

Output-Sensitive Construction of Reeb Graphs

Harish Doraiswamy and Vijay Natarajan, Member, IEEE

Report by- Karan Raj Pandey-IMT2021014

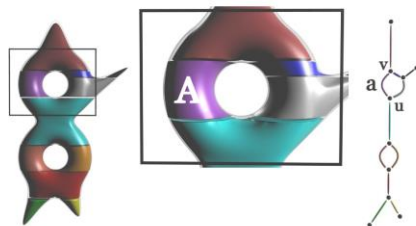
Prasad Vijay Jore-IMT2021104

The steps to compile the code are given in a “README.md” file in the “.zip” folder submitted.

This algorithm maximally leverages the efficient algorithms available to compute the loop-free version of the Reeb graph. It partitions the input into subsets whose Reeb graph do not contain any loops and merges the loop-free graphs in linear time to construct the Reeb graph of the input. The algorithm has a running time of $O(t \log t + l + n)$, which in the worst-case is $O(t \log t + sn)$ or close to the lower bound $\Omega(t \log t + n)$ as the size of critical level sets is usually $O(n)$ in practice and s is the number of saddles. Here t is the number of vertices or critical points, n is the number of triangles, and l is the total size (number of edges) of all critical level sets. In practice, the running times scale linearly with the number of triangles. The algorithm is up to two orders of magnitude faster than existing generic algorithms that support higher dimensional and non-manifold data. The performance is on par with algorithms that are catered to restricted classes of input. The algorithm is also amenable to handle large data that do not fit in memory. It supports 2D, 3D, and higher dimensional meshes as input.

Input

We assume that the input manifold is represented by a triangulated mesh, the function is sampled at vertices, and linearly interpolated within each simplex. In the case of higher dimensional manifolds ($d \geq 3$), the algorithm requires only the 2- skeleton (vertices, edges, and triangles) of the mesh.



The library currently supports the following format for the input mesh:

OFF

*

- 1) Optional first line containing "OFF".
- 2) Next line specifies the no. of vertices (nv) followed by the number of triangles (nt) (space separated).
- 3) The next nv lines contain $x\ y\ z\ [f]$ where x, y & z specify the co-ordinates of the vertex and f specifies the function value. (If the input type is not f , then the function value is optional).
- 4) The next nt lines have $[3]\ v1\ v2\ v3$ where $v1, v2$ and $v3$ are the vertex indices of the vertices that form the triangles (the 3 is optional).

The first step of the algorithm is to construct the augmented simplicial complex. This is done by adding an auxiliary vertex in each simplex of the original simplicial complex, and connecting these auxiliary vertices to form a new set of edges. The auxiliary vertices are assigned the function values of their corresponding simplices. The augmented simplicial complex is then sorted in non-decreasing order of the function values at the auxiliary vertices.

The second step of the algorithm is to compute the Reeb graph by collapsing the augmented simplicial complex. The algorithm proceeds by iterating over the edges of the sorted augmented simplicial complex. For each edge, the algorithm checks if the vertices of the edge belong to the same connected component in the current Reeb graph. If they do, the edge is discarded. If they belong to different connected components, the algorithm adds the edge to the Reeb graph and merges the connected components.

To ensure the topological correctness of the resulting Reeb graph, the algorithm applies a set of rules during the edge collapse process. These rules ensure that the Reeb graph is a connected, acyclic graph that preserves the homotopy type of the original scalar function. Specifically, the algorithm ensures that the Reeb graph has a single root and that each non-leaf node in the Reeb graph corresponds to a critical point of the scalar function.

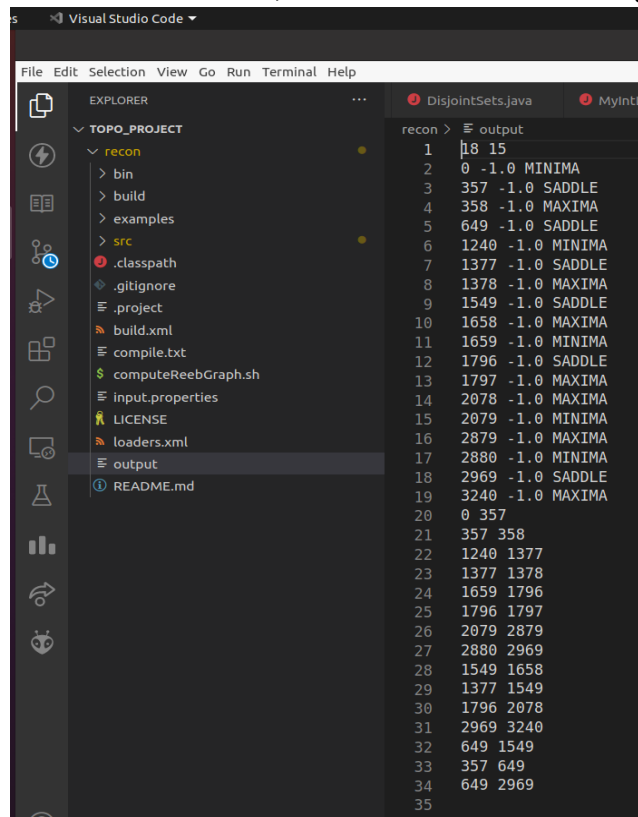
Table shows the time taken by our implementation to compute the Reeb graph for various models. We compare the performance of our algorithm with the online algorithm of Pascucci et al. While the online algorithm performs well for 2D data, our algorithm performs substantially better for 3D data. We expect that the algorithm will also be efficient in practice for higher dimensional input. The running time depends on the number of critical points, clearly indicating the output sensitivity of our algorithm.

