Efficient Boolean Operation on Manifold Mesh Surfaces

Article *in* Computer-Aided Design and Applications · August 2013

DOI: 10.3722/cadaps.2010.405-415

Ming Chen
Guangxi Normal University
28 PUBLICATIONS **140** CITATIONS
SEE PROFILE

Kai Tang
The Hong Kong University of Science and Technology
194 PUBLICATIONS **2,712** CITATIONS
SEE PROFILE

Matthew Yuen
The Hong Kong University of Science and Technology
222 PUBLICATIONS **6,947** CITATIONS
SEE PROFILE

Efficient Boolean Operation on Manifold Mesh Surfaces

Ming Chen, Xiao Yu Chen, Kai Tang and Matthew M. F. Yuen

Hong Kong University of Science and Technology, mektang@ust.hk

ABSTRACT

We present an efficient Boolean operation algorithm for mesh models. Compared with existing ones, the proposed has several distinct features that include: (1) the inside/outside classification for each triangle element is determined by both LDI (Layered Depth Images) technology and vertex connecting propagating rules – thus the heavy numerical and analytical computing for the classification of triangles is avoided, and (2) the important triangulation is specially treated which efficiently expedites the downstream processing. Test examples show that our algorithm can obtain a Boolean result at an interaction rate for a medium-sized mesh model (around 10,000 triangle faces) and for large mesh models the speed is also found to be acceptable in practice.

Keywords: Boolean operation, LDI model, point model, inside/outside classification.
DOI: 10.3722/cadaps.2010.405-415

1 INTRODUCTION

In a mesh modeler, Boolean operation is considered the most important and powerful modeling means for modeling complex shapes. Much work has been done in this area based on intersection and boundary evaluation [1]. In order to express Boolean operations more elegantly, some researchers try to perform Boolean operations based on implicit representation [2]. However, this implicit method can not generate an explicit representation model, which though is required in many industry applications such as rapid prototyping. Recently, volume based representations become popular [5-9], in which the objective mesh model is sampled into a volume model of some type, such as voxel [8, 9] and surfel [3-4]. Based on this technology, Boolean operations can be carried out in a more robust and efficient way; but as a trade-off, in order to output a triangle mesh model; all these methods require two time-consuming steps: adaptive sampling and triangulation. In the adaptive sampling step, heavy computing cost is paid for preserving model features so that the final sampled model will keep a high fidelity with the original model. After the sampling step, the Boolean operation is performed that results in a "BOOLEANED" volume model, which must be rebuilt through triangulation. For a large volume model, this triangulation step usually takes too much time, although from the designer's point of view it should be at an interaction rate. Moreover, the triangulation result is often error-prone [5].

In this paper, the mesh models to be considered are manifold triangle mesh models that are topologically complete – in other words, all the facets can be iterated by enquiring the edge connecting, and vertex incident relationship. Generally, the cost of Boolean operation is mainly due to three stages: triangulation, inside/outside classification of triangles and facet-facet intersection. Since, as it is well known, in a CAD model the facet-facet intersection calculation is virtually impossible to avoid in a Boolean operation, in this paper we aim to reduce the computing cost of the other two stages, i.e., triangulation and inside/outside classification.

1.1 Contributions

The contributions of this work are in two aspects:

- The inside/outside classification is efficiently performed by combining the LDI (Layered Depth Images) technology and the triangles' neighboring information, which greatly improve the computing time due to the absence of other computation otherwise needed.
- The triangulation is efficiently carried out, in which special treatment is paid according to practical use.

1.2 Relevant Work

Intensive research has been carried out in Boolean operations based on trimmed surfaces. Regardless of their implementation specifics, most Boolean algorithms proceed in three steps [10]: (1) calculating intersections between two B-rep based models, either numerically or analytically; (2) identifying a region within the domain whose evaluation is skipped; (3) rebuild a new B-rep model through the valid domain region. Most of the current geometric modeling kernels, such as Parasolid and ACIS, have adopted this paradigm and are widely used in commercial CAD software. While this exact approach based on surface intersection and boundary classification provides accuracy, the evaluation of the trim curves in the parametric domain is however notoriously difficult and requires potentially very expensive machinery to deal with topological inconsistencies such as cracks. Thus, conceivably, the exact-intersection-based approach is prone to robustness problems [11]. In practice, the number of complex geometries involved is usually very large, and many degenerated cases occur, hence the exact-trimmed-surface-based Boolean operation is rather difficult and challenging for modeling complex shapes.

