DOI: 10.3969/j. issn. 1673-5188. 2015. 03. 007
http://www.cnki.net/kcms/detail/34.1294.TN.20150909.1509.006.html, published online September 9, 2015

Fast, Exact and Robust Set Operations on Polyhedrons Using Localized Constructive Solid Geometry Trees

Ping Lu[1], Xudong Jiang[2], Wei Lu[2], Ran Wei[1], and Bin Sheng[3]
(1. ZTE Corporation, Nanjing 210012, China;
2. Autodesk China Research & Development Center, Shanghai 200061, China;
3. Shanghai Jiao Tong University, Shanghai 200240, China)

Abstract

Regularized Boolean operations have been widely used in 3D modeling systems. However, evaluating Boolean operations may be quite numerically unstable and time consuming, especially for iterated set operations. A novel and unified technique is proposed in this paper for computing single and iterated set operations efficiently, robustly and exactly. An adaptive octree is combined with a nested constructive solid geometry (CSG) tree by this technique. The intersection handling is restricted to the cells in the octree where intersection actually occurs. Within those cells, a CSG tree template is instanced by the surfaces and the tree is converted to plane-based binary space partitioning (BSP) for set evaluation; Moreover, the surface classification is restricted to the cells in the octree where the surfaces only come from a model and are within the bounding-boxes of other polyhedrons. These two ways bring about the efficiency and scalability of the operations, in terms of runtime and memory. As all surfaces in such a cell have the same classification relation, they are classified as a whole. Robustness and exactness are achieved by integrating plane-based geometry representation with adaptive geometry predicate technique in intersection handling, and by applying divide-and-conquer arithmetic on surface classification. Experimental results demonstrate that the proposed approach can guarantee the robustness of Boolean computations and runs faster than other existing approaches.

Keywords

Boolean operations; polyhedrons; constructive solid geometry; binary space partitioning tree

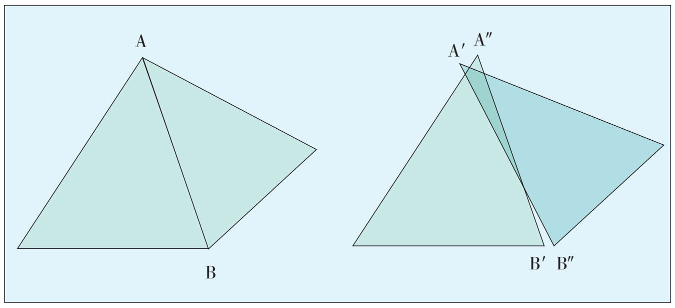
The work is supported by the Natural Science Foundation of China under Grant No. 61202154 and No. 61133009, the National Basic Research Project of China under Grant No. 2011CB302203, Shanghai Pujiang Program under Grant No.13PJ1404500, the Science and Technology Commission of Shanghai Municipality Program under Grant No. 13511505000, and the Open Project Program of the State Key Lab of CAD&CG of Zhejiang University under Grant No. A1401.

1 Introduction

Regularized Boolean operations [1] are defined as a closure of corresponding set-theoretic operations on the interior of two solids. The combination of set operations on polyhedrons, such as regularized union (U*), regularized intersection (∩*) and regularized difference (-*) can construct an arbitrary complex 3D model from simpler inputs. Thus, these operations have been widely used in interactive modeling systems, CAD/CAM applications, simulation systems and many other areas of computer graphics.

In order to evaluate the iterated set operations, an arbitrary set of polyhedrons is often organized into a constructive solid geometry (CSG) tree in which the leaf nodes are polyhedrons and the internal nodes are specific set operators. Then the evaluation of the CSG tree is generated by decomposing it into a combination of serial binary set operations with conventional methods.

The accumulation of numerical errors is introduced in the evaluation procedure of set operations. It often leads to system crashes or failure to generate correct result, such as holes and surface overlap (**Fig. 1**) when evaluating very complex models or when iterative set operations are required. Achieving geometric robustness by overcoming the problem is easy in the absence of efficiency. Thus, the major challenge of implementing such set operations is to take into account both robustness and efficiency for interactive applications. By carefully handling degeneracies and implementing set operations with arbitrary precision arithmetic, the algorithms [2]-[5] that can realize both goals have been proposed. However, they are too costly to be practical. In order to avoid these drawbacks, some methods [6]-[11] tried to implement set operations with voxels based on volumetric representations. However, all of these have to perform conversion between boundary representation (B-rep) and volumetric representation. Therefore, it is inevitable to lose geometric details and precision for input models in the conversion procedure. Another research stream for solving the robustness and exactness problem in set operations is to marry plane-based geometry representation and binary space partition (BSP) structure together with the adaptive geometry predicates technique [12], [13]. Nevertheless, the state-of-art BSP-based



▲**Figure 1. Topology inconsistency arising from accumulation of numerical error in Boolean operations.**

method [13] still suffers from robustness issues in some special cases because the method needs to split surfaces along with the boundary of critical cells, although the efficiency was improved by localized intersection handling. Moreover, in these methods, repetitive conversion between different representations and BSP merge operations may lead to robustness issues and poor performance when evaluating iterated set operations.

In contrast to B-rep-based methods [2], [3], [14], BSP-based methods have unique abilities for handling non-manifold surfaces and simplifying the procedure of processing all possible intersections and degeneracies among polyhedrons. Thus, the motivation of our study is to develop a fast and unified method for single and iterated set operations. This method can inherits the robustness and exactness of BSP-based approaches, and limits the cost of conversion between different representations.

In this paper, we exploit a new approach that uses localized CSG trees to efficiently compute Boolean operations on polyhedrons. Similar to the previous BSP-based work, our method implements robustness and exactness by integrating a plane-based geometry representation with adaptive geometry predicates technique. In order to significantly improve performance, we also apply an operation localization scheme to change the surface topology only at an intersection region. The surfaces at non-intersection region remain unchanged by employing a new adaptive octree construction algorithm which will be detailed discussed later. Unlike the method in [13], numeric errors are not introduced in the procedure of octree construction by avoiding split surfaces in ours. The main ingredient, which is the key to implementing the unified method for single or iterated set operations, is embedding a CSG tree into each octree cell where polyhedrons intersect with the help of an adaptive octree. Regularized set operations are evaluated on each CSG tree by converting it into a BSP tree and then extracting the boundary from the BSP tree.