	Model	#Triangles	#Critical points	Time taken (sec)	
				Our algorithm	Online algorithm [26]
2D	bunny	40000	217	0.7	0.06
	Laurent Hand	99999	92	1.0	0.24
	Neptune	998840	1757	9.0	1.76
3D	Engine	27252	160	1.2	0.26
	Solid 8-torus	34832	18	0.52	0.1
	PMDC	237291	902	2.7	4.9
	Blunt fin	451601	827	23.3	118.8
	Liquid Oxygen Post	1243200	132	13.0	481.1
	Bucky Ball	2524284	4378	197.9	9887.0 \pm
	Plasma	2646016	2852	396.3	11983.2 \pm
	SF Earthquake	4198057	11888	598.1	15949.1 \pm
Non-manifold	Crank (2D)	100056	253	1.5	0.56
	Armadillo-nonmanifold (2D)	331904	462	3.3	2.2
	Fighter (3D)	143881	3618	123.8	44.7

Output

It will be stored in the output file whose path is given in the "output" attribute of input.properties file.

1. The first line of output contains the number of critical points or vertices of the reeb graph n and the number of edges m of the reeb graph separated by a space.
2. The next n lines will display the critical points of the reeb graph and shows whether it is maxima, minima or saddle.
3. After those n lines, the next m lines contain edges of the graph.

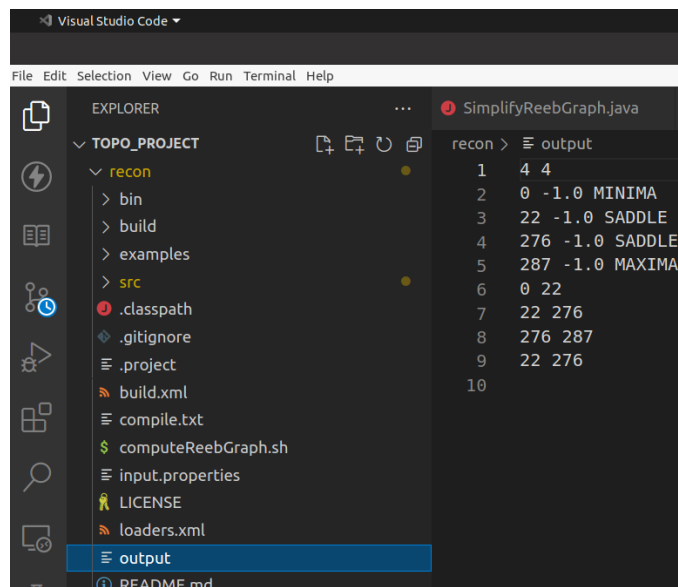


```

recon > output
1 18 15
2 0 -1.0 MINIMA
3 357 -1.0 SADDLE
4 358 -1.0 MAXIMA
5 649 -1.0 SADDLE
6 1240 -1.0 MINIMA
7 1377 -1.0 SADDLE
8 1378 -1.0 MAXIMA
9 1549 -1.0 SADDLE
10 1658 -1.0 MAXIMA
11 1659 -1.0 MINIMA
12 1796 -1.0 SADDLE
13 1797 -1.0 MAXIMA
14 2078 -1.0 MAXIMA
15 2079 -1.0 MINIMA
16 2879 -1.0 MAXIMA
17 2880 -1.0 MINIMA
18 2969 -1.0 SADDLE
19 3240 -1.0 MAXIMA
20 0 357
21 357 358
22 1240 1377
23 1377 1378
24 1659 1796
25 1796 1797
26 2079 2879
27 2880 2969
28 1549 1658
29 1377 1549
30 1796 2078
31 2969 3240
32 649 1549
33 357 649
34 649 2969
35

```

The above screenshot represents the Reeb graph, critical points, and edges for a teapot.



The above screenshot represents the Reeb graph, critical points, and edges for a torus.