Self-intersection Removal in Triangular Mesh Offsetting

Wonhyung Jung[1], Hayong Shin [2] and Byoung K. Choi [3]

[1]Korea Advanced Institute of Science and Technology, jcircle@vmslab.kaist.ac.kr
[2] Korea Advanced Institute of Science and Technology, hyshin@kaist.ac.kr
[3] Korea Advanced Institute of Science and Technology, bkchoi@vmslab.kaist.ac.kr

ABSTRACT

Proposed in this paper is an efficient algorithm to remove self-intersections from the raw offset triangular mesh. The resulting regular mesh can be used in shape inflation, tool path generation, and process planning to name a few. Objective is to find the valid region - set of triangles defining the outer boundary of the offset volume from the raw offset triangular mesh. Starting with a seed triangle, the algorithm grows the valid region to neighboring triangles until it reaches triangles with self-intersection. Then the region growing process crosses over the self-intersection and moves to the adjacent valid triangle. Therefore the region growing traverses valid triangles and intersecting triangles adjacent to valid triangles only. This property makes the algorithm efficient and robust, since this method omits unnecessary traversing invalid region, which usually has very complex geometric shape and contains many meaningless self-intersections.

Keywords: regularization, regular triangular mesh, offset, self-intersection

1. INTRODUCTION

While NURBS is de facto standard for exact curve and surface, triangular mesh (T-mesh for short) is probably the most popular choice for approximate shape representation in many engineering applications including FE analysis, tool path generation, and reverse engineering, as well as computer graphics and geographical information system. It is often required to offset a T-mesh, which consists of two major steps: (1) raw offsetting, and (2) regularization. Raw offsetting is to obtain a T-mesh apart from the original mesh by the given distance, and the resulting mesh may have degenerate triangles and/or self-intersections. Regularization is the step to remove those abnormalities. Then, we obtain a regular T-mesh which is a 2-manifold triangular mesh free from degenerate triangles and self-intersections (See Fig. 1).

Computing offset model of a shape represented by a T-mesh can be used for tool path generation and process planning of a sculptured surface such as mold & die (Choi et al. [2]).

Boolean operation between T-meshes (Cardan et al. [1]) is very similar to T-mesh regularization in that it finds intersections between T-meshes and selectively collects portions as specified by the Boolean operator. Hence the algorithm described in this paper can be applied to Boolean operation between T-meshes. The main distinction is that the triangle set in Boolean operation problem is already separated into two groups, which makes the self-intersection search a little easier.

Simulation of deformable objects in Ref. [6],[7],[10], [13] deals with self-collision detection and collision response. This is similar to T-mesh regularization in that it needs to detect self-collision efficiently. However, self-collision points are not to be removed, but to be repositioned and the resemblance between frames can be exploited in order to expedite self-collision computation because the simulation of deformable objects needs to compute multiple frames.

In this paper, we present an efficient algorithm to obtain the regular T-mesh from the raw offset T-mesh. A raw offset mesh is usually achieved by simply moving vertices in a smooth region along their normal

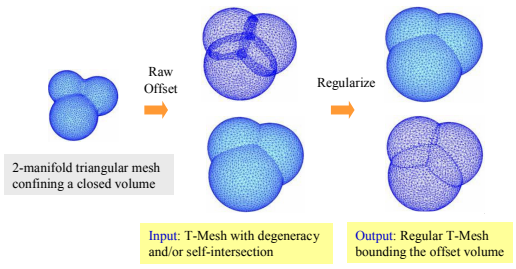

Fig. 1. Mesh regularization.

vector estimated using their incident triangles. (Smooth region means where the angle between adjacent triangles is small enough to be ignored.) However, vertices in a sharp region should be handled differently. The vertices in a sharp region are offset along the incident triangle's normal and the gap is filled by inserting a spherical mesh and a cylindrical mesh during the raw offset. For the details of raw offset process, the readers are referred to Jun [5].

2. BASIC OBSERVATIONS

Objective is to find the valid region - set of triangles defining the outer boundary of the offset volume from the raw offset T-mesh. Triangles in the input raw offset mesh can be classified into three groups: valid triangles, invalid triangles, and partially valid triangles. Fig. 2 shows these groups and related basic observations. Valid triangles are the ones to be entirely contained in the valid region and remain in the mesh after the self-intersection removal. Invalid triangles are the ones not participating in the valid region and should be deleted entirely. Partially valid triangles lie on the boundary between valid region and invalid region. A partially valid triangle has intersections with other triangles, and a portion of a partially valid triangle is to be included in the resulting mesh. A partially valid triangle needs to be split into sub-triangles. Then, sub-triangles are to be classified: valid and invalid sub-triangles.