A side effect caused by the operation localization scheme is that surface classification has to be performed for the polygon surface at non-intersecting regions. However, the procedure is quite time consuming when the number of those surfaces is very big. In order to improve classification efficiency, a partial surface classification scheme is employed based on the fact that classification of actual surfaces can be made only at the regions that are completely inside the bounding box of other polyhedrons, and the regions are covered by a set of octree cells. By classifying each cell as a whole, surface classification is further accelerated with the help of the octree. Through this divide-and-conquer strategy, the accuracy and robustness of classification is guaranteed because classification always performs between surfaces with original polyhedrons.

2 Related Work

Regularized Boolean operations have been investigated for many years. Corresponding methods can be classified into three categories: volumetric methods, approximate methods, and exact methods.

By converting B-rep into a volumetric representation, regularized set operations can be easily and robustly evaluated with voxels [6], [7]. However, the precision for models is inversely proportional to the size of voxel. Small voxels are quite expensive in terms of sampling time and memory consumption. Moreover, the sharp edges and corners of input models are lost in the converting process. In order to alleviate the problems, the methods in [8], [15] reconstruct the geometric details on surfaces of the resultant model by encoding the normal information of surfaces into the sampling process. Some [9]–[11], [16] also made efforts to preserve topology or manifold information in the results by applying dual-contour algorithms. Nevertheless, it is unavoidable for those methods to damage the geometric details and precision of the resulting models due to the conversion between different representations.

The performance penalty of exact Boolean operation is inevitable, so some researchers try to compute approximate Boolean operations instead. Biermann *et al*. [17] implemented approximated set operations on two free-form solids bounded by parametric surfaces based on a multi-resolution subdivision representation [18]. Robustness is achieved by applying the numeric perturbation [19] in intersection computation between the coarse meshes in the method. However, in the case when an intersection result is uncertain, the technology has to use new perturbation to compute the intersection again. Thus, the method is time-consuming for complex models. Smith and Dodgson [20] presented a topologically robust method for set operations on B-rep models. By carefully defining a series of interdependent operations, this approach can always guarantee the result with correct connectivity if the input models have valid connectivity. However, due to interdependent operations, this method cannot benefit from the power of parallel computation, which is common for modern processors,. Until recently, Wang [21] has proposed an approach to efficiently evaluate approximated set operations on two polygon meshes with the help of Layered Depth Images (LDI) [22]. A trimmed adaptive contouring algorithm is used to reconstruct the surfaces in the intersected region from the LDI/mesh hybrid, and then the surfaces are stitched together with the surfaces in non-intersected regions. In this way, the same robustness with that in the methods based on volumetric representation is obtained and geometric details are preserved at meaning time. Nevertheless, the trimmed adaptive contouring algorithm may damage the topological consistency of the resultant model.

The algorithms for exact Boolean operations have been accompanied with notorious robustness issues since they were introduced in 1980s [2]–[5]. Even though the arbitrary precision arithmetic and careful handling of degeneracies can be used in these algorithms to implement robustness, such implementations are too costly to be practical.

Naylor and Thibaut [23]–[25] have found that a BSP tree can facilitate Boolean operations of manifold or non-manifold solids. They proposed a much simpler alternative algorithm for B-rep ones by converting Boolean operation into BSP structure merging. BSP-based methods avoid handling all possible intersections and degeneracies. However, this approach is fragile due to error accumulation in the merging stage. Sugihara and Iri [26] introduced plane representation for polyhedrons for solving the robustness issue with geometry computation. By representing polygons with a supporting plane and a set of bounding planes, the rudimentary modeling operation can robustly performs. In 2009, Bernstein and Fusel [12] combined these two methods—plane based geometry representation and BSP structure—together with the adaptive geometry predicates technique [27], and proposed a robust BSP-based Boolean operation method in the true sense. However, this approach spends too much time on pre-computation. Both the time and space complexity of the merging algorithm is almost $O(n^2)$, making it impractical for large scale meshes. A year later, Camped and Kobbelt [13] improved this approach by introducing operations localization scheme. The improved algorithm subdivides input polyhedrons by an adaptive tree and marks the cells in which operant polyhedrons intersect as critical cells. BSP merge is then performed only in those critical cells. This optimization dramatically saves time and memory space for plane-based BSP Boolean operation while keeping robustness. However, the approach still suffers from the robustness issue in some special cases because it needs to split surfaces along with the boundary of critical cells. Moreover, in these methods, repetitive conversion between different representations and BSP merge operations may lead to robustness issues and poor performance when evaluating iterated set operations.

3 Overview

Given any two polyhedrons P_i and P_j, the evaluation of regularized Boolean operations between them implies the selection of boundary surfaces according to following equations [28].

$$P_i \cup^* P_j = \{F_i\ Out\ P_j\} \cup \{F_j\ Out\ P_i\} \cup \{F_i\ With\ P_j\} \quad (1)$$

$$P_i \cap^* P_j = \{F_i\ In\ P_j\} \cup \{F_j\ In\ P_i\} \cup \{F_i\ With\ P_j\} \quad (2)$$

$$P_i -^* P_j = \{F_i\ Out\ P_j\} \cup \{F_j\ In\ P_i\} \cup \{F_i\ Anti\ P_j\} \quad (3)$$

The surfaces set F_x is the boundary surfaces of P_x. The classification sets $F_x Out P_y, F_x In P_y, F_x\ With\ P_y, and\ F_x\ Anti P_y$ correspond to the subsets of boundary surfaces F_x that are respectively outside, inside, on the boundary with same orientation, and on the boundary with an orientation opposite to the polyhedron P_y. Moreover, the boundary surfaces of P_x can be classified into the union of three kinds of disjoint sets, i.e. $Px = \{S_{in}, S_{out}, S_{intersected}\}$, $S_{in} \cap S_{out} = S_{in} \cap S_{intersected} = S_{out} \cap S_{intersected} = \varnothing$, where S_{in}, S_{out}, and $S_{intersected}$ correspond to the set of surfaces that are completely inside, outside, and intersect with the minimum Axis-Aligned Bounding Box (AABB) of other models respectively. Then through the use of the classification, F_x can be generated from the classification set $S_{in}, S_{out}, S_{intersected}$ respectively.

$$F_x = F_{x1} \cup F_{x2} \cup F_{x3},\ F_{x1} \subset S_{in},\ F_{x2} \subset S_{out},\ F_{x3} \subset S_{intersected} \quad (4)$$

Thus, the evaluation of regularized Boolean operations between P_i and P_j can be discomposed to three procedures:

- to get F_{x2} from S_{out} of each polyhedron through expression simplification rules given in **Table 1** since all surfaces in S_{out} are outside the others.
- to collect F_{x1} from S_{in} of each polyhedron by classifying surfaces in the set since they are either completely outside or inside the other polyhedrons.
- to obtain F_{x3} form $S_{intersected}$ of each polyhedron by intersection handling.