In order to solve the robustness problem, the volumetric representation is widely adopted to approximately represent the model and the representative representation types include voxel [9], distance field [16], surfel [3-4], and ray-rep [6-8]. The geometric operations based on volumetric representation are easy to implement and robust. However, most of the current volumetric methods fall short in performance and accuracy. With a volumetric method, usually a "triangulation" phase is required to rebuild the resulting volume model into a triangle meshed model for further manufacturing consideration. This phase is usually error-prone and time consuming.

2 METHOD

This section presents an overview of our algorithm followed by a detailed description of four stages comprising the algorithm.

Input. The algorithm takes a 3D water tight manifold triangle mesh model as an input. In addition, the input mesh model should be topologically complete. More explicitly, we assume that a mesh model M = $\{V_i, M_i\}$, where V_i represents all the vertices of the mesh, and M_i is a look-up table which records the neighboring and owner relationship among triangle faces, edges, and vertices. In order words, for each triangle face, we can obtain its neighboring triangle faces sharing with a same vertex or edge; for each edge, we can find two faces, which share the edge; for each vertex, we can get all faces which contain the vertex.

Output. The algorithm outputs a final resultant mesh model per the given Boolean operator, i.e., union, difference or intersection.

Algorithm overview. Our algorithm proceeds in four stages.

Stage 1. This stage calculates the intersection of axis-aligned bounding boxes of pairs of objects. If the intersection is non-empty, using the method in [7] we compute an LDI (Layered Depth Images) model within the intersected bounding-box, with the help of graphic hardware that can quickly decide which pair of triangle facets intersect each other. One point worth mentioning is that the LDI model can be efficiently constructed with preset resolution using graphic hardware.

Stage 2. This stage computes the intersection line-segments with each pair of intersecting facets and connects them into multiple poly-lines, which split each facet into many inside or outside sub-domains.

Stage 3. This stage determines the inside/outside of sub domains with the information of LDI, which converts the classification from 3D to 1D, thus greatly improves the algorithmic performance. Further, the inside/outside classification is propagated to all the other remaining unmarked triangles or sub-domains until all of them are inside/outside tagged.

Stage 4. This stage triangulates the kept sub-domains into new triangles. All these kept triangles (some triangles are discarded based on their inside/outside information) together with those newly generated triangles collectively represent the final result model. In this stage, we find that in practice some typical cases frequently occur in which the triangulation can be simplified that in turn can greatly enhance the computing efficiency.

In the following, we give detailed description of each stage.

2.1 Efficient Sampling with LDIs

Layered Depth Images (LDI) [15] was first proposed by Heidelberger [7] for the task of collision-detection in solid modeling. An LDI model can be efficiently obtained from its original counterpart by shooting (many) rays through each pixel center along primal axis in regions of interest and calculating the intersections between the rays and the surfaces. For each ray, the intersection points on it will be stored in depth order. If the sampling is dense enough, the intersection points can approximately represent the solid model with enough accuracy.

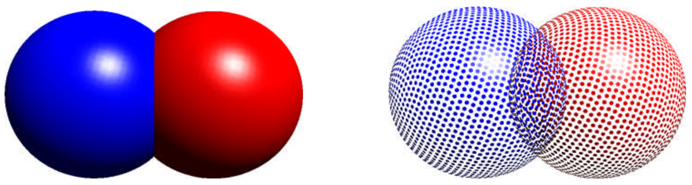

Fig. 1: LDI based point model. Left: the two original models; right: the point model.

With the help of graphics hardware, this procedure can be efficiently accomplished. In our application, the LDI model is used only to assist determining the inside/outside classification; therefore, only the interested region is sampled as an LDI model with a preset resolution. Actually, in this stage, we adopt the idea proposed by Heidelberger [7], which takes three steps: (1) calculating the intersection volume noted as VoI by the AABB (axis-aligned bounding-box) intersection method; (2) using graphics hardware to efficiently calculate the LDI model for VoI, in which each pixel contains the depth information of all the intersection points between the ray and the objects; and (3) based on the depth information, determining the collision.

