Mesh Generator for Curved Surfaces

October 11, 2017

Contents

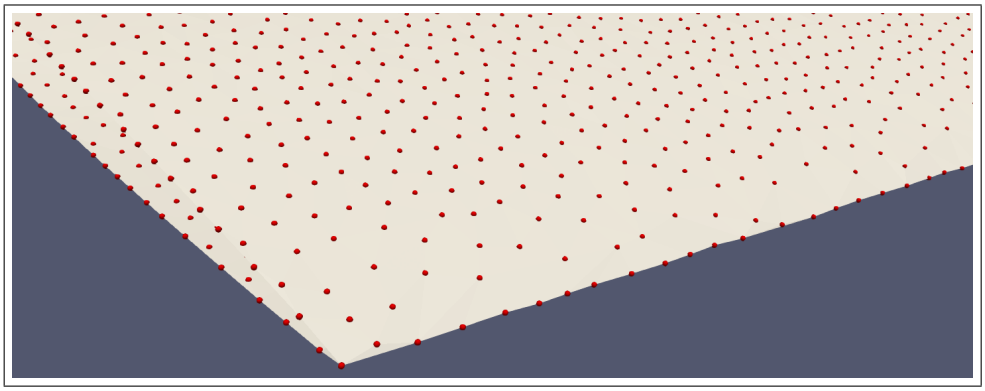

Figure 1: Initial points distribution on the surface

1 Summary

A mesh generator for creating triangulated representation of curved surface is described. The algorithm takes as an input a set of points on the surface and the corresponding surface-normal vectors and produces the list of surface triangles represented by the triples of node indexes.

2 Method

2.1 Overview

Figure 1 shows a typical node distribution provided as an input to the algorithm and Figure 2 shows the meshed surface representation produced by the algorithm.

The algorithm is based on a propagating-front (PF) method [1, 2] with the essential steps shown in Fig.3.

For this algorithm to work on curved surfaces, especially those with sharp edges, a measure of distance between the points should be modified from the conventional Euclidean distance to the one appropriate for a curved surface. In our case we found it convenient to specify a *bonding* function between the points, which represents the strength of connectivity between the points, and which depends on the positions and surface-normal vectors of the two points.

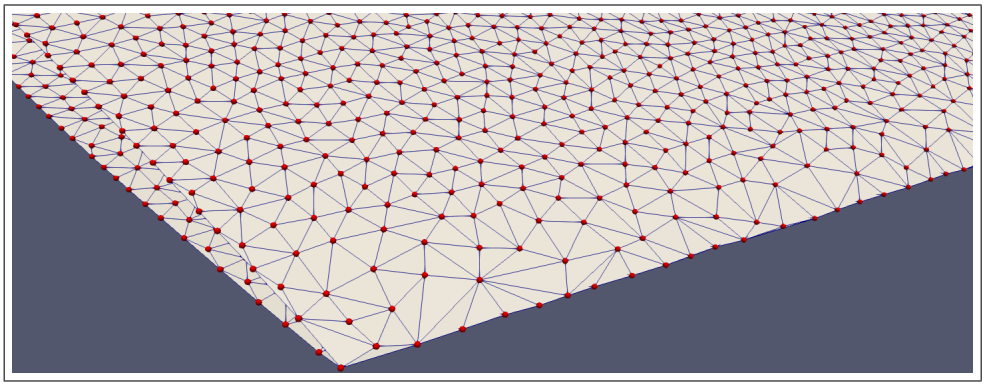

Figure 2: Triangulation from the initial set of points

In particular, the *bonding force* between points A and B is calculated as:

$$b = \frac{(\vec{N}_A \cdot \vec{N}_B)}{\mid \vec{X}_A - \vec{X}_B \mid} \tag{1}$$

where $\vec{N}$ is the surface normal vector, $\vec{X}$ is the position vector, $(\cdot)$ is a scalar product between the two vectors and the denominator has the Euclidean distance between two point positions. This bonding measure guarantees that points which can be close together but located on the opposite sides of the surface will not be considered as being close to each-other. Thus, a strong bonding will only be created between the points on the same side of the surface. This feature of measuring distance in an expanded space of distances and surface normals is essential to produces a continuous triangulation on a highly curved surface.

