v of \mathcal{M} .

Output:

- ▶ the **augmented** Reeb graph $\mathcal{R}(f)$.

graph + segmentation map

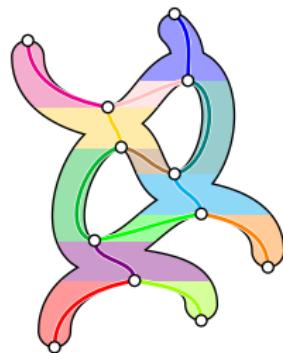


Reeb graph

Desired algorithm

Input:

- ▶ a PL manifold \mathcal{M}
~~ a triangulated mesh \mathcal{M} ;
- ▶ a non-degenerate PL scalar field f on \mathcal{M}
~~ a scalar value $f(v)$ for each vertex v of \mathcal{M} .



Output:

- ▶ the augmented Reeb graph $\mathcal{R}(f)$.

Time complexity:

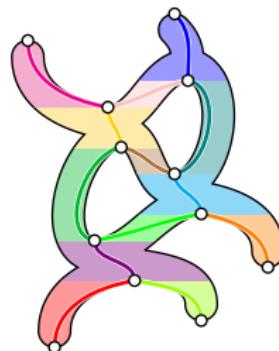
- ▶ $O(m \cdot \log m)$, where m is the size of the 2-skeleton of \mathcal{M} .
#vertices + #edges + #triangles ↗

Reeb graph

Desired algorithm

Input:

- ▶ a PL manifold \mathcal{M}
~~ a triangulated mesh \mathcal{M} ;
- ▶ a non-degenerate PL scalar field f on \mathcal{M}
~~ a scalar value $f(v)$ for each vertex v of \mathcal{M} .



Output:

- ▶ the augmented Reeb graph $\mathcal{R}(f)$.

Time complexity:

- ▶ $O(m \cdot \log m)$, where m is the size of the 2-skeleton of \mathcal{M} .

Parallel.

Reeb graph

Geometry of critical points

There are three kinds of critical points:

Reeb graph

Geometry of critical points

There are three kinds of critical points:

- ▶ (local) **maxima**



Reeb graph

Geometry of critical points

There are three kinds of critical points:

- ▶ (local) **maxima**



- ▶ (local) **minima**



Reeb graph

Geometry of critical points

There are three kinds of critical points:

- ▶ (local) **maxima**



- ▶ (local) **minima**



- ▶ **saddles**



Reeb graph

Geometry of critical points

There are three kinds of critical points:

- ▶ (local) **maxima**



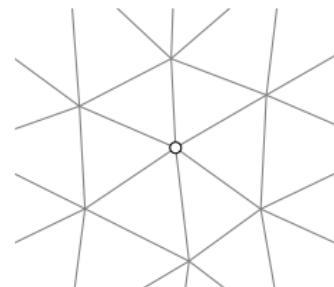
- ▶ (local) **minima**



- ▶ **saddles**



How to detect them on a PL manifold?



Reeb graph

Geometry of critical points

There are three kinds of critical points:

- ▶ (local) **maxima**



- ▶ (local) **minima**

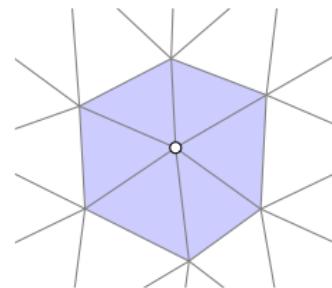


- ▶ **saddles**



How to detect them on a PL manifold?

Given a vertex v , the **star** of v is the union of all simplices containing v .



Reeb graph

Geometry of critical points

There are three kinds of critical points:

- ▶ (local) **maxima**



- ▶ (local) **minima**



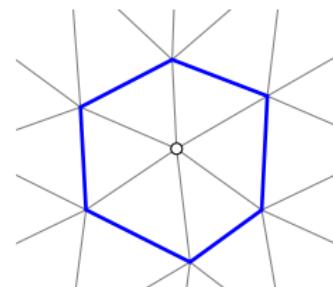
- ▶ **saddles**



How to detect them on a PL manifold?

Given a vertex v , the **star** of v is the union of all simplices containing v .

The **link** of v is the boundary of its star.



Reeb graph

Geometry of critical points

There are three kinds of critical points:

- ▶ (local) **maxima**



- ▶ (local) **minima**



- ▶ **saddles**



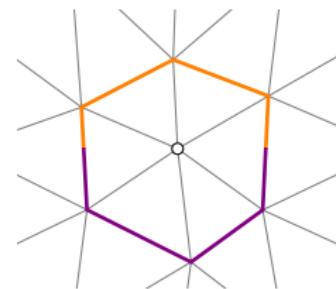
How to detect them on a PL manifold?

Given a vertex v , the **star** of v is the union of all simplices containing v .

The **link** of v is the boundary of its star.

$$\text{Link}^+(v) = \{x \in \text{Link}(v) : f(x) > f(v)\}$$

$$\text{Link}^-(v) = \{x \in \text{Link}(v) : f(x) < f(v)\}$$



Reeb graph

Geometry of critical points

There are three kinds of critical points:

- ▶ (local) **maxima**
~~ Link⁺ empty;
- ▶ (local) **minima**
- ▶ **saddles**



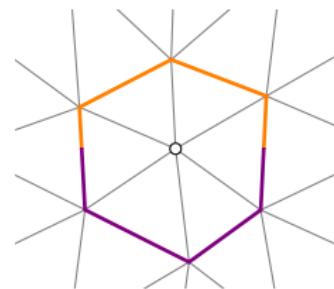
How to detect them on a PL manifold?

Given a vertex v , the **star** of v is the union of all simplices containing v .

The **link** of v is the boundary of its star.

$$\text{Link}^+(v) = \{x \in \text{Link}(v) : f(x) > f(v)\}$$

$$\text{Link}^-(v) = \{x \in \text{Link}(v) : f(x) < f(v)\}$$



Reeb graph

Geometry of critical points

There are three kinds of critical points:

- ▶ (local) **maxima**
~~~  $\text{Link}^+$  empty;
- ▶ (local) **minima**  
~~~  $\text{Link}^-$  empty;
- ▶ **saddles**



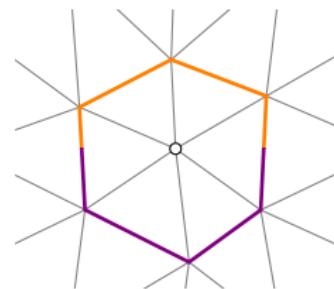
How to detect them on a PL manifold?

Given a vertex v , the **star** of v is the union of all simplices containing v .

The **link** of v is the boundary of its star.

$$\text{Link}^+(v) = \{x \in \text{Link}(v) : f(x) > f(v)\}$$

$$\text{Link}^-(v) = \{x \in \text{Link}(v) : f(x) < f(v)\}$$



Reeb graph

Geometry of critical points

There are three kinds of critical points:

- ▶ (local) **maxima**
~~ Link^+ empty;
- ▶ (local) **minima**
~~ Link^- empty;
- ▶ **saddles**
~~ Link^+ or Link^- disconnected.



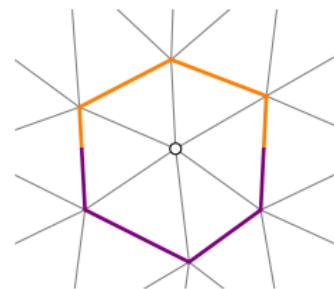
How to detect them on a PL manifold?

Given a vertex v , the **star** of v is the union of all simplices containing v .

The **link** of v is the boundary of its star.

$$\text{Link}^+(v) = \{x \in \text{Link}(v) : f(x) > f(v)\}$$

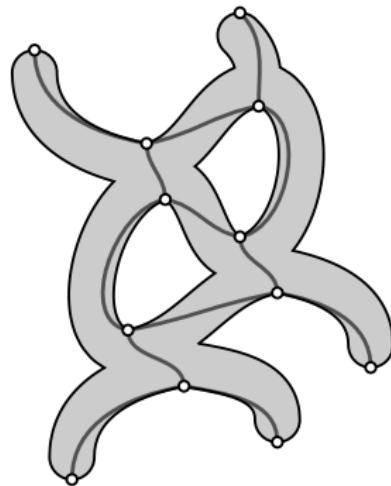
$$\text{Link}^-(v) = \{x \in \text{Link}(v) : f(x) < f(v)\}$$



Reeb graph

Significance of critical points

The critical points of f are closely related to the topology of the Reeb graph $\mathcal{R}(f)$.

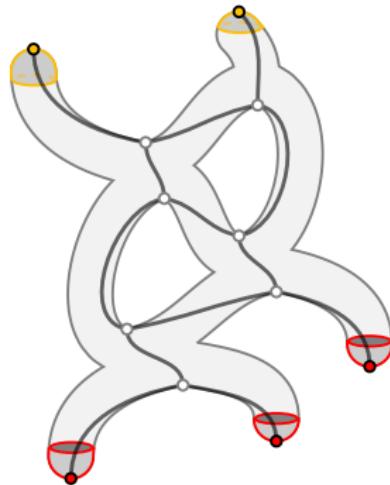


Reeb graph

Significance of critical points

The critical points of f are closely related to the topology of the Reeb graph $\mathcal{R}(f)$.

- ▶ **Maxima** and **minima** \rightsquigarrow nodes of valence 1 (leaves).

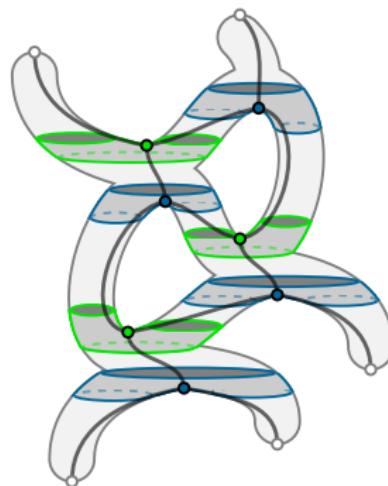


Reeb graph

Significance of critical points

The critical points of f are closely related to the topology of the Reeb graph $\mathcal{R}(f)$.