As the step (2) is taken using graphics hardware in GPU, its speed is much faster compared with the traditional computational methods. But this method (from [7]) was originally devised only for collision detection, not for Boolean operation, in which the line-segment intersection between each pair of collided triangles should be calculated – in other words, we must now determine which pair of triangles is intersected with each other. In order to solve this problem, we further encode a unique ID for every triangle face into a RGB color. After rendering all the faces by the preset colors, by reading the color of the pixel, for any intersection point it can be easily figured out to which face it belongs. The inside/outside classification of a triangle can also be straightforwardly determined by examining the colors of the relevant intersection points.

After getting the LDI model, we next scan all the pixels and test the depth difference of each pair of neighboring intersection points. If the difference is within a preset (small) threshold, the two corresponding triangles are counted as intersecting each other. All the intersecting triangles are tagged as boundary triangles (BT) and are added into a boundary triangle pool. For each pair of interesting BTs, we calculate the intersection line segments, and for each BT, we use a list to record its intersection line segments.

Next, we need to determine for all the triangles their inside/outside classification. This information will help us determine which triangles should be kept or discarded and which should be triangulated for further process.

2.2 The Inside/outside Classification

After the LDI sampling, we partition all the triangle faces into three types: the outside triangle denoted as OT which is outside the overlapping bounding box, the boundary triangle denoted as BT (which are intersecting triangles), and LT for all the rest triangles. In a Boolean operation, the inside/outside classification is a critical task of identifying the relative position relation between the objects. After all the parts have been classified, all their inside/outside tagged parts will be preserved or discarded according to the specific Boolean operation type, and the preservation scheme is shown in Tab. 1.

Operation	*Points of A kept*	*Points of B kept*
$A \cup B$	Outside B	Outside A
$A \cap B$	Inside B	Inside A
A-B	Outside B	Inside A
B-A	Inside B	Outside A

Tab. 1: The preservation table of Boolean operations.

After the classification, all the kept parts of A and B will collectively represent the final Boolean result. From the above description, we can easily mark all OT triangles as outside triangles. But the classification for LT and BT is still uncertain at this point.

2.2.1 The classification of LT triangles

Among the LT triangles, many have been obtained by the intersecting ray, but some may have been just missed. For those successfully hit LTs, we can easily determine their classification via their depth information. E.g., as illustrated in Fig.2, there are four recorded points relating object A with B, i.e., P_1, P_2, Q_1 and Q_2, which are ordered according to their depth values.

For each ray, the entry and exit points alternately appear, and an interval bounded by the entry and exit point should represent an inside portion of the object in 1D, e.g., P_1 and P_2 delimit an inside interval of A along the ray. Based on this information, we can easily classify all the LT triangles: referring to Fig. 2, the yellow point is the sample point for a LT triangle of object B; if this sample point stays on the interval of line segment P_1P_2, the LT triangle will be marked as being inside object A; otherwise, it is outside A.

Due to the limit of sampling resolution, some LT triangles may have been missed by the rays, and these left out LT triangle together with all the BT triangles will be pushed into a pool Ω for further classification.

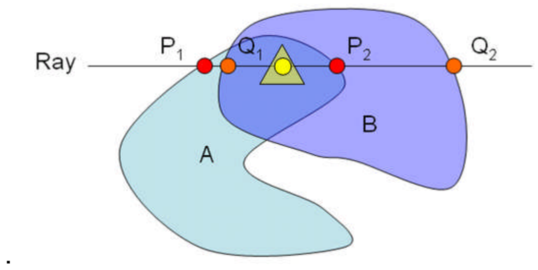

Fig. 2: Illustration of inside/outside point classification.

2.2.2 The classification of BT sub domains

For a BT triangle, we can not classify its inside/outside as we do for an LT triangle, since it contains both inside and outside parts which are separated by the intersection line-segments. Thus, we first connect all the intersection line-segments end-to-end to form splitting poly-lines, which split the involved triangle into outside and inside parts, referring to Fig. 3

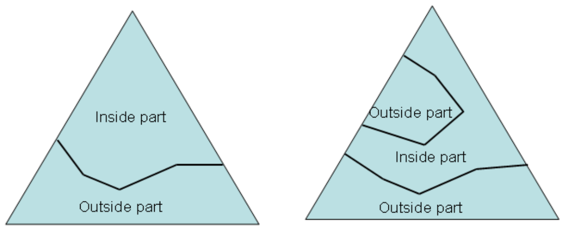

Fig. 3: Illustration of poly-lines splitting a triangle domain into inside/outside sub-domains that alternately occur: (left) Two split parts and (right) Three split parts.

