# Efficient Boundary Extraction of BSP Solids Based on Clipping Operations

Charlie C.L. Wang, *Member*, *IEEE*, and Dinesh Manocha, *Fellow*, *IEEE*

**Abstract**—We present an efficient algorithm to extract the manifold surface that approximates the boundary of a solid represented by a *Binary Space Partition* (BSP) tree. Our polygonization algorithm repeatedly performs clipping operations on volumetric cells that correspond to a spatial convex partition and computes the boundary by traversing the connected cells. We use point-based representations along with finite-precision arithmetic to improve the efficiency and generate the B-rep approximation of a BSP solid. The core of our polygonization method is a novel clipping algorithm that uses a set of logical operations to make it resistant to degeneracies resulting from limited precision of floating-point arithmetic. The overall BSP to B-rep conversion algorithm can accurately generate boundaries with sharp and small features, and is faster than prior methods. At the end of this paper, we use this algorithm for a few geometric processing applications including Boolean operations, model repair, and mesh reconstruction.

**Index Terms**—BSP to B-rep conversion, efficient, clipping, approximation, solid modeling

## 1 Introduction

Binary Space Partition (BSP) trees are regarded as an effective representation of polyhedra based on the use of spatial subdivisions. They were initially proposed for visibility computations [1], [2], and are used for Boolean operations [3], [4], [5], [6], [7], [8], mesh repair [9], [10], model simplification [11], collision detection, etc. Most current modeling and simulation systems (e.g., [12], [13], [14]) are based on boundary representation (B-rep) and use mesh-based B-reps. In applications like Boolean operations [3], [4], [5], [6], [7], [8] and mesh repair [9], [10], BSPs are used to perform geometric operations to compute a topologically valid mesh representation of the final solid. Such algorithms usually represent intermediate solids using BSP trees. Therefore, it is important to design efficient algorithms to compute the B-rep corresponding to the BSP tree (i.e., BSP => B-rep conversion or boundary extraction).

In this paper, we present a novel algorithm to perform clipping operations on convex polyhedra and use that formulation for efficient BSP => B-rep conversion. The basic computation involves clipping the boundary of convex polytopes with a plane. We extend the algorithm proposed by Bajaj and Pascucci [15] and use a set of logical operations to ensure we always output two objects with genus zero topology. Our clipping algorithm is resistant to degeneracies resulting from limited precision floating-point arithmetic. As compared to prior BSP => B-rep conversion algorithms, our approach offers the following benefits:

- By repeatedly clipping the volumetric cells that correspond to convex regions defined by nodes of a BSP tree, we are able to construct connected volumetric cells and two-manifold boundary surfaces are extracted from these connected cells as B-reps of BSP solids. A clipping algorithm is described in this paper to guarantee that the two resulting volumetric cells have genus zero, which results in a stable implementation of BSP => B-rep conversion. During the clipping operation, our algorithm combinatorially ensures that the intersection curve has a single component and is closed even when the object being clipped is not exactly convex due to the limited precision of floating-point arithmetic.
- All the operations are performed using finite-precision or IEEE double precision arithmetic. As a result, our approach is 5 to 8 times faster than an implementation of prior method [15] using exact arithmetic. For models of moderate size (e.g., less than 100k BSP nodes), we can perform the boundary extraction in a few seconds, thereby making it feasible to perform interactive editing (e.g., solid modeling, mesh repair, and surface reconstruction) on such models. Experimental results show that our BSP => B-rep algorithm can generate approximate B-rep models with very small geometric error (i.e., less than $10^{-5}$ on all tested examples). In terms of overall performance, our algorithm can compute the approximate B-rep from the BSP-tree (with 430k nodes) of a solid model (with 100k triangles) in 10 s and has a memory overhead smaller than 500 MB.

The rest of the paper is organized in the following manner. We survey related work in Section 2. Section 3 gives an overview of BSP => B-rep extraction issues. The clipping operation and the B-rep extraction algorithms are presented in Section 4. Experimental results and discussion are given in Section 5, and Section 6 highlights several applications of our algorithm.