- ▶ **Maxima** and **minima** \rightsquigarrow nodes of valence 1 (leaves).
- ▶ **Saddles** \rightsquigarrow nodes of valence ≥ 2 .

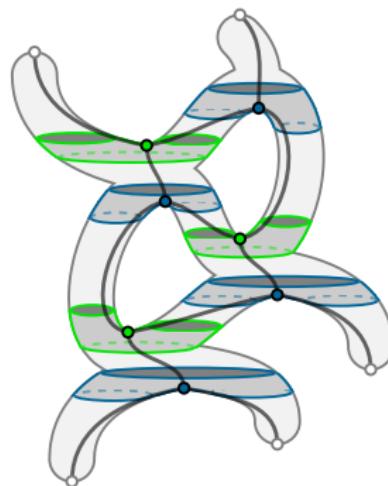


Reeb graph

Significance of critical points

The critical points of f are closely related to the topology of the Reeb graph $\mathcal{R}(f)$.

- ▶ **Maxima** and **minima** \rightsquigarrow nodes of valence 1 (leaves).
- ▶ **Saddles** \rightsquigarrow nodes of valence ≥ 2 .
 - ▶ **Join saddles**: multiple components below.
 - ▶ **Split saddles**: multiple components above.

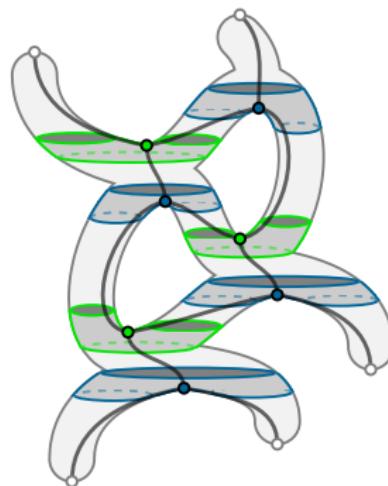


Reeb graph

Significance of critical points

The critical points of f are closely related to the topology of the Reeb graph $\mathcal{R}(f)$.

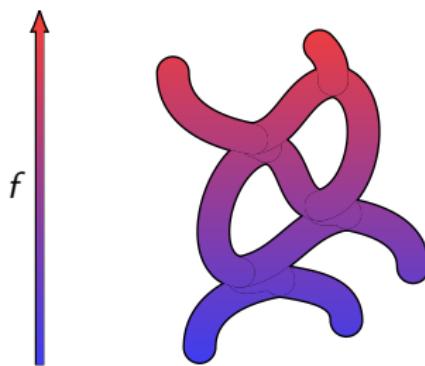
- ▶ **Maxima** and **minima** \rightsquigarrow nodes of valence 1 (leaves).
 - ▶ **Saddles** \rightsquigarrow nodes of valence ≥ 2 .
 - ▶ **Join saddles:** multiple components below.
 - ▶ **Split saddles:** multiple components above.
- } non-mutually exclusive
in dimension ≥ 3



Sequential algorithm

Informal description

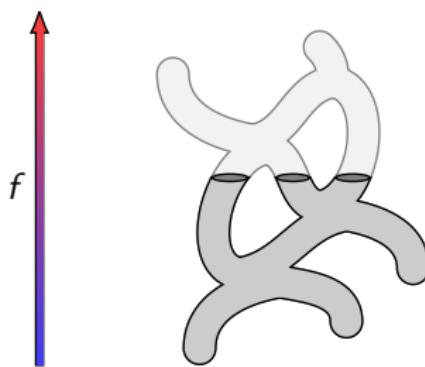
- ▶ Process the vertices of the mesh by **increasing** value of f .



Sequential algorithm

Informal description

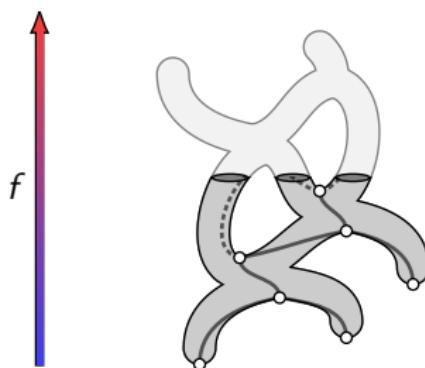
- ▶ Process the vertices of the mesh by **increasing** value of f .
- ▶ Construct the Reeb graph $\mathcal{R}(f)$ incrementally.



Sequential algorithm

Informal description

- ▶ Process the vertices of the mesh by **increasing** value of f .
- ▶ Construct the Reeb graph $\mathcal{R}(f)$ incrementally.
- ▶ While sweeping upwards, keep:
 - ▶ the **partial Reeb graph** constructed so far;

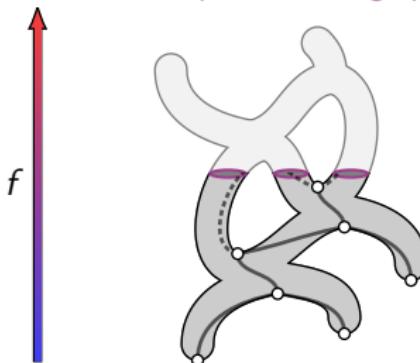


Sequential algorithm

Informal description

- ▶ Process the vertices of the mesh by **increasing** value of f .
- ▶ Construct the Reeb graph $\mathcal{R}(f)$ incrementally.
- ▶ While sweeping upwards, keep:
 - ▶ the **partial Reeb graph** constructed so far;
 - ▶ the current **level set** $f^{-1}(r)$.

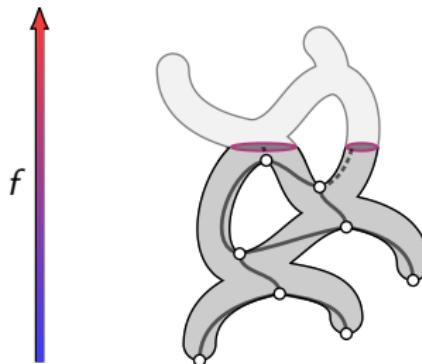
each connected component
corresponds to an open arc
of the partial Reeb graph



Sequential algorithm

Informal description

- ▶ Process the vertices of the mesh by **increasing** value of f .
- ▶ Construct the Reeb graph $\mathcal{R}(f)$ incrementally.
- ▶ While sweeping upwards, keep:
 - ▶ the **partial Reeb graph** constructed so far;
 - ▶ the current **level set** $f^{-1}(r)$.
- ▶ When processing a vertex, **update** the level set and the Reeb graph accordingly.



Sequential algorithm

The preimage graph

The level set $f^{-1}(r)$ can be represented by an abstract **graph** G_r :

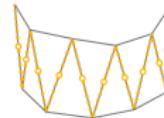


Sequential algorithm

The preimage graph

The level set $f^{-1}(r)$ can be represented by an abstract **graph** G_r :

- ▶ **nodes** \rightsquigarrow edges of the mesh \mathcal{M} ;



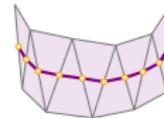
Sequential algorithm

The preimage graph

The level set $f^{-1}(r)$ can be represented by an abstract **graph** G_r :

- ▶ **nodes** \rightsquigarrow edges of the mesh \mathcal{M} ;
- ▶ **arcs** \rightsquigarrow triangles of \mathcal{M} intersecting $f^{-1}(r)$.

→ a triangle connects its two sides intersecting $f^{-1}(r)$

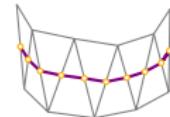


Sequential algorithm

The preimage graph

The level set $f^{-1}(r)$ can be represented by an abstract **graph** G_r :

- ▶ **nodes** \rightsquigarrow edges of the mesh \mathcal{M} ;
- ▶ **arcs** \rightsquigarrow triangles of \mathcal{M} intersecting $f^{-1}(r)$.



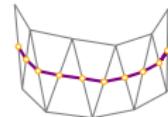
Updating G_r

Sequential algorithm

The preimage graph

The level set $f^{-1}(r)$ can be represented by an abstract **graph** G_r :

- ▶ **nodes** \rightsquigarrow edges of the mesh \mathcal{M} ;
- ▶ **arcs** \rightsquigarrow triangles of \mathcal{M} intersecting $f^{-1}(r)$.



Updating G_r

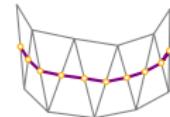
- ▶ **Trigger:** **update** when processing a vertex v .
 - ▶ from $r = f(v) - \epsilon$ to $r = f(v) + \epsilon$

Sequential algorithm

The preimage graph

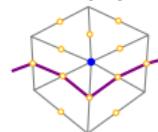
The level set $f^{-1}(r)$ can be represented by an abstract **graph** G_r :

- ▶ **nodes** \rightsquigarrow edges of the mesh \mathcal{M} ;
- ▶ **arcs** \rightsquigarrow triangles of \mathcal{M} intersecting $f^{-1}(r)$.



Updating G_r

- ▶ **Trigger:** update when processing a vertex v .
- ▶ **Action:** process each triangle T of $\text{Star}(v)$ separately.

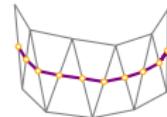


Sequential algorithm

The preimage graph

The level set $f^{-1}(r)$ can be represented by an abstract **graph** G_r :

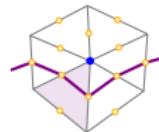
- ▶ **nodes** \rightsquigarrow edges of the mesh \mathcal{M} ;
- ▶ **arcs** \rightsquigarrow triangles of \mathcal{M} intersecting $f^{-1}(r)$.



Updating G_r

- ▶ **Trigger:** update when processing a vertex v .
- ▶ **Action:** process each triangle T of $\text{Star}(v)$ separately.

1. v is the upper vertex of T .