The process of connecting line-segments into a poly-line can be carried out easily and efficiently with the triangle edge connecting information. More explicitly, referring to Fig. 4, the two intersection line-segments in the yellow triangle can be connected end-to-end, as the other two involved intersected triangles (the aqua colored) are two neighboring triangles sharing a common edge. After poly-lines are obtained, the whole domain of a BT triangle can be split into multiple inside/outside sub-domains which alternately appear (thus once one sub domain's classification is determined, so are the rest). Therefore, the problem can be simplified as how to decide the classification of the first sub domain, as we describe next.

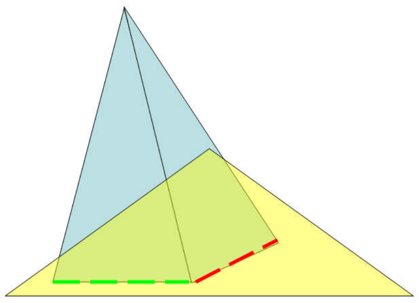

Fig. 4: Connecting intersection line segments into poly-lines with the help of edge connecting information.

By now, we have certain LT triangles and sub-domains of BT triangles that need to be classified so that the entire inside/outside classification can be completed. We observe that if one point is classified inside (outside), its owner triangle should also be classified inside (outside). Thus, previously obtained classification information of triangles in Ω can be reused to quickly identify its neighboring un-marked triangles' inside/outside classification. Based on this knowledge, the inside/outside classification can be expedited by taking advantage of the triangle connectivity information.

More specifically, in the left figure of Fig. 5, if the red triangle is classified as inside (outside), all the light blue triangles should also be classified as inside (outside), as all the light blue triangles share a common vertex with the red triangle. The classification of the sub-domain can also be determined with the same reasoning: in the right figure of Fig. 5, the hatched sub-domain containing the red point should have the same classification as the red triangle. Once one sub-domain is determined, all the other sub-domains can be determined using the alternation rule. Thus, through the vertex and edge propagation, the inside/outside attributes of all the left-over LT triangles and the sub-domains of the BT triangles can be effectively determined.

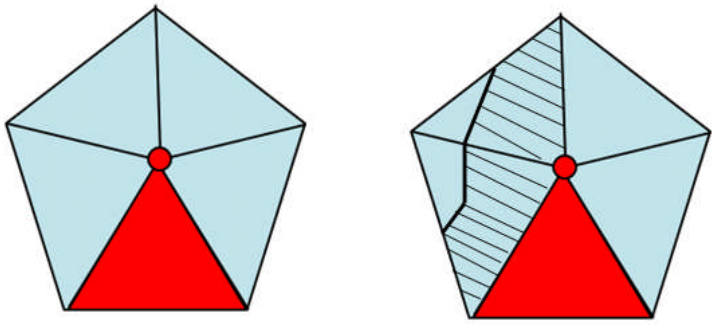

Fig. 5: Triangles sharing a common vertex: (Left) All the neighboring triangles will have the same classification as the red triangle; (Right) The two hatched sub-domains will have the same classification as the red triangle.

In practice, we find this speed-up strategy to be very useful. The choice of the seed (initial) triangle or sub-domain is critical. Conceivably, if the seed is selected arbitrarily and happens to belong to an already classified triangle, it may take quite a long time in propagation before an unclassified triangle/sub-domain is reached. Thus, naturally, we take the opposite direction: we select a BT or LT triangle one by one in Ω , search all its neighboring triangles sharing a common vertex with it. If any neighboring triangle (or sub domain) is classified, we are done, and meanwhile mark all these triangles (or sub domains) the same classification with this already tagged one; if neither neighboring triangles are tagged, we will search further based on the connecting information until we reach one tagged triangles or sub domain, and then all the previously visited triangles will be tagged with the same inside/outside classification and all these newly tagged triangles or sub-domains will be removed from Ω . This step will be repeated until the pool Ω is empty. As the classification is obtained by the connecting information rather than through direct computing, we are able to avoid notorious floating-errors.