Problem is how to classify triangles of the raw offset T-mesh into these three groups efficiently.

Intersecting triangles are the ones intersecting with other(s). A partially valid triangle is an intersecting triangle. However, the converse is not true as shown in Fig. 2. There are numerous invalid intersecting triangles. Therefore, it is important to compute triangle-triangle intersection only when necessary. The focus of our algorithm to be explained in the following section is to avoid unnecessary traversing and splitting invalid triangles.

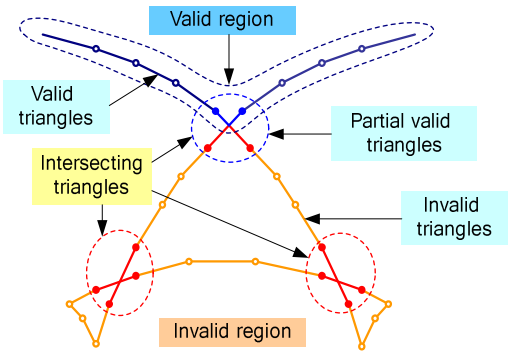

Fig. 2. Basic observations: three triangle groups & valid/invalid region.

3. MESH REGULARIZATION ALGORITHM

We assume that the original T-mesh before raw offset is a closed surface, namely, 2-manifold T-mesh not including internal void and the input mesh for this algorithm is obtained by offsetting the original T-mesh outward.

Fig. 3 shows the overall procedure of the proposed algorithm and Fig. 4 explains the steps with an example. Step 1 is to find and delete degenerate triangles with virtually zero area. Step 2 is to compute self-intersection segments efficiently by constructing a bucket structure. Then the valid region search starts with finding a valid seed triangle in step 3. The seed triangle forms the initial valid region. In step 4, the valid region grows from the seed triangle to neighboring triangles. This step also includes splitting the partially valid triangles. Once all valid triangles are marked, the remaining invalid triangles are removed and the self-intersection edges are stitched by assigning topological relation between adjacent valid triangles in the step.

3.1 Removing degenerate triangles

A degenerate triangle is the triangle with (almost) zero-area. Removal of edges with length $l < \varepsilon_1$ (zero length tolerance) by edge collapse as in Hoppe [4] and swapping diagonal edges if the minimum angle $\alpha < \varepsilon_2$ (zero angle tolerance) are used to remove degenerate triangles as shown Fig. 5.

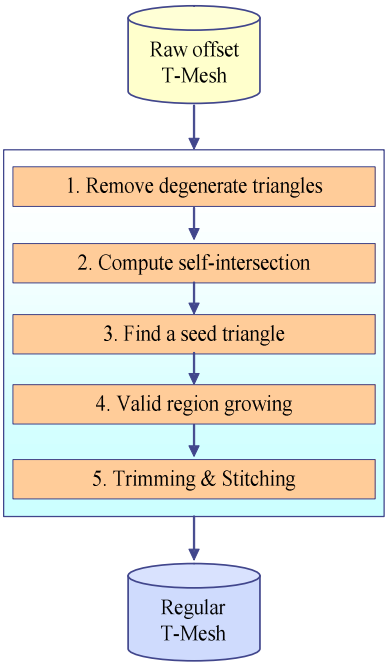

Fig. 3. Overall procedure

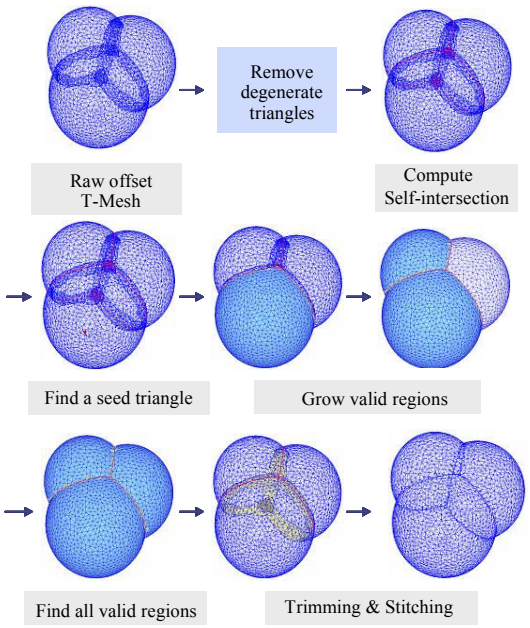

Fig. 4. An explanatory example of overall procedure.