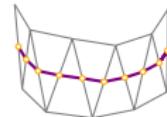


Sequential algorithm

The preimage graph

The level set $f^{-1}(r)$ can be represented by an abstract **graph** G_r :

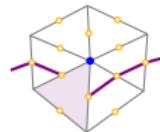
- ▶ **nodes** \rightsquigarrow edges of the mesh \mathcal{M} ;
- ▶ **arcs** \rightsquigarrow triangles of \mathcal{M} intersecting $f^{-1}(r)$.



Updating G_r

- ▶ **Trigger:** update when processing a vertex v .
- ▶ **Action:** process each triangle T of $\text{Star}(v)$ separately.

1. v is the upper vertex of T .

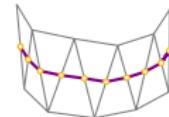


Sequential algorithm

The preimage graph

The level set $f^{-1}(r)$ can be represented by an abstract **graph** G_r :

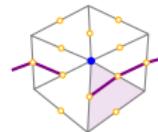
- ▶ **nodes** \rightsquigarrow edges of the mesh \mathcal{M} ;
- ▶ **arcs** \rightsquigarrow triangles of \mathcal{M} intersecting $f^{-1}(r)$.



Updating G_r

- ▶ **Trigger:** update when processing a vertex v .
- ▶ **Action:** process each triangle T of $\text{Star}(v)$ separately.

1. v is the upper vertex of T .

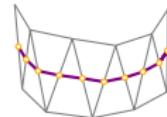


Sequential algorithm

The preimage graph

The level set $f^{-1}(r)$ can be represented by an abstract **graph** G_r :

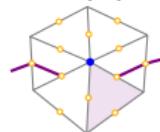
- ▶ **nodes** \rightsquigarrow edges of the mesh \mathcal{M} ;
- ▶ **arcs** \rightsquigarrow triangles of \mathcal{M} intersecting $f^{-1}(r)$.



Updating G_r

- ▶ **Trigger:** update when processing a vertex v .
- ▶ **Action:** process each triangle T of $\text{Star}(v)$ separately.

1. v is the upper vertex of T .

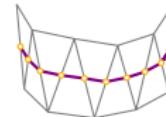


Sequential algorithm

The preimage graph

The level set $f^{-1}(r)$ can be represented by an abstract **graph** G_r :

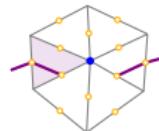
- ▶ **nodes** \rightsquigarrow edges of the mesh \mathcal{M} ;
- ▶ **arcs** \rightsquigarrow triangles of \mathcal{M} intersecting $f^{-1}(r)$.



Updating G_r

- ▶ **Trigger:** update when processing a vertex v .
- ▶ **Action:** process each triangle T of $\text{Star}(v)$ separately.

1. v is the upper vertex of T .
2. v is the middle vertex of T .

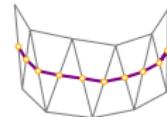


Sequential algorithm

The preimage graph

The level set $f^{-1}(r)$ can be represented by an abstract **graph** G_r :

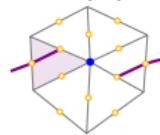
- ▶ **nodes** \rightsquigarrow edges of the mesh \mathcal{M} ;
- ▶ **arcs** \rightsquigarrow triangles of \mathcal{M} intersecting $f^{-1}(r)$.



Updating G_r

- ▶ **Trigger:** update when processing a vertex v .
- ▶ **Action:** process each triangle T of $\text{Star}(v)$ separately.

1. v is the upper vertex of T .
2. v is the middle vertex of T .

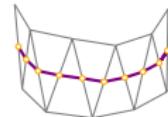


Sequential algorithm

The preimage graph

The level set $f^{-1}(r)$ can be represented by an abstract **graph** G_r :

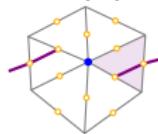
- ▶ **nodes** \rightsquigarrow edges of the mesh \mathcal{M} ;
- ▶ **arcs** \rightsquigarrow triangles of \mathcal{M} intersecting $f^{-1}(r)$.



Updating G_r

- ▶ **Trigger:** update when processing a vertex v .
- ▶ **Action:** process each triangle T of $\text{Star}(v)$ separately.

1. v is the upper vertex of T .
2. v is the middle vertex of T .

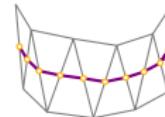


Sequential algorithm

The preimage graph

The level set $f^{-1}(r)$ can be represented by an abstract **graph** G_r :

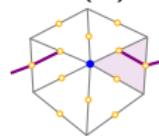
- ▶ **nodes** \rightsquigarrow edges of the mesh \mathcal{M} ;
- ▶ **arcs** \rightsquigarrow triangles of \mathcal{M} intersecting $f^{-1}(r)$.



Updating G_r

- ▶ **Trigger:** update when processing a vertex v .
- ▶ **Action:** process each triangle T of $\text{Star}(v)$ separately.

1. v is the upper vertex of T .
2. v is the middle vertex of T .

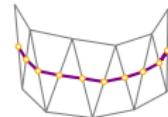


Sequential algorithm

The preimage graph

The level set $f^{-1}(r)$ can be represented by an abstract **graph** G_r :

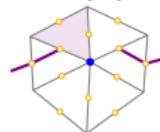
- ▶ **nodes** \rightsquigarrow edges of the mesh \mathcal{M} ;
- ▶ **arcs** \rightsquigarrow triangles of \mathcal{M} intersecting $f^{-1}(r)$.



Updating G_r

- ▶ **Trigger:** update when processing a vertex v .
- ▶ **Action:** process each triangle T of $\text{Star}(v)$ separately.

1. v is the upper vertex of T .
2. v is the middle vertex of T .
3. v is the lower vertex of T .

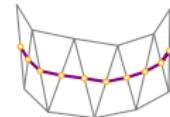


Sequential algorithm

The preimage graph

The level set $f^{-1}(r)$ can be represented by an abstract **graph** G_r :

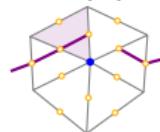
- ▶ **nodes** \rightsquigarrow edges of the mesh \mathcal{M} ;
- ▶ **arcs** \rightsquigarrow triangles of \mathcal{M} intersecting $f^{-1}(r)$.



Updating G_r

- ▶ **Trigger:** update when processing a vertex v .
- ▶ **Action:** process each triangle T of $\text{Star}(v)$ separately.

1. v is the upper vertex of T .
2. v is the middle vertex of T .
3. v is the lower vertex of T .

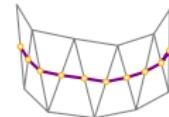


Sequential algorithm

The preimage graph

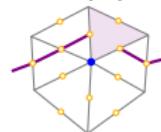
The level set $f^{-1}(r)$ can be represented by an abstract **graph** G_r :

- ▶ **nodes** \rightsquigarrow edges of the mesh \mathcal{M} ;
- ▶ **arcs** \rightsquigarrow triangles of \mathcal{M} intersecting $f^{-1}(r)$.



Updating G_r

- ▶ **Trigger:** update when processing a vertex v .
- ▶ **Action:** process each triangle T of $\text{Star}(v)$ separately.
 1. v is the upper vertex of T .
 2. v is the middle vertex of T .
 3. v is the lower vertex of T .

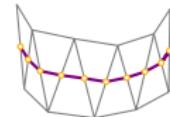


Sequential algorithm

The preimage graph

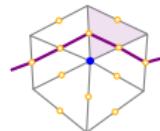
The level set $f^{-1}(r)$ can be represented by an abstract **graph** G_r :

- ▶ **nodes** \rightsquigarrow edges of the mesh \mathcal{M} ;
- ▶ **arcs** \rightsquigarrow triangles of \mathcal{M} intersecting $f^{-1}(r)$.



Updating G_r

- ▶ **Trigger:** update when processing a vertex v .
- ▶ **Action:** process each triangle T of $\text{Star}(v)$ separately.
 1. v is the upper vertex of T .
 2. v is the middle vertex of T .
 3. v is the lower vertex of T .

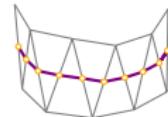


Sequential algorithm

The preimage graph

The level set $f^{-1}(r)$ can be represented by an abstract **graph** G_r :

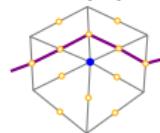
- ▶ **nodes** \rightsquigarrow edges of the mesh \mathcal{M} ;
- ▶ **arcs** \rightsquigarrow triangles of \mathcal{M} intersecting $f^{-1}(r)$.



Updating G_r

- ▶ **Trigger:** update when processing a vertex v .
- ▶ **Action:** process each triangle T of $\text{Star}(v)$ separately.

1. v is the upper vertex of T .
2. v is the middle vertex of T .
3. v is the lower vertex of T .



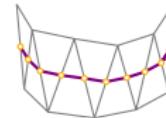
- ▶ **Data structure:** the following operations are required;

Sequential algorithm

The preimage graph

The level set $f^{-1}(r)$ can be represented by an abstract **graph** G_r :

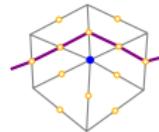
- ▶ **nodes** \rightsquigarrow edges of the mesh \mathcal{M} ;
- ▶ **arcs** \rightsquigarrow triangles of \mathcal{M} intersecting $f^{-1}(r)$.



Updating G_r

- ▶ **Trigger:** update when processing a vertex v .
- ▶ **Action:** process each triangle T of $\text{Star}(v)$ separately.

1. v is the upper vertex of T .
2. v is the middle vertex of T .
3. v is the lower vertex of T .



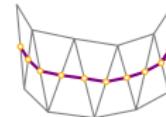
- ▶ **Data structure:** the following operations are required;
 - ▶ find the connected component of a node e ;

Sequential algorithm

The preimage graph

The level set $f^{-1}(r)$ can be represented by an abstract **graph** G_r :

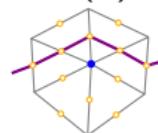
- ▶ **nodes** \rightsquigarrow edges of the mesh \mathcal{M} ;
- ▶ **arcs** \rightsquigarrow triangles of \mathcal{M} intersecting $f^{-1}(r)$.