Classification Algorithm

```
While (Ω is not empty)
Do
        Step 1: select one triangle T from Ω
        Step 2: searching a ring of neighboring triangles sharing a common vertex
                Step 2.1: If the classification is already marked for some neighboring triangle
                              Mark T or all the sub domains (if T is a BT)
                Step 2.2: If none of the neighboring triangles is marked
                              Search further until reaching an already marked triangle via the
                              connecting information, and mark T or all the sub domains (if T is a
                              BT) the same classification as the reached triangle.
                Step2.3:  Retrace all the previously visited triangles. All these previously visited
                          triangles in Step 2.2 including T or all sub domains (if T is a BT) are
                          tagged with the same classification and subsequently removed from Ω .
End
```

2.3 Triangulation

After the inside/outside classification of all the triangles (LT and OT) and all the sub domains of BTs are determined, we discard all the unnecessary triangles or sub domains based on their classifications and the specific Boolean operator type according to the rules in table 1. As the final result should be output as a triangle mesh model, we need to triangulate the kept sub domains of BTs. Referring to Fig. 6(a), a closed poly-line split a triangle into two parts of which the red hatched sub domain (Fig. 6(b)) is to be kept and thus needs to be triangulated (Fig. 6(c)). This is a typical computational geometry problem, i.e., simple-polygon triangulation [12-14], and can be solved theoretically in nearly linear time [14]. But in practice, we find that the computing time for this task can be greatly improved if some special treatments are carried out before hand. In practice, the case shown in Fig. 6(d) occurs most frequently, in which a poly-line traverses across a triangle, and only one portion in the triangle needs to be kept, i.e., either Fig. 6(e) or Fig. 6(f). In both situations of Fig. 6(e) or Fig. 6(f), the triangulation can be significantly simplified.

In the cases of both Fig. 6(e) and Fig. 6(f), the triangulation can be efficiently accomplished by tracing the connecting line segments between a vertex of the triangle and the edges of the poly-line. Specifically, for example in the case of Fig. 6(e), we first generate triangles that are made of vertex V_1 and those edges (some may be partial) that are “visible” to V_1. Since the region is simple, this visibility information can be quickly computed [13]. In most cases, these triangles are already a triangulation of the region and we are thus done. Occasionally there are maybe some pockets on the poly-line that are invisible to V_1. These pockets (though very few) then are triangulated with a general simple-polygon-triangulation algorithm. The special treatment for the case of Fig. 6(f) is similar and is omitted here.

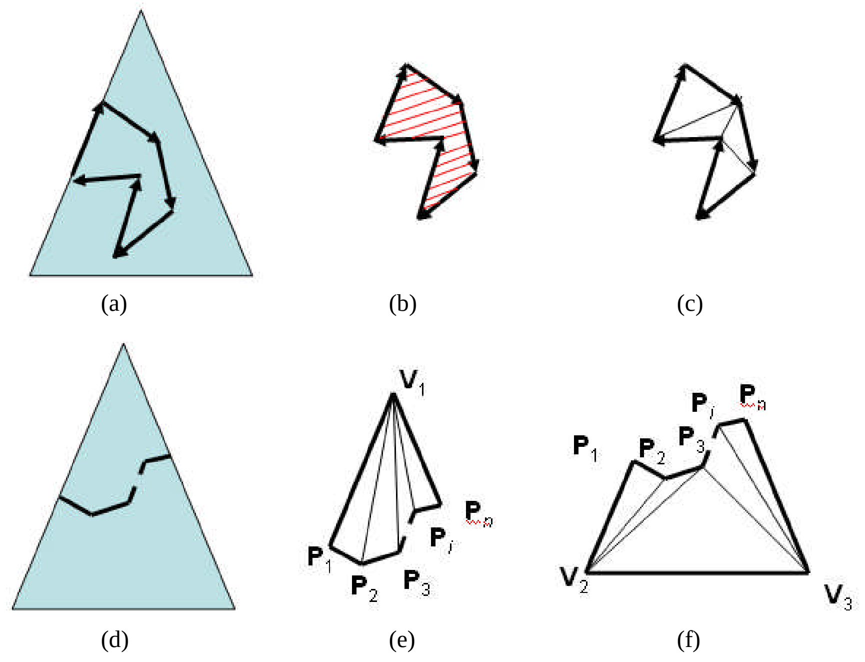

Fig. 6: Special treatment of triangulation: (a) A rare case in which poly-lines together with triangle edge(s) form a simple polygon, (b)&(c) The region to be triangulated and the triangulation, (d) The common case in which only one poly-line exists and it splits the triangle into two parts, (e) Special triangulation by connecting the vertex and the visible edges on the poly-line and (f) Another special triangulation by considering two vertices.