3.2 Computing self-intersections

Brute force intersection computation requires to test all triangle pairs, which takes $O(N_2)$ intersection comparison for T-mesh with N triangles. Since we want to avoid computing intersections between invalid triangles, we need an efficient structure for reducing the number of triangle-triangle intersection (TTI) tests. Among many structures for such purpose, we use a bucket structure for its simplicity, which partitions the input T-mesh into buckets, where each bucket contains geometrically coherent triangles less than a fixed number (the bucket capacity *C*).

Constructing bucket structure starts with a single bucket containing all triangles. If the number of triangles in a bucket is bigger than *C*, the bucket is subdivided into two by the plane splitting the longest side of its AABB (Axis Aligned Bounding Box). Triangles crossing the plane are stored in both of the buckets. The bucket subdivision process is applied recursively until each bucket contains less than *C* triangles or no improvement can be made. Once the bucket structure is constructed, the self-intersection test can be confined within a bucket.

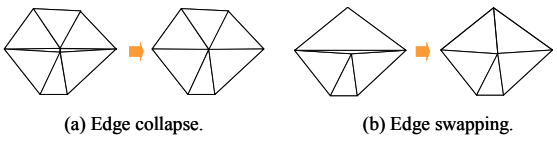

Fig. 5. Deleting degenerate triangles.

For each bucket, we simply compare all pairs of triangles within a bucket. For the fast TTI test, we use 'interval overlap method' suggested by Moeller [8],[9]. Each intersection segment stores the pointers to both participating triangles, and a triangle maintains the list of intersection segments which belongs to it.

The bucket subdivision process requires the computation time of $T_1 N \log_2 {}^{N}\!/_{C}$, where T_1 is intersection testing time between the bounding box and a triangle. Then the self-intersection test takes the computation time of $T_2 CN$, where T_2 is intersection testing time between two triangles.

Therefore, self-intersection computation using bucket structure requires the computation time:

$$O(N \log_2 \frac{N}{C}) + O(CN) \quad (1)$$

Our empirical test shows that the self-intersection computation time increases almost linearly given the bucket capacity *C*. However, *C* value to minimize the calculation time varies with the mesh size of the model. Also, as one can see in Section 4 (empirical results), this step (computing self-intersections) takes the majority of total time even with bucket structure. Avenues for further improvement on this step will be discussed in the conclusion.

3.3 Finding a seed triangle

Input T-mesh $TM =< V, T >$ is the raw offset T-mesh satisfying the previously stated assumptions, where *V* is the set of vertices of *TM*. *T* is the set of triangles and their adjacency information. A seed triangle is a valid triangle to initiate the valid region growing. Let $VC \subset V$ be the set of vertices on the convex hull of *V*.

Obviously *VC* belongs to the valid region. Any triangle $t \in T$ having at least a vertex in *VC* is valid or partially valid and can serve as the seed triangle. Even more simply, a triangle touching AABB of the input T-mesh is valid or partially valid. Among those triangles, we select a valid triangle as the seed. (For the simplification of the subsequent discussion, we assume that at least one triangle touching AABB is a valid one. Note that this assumption can be easily removed by slightly modifying the algorithm in section 3.4.)

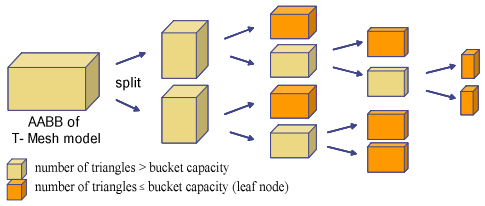

Fig. 6. Bucket subdivision process

3.4 Valid region growing

To find the valid region, we use region growing approach starting from the seed triangle found in the previous section. Instead of splitting all intersecting triangles beforehand, we split only partially valid triangles at the point of valid region growing in order not to hassle with unnecessary invalid triangles. Detailed algorithm of the valid region growing is composed of the steps as follows:

1. Each triangle has a three states : unvisited, valid, partially valid. Initially, all triangles are marked as unvisited.
2. The seed triangle (found in Section 3.3) is marked as valid and inserted into **S**, the set of wave front triangles for region growing.
3. Go to 5 if **S** is empty. Otherwise, remove a triangle T from **S**.
4. For each unvisited T_a adjacent to T, if T_a has no intersections, then it is marked as valid and inserted into **S**. Otherwise, T_a is marked as partially valid and inserted into **P**, the set of partially valid triangle confronted. T_a is stored in **P** together with the information to indicate the 'entrance edge' e_p, which is the edge shared by T and T_p. Then go to 3.
5. Go to 7 if **P** is empty. Otherwise, remove a partially valid triangle T_p and its entrance edge e_p from **P**.
6. Sub-triangulate T_p (details in Section 3.5). Propagate the valid region over T_p and its counterpart triangles (as detailed in Section 3.6). If another seed triangle is found, it is inserted into the seed queue and then go to 2. Otherwise, go to 4.
7. The valid region growing step is completed.

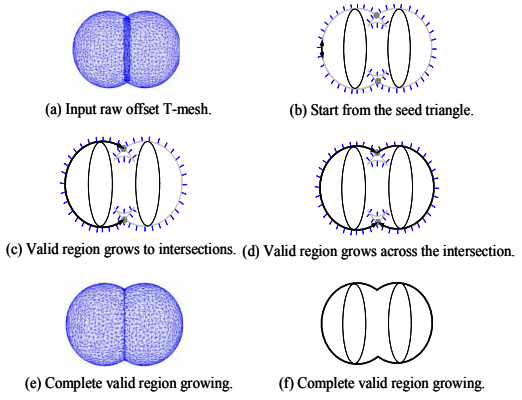

Fig. 7. An explanatory example of the valid region growing.

3.5 Sub-triangulation

A partially valid triangle T_p has (possibly many) intersection segments in it. This step is to perform two tasks : (1) to split T_p into sub-triangles which contains the intersection segments as their edges (Fig.8(a)), (2) to propagate valid region within the sub-triangular mesh (Fig.8(b)).

The detailed steps for the first task is as following :

1. Split each edges e_i of T_p by all intersection segments $\{s_i\}$.
2. Split each s_i at the intersection points among them.
3. Sub-triangulate T_p by 2D constrained Delaunay triangulation [11] together with edges and intersection segments split from 1 and 2.

As shown in Fig. 8(b), the valid region growing in the sub-triangular mesh starts from the entrance edge e_p. The sub-triangle t_{sub0} which is adjacent to e_p is marked as valid and becomes the seed for the valid region growing in the sub-triangular mesh. (Note that t_* denote a sub-triangle.) Then the valid portion of T_p grows into neighboring sub-triangles until it reaches intersection segments, which play the role of the entrance edge for the counterpart (partially valid) triangle T_c in the next step described in Section 3.6.

3.6 Crossing the river

The region growing process crosses over the self-intersection and moves to the sub-triangles of the counterpart triangle. Fig. 9 illustrates detailed steps of propagating into the sub-triangles of the counterpart triangle across the intersection of a partially valid triangle.

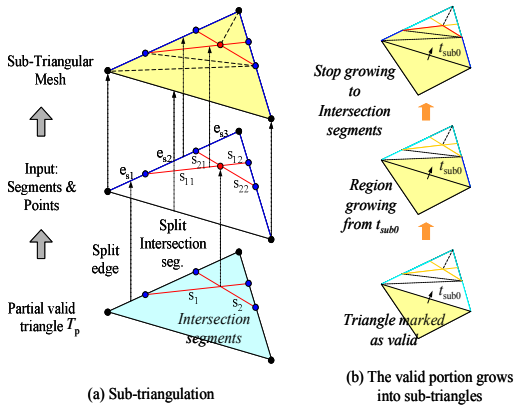

Fig. 8. Sub-triangulation & valid portion growing in sub-triangular mesh.