Updating G_r

- ▶ **Trigger:** update when processing a vertex v .
- ▶ **Action:** process each triangle T of $\text{Star}(v)$ separately.

1. v is the upper vertex of T .
2. v is the middle vertex of T .
3. v is the lower vertex of T .



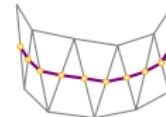
- ▶ **Data structure:** the following operations are required;
 - ▶ find the connected component of a node e ;
 - ▶ insert a new arc between nodes e_1, e_2 ;

Sequential algorithm

The preimage graph

The level set $f^{-1}(r)$ can be represented by an abstract **graph** G_r :

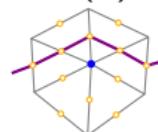
- ▶ **nodes** \rightsquigarrow edges of the mesh \mathcal{M} ;
- ▶ **arcs** \rightsquigarrow triangles of \mathcal{M} intersecting $f^{-1}(r)$.



Updating G_r

- ▶ **Trigger:** update when processing a vertex v .
- ▶ **Action:** process each triangle T of $\text{Star}(v)$ separately.

1. v is the upper vertex of T .
2. v is the middle vertex of T .
3. v is the lower vertex of T .



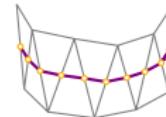
- ▶ **Data structure:** the following operations are required;
 - ▶ find the connected component of a node e ;
 - ▶ insert a new arc between nodes e_1, e_2 ;
 - ▶ delete the arc between nodes e_1, e_2 ;

Sequential algorithm

The preimage graph

The level set $f^{-1}(r)$ can be represented by an abstract **graph** G_r :

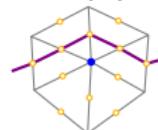
- ▶ **nodes** \rightsquigarrow edges of the mesh \mathcal{M} ;
- ▶ **arcs** \rightsquigarrow triangles of \mathcal{M} intersecting $f^{-1}(r)$.



Updating G_r

- ▶ **Trigger:** update when processing a vertex v .
- ▶ **Action:** process each triangle T of $\text{Star}(v)$ separately.

1. v is the upper vertex of T .
2. v is the middle vertex of T .
3. v is the lower vertex of T .



- ▶ **Data structure:** the following operations are required;

- ▶ find the connected component of a node e ;
- ▶ insert a new arc between nodes e_1, e_2 ;
- ▶ delete the arc between nodes e_1, e_2 ;

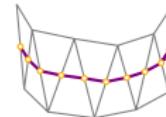
\rightsquigarrow offline dynamic connectivity problem

Sequential algorithm

The preimage graph

The level set $f^{-1}(r)$ can be represented by an abstract **graph** G_r :

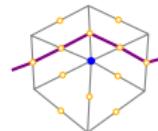
- ▶ **nodes** \rightsquigarrow edges of the mesh \mathcal{M} ;
- ▶ **arcs** \rightsquigarrow triangles of \mathcal{M} intersecting $f^{-1}(r)$.



Updating G_r

- ▶ **Trigger:** update when processing a vertex v .
- ▶ **Action:** process each triangle T of $\text{Star}(v)$ separately.

1. v is the upper vertex of T .
2. v is the middle vertex of T .
3. v is the lower vertex of T .



- ▶ **Data structure:** the following operations are required;

- ▶ find the connected component of a node e ;
- ▶ insert a new arc between nodes e_1, e_2 ;
- ▶ delete the arc between nodes e_1, e_2 ;

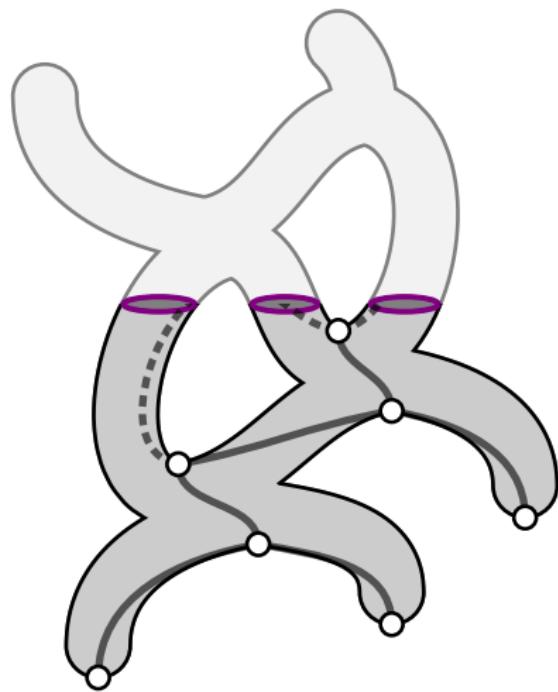
\rightsquigarrow offline dynamic connectivity problem \rightsquigarrow **ST-trees**

support all the operations in $O(\log m)$

Sequential algorithm

The augmented Reeb graph

The partial augmented Reeb graph is represented by a pair (\mathcal{R}, Φ) .

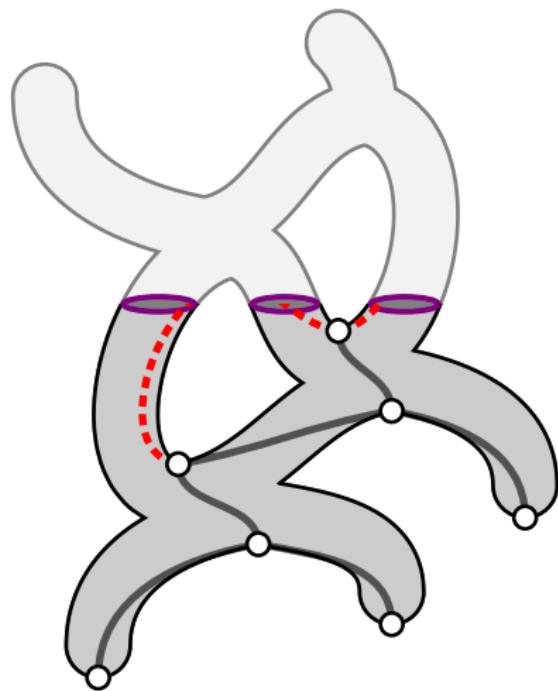


Sequential algorithm

The augmented Reeb graph

Reeb graph constructed so far;
has one **open arc** for each component of G_r

The partial augmented Reeb graph is represented by a pair (\mathcal{R}, Φ) .



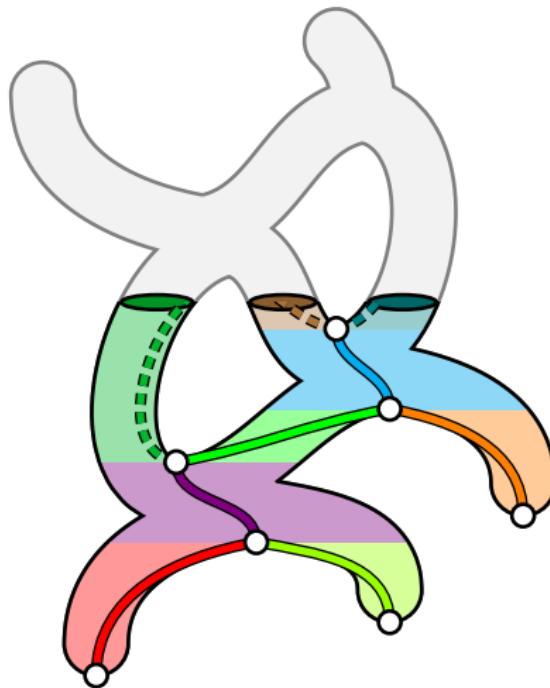
Sequential algorithm

The augmented Reeb graph

The partial augmented Reeb graph is represented by a pair (\mathcal{R}, Φ) .

partial segmentation map;

maps each vertex of the mesh to a node or an arc of \mathcal{R}



Sequential algorithm

The augmented Reeb graph

The partial augmented Reeb graph is represented by a pair (\mathcal{R}, Φ) .

Updating (\mathcal{R}, Φ)

When processing a vertex v :

Sequential algorithm

The augmented Reeb graph

The partial augmented Reeb graph is represented by a pair (\mathcal{R}, Φ) .

Updating (\mathcal{R}, Φ)

When processing a vertex v :

1. Let $\text{Lc} = \{G_{f(v)-\epsilon}.\text{find}([vv']): v' \in \text{Link}^-(v)\}$.

lower components

Sequential algorithm

The augmented Reeb graph

The partial augmented Reeb graph is represented by a pair (\mathcal{R}, Φ) .

Updating (\mathcal{R}, Φ)

When processing a vertex v :

1. Let $L_c = \{G_{f(v)-\epsilon}.\text{find}([vv']): v' \in \text{Link}^-(v)\}$.
2. Let $U_c = \{G_{f(v)+\epsilon}.\text{find}([vv']): v' \in \text{Link}^+(v)\}$.

upper components

Sequential algorithm

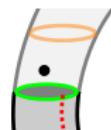
The augmented Reeb graph

The partial augmented Reeb graph is represented by a pair (\mathcal{R}, Φ) .

Updating (\mathcal{R}, Φ)

When processing a vertex v :

1. Let $L_c = \{G_{f(v)-\epsilon}.\text{find}([vv']): v' \in \text{Link}^-(v)\}$.
2. Let $U_c = \{G_{f(v)+\epsilon}.\text{find}([vv']): v' \in \text{Link}^+(v)\}$.
3. If $|L_c| = |U_c| = 1$ then:



Sequential algorithm

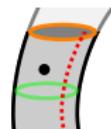
The augmented Reeb graph

The partial augmented Reeb graph is represented by a pair (\mathcal{R}, Φ) .