3 EXPERIMENTAL RESULTS

All of the experiments described in this section are performed on a PC with a 2.0GHz Intel Pentium IV, 1GB main memory and NVIDIA GeForce 4 Ti4600 graphics card with 128MB on board memory.

The first example is an application in toy design, where mesh models are usually used. Here ten letters are intended to be carved into the mesh surface model (see Fig. 7(a), (c) and (d)) which is a penguin with 6,834 vertices and 7, 790 triangle faces. The ten letters themselves are made up of 1, 682 triangle faces and 1, 520 vertices. The whole process of the subtract operation takes only 1.19s only, which is interaction rate. From the enlarged figure shown in Fig. 7(b), it can be seen that most newly generated triangles belong to the type of Fig. 6(e), i.e., connecting an apex vertex with a poly-line.

The second example is meant to test the validity and the performance of the algorithm. In this example, we intentionally make the penguin model much denser, with 78, 722 vertices and 79, 599 triangles. We then make two copies of this model, A and B, that are some distance apart but intersect each other. The result of $A \cap B$, A-B and B-A are shown in Fig. 8(c), Fig. 8(d) and Fig. 8(e), respectively. The triangulated eye portion is shown in Fig. 8(b), herein, the type of triangulation in Fig. 6(e) makes up a large percentage of all the triangles. Since, in practice, the computing cost of triangulation is high, this kind of special treatment saves tremendous amount of time: in this example, we find that when all the triangulations are carried out indiscriminatingly as a general simple polygon triangulation, the total triangulation time is 5.21s, but with our special treatment, the time is reduced to only 0.67s. The entire process of Boolean operation for Fig. 8(c), Fig. 8(d) and Fig. 8 (e) takes 10.12s, 11.02s and 11.33s, respectively.

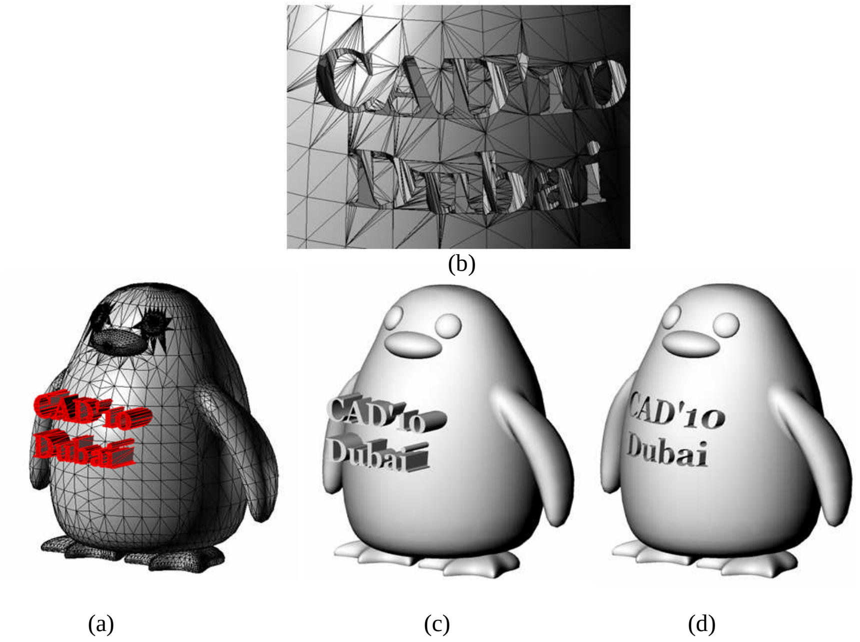

Fig. 7: carving a word on a penguin mesh model: (a) & (c) Two mesh models for the Boolean operation, i.e., one penguin and word, (b) The triangulations of BT sub domains and (d) The final rendered result of the carved penguin.

The third example is from a practical jewelry design project. A series of Boolean operations are performed on 16 solid bodies to build a ring model. The total vertex and facet numbers of all the involved bodies are around 80k, and the total spent time is 18.60s. The time cost is relatively much higher than that of example II although their mesh sizes are of the same order. We find that much extra computing time is spent in setting up the LDI models of two involved bodies for each Boolean operation, and each setup takes around 0.24s; thus, compared with example II, more than 15 LDI-setups are needed in example III. Actually, if all the Boolean operations and involved solid bodies are known in advance, all LDI models of involved bodies can be generated after one LDI step-up, which will greatly improve the algorithm efficiency.