- *C.C.L. Wang is with the Department of Mechanical and Automation Engineering, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong. E-mail: cwang@mae.cuhk.edu.hk.*
- *D. Manocha is with the Department of Computer Science, University of North Carolina, Chapel Hill, NC 27599-3175. E-mail: dm@cs.unc.edu.*

*Manuscript received 5 Nov. 2010; revised 15 Apr. 2011; accepted 29 Mar. 2012; published online 10 Apr. 2012.*
*Recommended for acceptance by K. Bala.*
*For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCG-2010-11-0264.*
*Digital Object Identifier no. 10.1109/TVCG.2012.104.*

## 2 Related Work

Most of the earlier work related to BSPs in geometric modeling dealt with the problem of computing a BSP tree from a polygonal representation [2], [8], [16], [17]. The algorithm proposed by Thibault and Naylor [8] is based on repeatedly selecting a planar polygon on the surface of a given model as the clipping plane (i.e., the nonleaf node of the BSP tree) to separate other polygons into left and right half-spaces, where the tree itself is not well balanced. Bajaj et al. present a progressive conversion from B-rep to BSP tree for streaming geometric applications [16]. They use the Eigen-decomposition of the Euler tensor, which represents the surface inertia, to select the clipping plane.

Compared with the problem of B-rep => BSP conversion, there is relatively little work on boundary extraction from BSP trees [8], [15], [17]. These methods are prone to robustness problems due to round-off errors. Recently, Bernstein and Fussell presented a plane-based approach [3] to perform robust Boolean operations on BSP solids by using finite-precision arithmetic. Their approach assumes that the boundary of the model has been explicitly stored in the nonleaf nodes of a BSP tree, and the final surface generated by their algorithm corresponds to a polygonal soup without the connectivity information. The reconstruction of connectivity would take extra time and effort. Campen and Kobbelt [9] describe an algorithm to construct BSP trees locally in a few regions of a spatial subdivision, which can reduce the memory overhead and the computation cost. More discussions about the geometric predicates and the point-based versus plane-based representation can be found in [18].

It is possible to convert a BSP tree into a CSG tree and use CSG => B-rep conversion algorithms [6], [19] to compute the final boundary. Banerjee and Rossignac [6] present a method to evaluate the B-rep of a CSG solid by first converting each primitive in the CSG tree into a subtree that defines primitives in terms of planar half-spaces and then computing the B-rep [19]. The robustness of their approach is guaranteed by performing Boolean operations of the "loose" primitives, whose exact dimensions, positions, and topology are not fixed. Based on these loose primitives, the robustness of an algorithm can be decoupled from numerical accuracy of the operations. We also use a similar notion of loose primitives in our approach. As opposed to CSG trees, BSP trees are used in many other geometry processing applications besides Boolean operations. Fig. 1 highlights a mesh repair example, where the repaired solid represented by a BSP tree is composed by 101k convex solids. One may wish to use the pipeline of "BSP => CSG => B-rep" as an alternative of direct BSP => B-rep conversion in order to use the existing techniques of CSG => B-rep conversion. However, as reported in the recent work of Bernstein and Fussell [3], performing around 9,000 Boolean operations on a model would take about 3 hours by their approach and almost 3 days by CGAL [20]. The efficient CSG => B-rep techniques in the literature (e.g., [21], [22], [23], [24]) mainly focus on rendering the final boundary, and may not be able to generate a valid mesh representation. As a result, using CSG tree as the intermediate representation for the BSP => B-rep conversion can be relatively slow.

There is a long history of research on robust polygon clipping in computational geometry and solid modeling

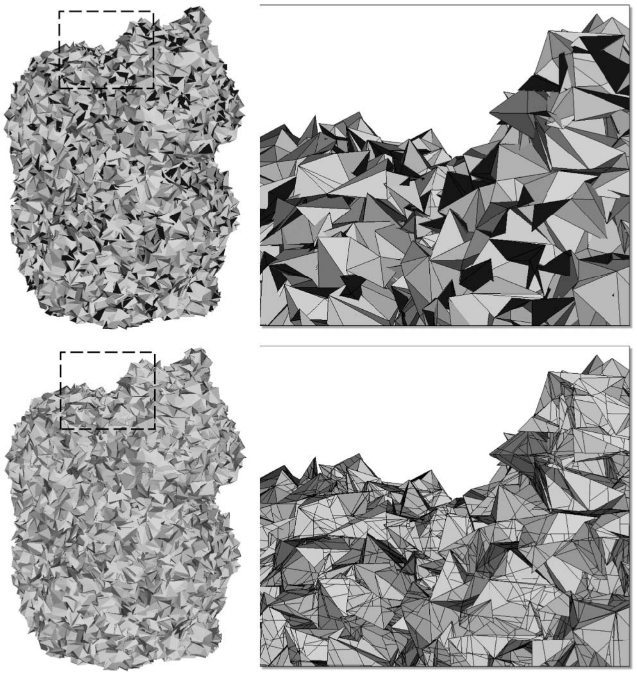


Fig. 1. A highly noisy squirrel model having 20k triangles (top) with many flipped triangles and self-intersections can be fixed by the solid repair algorithm [10] plus our approach in about 26 s, where our BSP => B-rep takes about 12 s. We use the model repair algorithm [10] to construct a BSP tree with 370k nodes (including about 101k solid leaf-nodes) from the squirrel model.