Updating (\mathcal{R}, Φ)

When processing a vertex v :

1. **Let $L_c = \{G_{f(v)-\epsilon}.\text{find}([vv']): v' \in \text{Link}^-(v)\}$.**
2. **Let $U_c = \{G_{f(v)+\epsilon}.\text{find}([vv']): v' \in \text{Link}^+(v)\}$.**
3. **If $|L_c| = |U_c| = 1$ then:**
 - ▶ \mathcal{R} is unchanged;



Sequential algorithm

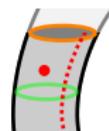
The augmented Reeb graph

The partial augmented Reeb graph is represented by a pair (\mathcal{R}, Φ) .

Updating (\mathcal{R}, Φ)

When processing a vertex v :

1. Let $L_c = \{G_{f(v)-\epsilon}.\text{find}([vv']): v' \in \text{Link}^-(v)\}$.
2. Let $U_c = \{G_{f(v)+\epsilon}.\text{find}([vv']): v' \in \text{Link}^+(v)\}$.
3. If $|L_c| = |U_c| = 1$ then:
 - ▶ \mathcal{R} is unchanged;
 - ▶ $\Phi(v) =$ the open arc associated to the lower component.



Sequential algorithm

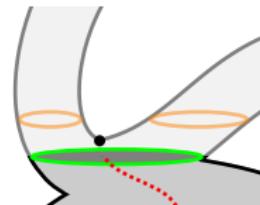
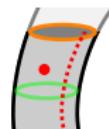
The augmented Reeb graph

The partial augmented Reeb graph is represented by a pair (\mathcal{R}, Φ) .

Updating (\mathcal{R}, Φ)

When processing a vertex v :

1. Let $L_c = \{G_{f(v)-\epsilon}.\text{find}([vv']): v' \in \text{Link}^-(v)\}$.
2. Let $U_c = \{G_{f(v)+\epsilon}.\text{find}([vv']): v' \in \text{Link}^+(v)\}$.
3. If $|L_c| = |U_c| = 1$ then:
 - ▶ \mathcal{R} is unchanged;
 - ▶ $\Phi(v) =$ the open arc associated to the lower component.
4. Otherwise:



Sequential algorithm

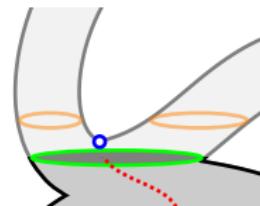
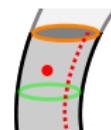
The augmented Reeb graph

The partial augmented Reeb graph is represented by a pair (\mathcal{R}, Φ) .

Updating (\mathcal{R}, Φ)

When processing a vertex v :

1. Let $L_c = \{G_{f(v)-\epsilon}.\text{find}([vv']): v' \in \text{Link}^-(v)\}$.
2. Let $U_c = \{G_{f(v)+\epsilon}.\text{find}([vv']): v' \in \text{Link}^+(v)\}$.
3. If $|L_c| = |U_c| = 1$ then:
 - ▶ \mathcal{R} is unchanged;
 - ▶ $\Phi(v) =$ the open arc associated to the lower component.
4. Otherwise:
 - ▶ create a new node w in \mathcal{R} ;



Sequential algorithm

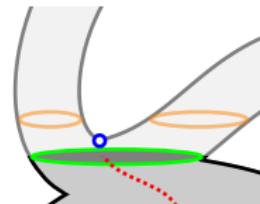
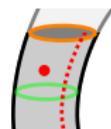
The augmented Reeb graph

The partial augmented Reeb graph is represented by a pair (\mathcal{R}, Φ) .

Updating (\mathcal{R}, Φ)

When processing a vertex v :

1. Let $L_c = \{G_{f(v)-\epsilon}.\text{find}([vv']): v' \in \text{Link}^-(v)\}$.
2. Let $U_c = \{G_{f(v)+\epsilon}.\text{find}([vv']): v' \in \text{Link}^+(v)\}$.
3. If $|L_c| = |U_c| = 1$ then:
 - ▶ \mathcal{R} is unchanged;
 - ▶ $\Phi(v) =$ the open arc associated to the lower component.
4. Otherwise:
 - ▶ create a new node w in \mathcal{R} ;
 - ▶ $\Phi(v) = w$;



Sequential algorithm

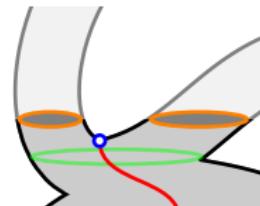
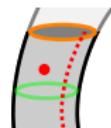
The augmented Reeb graph

The partial augmented Reeb graph is represented by a pair (\mathcal{R}, Φ) .

Updating (\mathcal{R}, Φ)

When processing a vertex v :

1. Let $L_c = \{G_{f(v)-\epsilon}.\text{find}([vv']): v' \in \text{Link}^-(v)\}$.
2. Let $U_c = \{G_{f(v)+\epsilon}.\text{find}([vv']): v' \in \text{Link}^+(v)\}$.
3. If $|L_c| = |U_c| = 1$ then:
 - ▶ \mathcal{R} is unchanged;
 - ▶ $\Phi(v) =$ the open arc associated to the lower component.
4. Otherwise:
 - ▶ create a new node w in \mathcal{R} ;
 - ▶ $\Phi(v) = w$;
 - ▶ all the open arcs associated to the lower components end at w ;



Sequential algorithm

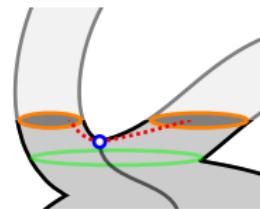
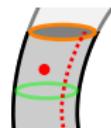
The augmented Reeb graph

The partial augmented Reeb graph is represented by a pair (\mathcal{R}, Φ) .

Updating (\mathcal{R}, Φ)

When processing a vertex v :

1. Let $L_c = \{G_{f(v)-\epsilon}.\text{find}([vv']): v' \in \text{Link}^-(v)\}$.
2. Let $U_c = \{G_{f(v)+\epsilon}.\text{find}([vv']): v' \in \text{Link}^+(v)\}$.
3. If $|L_c| = |U_c| = 1$ then:
 - ▶ \mathcal{R} is unchanged;
 - ▶ $\Phi(v) =$ the open arc associated to the lower component.
4. Otherwise:
 - ▶ create a new node w in \mathcal{R} ;
 - ▶ $\Phi(v) = w$;
 - ▶ all the open arcs associated to the lower components end at w ;
 - ▶ open a new arc in \mathcal{R} starting at w for each upper component.



Sequential algorithm

Full implementation

```
1 input : a triangulated mesh  $\mathcal{M}$ 
      a scalar field  $f$  on  $\mathcal{M}$ 
2 output: the augmented Reeb graph  $(\mathcal{R}, \Phi)$ 
3 begin
4    $\mathcal{R}, \Phi \leftarrow \emptyset$  [graph],  $\emptyset$  [function]
5    $G_r \leftarrow \emptyset$  [ST-tree]
6   sort the vertices of  $\mathcal{M}$  by increasing value of  $f$ 
7   foreach  $v$  vertex of  $\mathcal{M}$  do
8      $L_c \leftarrow \text{GetLowerComponents}(v)$ 
9      $U_c \leftarrow \text{GetUpperComponents}(v)$ 
10    if  $|L_c| = |U_c| = 1$  then update  $\Phi(v)$ 
11    else  $\text{UpdateReebGraph}(v, L_c, U_c)$ 
12  end
13 return  $(\mathcal{R}, \Phi)$ 
14 end
```

Parallel algorithm

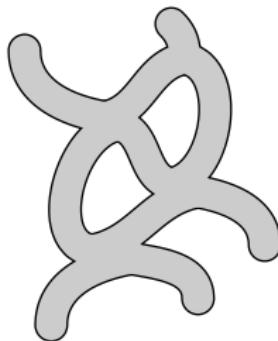
Core ideas

- ▶ **Sequential:** single procedure sweeping all the vertices sequentially.

Parallel algorithm

Core ideas

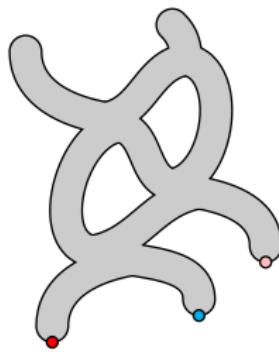
- ▶ **Sequential:** single procedure sweeping all the vertices sequentially.
- ▶ **Parallel:** multiple procedures (local growths) running simultaneously.



Parallel algorithm

Core ideas

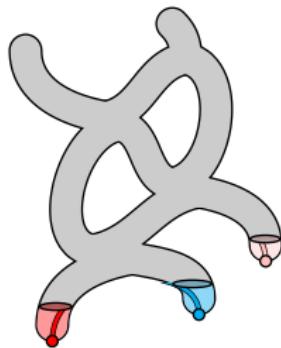
- ▶ **Sequential:** single procedure sweeping all the vertices sequentially.
- ▶ **Parallel:** multiple procedures (local growths) running simultaneously.
 - ▶ A local growth is started at every minimum.



Parallel algorithm

Core ideas

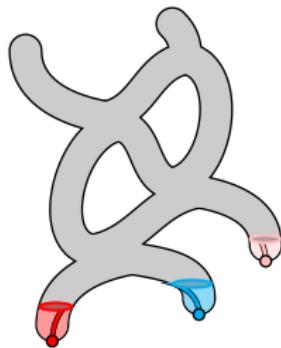
- ▶ **Sequential:** single procedure sweeping all the vertices sequentially.
- ▶ **Parallel:** multiple procedures (local growths) running simultaneously.
 - ▶ A local growth is started at every minimum.
 - ▶ Each local growth spreads independently with an ordered BFS.



Parallel algorithm

Core ideas

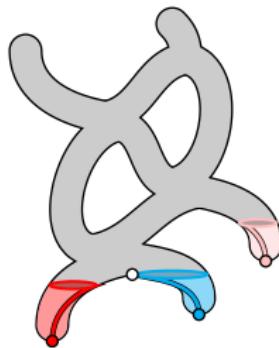
- ▶ **Sequential:** single procedure sweeping all the vertices sequentially.
- ▶ **Parallel:** multiple procedures (local growths) running simultaneously.
 - ▶ A local growth is started at every minimum.
 - ▶ Each local growth spreads independently with an ordered BFS.
 - ▶ Each local growth updates its own preimage graph G_r .