4 CONCLUSION

We have presented a fast hardware-accelerated method for Boolean operations on triangle mesh manifolds. Unlike many existing Boolean operation algorithms that depend on exact computational or analytical algorithmic, our algorithm takes advantage of graphics hardware – which is originally mainly used for collision detection – and the idea of topological propagation that help tremendously expedite the inside/outside classification process. The specially designed triangulation algorithms for the most common situations of intersecting triangles also help greatly enhance the computation efficiency. In our preliminary experiments, our program can achieve an interaction rate for medium size mesh models, and for large mesh models, its speed is also acceptable. As a caveat, nevertheless, our algorithm strictly requires the mesh models to be water tight and topologically complete. A further study on how to loosen up this restriction is needed.

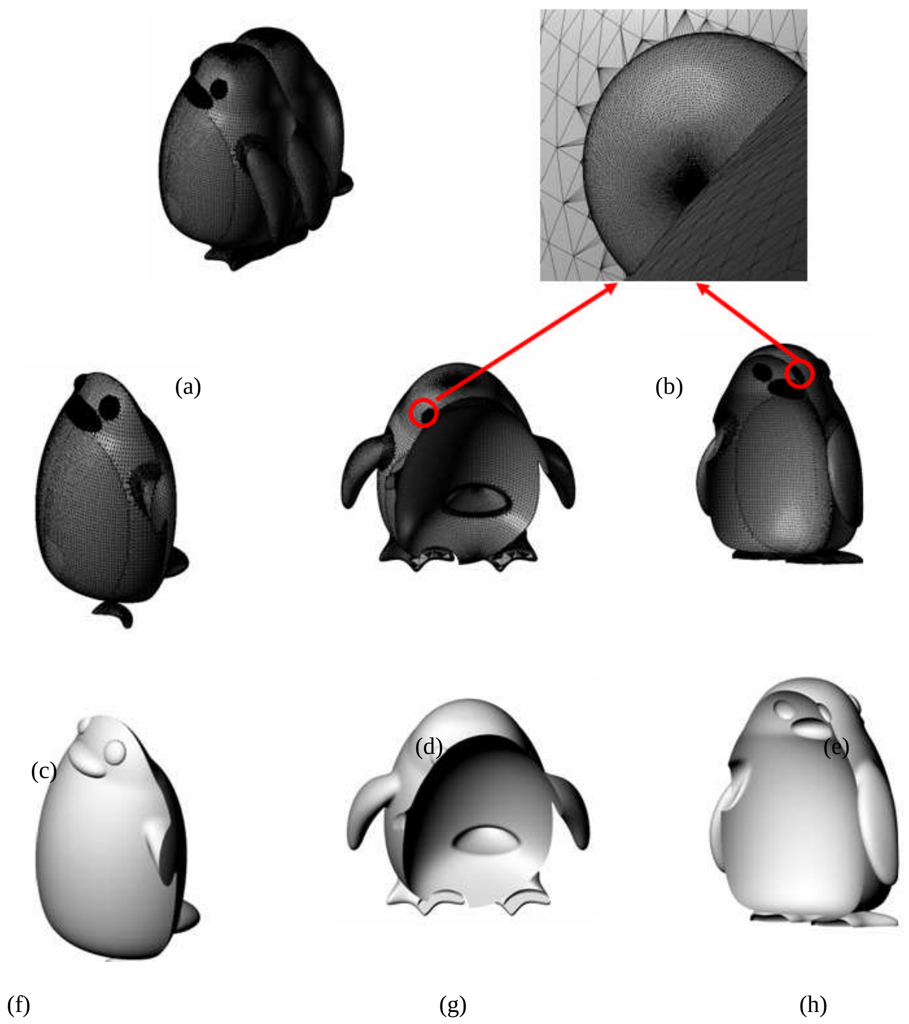

Fig. 8: Boolean operations on two dense mesh models. (a) The two copies of an identical penguin mesh model that are some distance apart, (b) The enlarged figure showing the triangulation of the BT sub domains, (c)&(f) A$\cap$B; (d)&(g) A-B, and (e)&(h) B-A.