(e.g., [25], [26]). However, reliable polygon clipping algorithms are not sufficient for boundary extraction from BSP trees, as the numerical pruning here mainly arises from the inexact representation of a convex polyhedron generated after the clipping operations.

Some of the widely used spatial subdivision representations in geometric modeling and visualization are based on uniform grids or octrees. They are relatively simple to implement and there is extensive literature on computing a reliable polygonal approximation of the resulting surface [27], [28], [29], [30], [31]. In contrast, BSP trees are more flexible in terms of representing (and reconstructing) solids with fine features, which include thin-shell like shapes with one small dimension, or thin-rope like shape with two small dimensions. For example, in Fig. 3, the fine features are missed even when we use very high-resolution octrees. However, they are faithfully reconstructed in the B-rep models of BSP solids. Most recently, the problem of thin features in grid-based approaches was also addressed by using the hybrid representation of grids and mesh surfaces (see [32]).

## 3 Problem of B-Rep Extraction

In this section, we give a brief overview of BSP trees and discuss the issues related to geometric computation in BSP => B-rep conversion.

### 3.1 BSP Trees

A BSP solid is represented by a binary tree, where each nonleaf node corresponds to a plane that partitions the space into two half-spaces. Each BSP node represents a convex region that corresponds to the intersection of all half-spaces

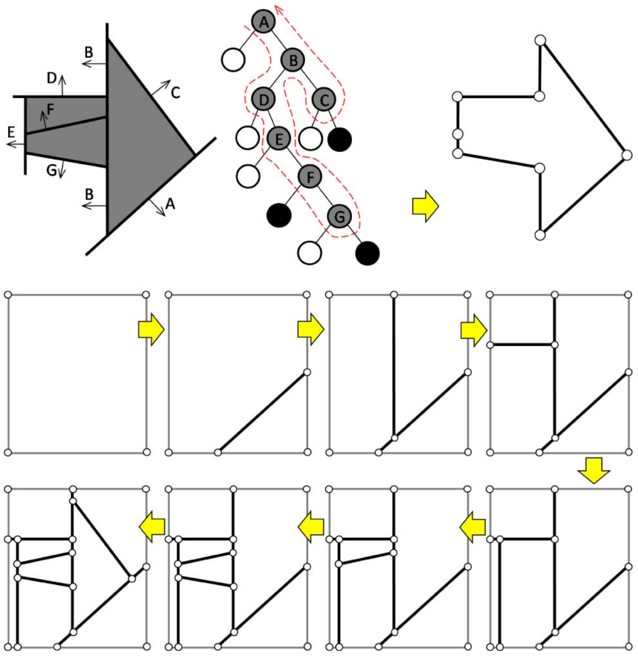
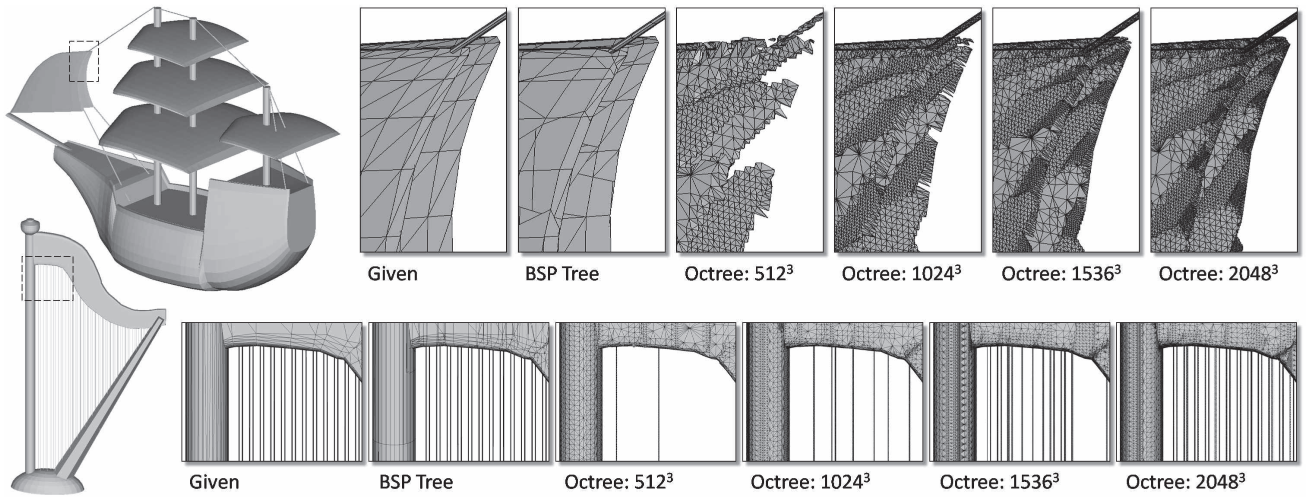