2.2 Initializing

The following data sets are initialized at the beginning of the algorithm:

1. Points (`points`).
2. Front nodes (`fnodes`).
3. Mesh nodes (`mnodes`).
4. Front segments (`fsegs`).

```
Create the first triangle and assign its edges to PF
WHILE there are still segments in PF DO
| FOR each segment in PF DO
| | Find its closest point
| | IF found THEN
| | | IF the new point belongs to PF THEN
| | | | IF this point is neigboring the current segment THEN
| | | | | IF the point is neibhboring segment on both ends THEN
| | | | | | Add new triangle made of the current segment and
| | | | | |     its two neighbors
| | | | | | Delete current segment and its neighbors from PF
| | | | | ELSE (point is neighboring on one end only)
| | | | | | Add new mesh triangle at the corner of the segment
| | | | | | Delete current segment and its neighbor from PF
| | | | | | Add one new segment to PF
| | | | ELSE
| | | | | Add new trianle with the segment at its base
| | | | | Delete current segment from PF
| | | | | Add two side segments to PF
| | | ELSE (the new point does not belong to PF)
| | | | Add new trianle with the segment at its base
| | | | Delete current segment from PF
| | | | Add two side segments to PF
| | ELSE (point not found)
| | | Assign current segment to the boundary
| | | Remove current segment from PF
```

Figure 3: Propagating Front Algorithm

5. Boundary segments (`bsegs`).

6. Surface mesh triangles (`mesh`).

All the data sets are implemented as doubly linked lists, capable of growing and shrinking as needed. The set of points is loaded from the external file, while other data sets are initially empty.

At the beginning the first triangle is created by selecting a first arbitrary point, finding a second closest point, forming the first segment between the two points, and then picking a third point closest to the middle of the segment. The three segments forming the first triangle are added to the list of front segments (`fsegs`) and the three vertices of the triangle are added to the list of front nodes (`fnodes`). After that the triangle is added to the list of mesh triangles (`mesh`).

Then the algorithm starts looping over the list of front segments as long as the list is not empty.

2.3 Finding Closest Point

For each segment of the front a middle point of the segment is considered and a point with the strongest bonding according to (1) is searched among the lists of free points and among the front nodes. To speed up the search only points at locations within a specified range are considered. That range value can be made dependent on location but in the current implementation it is constant throughout the domain.

There are two additional constraints applied to candidate points in the bond calculation:

1. New points should be located ahead of the front.

2. New point should not create intersection with an existing front segment (Fig.4).

The first constraint is realized by considering only the points which are on the positive side of the plane formed by the surface normal and the segment line. The surface normal vectors supplied as input to the algorithm are used for this purpose.

The second constraint is realized by using standard line intersection algorithm on a plane adapted for a 3D case by projecting all the points to the plane perpendicular to the surface normal vector.

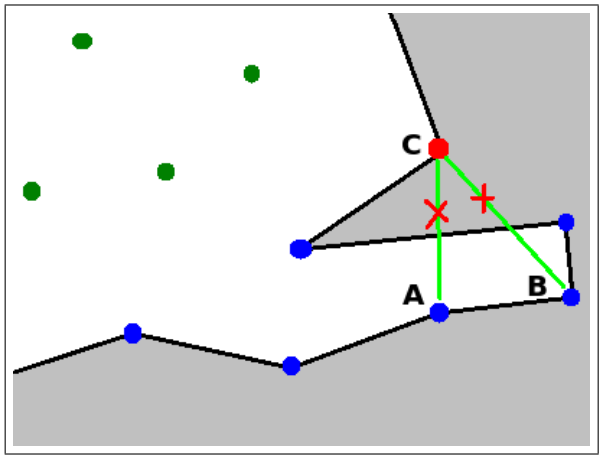

Figure 4: New point creates intersection

After the two searches over two respective lists (`points, bnodes`) are completed, three possibilities are considered: (1) there are no candidates for the next mesh node, (2) only one list has a candidate, (3) both lists have candidates for the next mesh node.

In the first case the current segment is added to the list of boundary segments (`bsegs`) and is removed from the list of front segments (`fsegs`). In the second case the only available candidate is considered, and in the last case the candidate with the strongest bond is considered.

After the candidate point is selected further processing depends on whether this point is on the front or not.