The same strategies can be adopted to the evaluation of regularized set operations based on a CSG tree. Given a CSG tree T and a set of polyhedrons $P_{i=(1..n)}$ in its leaf nodes, T can also be decomposed into the union of three disjoint sub-trees, i.e. $T = T_{in} \cup T_{out} \cup T_{intersected}$, where T_{in}, T_{out}, $T_{intersected}$ are CSG trees composed of S_{in}, S_{out}, $S_{intersected}$ from all polyhedrons respectively. A leaf node of any sub-tree can contain an empty surface set in case of the corresponding S is $\varnothing$. The evaluation of T can be simplified to the results collection of all sub-tree's evaluation. **Fig. 2** shows an example of CSG decomposition.

By using an octree, T can efficiently be decomposed into a series of sub-trees. Thus, our algorithm can compute set operations on a set of polyhedrons in four steps.

1) CSG tree construction

The first step of our algorithm is to convert input Boolean expression into a CSG tree in which the leaf nodes are polyhedrons and the internal nodes are specific Boolean operators. For single set operations, the CSG tree is trivial, which only has two leaf nodes and one non-leaf node.

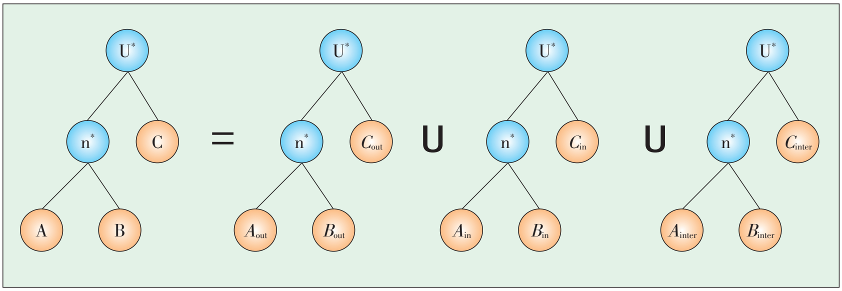
2) Localized intersection handling

The purpose of this step is to decompose the CSG tree into disjoint sub-trees, and restrict intersection handling in the trees which contain the surfaces in an intersection region with

▼**Table 1. Boolean operation simplification rules for the polygons outside other polyhedrons**

Operation	Left operand	Right operand	Result
∪*	a	b	a ∪ b
∩*	a	b	∅
−*	a	b	a

* a and b mean the polygon from left polyhedrons and that from right polyhedrons respectively.



▲Figure 2. An example of CSG tree decomposition.

the help of an adaptive octree. We first construct an octree to classify input polyhedrons into three kinds of nodes: internal nodes, external nodes, and intersected nodes (Section 4.1). Intersection computation is restricted to those intersected nodes where surfaces belong to at least two models. Before processing each intersected node, a CSG template is created based on the CSG tree constructed in the first step in order to improve performance (Section 4.2). Within each intersected node, all surfaces are converted to plane-based representation, then grouped according to the polyhedrons they belong to. If a surface spans at least two intersected nodes, the surface will be clipped along with six bounding planes of current node, and the remaining parts will associate to original surface. The CSG template is instanced with the grouped surfaces, then converted to BSP. The result of Boolean operations on the instanced CSG tree is obtained by extracting the boundary from the BSP.

3) Partial surfaces classification

In order to evaluate the CSG sub-tree where all surfaces are either entirely inside other models, or entirely outside other models, or on other models, we need to determine every surface relation with respect to other polyhedrons by a point-in-polyhedron test. Several strategies are employed in surface classification either to speed up the classification or to make the procedure robust. First, the classification is restricted to the internal nodes where surfaces only come from a model and are within the AABBs of other polyhedrons. All surfaces in each internal node are clustered as a whole for classification. The point-in-polyhedron test arithmetic based on spatial structure is then employed to further improve efficiency.

Distinct from conventional classification based on CSG, The divide-and-conquer method is used to achieve the robust and exact classification (Section 4.3).

4) Boolean operation result generation

The result of a Boolean operation is obtained by collecting the evaluation results from different CSG sub-trees. In this step, all surfaces in the intersected nodes and internal nodes are evaluated. Therefore, we only need to evaluate the set operations on the surfaces within external nodes where all surface only come from a model and are outside the AABBs of other polyhedrons. Knowing the relations of surfaces in each external node with respect to other polyhedrons, we can quickly evaluate those surfaces according to the simplification rules in

Table 1.

4 Localized Evaluation and Classification

4.1 Adaptive Octree Construction

In recent years, an adaptive octree has been employed to find intersected areas of input polyhedrons by several Boolean algorithms [13], [20], [27]. Different from those methods, the adaptive octree constructed in this paper allows a single surface to add into multiple nodes, which avoids the numerical error caused by clipping the surface during the tree construction. Meanwhile, the nodes of the octree are classified into three categories: external nodes, internal nodes, and intersected nodes, and different schemes for different nodes can be applied to accelerate the algorithm execution.

The adaptive octree algorithm takes all the surfaces of all polyhedrons as input, recursively subdivides the minimum AABB encompassing whole input polyhedrons along X axis, Y axis and Z axis of its Cartesian coordinates, and then classifies the surfaces according to the sub-bounding-boxes to generate the tree. Different from the conventional construction strategy, the procedure subdivides the current node if and only if the node contains surfaces from different models and the surfaces in the nodes exceed an adjustable parameter m. In this way, the octree can automatically adapt to the complexity of the model. During the generation period of the octree, each node is classified to one type of the following three categories:

- Intersected node: surfaces in the node belong to at least two models;
- Internal node: surfaces in the node only come from a model and are within the AABBs of other polyhedrons;
- External node: surfaces in the node only come from a model and are outside the AABBs of other polyhedrons.

All surfaces of the octree spread over the leaf nodes classified by spatial relation, and the surfaces in each leaf node are either entirely within the cell or intersected with it. Therefore, a surface can span multiple leaf nodes. To manage this case, our algorithm defines the priorities of different node types: the priority of an intersected node is higher than that of an internal node, while internal node is higher than external node. It means when a surface is shared by several nodes with different types, the surface is considered to belong to the node with highest priority and will be handled in this node. For example, if a surface is shared by an internal node and an external node, it will be handled in the internal node. To determine the ownership of each shared surface, those surfaces are set one of the following flags:

- Multi-intersected: the surface spans at least two intersected nodes.
- Single-Intersected: the surface only spans a intersected node;

- Internal: the surface can at least be shared by an internal node while cannot be shared by intersected nodes;
- External: the surface can only be shared by external nodes.