Fig. 2. A 2D illustration of the B-rep extraction from a BSP solid (top). Bottom rows show the steps for constructing the connected regions defined by the leaf-nodes on a BSP tree. The B-rep is extracted from the boundary of these regions. The construction procedure follows the order of nodes visited on the BSP tree (displayed in a red dashed curve)—has a linear complexity in terms of node number. Reconstruction of the connectivity between convex regions is guaranteed by our novel clipping algorithm (see Section 4).

on the path from the tree's root to that node. The leaf-nodes are labeled as either solid (in *black*) or empty (in *white*) according to whether their corresponding convex regions are part of the solid or not (see Fig. 2). The connectivity between these convex regions on a BSP tree is not stored explicitly in the data structure, which makes it difficult to robustly extract the connectivity of B-reps as reconstructing the connectivity requires making decisions-based on geometric tests (i.e., predicates). Specifically, for two convex regions corresponding to two leaf-nodes which have different parents, the detection of whether their boundary surfaces are connected is affected by round-off errors. A node on a BSP tree $\Gamma$ is defined as *valid* if its plane separates the corresponding convex region into two convex regions with nonzero volume. We assume that all the nodes on a given BSP tree are defined as *valid*.

## 3.2 Polygonization Algorithms

Thibault and Naylor [8], [17] present an approach to transmit the polygon of a splitting plane node in the BSP tree to the descendant nodes. The convex polygon corresponding to a node is clipped into smaller convex polygons (called fragments in [9]) by the half-spaces defined on the nodes of its subtree. If a fragment lies between the inside cell and the outside cell, it corresponds to a polygon on the boundary surface of the BSP solid. The connectivity of the resultant polygonal model is computed by matching the coordinates between vertices. However, this matching step can result in two problems [33], [34].



| | Dutch-ship | | | | Harp | | | |
|---|---|---|---|---|---|---|---|---|
| Example | Cell # | Time (Sec.) | Peak Memory | Face # | Cell # | Time (Sec.) | Memory | Face # |
| Given Model | - | - | - | 15,812 | - | - | - | 10,652 |
| BSP Tree* | 74,955 | 4.751 | 168 MB | 47,076 | 36,413 | 4.438 | 186 MB | 51,948 |
| Octree: $512^3$ | 1,184,192 | 2.237 | 265 MB | 147,532 | 1,238,464 | 1.359 | 142 MB | 81,520 |
| Octree: $1024^3$ | 2,721,728 | 6.225 | 579 MB | 305,218 | 4,416,128 | 5.071 | 441 MB | 278,618 |
| Octree: $1536^3$ | 4,662,016 | 12.351 | 1,024 MB | 469,824 | 8,750,976 | 11.158 | 1,026 MB | 586,634 |
| Octree: $2048^3$ | 7,144,064 | 21.079 | 1,543 MB | 621,434 | 12,810,368 | 19.783 | 1,572 MB | 1,033,754 |

Fig. 3. The mesh surfaces reconstructed from BSP tree by our method are compared with the results contoured from octrees on two benchmarks—the Dutch-ship and the harp. We use different resolutions for the octree and compare the performance with the BSP-tree. Our BSP => B-rep conversion algorithm can successfully reconstruct small features, which may be difficult even with a high resolution octree. Moreover, the use of BSP tree results in a B-rep with fewer triangles and has lower running time and memory overhead than octrees, though octree-based algorithms are easier to implement. In this case, the mesh surface reconstructed from octree is generated based on the Dual Contouring (DC) algorithm [30].

- First, repeated intersection computations result in accumulated round-off errors in the vertex coordinates. As different fragments that share a vertex $V$ are generated by performing intersection computations on a different sequence of planes, these computations can result in varying round-off errors corresponding to the vertex $V$.
- Second, epsilon error-based predicates are typically used to detect coincident points, i.e., two points are identified as coincident points if the euclidean distance between them is less than $\epsilon$, a small constant. However, the use of such predicates can 1) mismatch the different coordinates of $V$ generated from different fragments and this could result in cracks in the B-rep, or 2) classify topologically separated vertices as coincident due to inaccuracies—this can result in nonmanifold entities as with vertex clustering-based mesh simplification algorithms (e.g., [35]).