Parallel algorithm

Core ideas

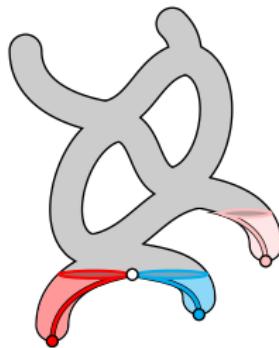
- ▶ **Sequential:** single procedure sweeping all the vertices sequentially.
- ▶ **Parallel:** multiple procedures (local growths) running simultaneously.
 - ▶ A local growth is started at every minimum.
 - ▶ Each local growth spreads independently with an ordered BFS.
 - ▶ Each local growth updates its own preimage graph G_r .
 - ▶ **Join saddles:** wait until all the involved local growths have reached the saddle, then join them.



Parallel algorithm

Core ideas

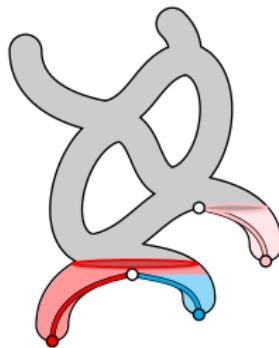
- ▶ **Sequential:** single procedure sweeping all the vertices sequentially.
- ▶ **Parallel:** multiple procedures (local growths) running simultaneously.
 - ▶ A local growth is started at every minimum.
 - ▶ Each local growth spreads independently with an ordered BFS.
 - ▶ Each local growth updates its own preimage graph G_r .
 - ▶ **Join saddles:** wait until all the involved local growths have reached the saddle, then join them.



Parallel algorithm

Core ideas

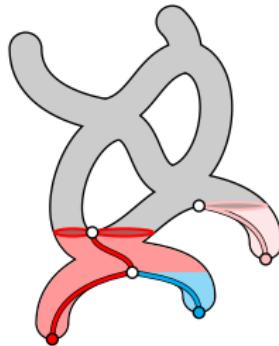
- ▶ **Sequential:** single procedure sweeping all the vertices sequentially.
- ▶ **Parallel:** multiple procedures (local growths) running simultaneously.
 - ▶ A local growth is started at every minimum.
 - ▶ Each local growth spreads independently with an ordered BFS.
 - ▶ Each local growth updates its own preimage graph G_r .
 - ▶ **Join saddles:** wait until all the involved local growths have reached the saddle, then join them.



Parallel algorithm

Core ideas

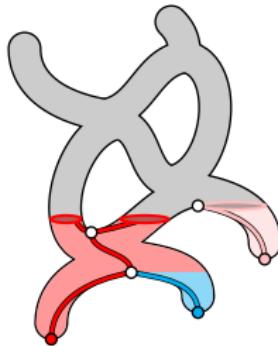
- ▶ **Sequential:** single procedure sweeping all the vertices sequentially.
- ▶ **Parallel:** multiple procedures (local growths) running simultaneously.
 - ▶ A local growth is started at every minimum.
 - ▶ Each local growth spreads independently with an ordered BFS.
 - ▶ Each local growth updates its own preimage graph G_r .
 - ▶ **Join saddles:** wait until all the involved local growths have reached the saddle, then join them.
 - ▶ **Split saddles:** the new open arcs in $\mathcal{R}(f)$ are handled by the same local growth.



Parallel algorithm

Core ideas

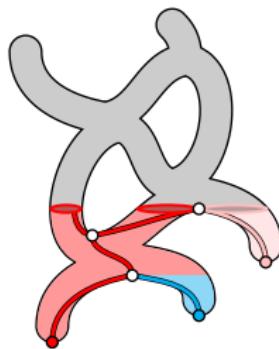
- ▶ **Sequential:** single procedure sweeping all the vertices sequentially.
- ▶ **Parallel:** multiple procedures (local growths) running simultaneously.
 - ▶ A local growth is started at every minimum.
 - ▶ Each local growth spreads independently with an ordered BFS.
 - ▶ Each local growth updates its own preimage graph G_r .
 - ▶ **Join saddles:** wait until all the involved local growths have reached the saddle, then join them.
 - ▶ **Split saddles:** the new open arcs in $\mathcal{R}(f)$ are handled by the same local growth.



Parallel algorithm

Core ideas

- ▶ **Sequential:** single procedure sweeping all the vertices sequentially.
- ▶ **Parallel:** multiple procedures (local growths) running simultaneously.
 - ▶ A local growth is started at every minimum.
 - ▶ Each local growth spreads independently with an ordered BFS.
 - ▶ Each local growth updates its own preimage graph G_r .
 - ▶ **Join saddles:** wait until all the involved local growths have reached the saddle, then join them.
 - ▶ **Split saddles:** the new open arcs in $\mathcal{R}(f)$ are handled by the same local growth.



Parallel algorithm

Local growths

Data structures

Each local growth keeps:

Parallel algorithm

Local growths

Data structures

Each local growth keeps:

- ▶ a **Fibonacci heap** θ to store candidates for the ordered BFS;

Parallel algorithm

Local growths

Data structures

Each local growth keeps:

candidates are
sorted by f value

- ▶ a **Fibonacci heap** θ to store candidates for the **ordered BFS**;

Parallel algorithm

Local growths

Data structures

Each local growth keeps:

- ▶ a **Fibonacci heap** θ to store candidates for the ordered BFS;
 - can be merged in $O(1)$

Parallel algorithm

Local growths

Data structures

Each local growth keeps:

- ▶ a **Fibonacci heap** θ to store candidates for the ordered BFS;
- ▶ an **ST-tree** G_r to store the preimage graph.

Parallel algorithm

Local growths

Data structures

Each local growth keeps:

- ▶ a **Fibonacci heap** θ to store candidates for the ordered BFS;
- ▶ an **ST-tree** G_r to store the preimage graph.
 - ↗ can be merged in $O(1)$

Parallel algorithm

Local growths

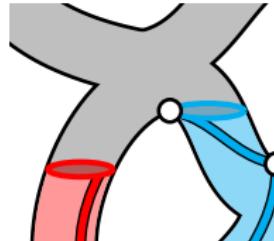
Data structures

Each local growth keeps:

- ▶ a **Fibonacci heap** θ to store candidates for the ordered BFS;
- ▶ an **ST-tree** G_r to store the preimage graph.

Join saddles

What if a saddle joins components from different local growths?



Parallel algorithm

Local growths

Data structures

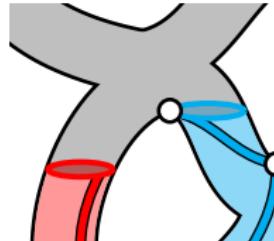
Each local growth keeps:

- ▶ a **Fibonacci heap** θ to store candidates for the ordered BFS;
- ▶ an **ST-tree** G_r to store the preimage graph.

Join saddles

What if a saddle joins components from different local growths?

- ▶ **Detection:** before processing a vertex v , check whether all the vertices in $\text{Link}^-(v)$ have already been visited.



Parallel algorithm

Local growths

Data structures

Each local growth keeps:

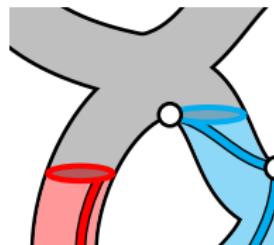
- ▶ a **Fibonacci heap** θ to store candidates for the ordered BFS;
- ▶ an **ST-tree** G_r to store the preimage graph.

Join saddles

What if a saddle joins components from different local growths?

- ▶ **Detection:** before processing a vertex v , check whether all the vertices in $\text{Link}^-(v)$ have already been visited.

→ concretely, update an atomic counter
`visitedLower[v]` and check whether
 $\text{visitedLower}[v] = |\text{Link}^-(v)|$



Parallel algorithm

Local growths

Data structures

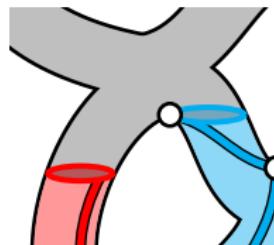
Each local growth keeps:

- ▶ a **Fibonacci heap** θ to store candidates for the ordered BFS;
- ▶ an **ST-tree** G_r to store the preimage graph.

Join saddles

What if a saddle joins components from different local growths?

- ▶ **Detection:** before processing a vertex v , check whether all the vertices in $\text{Link}^-(v)$ have already been visited.
- ▶ **Stopping:** if not, terminate **this local growth**.



Parallel algorithm

Local growths

Data structures

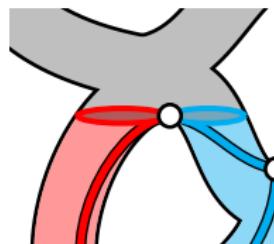
Each local growth keeps:

- ▶ a **Fibonacci heap** θ to store candidates for the ordered BFS;
- ▶ an **ST-tree** G_r to store the preimage graph.

Join saddles

What if a saddle joins components from different local growths?

- ▶ **Detection**: before processing a vertex v , check whether all the vertices in $\text{Link}^-(v)$ have already been visited.
- ▶ **Stopping**: if not, terminate **this local growth**.
- ▶ **Processing**: otherwise, **this local growth** is in charge of proceeding;



Parallel algorithm

Local growths

Data structures

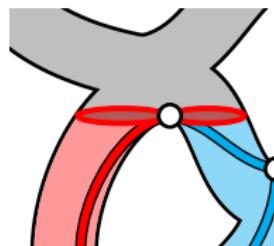
Each local growth keeps:

- ▶ a **Fibonacci heap** θ to store candidates for the ordered BFS;
- ▶ an **ST-tree** G_r to store the preimage graph.

Join saddles

What if a saddle joins components from different local growths?

