

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 $\Phi: \mathcal{M} \rightarrow \mathcal{R}(f)$.
 $x \sim y$ if they belong to the same connected component of $f^{-1}(f(x))$

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

Reeb graph

Desired algorithm

Input:

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

Output:

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

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

graph + segmentation map

Time complexity:

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

#vertices + #edges + #triangles

Parallel.

Reeb graph

Geometry of critical points

There are three kinds of critical points:

- ▶ (local) **maxima**
 $\rightsquigarrow \text{Link}^+$ empty;
- ▶ (local) **minima**
 $\rightsquigarrow \text{Link}^-$ empty;
- ▶ **saddles**
 $\rightsquigarrow \text{Link}^-$ 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 $\mathcal{R}(f)$.

- ▶ **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 .
- ▶ 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.
 - ▶ each connected component corresponds to an open edge of the partial Reeb graph

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

Updating G_r

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

- ▶ **Trigger:** **update** when processing a vertex v
- ▶ **Action:** process each triangle \mathcal{T} of $\text{Star}(v)$ separately.
 1. v is the lower vertex of \mathcal{T} .
 2. v is the middle vertex of \mathcal{T} .
 3. v is the upper vertex of \mathcal{T} .
- ▶ **Data structure:** the following operations are required;
 - ▶ find the connected component of a node e ;
 - ▶ insert a new edge between nodes e_1, e_2 ;
 - ▶ delete the edge 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

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

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

Updating (\mathcal{R}, Φ)

partial segmentation map;
maps each vertex of the mesh to a node or an edge of \mathcal{R}

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 edge associated to the lower component.
4. Otherwise:
 - ▶ create a new vertex w in \mathcal{R} ;
 - ▶ all the open edges associated to the lower components end at w ;
 - ▶ open a new edge in \mathcal{R} starting at w for each upper component.
 - ▶ $\Phi(v) = w$.

Sequential algorithm

Full implementation

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

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 involved local growths have reached the saddle, then join them.
 - ▶ **Split saddles:** the new open edges in $\mathcal{R}(f)$ are handled by the same 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.
- candidates are sorted by f value
- can be merged in $O(1)$
- can be merged in $O(1)$

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.
- coarsely: update an element's counter
- visitedLower[v] and check whether visitedLower[v] = |Link⁻(v)|

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 → add  $|\{w \in \text{Link}^-(v) : w \text{ visited by this local growth}\}|$ 
4      $v \leftarrow$  vertex in  $\theta$  with minimal  $f$  value
5     update visitedLower[ $v$ ]
6     if visitedLower[ $v$ ] <  $|\text{Link}^-(v)|$  then
7       append  $(\theta, G_r)$  to pending[ $v$ ]
8       terminate
9     end
10    foreach  $(\theta', G'_r) \in \text{pending}[v]$  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  $\text{Link}^+(v)$  to  $\theta$ 
15  end just as in the sequential algorithm ←
16 end
```

critical section

Parallel algorithm

Full implementation

input : a triangulated mesh \mathcal{M}
a scalar field f on \mathcal{M}
output : the augmented Reeb graph (\mathcal{R}, Φ)

```
1 begin
2    $\mathcal{R}, \Phi \leftarrow \emptyset$  [graph],  $\emptyset$  [function]
3    $V \leftarrow \text{FindMinima}(\mathcal{M}, f)$   $\longrightarrow$  easy to run in parallel
4   foreach  $v_0 \in V$  in parallel do
5     | start procedure LocalGrowth( $v_0, \mathcal{R}, \Phi$ )
6   end
7   return  $(\mathcal{R}, \Phi)$ 
8 end
```

Parallel algorithm

Optimizations

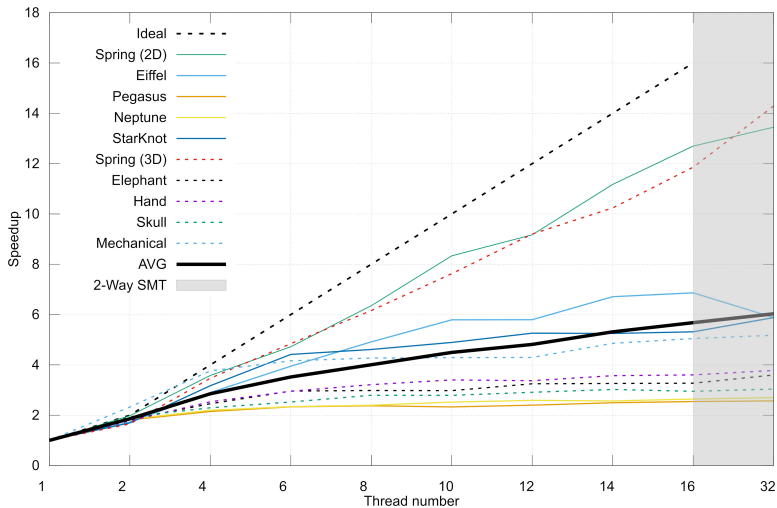
Laziness —————> decreased number of operations on ST-trees

- ▶ Instead of inserting and removing edges in G_r , keep a **history list**.
- ▶ When a **critical point** is reached, discard all edges marked as both added and deleted.
one for each open edge in \mathcal{R}
- ▶ Perform all the other operations on G_r .
requires a pre-processing step

Dual sweep —————> increased number of independent local growths
to identify the critical points

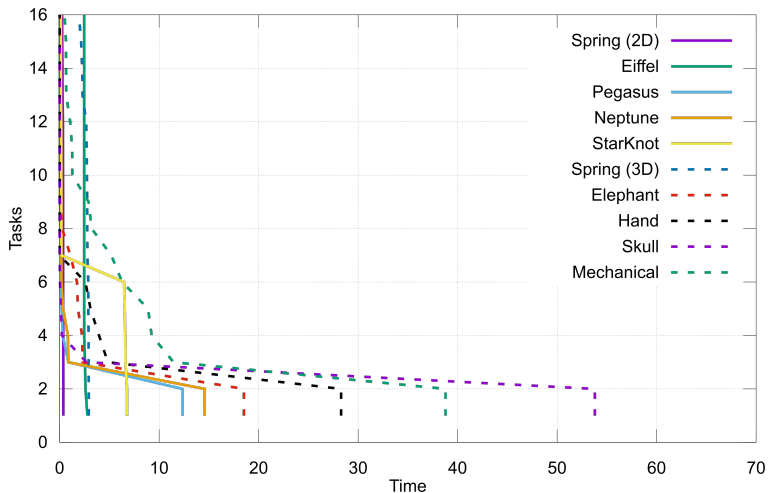
- ▶ Two kinds of local growths:
 - ▶ **upward local growths** \rightsquigarrow as described before;
 - ▶ **downward local growths** \rightsquigarrow start from maxima and expand downwards.
- ▶ Post-processing step to merge edges that have been created twice.

Speedup



Results

Number of tasks



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.