Therefore, we investigate a boundary surface extraction algorithm that generates global topology information as part of the polygonization.

#### 3.2.1 Polygonization with Topology Information

The boundary extraction algorithm presented by Comba and Naylor [36] first generate polygonal faces for all solid leaf-nodes of a BSP tree. After that, their algorithm relies on the nonrobust "glue" operator to compute a correct boundary surface from polygonal meshes with overlapped faces.

Bajaj et al. [15], [16] use a nonmanifold data structure to store the convex polytopes and their neighbor information, where each polytope is generated while constructing the BSP from a B-rep. Each polytope in the data structure corresponds to a leaf-node of the BSP tree, and is stored as a cell. The boundary surface can be computed by collecting the boundary faces between the solid and the empty polytopes. Nevertheless, robust computation of these cells with the neighboring information can be challenging due to finite-precision arithmetic.

#### 3.2.2 Problems of Polyhedron Clipping

According to the convex clipping algorithm used in [15] and [16], the polytopes and the adjacency information between them can be computed recursively by clipping the convex polyhedra. A consistent classification technique is introduced in [15] to deal with numerical inaccuracies; however, the robustness of this technique is based on the assumption that the input polyhedron to be clipped is exactly convex, which is not always true when the vertex coordinates are represented by finite-precision arithmetic.

Ideally, performing the clipping operation on a convex polyhedron $C$ by a plane $P$ should result in two convex polyhedra $C^+$ and $C^-$ (whenever $P$ intersects $C$). However, when clipping operations are performed using finite-precision arithmetic and point-based representations, the resultant polyhedra $C^+$ and $C^-$ may actually be nonconvex due to round-off errors in the coordinates of the new vertices. More specifically, the truncated coordinates of the vertices make them not lie "exactly" on the edges (and faces) of $C$.

When a convex polyhedron is separated by a plane into two polyhedra, the intersection curve between the convex polyhedron and the plane corresponds to a simple intersection curve that is homeomorphic to a circle. Nevertheless, when clipping a nearly convex cell, the intersection curve may have a more complicated topology, e.g., separate loops, loops joining at a single point, or nested loops. Such a clipping operation may generate more than two polyhedra, which is inconsistent with the operation of clipping a convex volume defined by a node of the BSP tree. Fig. 4 shows some examples of intersection curves with complex topologies. Although such cases may only occur in small or localized regions, inappropriate topology manipulation on these regions can easily make the program of a clipping algorithm crash when the clipping algorithm makes assumptions on the input model's topology (e.g., genus zero). Without using a carefully designed method (like our clipping algorithm) to clip degenerate convex cells which are in fact nonconvex, the clipping results, which are the input for further clipping, could result in 1) a two-manifold polygonal model (but may not be genus zero), 2) a model with several separated components, or 3) even a nonmanifold model. The consistent classification technique proposed by Bajaj and Pascussi [15] does not solve the stability problem of clipping degenerate convex cells. An approach to make their consistent classification technique work is to choose the strategy of plane-based representation plus exact predicates; however, this can be slower in practice.

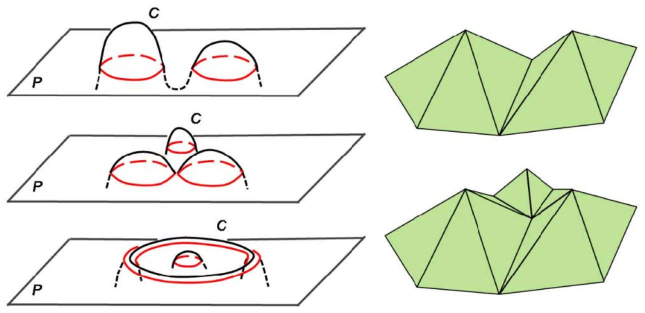


Fig. 4. (Left) Examples of some complex topologies that can be generated on the intersection curve between a plane $P$ and a nearly convex cell: (top) separate loops, (middle) loops joined at a single point and (bottom) nested loops. Since the vertices are generated by intersecting different planes, their coordinates may include varying round-off errors. This can result in a "terrain" like shape (right) in a localized region of the boundary. Using a plane to cut the "terrain" leads to the intersection curves having complex topology.

## 4 Boundary Extraction

This section presents the details of our boundary extraction algorithm (i.e., BSP => B-rep) and the clipping operation.