The four flags correspond to the node categories respectively. These flags define how to traverse the surfaces of the octree. When visiting external nodes, all the surfaces with single-intersected or internal flags are skipped because they are traversed by the corresponding intersected or internal nodes; the surfaces with multi-intersected flag are checked if and only if there are the remaining sub-surfaces after clipping. When visiting internal nodes, all single-intersected surfaces are skipped because they are traversed by the corresponding intersected nodes, and the surfaces with multi-intersected flag are processed as before. **Fig. 3** illustrates spatial subdivision on two 2D polygons by using adaptive octree.

The octree generated by this algorithm has three properties:

1) The internal and external nodes can only be leaf nodes, and only the intersected nodes may have children.

During the octree construction, the node is not be recursively subdivided unless it is an intersected node with the surfaces that exceed the threshold. This explains why the parent of an internal or external node can only be an intersected node, and the node cannot be subdivided any more.

2) All the surfaces within an internal node have the same relationship. They are either totally outside other models, or totally inside other models, or totally on the same surface with other models.

According to the definition and property 1, an internal node is a leaf node with all surfaces belonging to a single model. Assuming there exists a surface outside other models while the remaining surfaces are inside the model, those surfaces will cross the boundary of model, which means the surfaces intersecting with them are also in this node. This assumption does not meet the definition of internal node, so this propositions is true.

3) Only the surfaces of intersected nodes have the probability to intersect with other models.

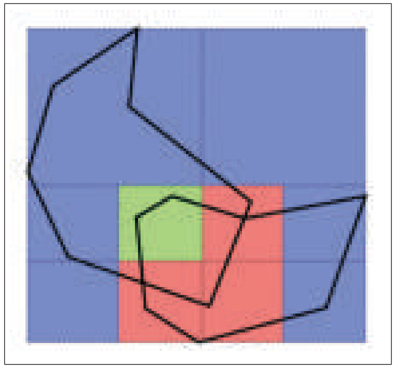
Assuming there is a surface in an internal or external node of the octree intersecting with models, the node contains surfaces from other models, or the surface is shared by a single/multi-intersected node. In the first situation, the node is an intersected node, unconformable to the assumption; in the second situation, the surface is handled by the intersected node, and this propositions is true.

The octree construction strategy implicitly divides the surfaces of each input model into three different regions: the intersected region, internal region, and external region. The regions consist of the surfaces from same model among all intersected nodes, all internal nodes and all external nodes respectively. **Fig. 4** shows an example of different regions on each polyhedron. Different strategies can be applied on the surfaces of different regions. In summation, the algorithm only needs to do the intersection handling on the surfaces of intersected regions, classifies the surfaces of internal regions, and directly evaluate the surfaces of external regions. The triangles from the exclusive group can be evaluated in early stage according to Table 1. Triangle-triangle intersection tests are limited to within each intersected group. A point-in-polyhedron test is performed for the triangulating results of every intersection test and performed only once for each inclusive group.

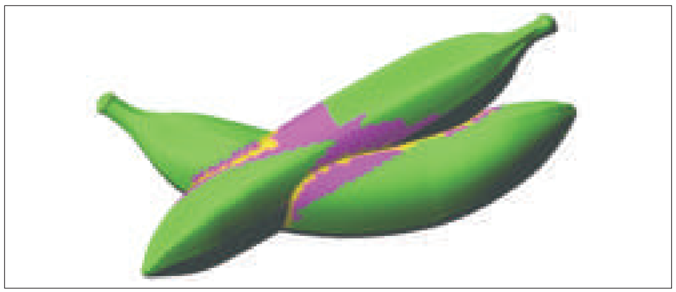
4.2 Localized Intersection Handling

In intersection handling, it is inevitable that operations change the topology of polyhedrons. By using predicates rather than constructions, it is easier to make the operations robust since no new geometric data are generated from existing geometric information [29]. Hence, vertex coordinates are not used in the operations such as intersection computation and clipping polygons, but the surfaces are converted to planed-based representation before further processing. The plane-based representation of a polyhedron consists of a support plane and a set of bounding planes. With the representation, a vertex of the polyhedron is defined implicitly by the intersections among the support plane and two bounding planes, while an edge is defined implicitly by the support plane and a bounding plane. Given a polygon $P=(V_1, V_2, \ldots V_n)$, where V_i is a vertex defined in counter clock wise order, we can get its plane-based representation $P=\{S, B_1, B_2, \ldots B_n\}$, where S is the support plane and B_i is a bounding plane by the plane equations in the forms:

$$f(S)=((V_3-V_2)\times(V_1-V_2))\cdot(p-V_1)=0 \qquad (5)$$



◀Figure 3.
2D illustration of subdividing two polygons by adaptive octree. Blue denotes exclusive cells; red denotes intersected cells; and green denotes inclusive cell.



▲ Figure 4. An example of different regions on each polyhedron constructed by an adaptive octree. Green, pink and yellow denote the external, internal, intersected region respectively.

$$f(B_i) = \left(\left(V_{i+1} - V_i \right) \times \left(\left(V_3 - V_2 \right) \times \left(V_1 - V_2 \right) \right) \right) \cdot \left(p - V_i \right) = 0 \quad (6)$$

where f is the implicit function of a plane and p is a point in the defined plane. Related geometric operations based on the representation were proposed in [12].

With the constructed octree, the intersection handling is restricted to the intersected nodes. Before processing intersected nodes, a CSG tree template is constructed in order to improve the efficiency in terms of runtime and memory. The CSG tree template is instanced by filling the polygons based on plane representation, and then converted to BSP for set operations evaluation in each intersected node. The evaluation results of are obtained by extracting the boundary from the BSP tree. The procedure is implemented in the three steps: constructing a CSG tree template, instancing the template in each intersected node, and converting CSG to BSP and extracting boundary.

4.2.1 Constructing CSG Tree Template

A general CSG template inherits the CSG tree constructed in the first step (Section 3). It is reused in the whole procedure of intersection handling, and instanced with the surfaces based on plane representation in each intersected node. In order to construct the template, we copy the original CSG tree, and replace the primitive in each leaf node with a pair of key-value, where the key is the identifier of the primitive while the value is a list of faces from each intersected node of the octree. **Fig. 5** shows a CSG tree template.