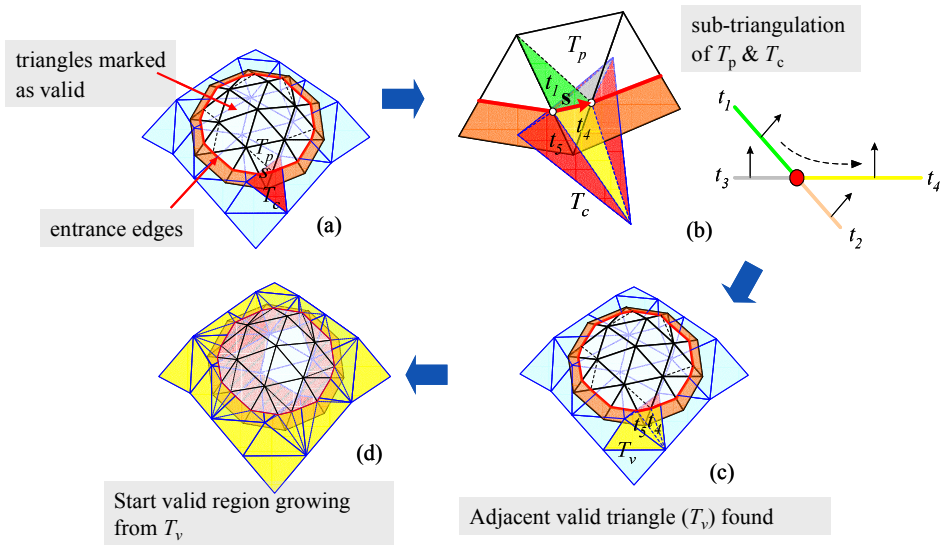

Fig. 9. Detailed steps of crossing the river.

This process starts by sub-triangulate the counterpart triangle T_c as in Section 3.5. In Fig.9 (b), there are two sub-triangles t_3 and t_4 of T_c adjacent to the previously found entrance edge. (Note that t_1 and t_2 are sub-triangles of T_p.) By considering the normal orientation compatibility with T_p, the sub-triangle t_4 is selected as valid one, which serves as the valid seed triangle in the region growing within the sub-triangular mesh of T_c, which is similar to Section 3.5. Eventually, as shown in Fig.9(c), T_v is found as a valid triangle and this is inserted into **S**.

3.7 Trimming & Stitching

Since all partially valid triangles are replaced by sub-triangles and all valid triangles are marked, the trimming and stitching can be done very simply. The trimming step is to retain valid triangles only and to remove invalid ones. And the next step is to stitch the self-intersection edges by assigning topological relation between adjacent valid triangles. Fig. 10 shows the result of trimming and stitching.

Fig. 10. Trimming & stitching example.

4. RESULTS

Fig. 11 – 14 show some mesh regularization examples removing self-intersections from the raw offset T-mesh by the valid region growing proposed in this paper. In each figure, the original T-mesh, its raw offset T-mesh and self-intersections, the invalid region left after all the valid triangles is visited by region growing, and the resulting regular T-mesh are showed. Table 1 lists the experimental results of these examples. As shown in Tab. 1, most of the execution time is spent by self-intersection calculation step. The experiments were done on a PC with Pentium 1.8GHz CPU and 1GB memory.

Name	# of Triangles	# of Intersection Segments	Time (sec)		
			Self-intersection Calculation	Region growing & Stitching	Total
Bunny	15,224	4,253	2	0.454	2.454
Knot	150,776	44,211	11.766	0.656	12.422
C-arm lower	173,880	14,141	19.266	2.591	21.857
Pump	256,672	35,401	50.781	3.140	53.921

Tab. 1. The experimental results

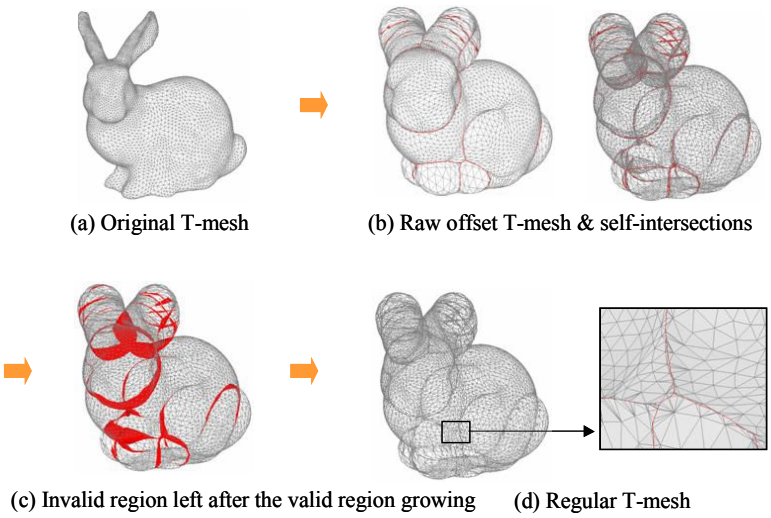

Fig. 11. Bunny.