### 4.1 Overview

Our BSP => B-rep algorithm is built on the earlier work of Bajaj and Pascussi [15]. However, we develop a novel clipping algorithm to avoid the numerical accuracy problems of [15] when using finite-precision arithmetic. We start with a polyhedron $C$ (e.g., a rectangular box) that is larger than the bounding box of the solid $H$ represented by the BSP tree, $\Gamma$. Because of the round-off error, using the bounding box as the initial polyhedron $C$ may cut out some small volumes of $H$ near the boundary of $C$. The bounding box of $H$ can be given by the users as an input or computed from the vertices of convex-hulls according to the solid leaf-nodes of $\Gamma$, where the convex-hulls defined by a sequence of half-spaces can be determined by the Quick-Hull algorithm

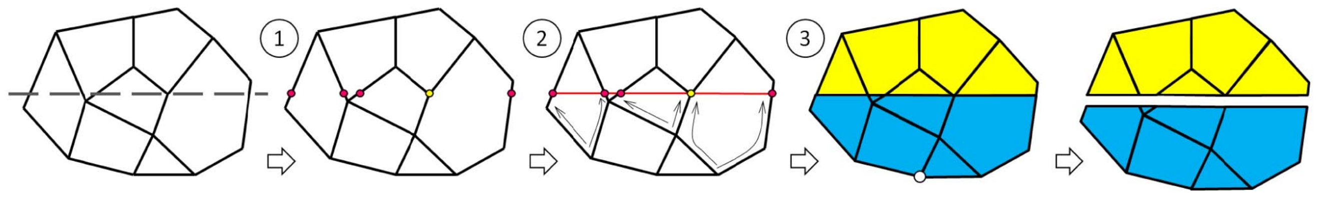


Fig. 5. Three steps of our clipping algorithm: 1) edge splitting, 2) face clipping and 3) cell separation. The vertices in red are new vertices introduced by edge splitting, and the yellow ones are existing vertices that are classified as vertices “on” the clipping plane. New algorithms are developed for steps 2 and 3 to guarantee the topology and the number of resulting cells, where a cell in yellow is above the clipping plane and a blue one is below the plane.

[37]. In our BSP => B-rep algorithm, we recursively traverse the BSP tree and clip $C$ into smaller cells based on the nodes of $\Gamma$. The resulting cells corresponding to the solid leaf-nodes are classified as *solid*, and the cells assigned to the empty leaf-nodes are classified as *empty*. The boundary surface of $H$ is then extracted from the faces between the solid and empty cells. Note that we use the concept of “loose” primitives [6] here. It is not necessary for all the vertices belonging to a face to be coplanar; in order to deal with round-off errors, the nonplanar faces are triangulated for intermediate geometric computation. Fig. 2 gives a 2D illustration of the algorithm. Pseudocode of the boundary extraction algorithm is given. Procedure 1 *BSPtoBrep* is based on the cell clipping procedure (see Section 4.2) and a nonmanifold data structure used to store the neighboring information between the cells (to be described in Section 4.4).

```
**Procedure 1.** BSPtoBRep (BSP tree $\Gamma$)
**Require:** the bounding box $B$ of $\Gamma$ is known
**Goal:** generate the B-rep of $\Gamma$ as a mesh surface $M_H$
 1: Compute a polyhedron $P$ that is slightly larger than $B$;
 2: Store $P$ as a cell $C$ in the complex-based non-manifold
    data structure;
 3: **Call** CellConstruction($C$, $\Gamma.rt$); {'$.rt$' is the root of $\Gamma$; this
    procedure constructs a complex of connected cells}
 4: **for all** faces $F$ in the complex of cells **do**
 5:     **if** the adjacent cells above and below $F$ are with
        different status **then**
 6:         $F$ is defined as a boundary face;
            {a polygonal face between *solid* and *empty* cells}
 7:     **end if**
 8: **end for**
 9: Generate a mesh $M_H$ consisting of all the boundary
    faces and the adjacent vertices;
10: **return** $M_H$;
```

```
**Procedure 2.** CellConstruction (a cell $C$, a BSP node $\gamma$)
**Require:** $C$ is stored in the data structure that can fetch its
    neighbors in constant time
**Goal:** generate a set of cells corresponding to the leaf-nodes
    of a given BSP tree node $\gamma$, along with connectivity
    information
 1: Assign the status of $C$ based on the status of $\gamma$; {*empty*,
    *solid* or *ambiguous*}
 2: **if** $\gamma$ is *ambiguous* **then**
 3:     Clip $C$ by $\gamma.pl$ into two cells $C^+$ and $C^-$; {"$.pl$" is the
        plane defined on a node of BSP tree}
 4:     **Call** CellConstruction($C^+$, $\gamma.leftChild$);
 5:     **Call** CellConstruction($C^-$, $\gamma.rightChild$);
 6: **end if**
 7: **return**;
```