4.2.2 Instancing Template in Each Intersected Node

Within each intersected node, an empty group list is created corresponding to the set of input polyhedrons, and each group has a unique identifier for quickly finding the polyhedron where the surfaces in the group come from. A copy of each surface is converted into the plane-based representation, and then different processing strategies are taken according to surface flags. If a surface has a multi-intersected flag, the plane-based representation is clipped by six boundary planes of the cell by the clipping algorithm presented in [12], and the inner parts are added into the corresponding group by matching, while the outer parts are associated to the original surface, which will be clipped again when another relevant intersected cell is processed, otherwise, it is added into the corresponding group directly. After all surfaces are processed, the CSG template is instanced by filling the faces list in each leaf node with the surfaces from the group. The identifier of a group is used to quickly locate the leaf node by matching it. Many technologies can be used for this purpose such as a hash table.

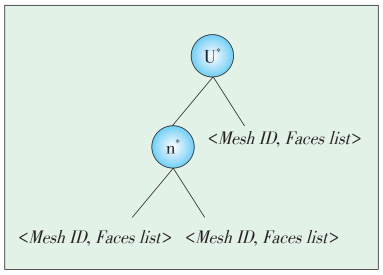


Figure 5.
In a CSG tree template, every primitive is replaced by a pair of key-value with mesh ID and face-list when instancing.

4.2.3 Converting CSG into BSP and Extract Boundary

There are two ways converting the instanced CSG into BSP. One is converting each primitive in leaf nodes into BSP, and then evaluating the tree down-to-up by performing BSP merging operation [24]. The other is converting the CSG tree into BSP tree directly [23]. In order to evaluate the CSG tree efficiently and avoid massive merging operations, we use the latter method. After the conversion, the results of set operations on the CSG tree are obtained by extracting the boundary from the BSP tree [23]. The results must be converted into B-rep and generate the final output by being combined with the evaluation results from other CSG sub-trees.

4.3 Partial Surfaces Classification

Such classification is used to determine the relationship of surfaces within an internal node of an octree with respect to other input models. This way helps determine whether the surfaces should be remained in the final results. The surfaces within an internal node are either entirely inside other models, or entirely outside other models, or on other models. Therefore, the relationship of a surface and a polygon is abstracted to specifying the relationship between the centroids of the polygon and the polyhedron. The classification based on CSG tree starts from bottom, and passes the classification results upwards to the parent node of the current node. The intermediate results are classified with respect to the models representing the brother nodes of the current node. This process ends at the root node and the final classification is achieved.

However, conventional approaches are confined to directly applying classification on the models that are represented by the two children of a CSG tree. When the two children are leaf nodes, the classification is executed between the models represented by them. If one of the two children is a non-leaf node, the intermediate result represented by the node always participates in the classification. This scheme makes the numerical errors propagated upwards, which may impair the topological consistency (such as holes or splits on the resultant polyhedron). In order to avoid such the issue, our algorithm uses a divide-and-conquer method to ensure the classification always happens between the surfaces and the original input model. We further optimize the performance of surface classification by using clustering strategy and octree.

Given a candidate surface p, ourmethod first decides whether the brother node n of the node containing p is a leaf node. If so, the algorithm of point-in-polyhedron is used to determine the relationship of the centroid of p with the input model corresponding to n . Otherwise, the relationship between t and the models represented by two children of n will be checked. The recursion is performed downwards to the leaf nodes to classify *t* with respect to the models corresponding to the two leaf nodes. After the classification, the results are propagated upwards again to the father node, and then the classification results are obtained from the predefined rule tables (**Tables 2–4**) based on required Boolean operations. The clasification results keep on propagating to the brother node of *n*. The concrete process is described in **Algorithm 1**, in which the function Combine makes use of the Boolean operations defined on *n* and the classification results of left and right sub-trees to get the classification results of the surface *p* by querying Tables 2 to 4.

▼**Table 2. Classification relation between model C and A ∪* B**

	CinB	CoutB	CwithB	CantiB
CinA	In	In	In	In
CoutA	In	Out	With	Anti
CwithA	In	With	With	In
CantiA	In	Anti	In	Out

"In" denotes a face is inside the model while "Out" has opposite meaning, "With" denotes a face is on the model with same normal, and "Anti" means a face is on the model with opposite normal.

▼**Table 3. Classification relation between model C and A ∩* B**

	CinB	CoutB	CwithB	CantiB
CinA	In	Out	With	Anti
CoutA	Out	Out	Out	Out
CwithA	With	Out	With	Out
CantiA	Anti	Out	Out	Anti

"In" denotes a face is inside the model while "Out" has opposite meaning, "With" denotes a face is on the model with same normal, and "Anti" means a face is on the model with opposite normal.

▼**Table 4. Classification relation between model C and A – * B**

	CinB	CoutB	CwithB	CantiB
CinA	Out	In	Anti	With
CoutA	Out	Out	Out	Out
CwithA	Out	With	Out	With
CantiA	Out	Anti	Anti	Out

"In" denotes a face is inside the model while "Out" has opposite meaning, "With" denotes a face is on the model with same normal, and "Anti" means a face is on the model with opposite normal.

Algorithm 1 ClassifyFacet (Polygon *p*, CSG-tree-node *n*)

```
1: if n is a leaf node then
2:     return Point-in-polyhedron (p.barycenter, n.mesh);
3: else
4:     return Combine (ClassifyFacet (p, n.left), ClassifyFacet
           (p, n.right), n.operator);
5: end if
```

Our algorithm uses the octree to complete classification in linear time by combining clustering strategy with the optimized point-in-polyhedron test based on spatial structure. First, the surfaces within each external node of an octree are known to be outside the bounding boxes of other models while the surfaces of each intersected node have been estimated through embedded in the CSG tree. Therefore, we only need to decide the surfaces within internal node with respect to other models. Regarding Property 2, all surfaces within the internal nodes of an octree have the same relationship. Therefore, we can cluster all surfaces as a whole in an internal node and take only one randomly for testing to decide the relationships of all surfaces of an internal node, which dramatically optimizes the performance of classification.

The point-in-polyhedron test is a basic geometric issue, and many researchers have proposed different methods for it [30]–[33]. Those approaches are divided into non-spatial structure methods and spatial structure methods, according to whether spatial structure for acceleration is used. Non-spatial structure methods test all surfaces of the model, while spatial structure methods only need to test a part of the surfaces from the model. Common spatial structure methods include octree, k-d tree, and BSP tree. Since the point-in-polyhedron test based on spatial structure only needs partial geometric information of the polyhedron, its performance is much better than a non-spatial structure method. However, it usually requires pre-process time to construct the spatial structure. The time complexity of classification between two models based on non-spatial structure method is $O(n*k)$, where n is the surface number of tested models and k is the number of tested surfaces. By adapting the ray-casting algorithm based on octree, the time complexity of classifying two models can be decreased to $O(n*\log n)$ yet plus the time of octree traversal. However, octree traversal can be accelerated by parameterization methods [34].