- ▶ **Detection**: before processing a vertex v , check whether all the vertices in $\text{Link}^-(v)$ have already been visited.
- ▶ **Stopping**: if not, terminate **this local growth**.
- ▶ **Processing**: otherwise, **this local growth** is in charge of proceeding:
 - ▶ join the priority queues (θ) and the preimage graphs (G_r) of all local growths terminated at v ;



Parallel algorithm

Local growths

Data structures

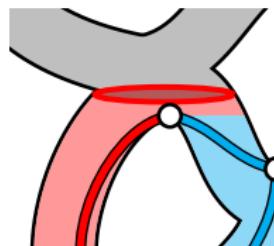
Each local growth keeps:

- ▶ a **Fibonacci heap** θ to store candidates for the ordered BFS;
- ▶ an **ST-tree** G_r to store the preimage graph.

Join saddles

What if a saddle joins components from different local growths?

- ▶ **Detection**: before processing a vertex v , check whether all the vertices in $\text{Link}^-(v)$ have already been visited.
- ▶ **Stopping**: if not, terminate **this local growth**.
- ▶ **Processing**: otherwise, **this local growth** is in charge of proceeding:
 - ▶ join the priority queues (θ) and the preimage graphs (G_r) of all local growths terminated at v ;
 - ▶ process v as usual.



Parallel algorithm

Local growth implementation

```
1 procedure LocalGrowth( $v_0$ ,  $\mathcal{R}$ ,  $\Phi$ )
2    $\theta$ ,  $G_r \leftarrow \{v_0\}$  [Fibonacci heap],  $\emptyset$  [ST-tree]
3   while  $\theta \neq \emptyset$  do
4      $v \leftarrow$  vertex in  $\theta$  with minimal  $f$  value
5     update visitedLower[ $v$ ]
6     if visitedLower[ $v$ ] < |Link-( $v$ )| then
7       append ( $\theta$ ,  $G_r$ ) to pending[ $v$ ]
8       terminate
9     end
10    foreach ( $\theta'$ ,  $G'_r$ )  $\in$  pending[ $v$ ] do
11       $\theta$ .join( $\theta'$ );  $G_r$ .join( $G'_r$ )
12    end
13    process  $v$ , updating  $G_r$ ,  $\mathcal{R}$  and  $\Phi$ 
14    add vertices in Link+( $v$ ) to  $\theta$ 
15  end
16 end
```

Parallel algorithm

Local growth implementation

```
1 procedure LocalGrowth( $v_0, \mathcal{R}, \Phi$ )
2    $\theta, G_r \leftarrow \{v_0\}$  [Fibonacci heap],  $\emptyset$  [ST-tree]
3   while  $\theta \neq \emptyset$  do
4      $v \leftarrow$  vertex in  $\theta$  with minimal  $f$  value
5     update visitedLower[ $v$ ]
6     if visitedLower[ $v$ ] < |Link-( $v$ )| then
7       append ( $\theta, G_r$ ) to pending[ $v$ ]
8       terminate
9     end
10    foreach ( $\theta', G'_r$ )  $\in$  pending[ $v$ ] do
11       $\theta.\text{join}(\theta');$   $G_r.\text{join}(G'_r)$ 
12    end
13    process  $v$ , updating  $G_r, \mathcal{R}$  and  $\Phi$ 
14    add vertices in Link+( $v$ ) to  $\theta$ 
15  end
16 end
```

A blue brace is drawn on the right side of the pseudocode, spanning from line 5 to line 9. To the right of the brace, the text "critical section" is written in blue.

Parallel algorithm

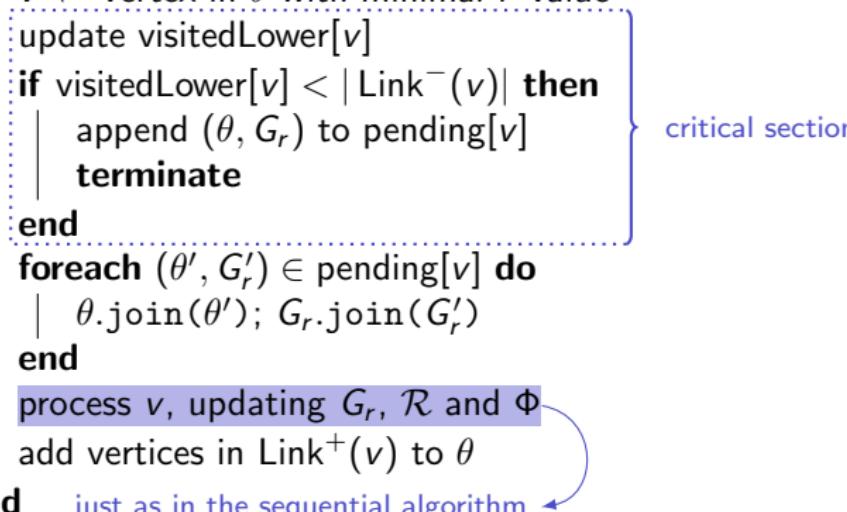
Local growth implementation

```
1 procedure LocalGrowth( $v_0, \mathcal{R}, \Phi$ )
2    $\theta, G_r \leftarrow \{v_0\}$  [Fibonacci heap],  $\emptyset$  [ST-tree]
3   while  $\theta \neq \emptyset$  do
4      $v \leftarrow$  vertex in  $\theta$  with minimal  $f$  value
5     update visitedLower[ $v$ ]
6     if visitedLower[ $v$ ] < |Link-( $v$ )| then
7       append ( $\theta, G_r$ ) to pending[ $v$ ]
8       terminate
9     end
10    foreach ( $\theta', G'_r$ )  $\in$  pending[ $v$ ] do
11       $\theta.\text{join}(\theta'); G_r.\text{join}(G'_r)$ 
12    end
13    process  $v$ , updating  $G_r, \mathcal{R}$  and  $\Phi$ 
14    add vertices in Link+( $v$ ) to  $\theta$ 
15  end
16 end
```

Parallel algorithm

Local growth implementation

```
1 procedure LocalGrowth( $v_0, \mathcal{R}, \Phi$ )
2    $\theta, G_r \leftarrow \{v_0\}$  [Fibonacci heap],  $\emptyset$  [ST-tree]
3   while  $\theta \neq \emptyset$  do
4      $v \leftarrow$  vertex in  $\theta$  with minimal  $f$  value
5     update visitedLower[ $v$ ]
6     if visitedLower[ $v$ ] < |Link-( $v$ )| then
7       append ( $\theta, G_r$ ) to pending[ $v$ ]
8       terminate
9     end
10    foreach ( $\theta', G'_r$ )  $\in$  pending[ $v$ ] do
11       $\theta.\text{join}(\theta');$   $G_r.\text{join}(G'_r)$ 
12    end
13    process  $v$ , updating  $G_r, \mathcal{R}$  and  $\Phi$ 
14    add vertices in Link+( $v$ ) to  $\theta$ 
15  end just as in the sequential algorithm
16 end
```



Parallel algorithm

Full implementation

```
1 input : a triangulated mesh  $\mathcal{M}$ 
      a scalar field  $f$  on  $\mathcal{M}$ 
2 output: the augmented Reeb graph  $(\mathcal{R}, \Phi)$ 
3 begin
4    $\mathcal{R}, \Phi \leftarrow \emptyset$  [graph],  $\emptyset$  [function]
5    $V \leftarrow \text{FindMinima}(\mathcal{M}, f)$ 
6   foreach  $v_0 \in V$  in parallel do
7     | start procedure LocalGrowth( $v_0, \mathcal{R}, \Phi$ )
8   end
9   return  $(\mathcal{R}, \Phi)$ 
10 end
```

Parallel algorithm

Full implementation

```
1 input : a triangulated mesh  $\mathcal{M}$ 
      a scalar field  $f$  on  $\mathcal{M}$ 
2 output: the augmented Reeb graph  $(\mathcal{R}, \Phi)$ 
3 begin
4    $\mathcal{R}, \Phi \leftarrow \emptyset$  [graph],  $\emptyset$  [function]
5    $V \leftarrow \text{FindMinima}(\mathcal{M}, f)$  ————— easy to run in parallel
6   foreach  $v_0 \in V$  in parallel do
7     | start procedure LocalGrowth( $v_0, \mathcal{R}, \Phi$ )
8   end
9   return  $(\mathcal{R}, \Phi)$ 
10 end
```

Parallel algorithm

Full implementation

```
1 input : a triangulated mesh  $\mathcal{M}$ 
      a scalar field  $f$  on  $\mathcal{M}$ 
2 output: the augmented Reeb graph  $(\mathcal{R}, \Phi)$ 
3 begin
4    $\mathcal{R}, \Phi \leftarrow \emptyset$  [graph],  $\emptyset$  [function]
5    $V \leftarrow \text{FindMinima}(\mathcal{M}, f)$ 
6   foreach  $v_0 \in V$  in parallel do
7     | start procedure LocalGrowth( $v_0, \mathcal{R}, \Phi$ )
8   end
9   return  $(\mathcal{R}, \Phi)$ 
10 end
```

Parallel algorithm

Optimizations

Laziness

Parallel algorithm

Optimizations

Laziness

- ▶ Instead of inserting and removing arcs in G_r , keep a **history list**.

Parallel algorithm

Optimizations

Laziness

- ▶ Instead of inserting and removing arcs in G_r , keep a **history list**.

one for each open arc in \mathcal{R}



Parallel algorithm

Optimizations

Laziness

- ▶ Instead of inserting and removing arcs in G_r , keep a **history list**.
- ▶ When a critical point is reached, discard all arcs marked as both added and deleted.

Parallel algorithm

Optimizations

Laziness

- ▶ Instead of inserting and removing arcs in G_r , keep a **history list**.
- ▶ When a **critical point** is reached, discard all arcs marked as both added and deleted.
 - requires a pre-processing step to identify the critical points

Parallel algorithm

Optimizations

Laziness

- ▶ Instead of inserting and removing arcs in G_r , keep a **history list**.
- ▶ When a critical point is reached, discard all arcs marked as both added and deleted.
- ▶ Perform all the other operations on G_r .