2.4 Point is not on the Front

Fig.5 shows a typical situation when the point selected for a new triangle does not belong to the current propagating mesh front. In this case two new segments (AC, BC) are created and the current segment (AB) is removed from the front. The resulting triangle ABC is added to the surface mesh.

2.5 Point is on the Front

When the selected point is on the front of the mesh, several situations are considered:

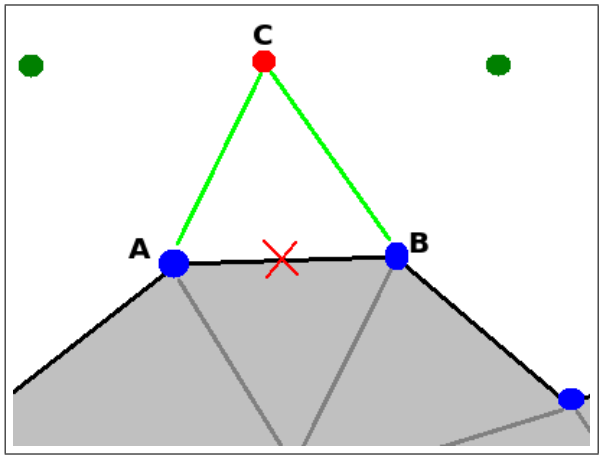

Figure 5: The closest point (C) is not on the front

1. *The point is not a neighbor of the current front segment* (Fig.6). In this case the new triangle ABC is formed, segment AB is removed from the front, and segments AC and BC are added to the front.

2. *The point is a neighbor of the current front segment* (Fig.7). In this case the new triangle ABC is formed, segments AB and BC are removed from the front, and segment AC is added to the front.

3. *The point is a neighbor to both vertices of the current segment* (Fig.8). This is a case of an isolated triangle. All three segments are removed from the front and the triangle is added to the mesh.

2.6 Closing Gaps

After the mesh generation is completed, some artificial boundaries can be produced at sharp edges of the mesh (Fig.9). This happens because sharp edges result in boundary segments which can be close together but don't join since the bonding force between them is too week (i.e. their surface normals point to the opposite directions). Thus, a separate routine is needed to mend those holes. This routine loops over all boundary segments, examines their proximity in Euclidean sense, and joins them if they are close enough (see Fig.10).