Algorithm 2 shows the pseudo-code of the classification procedure. This algorithm begins with the root node of the CSG

Algorithm 2 Evaluate (CSG-tree-node *n*)

```
1: if n.left is a leaf node then
2:     Store all facet groups in n.left into $G_l$;
3: else
4:     $G_l$ = Evaluate (n.left);
5: end if
6: if n.right is a leaf node then
7:     Store all facets groups in n.right into $G_r$;
8: else
9:     $G_r$ = Evaluate (n.right);
10: end if
11: for each facet group g in $G_l$ do
12:     store the first facet in g into f;
13:     if IsAcceptable(ClassifyFacet(f, n.right), n.operator)) then
14:         store g in node n;
```

```
15:     end if
16: end for
17: for each facet group g in $G_r$ do
18:     store the first facet in g into f;
19:     if IsAcceptable(ClassifyFacet(f, n.left), n.operator)) then
20:         store g in node n;
21:     end if
22: end for
23: return all facet groups in node n;
```

tree in which each leaf node has a list of facet groups. The lists are formed by grouping all surfaces from each internal node of the octree as a whole and then by storing the surfaces into the corresponding list. The recursive process collects the facet groups from child nodes at every stage. Since the surfaces in a group have the same membership, only the first facet in each group involves in the membership test with respect to the polyhedron representing by the brother child node implicitly. The test results, along with the specific selection rules (Section 3), determine whether the group that contains the tested facet is retained in the current node. This process ends at the root node and the final results are achieved in the node.

5 Experiments

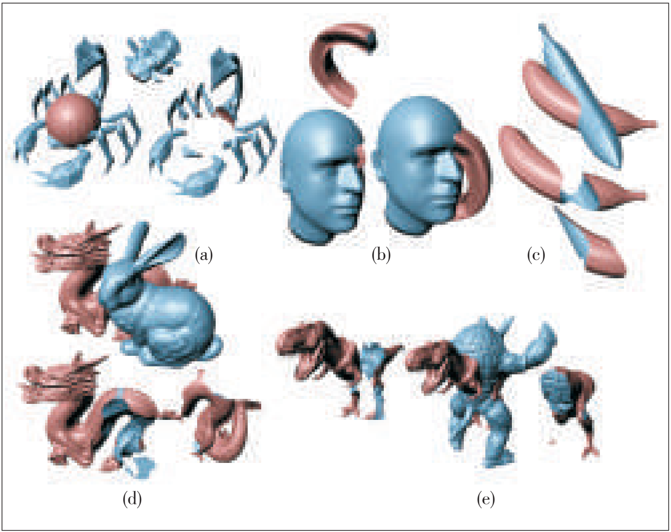
5.1 Setup

The implementation of our method is written in C++. Also, the Intel TBB multi-thread library is used to improve the performance. In order to reach a balance between the depth of octree and the number of polygon pairs for intersection test, the max polygon count in each octree leaf node is around 17.

We evaluated the performance of our method on a system with Intel i5-4200 1.53 GHz CPU and 16 GB RAM. In order to compare the quality of the results and the overall performance, we tested other systems including Maya Campen's method [13]. The input is closed triangle meshes with only vertices position and face indices information. The running time presented below includes pre-processing time.

5.2 Single Boolean Test

A single Boolean test is often used in interactive modeling systems which pursue robust and efficient solutions. We evaluate the robustness and speediness of our method from two aspects. First, we confirm the consistency of performance by performing regularized intersection, union and difference on a series of polyhedral pairs with increasing facet count (**Fig. 6**). The average execution time of the methods was then compared (**Table 5**). Second, we inspected the relationship between the execution time of our method and the number of the facets by constantly subdividing a polyhedron and by performing different set operations on it. The corresponding experimental results are shown in **Table 6**.



▲**Figure 6. Models and its Boolean results: (a) Scorpion; (b) Head; (c) Banana; (d) Bunny; and (e) Dino.**

▼**Table 5. Average time for Boolean operations (intersection, union, difference) of different polyhedrons in Fig. 6**

Models	Facet count	Maya2015 (ms)	Campen's (ms)	Our method (ms)
Scorpion	2400	62	113	31
	9600	219	272	79
	38,200	920	800	254
Head	10,000	281	349	97
	40,000	1217	1192	258
	160,000	5679	5481	934
Banana	48,000	983	1725	368
	192,000	4946	11,576	1303
	768,000	24,960	89,852	6757
Bunny	170,000	20,187	37,774	5370
	680,000	110,885	333,072	29,343
	4430,000	Fail	Out of memory	50,200
Dino	770,000	131,711	92,793	5,202

"Fail" means we get a wrong evaluation result from programs.
"Out of memory" means program crashed because system ran out of memory.

The results in Table 5 show that Maya2015 and Campen's approach went well for simple models. But their performance dropped significantly when models contain over about 200,000 facets. Moreover, the two methods also experienced robustness problems when models contain over 4000,000 facets.

On the contrary, our method can correctly evaluate all examples with high efficiency. Especially, our method is 5 times faster than Maya2015, and 10 times faster than Campen's approach for complex models (over 150,000 facets). This is due to the fact that our method only splits the facets that span over two intersected cells in intersection handling, and speeds up the classification by partial surface classification scheme.

▼Table 6. Average time for Boolean operations with increasing facet count

Facet count	Maya2015 (ms)	Campen's (ms)	Our method (ms)
8000	250	319	139
32,000	797	971	313
128,000	3406	4516	771
512,000	16,438	34,113	2554
2048,000	87,453	32,790	12,371

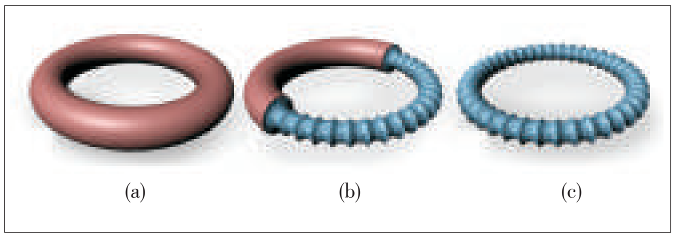
Moreover, our method spent much more processing time on Bunny - Dragon (680,000) than Dino - Monster (770,000) although they have similar facet count. The reason is that the computing time of our method is spent on both intersected handling and membership classification. Our method has to take more time to classify facets when there are a lot of internal cells in the constructed octree. Taking the model pair of Bunny -Dragon as an example, each model has many facets inside the other, which leads to many internal cells in the octree. In general, the system spends the least time on classification for two mutually orthogonal models.