## 4.2 Clipping Algorithm

Given a cell $C$ with the vertex coordinates stored by finite-precision representation, we present a novel clipping algorithm using finite-precision arithmetic. The clipping operation guarantees that the separating curve has a simple topology and ensures the generation of two genus zero cells, $C^+$ and $C^-$. $C^+$ denotes the cell above the clipping plane $P$ and $C^-$ is below. Our clipping algorithm is based on performing logical operations; therefore, it is resistant to degeneracies caused by evaluating numerical predicates with limited precision. The algorithm consists of three steps: edge splitting, face clipping, and cell separation (see Fig. 5).

### *4.2.1 Edge Splitting*

For every vertex $V$ on a given cell $C$, we check whether it is above (assigned as “$+$”), below (set as “$-$”) or on (denoted by the flag “0”) the plane $P$. We compute the signed distance $d_V$ from $V$ to the plane $P$. The vertex $V$ is classified as

1. *above* $P$ if $d_V > \epsilon$,
2. *below* $P$ when $d_V < -\epsilon$, and
3. *on* the plane $P$ otherwise.

The value of $\epsilon$ is chosen by the user. For example, $\epsilon = 5 \times 10^{-10}$ is selected when the computations are performed using IEEE double precision arithmetic. It should be noted that using a small $\epsilon$ splits a higher number of edges into very short segments, whereas a large value of $\epsilon$ will result in a high number of existing vertices being classified as “on” the plane $P$ (e.g., the yellow point in Fig. 5). Note that different $\epsilon$ will lead to different shape approximation errors in the models resulting from our approach. Thus, we tend to select a value that balances between the overall running time and accuracy (e.g., $\epsilon = 5 \times 10^{-10}$).

For every edge $E$ on $C$, if there is one endpoint above and another below the clipping plane based on our classification, a new vertex $V_i$ is inserted at the position of its intersecting point at $P$ to split $E$ into two edges (e.g., the red points shown in Fig. 5). $V_i$ is assigned the label “0.” When a new vertex is introduced to split an edge $E$, the topology of all faces linked to $E$ is updated accordingly. After this step, the cell $C$ still has genus zero.

### *4.2.2 Face Clipping*

The round-off errors corresponding to the positions of vertices could lead to the following problems on $C$: 1) the vertices of $C$ do not lie on their convex hull, 2) the vertices of a face $F$ are not coplanar, or 3) the vertices of $F$ do not lie on their convex hull. As a result, it is possible that more than two vertices on a face $F \in C$ are flagged as "0" after the edge splitting step is applied to $C$. To address this, we need a scheme to split the face $F$ simply into $F^+$ and $F^-$ (i.e., "above" and "below" the clipping plane $P$), and build a boundary edge between $F^+$ and $F^-$. For the degenerate cases having more than two "0" vertices, a face $F^-$ with no vertex having "+" flag will be constructed to prevent the flooding algorithm (in Section 4.2.3) from propagating the region "below" the clipping plane into $F^+$.

First, we find two vertices $V_p^+$ and $V_p^-$ on $F$ that are on different sides of the clipping plane and farthest away from $P$. If $V_p^+$ is not assigned with a "+" flag or if $V_p^-$ is not identified as "$-$" by the edge splitting step, we ignore the clipping of $F$ as its vertices are not classified as being on different sides of the clipping plane $P$.

Starting from $V_p^-$, we search the vertices on the boundary of $F$ in a clockwise manner until we find a vertex $V_e$ with flag "0." This vertex is considered as the vertex *entering* the region below the clipping plane $P$. Similarly, walking counter-clockwise starting from $V_p^-$, the first vertex $V_l$ with flag "0" is labeled as the vertex *leaving* the region below the clipping plane $P$. The search orders are illustrated in the middle of Fig. 5 by the arrows. Note that the $F^-$ face constructed in this way will only contain "$-$" and two consecutive "0" vertices.

Finally, an edge linking $V_e$ and $V_l$ is constructed to split $F$ into two new faces $F^+$ and $F^-$. Note that none of the operations in this computation involve the evaluation of numerical predicate. Therefore, the robustness of face clipping is not susceptible to numerical inaccuracies. Again, the newly constructed faces are loose primitives as described in [6], and the new cell $C'$ has genus zero. The proof is straightforward. According to the formula of Euler characteristic, the above face clipping operations add a new edge and split one face into two—both the edge number $n.E$ and the face number $n.F$ of the cell increase one, which will not change the genus: $G = 1 - (n.V - n.E + n.F)/2$.