Parallel algorithm

Optimizations

Laziness —————→ decreased number of operations on ST-trees

- ▶ Instead of inserting and removing arcs in G_r , keep a **history list**.
- ▶ When a critical point is reached, discard all arcs marked as both added and deleted.
- ▶ Perform all the other operations on G_r .

Parallel algorithm

Optimizations

Laziness —————→ decreased number of operations on ST-trees

- ▶ Instead of inserting and removing arcs in G_r , keep a **history list**.
- ▶ When a critical point is reached, discard all arcs marked as both added and deleted.
- ▶ Perform all the other operations on G_r .

Dual sweep



Parallel algorithm

Optimizations

Laziness —————→ decreased number of operations on ST-trees

- ▶ Instead of inserting and removing arcs in G_r , keep a **history list**.
- ▶ When a critical point is reached, discard all arcs marked as both added and deleted.
- ▶ Perform all the other operations on G_r .

Dual sweep

- ▶ Two kinds of local growths:



Parallel algorithm

Optimizations

Laziness —————→ decreased number of operations on ST-trees

- ▶ Instead of inserting and removing arcs in G_r , keep a **history list**.
- ▶ When a critical point is reached, discard all arcs marked as both added and deleted.
- ▶ Perform all the other operations on G_r .

Dual sweep

- ▶ Two kinds of local growths:
 - ▶ **upward local growths**
~~~ as described before;



# Parallel algorithm

## Optimizations

Laziness —————→ decreased number of operations on ST-trees

- ▶ Instead of inserting and removing arcs in  $G_r$ , keep a **history list**.
- ▶ When a critical point is reached, discard all arcs marked as both added and deleted.
- ▶ Perform all the other operations on  $G_r$ .

## Dual sweep

- ▶ Two kinds of local growths:
  - ▶ **upward local growths**  
~~ as described before;
  - ▶ **downward local growths**  
~~ start from maxima and expand downwards.



# Parallel algorithm

## Optimizations

Laziness —————→ decreased number of operations on ST-trees

- ▶ Instead of inserting and removing arcs in  $G_r$ , keep a **history list**.
- ▶ When a critical point is reached, discard all arcs marked as both added and deleted.
- ▶ Perform all the other operations on  $G_r$ .

## Dual sweep

- ▶ Two kinds of local growths:
  - ▶ **upward local growths**  
~~ as described before;
  - ▶ **downward local growths**  
~~ start from maxima and expand downwards.
- ▶ Post-processing step to merge arcs that have been created twice.



# Parallel algorithm

## Optimizations

Laziness —→ decreased number of operations on ST-trees

- ▶ Instead of inserting and removing arcs in  $G_r$ , keep a **history list**.
- ▶ When a critical point is reached, discard all arcs marked as both added and deleted.
- ▶ Perform all the other operations on  $G_r$ .

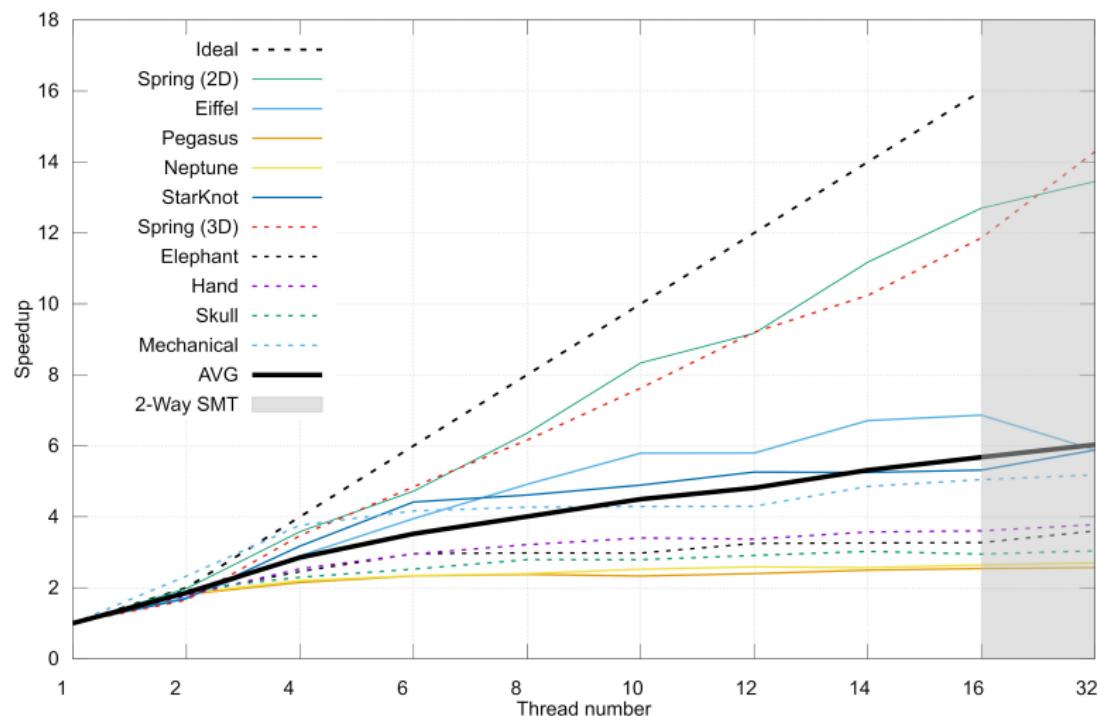
Dual sweep —→ increased number of independent local growths

- ▶ Two kinds of local growths:
  - ▶ **upward local growths**  
~~ as described before;
  - ▶ **downward local growths**  
~~ start from maxima and expand downwards.
- ▶ Post-processing step to merge arcs that have been created twice.



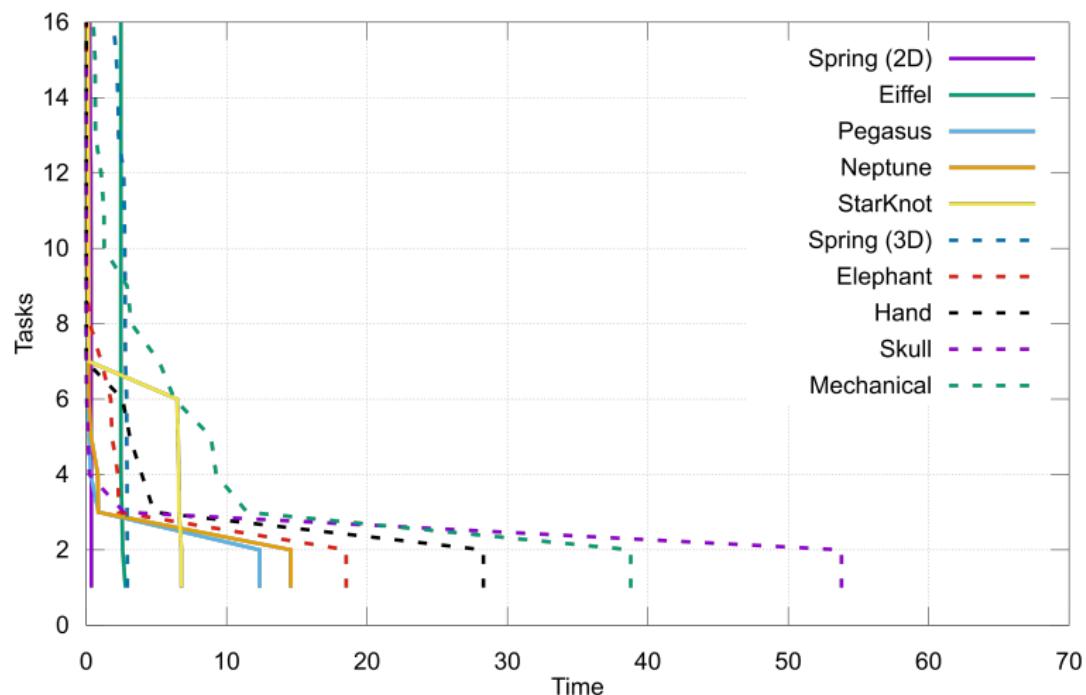
# Results

## Speedup



# Results

## Number of tasks



# Results

## Conclusion

- ▶ Parallel task-based algorithm for computing the augmented Reeb graph.

# Results

## Conclusion

- ▶ Parallel task-based algorithm for computing the augmented Reeb graph.
- ▶ Improved laziness mechanism thanks to the locality of growths.

# Results

## Conclusion

- ▶ Parallel task-based algorithm for computing the augmented Reeb graph.
- ▶ Improved laziness mechanism thanks to the locality of growths.
- ▶ Dual sweep for improved parallel performance.

# Results

## Conclusion

- ▶ Parallel task-based algorithm for computing the augmented Reeb graph.
- ▶ Improved laziness mechanism thanks to the locality of growths.
- ▶ Dual sweep for improved parallel performance.
- ▶ OpenMP/C++ implementation provided.

# Results

## Conclusion

- ▶ Parallel task-based algorithm for computing the augmented Reeb graph.
- ▶ Improved laziness mechanism thanks to the locality of growths.
- ▶ Dual sweep for improved parallel performance.
- ▶ OpenMP/C++ implementation provided.

What next?

# Results

## Conclusion

- ▶ Parallel task-based algorithm for computing the augmented Reeb graph.
- ▶ Improved laziness mechanism thanks to the locality of growths.
- ▶ Dual sweep for improved parallel performance.
- ▶ OpenMP/C++ implementation provided.

## What next?

- ▶ Number of tasks bounded by the number of leaves in the Reeb graph.

# Results

## Conclusion

- ▶ Parallel task-based algorithm for computing the augmented Reeb graph.
- ▶ Improved laziness mechanism thanks to the locality of growths.
- ▶ Dual sweep for improved parallel performance.
- ▶ OpenMP/C++ implementation provided.

## What next?

- ▶ Number of tasks bounded by the number of leaves in the Reeb graph.
- ▶ Number of independent local growths rapidly decreases to 2.