To evaluate the complexity of the algorithm, we prepared a list of mesh pairs with increasing facet count by iteratively subdividing mesh pair Banana (Fig. 6c). Each subdivision increases the facet count by four times. Then we performed set operations on each mesh pair and recorded the average processing time. Table 6 shows that our algorithm is more efficient than the other two methods. This is because the octree constructed in our method only increases the number of intersected cells, while the number of internal and external cells stay fairly constant with the increase of facet count, when the position of two models remains unchanged. As a result, the efficiency of our method is determined by the processing time of intersection handling. Table 6 also shows that the complexity is approximately $O(n)$, where n is the polygon count of mesh, because our method can complete intersection handling in linear time.

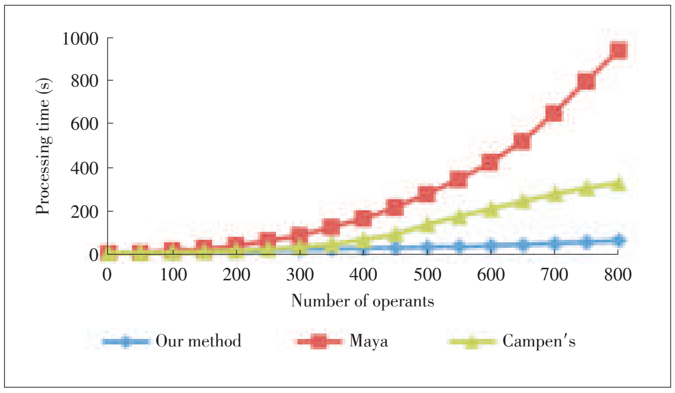
5.3 Iterative Boolean Test

In order to test the iterative Boolean operation performance and robustness of our method, we cut out a large polyhedral object, a ring, by performing iterative intersection operations among a series of small spheres and the ring. Meanwhile, we constantly increase the number of spheres so as to evaluate different methods. **Fig. 7** shows the ring and partial results of the intersection operation.

The experiment results (**Fig. 8**) show that Maya experiences an exponential growth in the process time with constantly increasing spheres, and performance of Campen's method falls dramatically after over 400 spheres. This is because the approaches decompose iterative set operations into a series of binary Boolean operation and then perform them one by one. However, each iteration needs to clip facets, and thereby generates new facets by intersection handling. The situation be-



▲Figure 7. Sculpting a ring: (a) original mesh (b) 400 spheres cut out (c) after 800 spheres cutting.



▲Figure 8. Iterative Boolean operation for different methods.

comes worse for Campen's method because the method requires extra effort on splitting surfaces along with the boundary of critical cells and performing conversion between different representations. Consequently, the time spent on iterative Boolean operations for the method is greater than the sum of time for all single operations, and the performance and robustness of the method get more and more serious with the growth in iterations. On the contrary, our approach can parse and execute the whole Boolean operation once for all by two major steps: constructing an adaptive octree and embedding a CSG tree into each intersected cell to evaluate all facets in the intersected region, and then using divide-and-conquer method and cluster classification strategy to classify all surfaces in the internal region with the help of the constructed octree. Thus, our method greatly increases the processing speed, and also implements robustness and exactness by integrating plane-based geometry representation with adaptive geometry predicates technique.

6 Conclusion

In this paper, we propose a unified method for efficiently computing single and iterated set operations on polyhedrons. By using localized CSG Trees evaluation strategy, the proposed method can evaluate very complex polyhedrons or massive iterated operations in a few seconds. This method uses the partial surface classification strategy to complete set membership classification in linear time with octree. Moreover, plane-based geometry computation is integrated in this method to make it robust. Experiments have verified that our system is very efficient for CSG trees with different sizes while keeping good quality and stability.