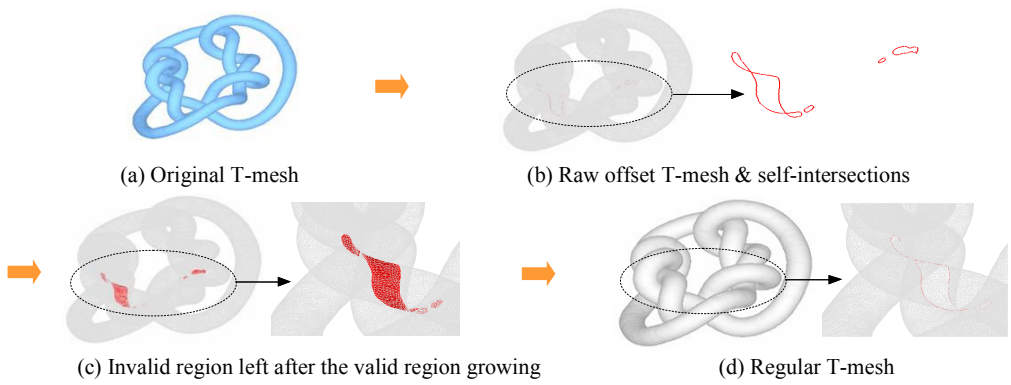

Fig. 12. Knot.

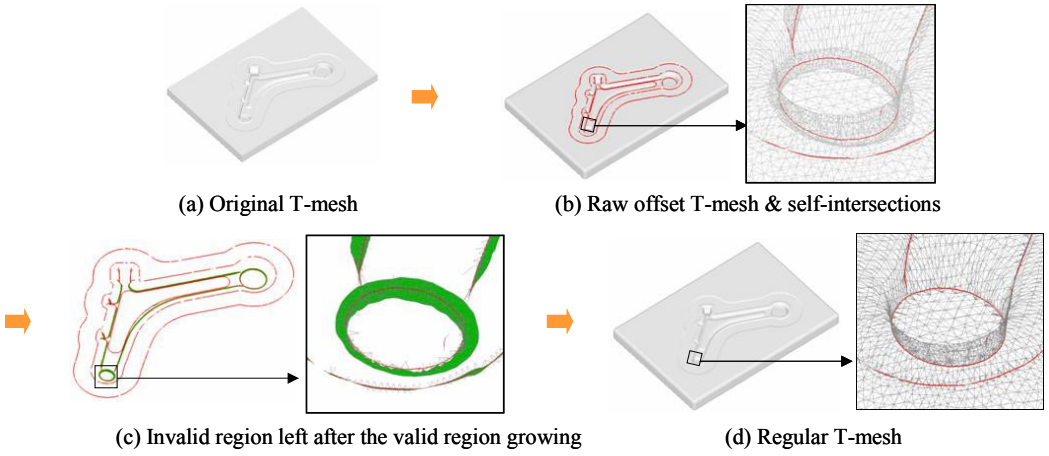

Fig. 13. C-arm lower.

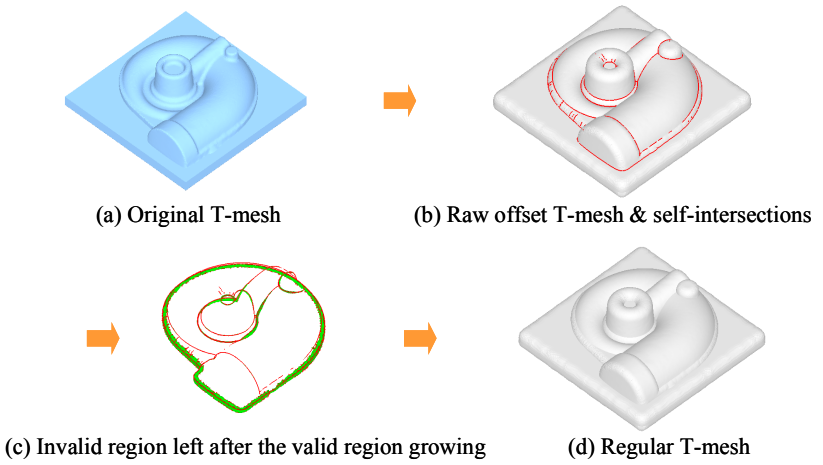

Fig. 14. Pump.