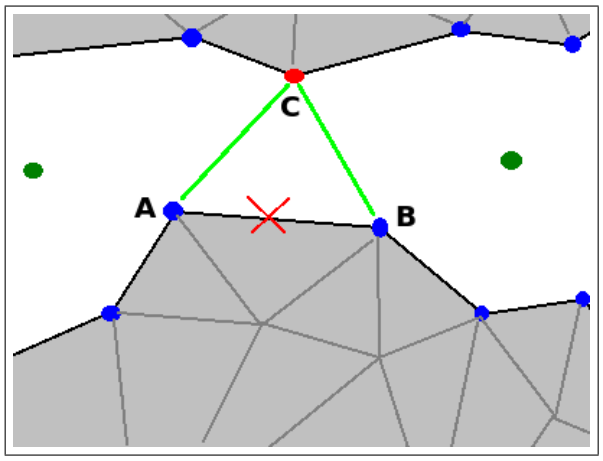

Figure 6: The closest point (C) on the front is not a neighbor

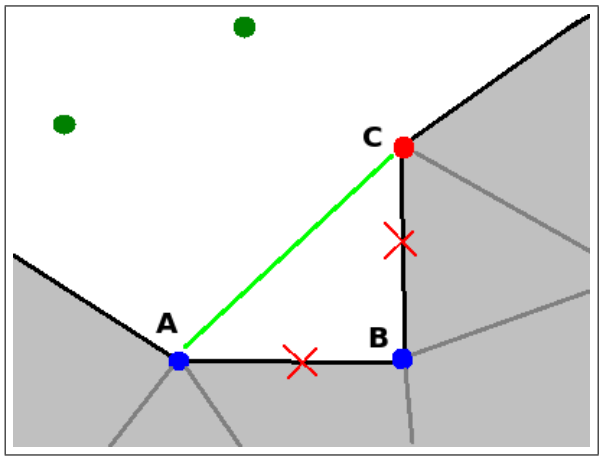

Figure 7: The closest point (C) on the front is also a neighbor

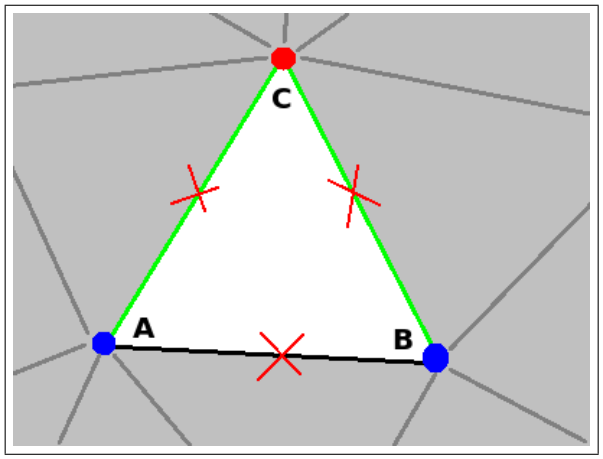

Figure 8: The closest point on the front is a double neighbor

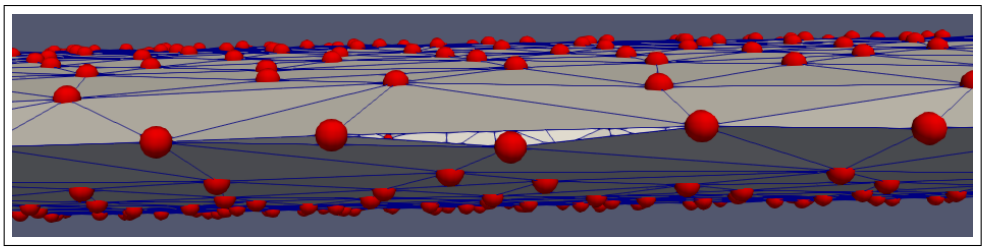

Figure 9: Sharp edges can produce artificial boundaries with gaps

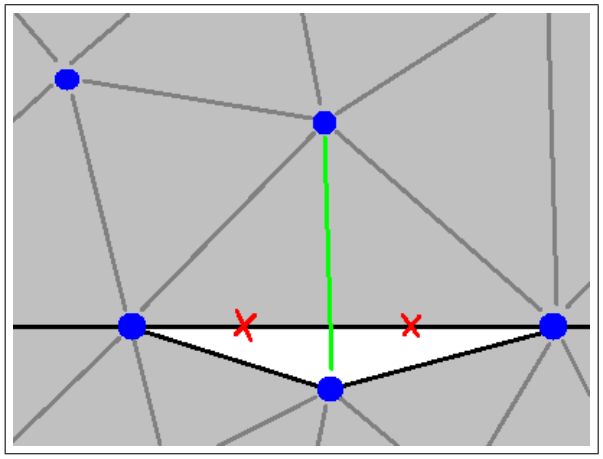

Figure 10: Closing gaps at the boundaries

3 Source Code

The source code is written in C++ and is provided in two files: `frontmesh.cc` and `frontmesh.h`. The former contains implementation of all essential mesh generation functions, while the latter includes description of basic data types such as: `Vector`, `Point`, `Node`, `Segment`, `Triangle`, and a general `Collection` class to provide link list operations on the basic data types. Several global constants defined in `frontmesh.h` file can be used for adjusting the behavior of the algorithm. In particular:

1. `MIN_ASPECT_RATIO`: the minimum aspec ratio for the triangles.
2. `ZIP_PROXIMITY`: the maximum distance to consider the boundary segments for joining.
3. `RANGE`: the distance beyond which the points are not considered for making a triangle with a current front segment.

The main routine reads node coordinates and surface normal vectors from text files in Tetgen *node* format [3] and outputs the resulting node connectivity information in Tetgen *face* format.

An additional converter `tetgen2stl` is provided to convert data from Tetgen to STL format.

Detailed information on compiling and running the code is provided in README.md file. Detailed list of data structures is provided in the accompanying code reference manual.

References

[1] S. Akkouche and E. Galin. Adaptive Implicit Surface Polygonization Using Marching Triangles. 20:67–80, 2001.

[2] S. Osher and J. A. Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on HamiltonJacobi formulations. *J. Comput. Phys.*, 79:12–49, 1988.

[3] WIAS. Tetgen File Formats. wias-berlin.de, 2017.