References

[1] T. Ertl, *Computer Graphics—Principles and Practice*. Berlin, Germany: Springer, 1996.
[2] A. A. Requicha and H. B. Voelcker, "Boolean operations in solid modeling: boundary evaluation and merging algorithms," *Proceedings of the IEEE*, vol. 73, no. 1, pp. 30–44, 1985. doi: 10.1109/PROC.1985.13108.
[3] D. H. Laidlaw, W. B. Trumbore, and J. F. Hughes, "Constructive solid geometry for polyhedral objects," *ACM SIGGRAPH Computer Graphics*, vol. 20, no. 4, pp. 161–170, 1986.
[4] F. Yamaguchi and T. Tokieda, "A unified algorithm for Boolean shape operations," *IEEE Computer Graphics and Applications*, vol. 4, no. 6, pp. 24–37, 1987. doi: 10.1109/MCG.1984.275959.
[5] P. Hachenberger and L. Kettner, "Boolean operations on 3D selective Nef complexes: optimized implementation and experiments," in *Proc. ACM Symposium on Solid and Physical Modeling*, Cambridge, USA, 2005, pp. 163–174.
[6] S. F. Frisken, R. N. Perry, A. P. Rockwood and T. R. Jones, "Adaptively sampled distance fields: a general representation of shape for computer graphics," in *Proc. 27th Annual Conference on Computer Graphics and Interactive Techniques*, New Orleans, USA, 2000.
[7] K. Museth, D. E. Breen, R. T. Whitaker, and A. H. Barr, "Level set surface editing operators," *ACM Transactions on Graphics*, vol. 21, no. 3, pp. 330–338, Jul. 2002.
[8] T. Ju, F. Losasso, S. Schaefer, and J. Warren, "Dual contouring of hermite data," *ACM Transactions on Graphics*, vol. 21, no. 3, pp. 339–346, Jul. 2002.
[9] G. Varadhan, S. Krishnan, Y. J. Kim, and D. Manocha, "Feature-sensitive subdivision and isosurface reconstruction," in *IEEE Visualization*, Seattle, USA, 2003, pp. 99–106. doi: 10.1109/VISUAL.2003.1250360.
[10] G. Varadhan, S. Krishnan, T. V. N. Sriram, and D. Manocha, "Topology preserving surface extraction using adaptive subdivision," in *Proc. Second Eurographics Symposium on Geometry processing*, Nice, France, 2004, pp. 235–244.
[11] N. Zhang, W. Hong, and A. Kaufman, "Dual contouring with topology-preserving simplification using enhanced cell representation," *IEEE Visualization*, pp. 505–512, Oct. 2004. doi: 10.1109/VISUAL.2004.27.
[12] G. Bernstein and D. Fussell, "Fast, exact, linear Booleans," *Computer Graphics Forum*, vol. 28, no. 5, pp. 1269–1278, Jul. 2009. doi: 10.1111/j.1467–8659.2009.01504.
[13] M. Campen and L. Kobbelt, "Exact and robust (self-) intersections for polygonal meshes," *Computer Graphics Forum*, vol. 29, no. 2, pp. 397–406, Jun. 2010. doi: 10.1111/j.1467–8659.2009.01609.
[14] F. R. Feito, C. J. Ogáyar, R. J. Segura, and M. Rivero, "Fast and accurate evaluation of regularized Boolean operations on triangulated solids," *Computer-Aided Design*, vol. 45, no. 3, pp. 705–716, Mar. 2013. doi: 10.1016/j.cad.2012.11.004.
[15] L. P. Kobbelt, M. Botsch, U. Schwanecke, and H. P. Seidel, "Feature sensitive surface extraction from volume data," in *Proc. 28th Annual Conference on Computer Graphics and Interactive Techniques*, New York, USA, 2001. doi: 10.1145/383259.383265.
[16] S. Schaefer, T. Ju, and J. Warren, "Manifold dual contouring," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 3, pp. 610–619, 2007. doi: 10.1109/TVCG.2007.1012.
[17] H. Biermann, D. Kristjansson, and D. Zorin, "Approximate boolean operations on free-form solid," in *28th Annual Conference on Computer Graphics and Interactive Techniques*, New York, USA, 2001, pp. 185–194. doi: 10.1145/383259.383280.
[18] M. Lounsbery, T. D. DeRose, and J. Warren, "Multiresolution analysis for surfaces of arbitrary topological type," *ACM Transactions on Graphics*, vol. 16, no. 1, pp. 34–73, 1997. doi: 10.1145/237748.237750.
[19] R. Seidel, "The nature and meaning of perturbations in geometric computing," *Discrete & Computational Geometry*, vol. 19, no. 1, pp. 1–17, Jan. 1998. doi: 10.1007/PL00009330.
[20] J. M. Smith and N. A. Dodgson, "A topologically robust algorithm for Boolean operations on polyhedral shapes using approximate arithmetic," *Computer-Aided Design*, vol. 39, no. 2, pp. 149–163, Feb. 2007. doi: 10.1016/j.cad.2006.11.003.
[21] C. C. Wang, "Approximate boolean operations on large polyhedral solids with partial mesh reconstruction," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 6, pp. 836–849, Jun. 2011. doi: 10.1109/TVCG.2010.106.
[22] J. Shade, S. Gortler, L. W. He, and R. Szeliski, "Layered depth images," in *Proc. 25th Annual Conference on Computer Graphics and Interactive Techniques*, Orlando, USA, 1998, pp. 231–242. doi: 10.1145/280814.280882.
[23] W. C. Thibault and B. F. Naylor, "Set operations on polyhedra using binary space partitioning trees," *ACM SIGGRAPH computer graphics*, vol. 21, no. 4, pp. 153–162, 1987. doi: 10.1145/37402.37421.
[24] B. Naylor, J. Amanatides, and W. Thibault, "Merging BSP trees yields polyhedral set operations," *ACM SIGGRAPH Computer Graphics*, vol. 24, no. 4, pp. 115–124, Aug. 1990. doi: 10.1145/97880.97892.
[25] W. C. Thibault, "Application of binary space partitioning trees to geometric modeling and ray-tracing," Ph.D. dissertation, Georgia Institute of Technology, Atlanta, Georgia, USA, 1987.
[26] K. Sugihara and M. Iri, "A solid modelling system free from topological inconsistency," *Journal of Information Processing*, vol. 12, no. 4, pp. 380–393, 1990.
[27] J. R. Shewchuk, "Adaptive precision floating-point arithmetic and fast robust geometric predicates," *Discrete & Computational Geometry*, vol. 18, no. 3, pp. 305–363, Oct. 1997. doi: 10.1007/PL00009321.
[28] K. Kuratowski and A. Mostowski, *Set Theory*. Waltham, USA: Elsevier, Academic Press, 1968.
[29] J. R. Shewchuk, "Lecture notes on geometric robustness," in *Eleventh International Meshing Roundtable*, 1999, pp. 115–126.
[30] F. R. Feito and J. C. Torres, "Inclusion test for general polyhedra," *Computers & Graphics*, vol. 21, no. 1, pp. 23–30, 1997. doi: 10.1016/S0097–8493(96)00067–2.
[31] J. Liu, Y. Q. Chen, J. M. Maisog, and G. Luta, "A new point containment test algorithm based on preprocessing and determining triangles," *Computer-Aided Design*, vol. 42, no. 12, pp. 1143–1150, 2010. doi: 10.1016/j.cad.2010.08.002.
[32] C. J. Ogayar, R. J. Segura and F. R. Feito, "Point in solid strategies," *Computers & Graphics*, vol. 29, no. 4, pp. 616–624, Aug. 2005. doi: 10.1016/j.cag.2005.05.012.
[33] W. Wang, J. Li, H. Sun, and E. Wu, "Layer-based representation of polyhedrons for point containment tests," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 1, pp. 73–83, 2008. doi: 10.1109/TVCG.2007.70407.
[34] J. Revelles, C. Urena, and M. Lastra, "An Efficient Parametric Algorithm for Octree Traversal," in *WSCG*, Plzen-Bory, Czech Republic, 2000, pp. 212–219.

Manuscript received: 2015-06-05

Biographies

Ping Lu (lu.ping@zte.com.cn) received his ME degree in automatic control theory and applications from South East University. He is the chief executive of the Cloud Computing and IT Institute of ZTE Corporation. His research interests include augmented reality and multimedia services technologies.

Xudong Jiang (denny.jiang@gmail.com) received his master's degree in computer science and technology from Shanghai Jiao Tong University. He is currently working at the Autodesk China Research & Development Center. His research interests include computer graphics and solid modeling.

Wei Lu (ddhansh@gmail.com) received her PhD degree in computer science and technology from Nanjing University. She is currently working at the Autodesk China Research & Development Center. Her research interests include computer graphics, mesh deformation, and virtual reality.

Ran Wei (wei.ran233@zte.com.cn) received his master's degree in communications and electronic information from Chongqing University of Posts and Telecommunications. He is currently a pre-research engineer of ZTE Corporation. His research interests include machine vision and graphics and image processing.

Bin Sheng (shengbin@cs.sjtu.edu.cn) received his MS degree in software engineering from University of Macau in 2007, and PhD degree in computer science from The Chinese University of Hong Kong in 2011. He is currently an associate professor at Department of Computer Science and Engineering, Shanghai Jiao Tong University. He also works with the Institute of Software, Chinese Academy of Sciences. His research interests include virtual reality, computer graphics, and image based techniques.