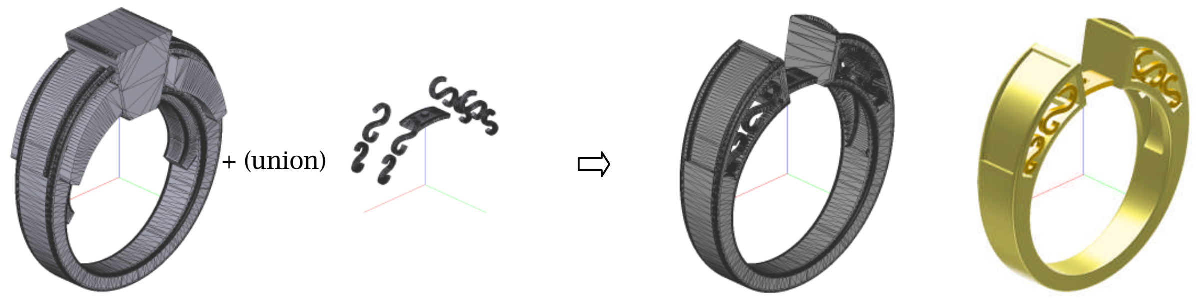

Fig. 9: A ring model generated by performing a series of Boolean operations on 16 solid bodies. For each Boolean operation, an extra LDI setup time is needed.

REFERENCES

[1] Hoffmann, C.: Geometric and solid modeling, Morgan Kaufmann, San Mateo, California, 1989.

[2] Pasko, A.; Adzhiev, V.; Sourin, A.; Savchenko, V.: Function representation in geometric modeling: concepts, implementation and applications, The Visual Computer, 11(8), 1995, 429-446.

[3] Pfister, H.; Zwicker, M.; Baar, J.V.; Gross, M.: Surfels: Surface elements as rendering primitives, ACM SIGGRAPH 2000, 2000, 335-342.

[4] Zwicker, M.; Pauly, M.; Knoll, O.; Gross, M.: Pointshop 3d: An interactive system for point-based surface editing, ACM SIGGRAPH, 2002, 322-329.

[5] Lorensen, W. E.; Cline, H. E.: Marching cubes: A high-resolution 3D surface construction algorithm. ACM SIGGRAPH, 1987, 163-169.

[6] Menon, J. P.; Voelcker H. B.: On the completeness and conversion of ray representations of arbitrary solids, Proceedings Of ACM Symposium on Solid Modeling and Applications, 1995, 175-286.

[7] Heidelberger, B.; Teschner, M.; Gross, M.: Detection of Collisions and Self-collisions Using Image-space Techniques, Journal of WSCG, 12(1-3), 2004, 145-152.

[8] Chen, Y.; Wang, C. C-L.: Layer depth-normal images for complex geometries - part one: accurate modeling and adaptive sampling, Proceedings of the ASME 2008 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, 2008.

[9] Muraki, S.: Multiscale 3D edge representation of volume data by a DOG wavelet, Symposium on Volume Visualization archive Proceedings of the 1994 symposium on Volume visualization, 1994, 35-42.

[10] Krishnan, S.; Manocha, D.: An efficient surface intersection algorithm based on lower dimensional formulation, ACM Transactions on Graphics, 16 (1), 1997, 74–106.

[11] Hoffmann, C.: Robustness in geometric computations, ASME Journal of computing and information science in engineering, 1, 2001, 143-156.

[12] Seidel, R.: A Simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons, Computational Geometry: Theory and Applications, 1, 1991, 51–64

[13] Berg, M. D.; Kreveld, M. V; Overmars, M.; Schwarzkopf, O.: Computational Geometry, Springer-Verlag, 2000.

[14] Amato, N.M.; Goodrich, M. T.; Ramos, E.A.: A Randomized Algorithm for Triangulating a Simple Polygon in Linear Time, Discrete & Computational Geometry, 26 (2), 2001, 245–265.

[15] Shade, J.; Gortler, S.; Wei, H. L.; Szeliski, R.: Layered depth images, Proceedings of the 25th annual conference on Computer graphics and interactive techniques, 1998, 231–242.

[16] Jones, M. W.; Baerentzen, J. A.; Sramek, M.: 3D distance fields: a survey of techniques and applications, IEEE Transactions on Visualization and Computer Graphics, 12(4), 2006, 581-599.