5. CONCLUSION & FURTHER RESEARCH

In this paper, we present the mesh regularization algorithm using region growing from a seed triangle. The main advantages of this method are that the region growing traverses valid or partially valid triangles only. This property makes the algorithm efficient and robust, since in many practical examples the invalid region tends to have very complex geometric shape and contains many meaningless self-intersections. Our algorithm skips over this invalid region traversing.

The proposed algorithm utilizes a bucket structure to facilitate self-intersection computation. However, as shown in Table 1, still the majority of the computation time is consumed in the intersection finding step. This can be improved by introducing more sophisticated hierarchical structure, such as octree-like space partitioning tree or object partitioning tree (OBB tree or k-DOP tree as explained in [9]).

Furthermore, our proposed algorithm finds all the self-intersections without separating local and global self-intersection tests. If we can segment the raw offset mesh model into areas having no local self-intersection, then by computing the intersections (global intersection test) between these areas, we may be able to reduce the number of self-intersection test between triangles. For developing such method, the visibility condition described in [13] can be helpful.

Though the current implementation of the proposed algorithm in this paper computes all intersections beforehand, it is not necessarily the best choice. The algorithm can be even further improved by performing the self-intersection test only with the triangle in the wave front of the region growing process on the fly. This will omit the unnecessary self-intersection computation between invalid triangles.

We assumed that the original T-mesh is a connected closed surface and the input T-mesh is obtained by offsetting the original T-mesh outward. These assumptions are mainly for obtaining initial seed triangle easily. If we can efficiently obtain seed triangles for general T-meshes, these assumptions can be relaxed without changing the rest part of the algorithm, so that we can handle open meshes, multi-component meshes, or inward offset meshes.

6. ACKNOWLEDGEMENTS

This research was supported by the Ministry of Science and Technology of Korean government through NRL grant.

7. REFERENCES

[1] Cardan, Y. and Perrin, E., An algorithm reducing 3D Boolean operatoins to a 2D problem: concepts and results, Computer-Aided Design, Vol. 28, No. 4, 1996, pp 277~287

[2] Choi, B.K., Kim, D.H. and Jerard, R.B., C-space approach to tool-path generation for die and mould machining, Computer-Aided Design, Vol.29, No.9, 1997, pp 657-669

[3] Choi, B.K. and Jerard, R.B., Sculptured Surface Machining – theory and applications, Kluwer Academic Publishers, 1998

[4] Hoppe, H., Progressive meshes, ACM SIGGRAPH, 1996, pp 99-108

[5] Jun, C.S., Exact Polyhedral Machining, IFIP WG5.3 International Conference on Sculptured Surface Machining (SSM98), 1998

[6] Lau, R. W.H., Chan, O., Luk, M., Li, F. W.B., A Collision Detection Framework for Deformable Objects, ACM VRST'02, November 11-13, 2002

[7] Mezger, J., Kimmerle, S. and Etzmub, O., Progress in Collision Detection and Response Techniques for Closth Animation, Proc. Of the 10th Pacific Conference on Computer Graphics and Application, 2002
[8] Moller, T., A Fasst Triangle-Triangle Intersection Test, Journal of graphics tools, Vol.2, No.2, 1997, pp 25-30
[9] Moeller, T. and Haines, E., Real-Time Rendering, AK Peters, 1999
[10] Provot, X., Collision and Self-Collision Handling in Cloth Model Dedicated to Design Garments, Graphics Interface, May 1997, pp 177~189
[11] Ruppert, J., A Delaunay Refinement Algorithm for Quality 2-Dimensional Mesh Generation, Journal of Algorithms, Vol.18 No.3, May 1995, pp 548-585
[12] Samet, H., Applications of Spatial Data Structures - Computer Graphcis, Image Processing and GIS, Addison-Wesley, 1990
[13] Volino, P., Thalmann, N.M., Efficient self-collision detection on smoothly discrretized surface animations using geometrical shape regularity, Computer Graphics Forum (EuroGraphics Proc.), Vol. 13, 1994, pp 155~166