The above algorithm works well when the vertices of $F$ are separated by "0" vertices into one simply connected "+" group and one simply connected "$-$" group (as shown in Fig. 6a). However, in some extreme case, more than one groups of "+" (or "$-$") vertices are generated due to numerical error. For such cases, we introduce a new "0" vertex in the center of $F$ and create more than one edges to separate these "+" and "$-$" vertices (see the illustration in Fig. 6b).

### 4.2.3 Cell Separation

The last step of our clipping algorithm generates a separating curve $\varsigma$ that is homeomorphic to a circle, which subdivides the given cell $C'$ into two open surfaces $M^+$ ("above" the clipping plane $P$) and $M^-$ ("below" $P$) with disk-like topology, where $C' = M^+ \cup M^-$. The holes on $M^+$ and $M^-$ are then filled by a face to create two cells. The new face and cells are all loose primitives [6].

We use a region flooding algorithm to generate the separating curve $\varsigma$. First, we find a seed vertex $V_{seed}$ with $V_{seed} = \arg\min_{V \in C'} d_V$. Then, we randomly select a face incident to $V_{seed}$ to serve as the seed face $F_{seed}$. The algorithm starts from $F_{seed}$, floods across the edges between faces, and stops at the edges not "below" $P$ (i.e., with no endpoint flagged as "$-$"). All the faces traversed by the flooding algorithm are assigned the "$-$" flag. Pseudocode of the flooding algorithm is presented in **Procedure** *RegionFlooding*.

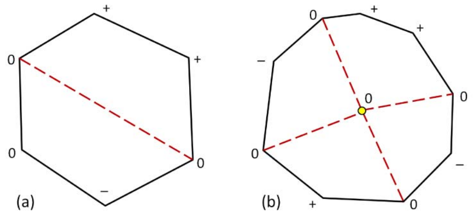


Fig. 6. The illustration of face clipping: (a) the "0" vertices classifies "+" and "$-$" vertices into two simple groups, and (b) a complex case with more than one groups of "+" and "$-$" vertices that can be "clipped" by introducing a new vertex (the yellow dot) at the center of $F$ and creating more than one edges (the red dash lines).

**Procedure 3.** RegionFlooding (a cell $C'$)
**Require:** the vertices of $C'$ has been classified as "+," "$-$," or "0"
**Goal:** compute the faces on $C'$ "below" the clipping plane

```
1:  Find a seed vertex based on $V_{seed} = \arg\min_{V \in C'} d_V$;
2:  Randomly select a face incident to $V_{seed}$ as $F_{seed}$, and
    assign the "$-$" flag to $F_{seed}$;
3:  Initialize a set of faces, $S = \{F_{seed}\}$;
4:  repeat
5:      for all faces $F \in S$ do
6:          Initialize an empty set of faces, $S^r$;
7:          for all faces $F_e$ incident to $F$ by an edge $E$ do
8:              if any endpoints of $E$ is with "$-$" flag then
9:                  if $F_e$ is NOT with "$-$" flag then
10:                     Insert $F_e$ into $S^r$;
11:                 end if
12:                 $F_e$ is assigned with "$-$" flag;
13:             end if
14:         end for
15:     end for
16:     Replace $S$ by $S^r$;
17: until $S$ is empty
18: return
```

The above flooding algorithm makes it possible to classify the surface of $C'$ into more than two disjoint regions. We further classify the regions into two separated surface areas with a simple boundary by the following *region processing* steps.

1. Identify the isolated regions with faces not assigned as "$-$" (e.g., two regions shown in Fig. 7)—each isolated region is circled by a boundary with one simple loop (i.e., no self-intersection).
2. Select the region $R_{\max}$ with the maximal area[1] and assign all the faces in this region with the flag "$-$."
3. Assign "$-$" to all of the remaining faces.

Based on this method, the curve $\varsigma$ that separates the "+" and the "$-$" regions is homeomorphic to a circle—the property is held even if there are round-off errors. The curve $\varsigma$ is in fact the boundary of the largest region not traversed by the flooding algorithm. In the second step of

1. To compute the area, a "loose" facet is temporarily triangulated into a set of triangles by introducing a new vertex at the average position of its existing vertices and constructing triangles connecting the existing boundary edges with the newly added vertex.

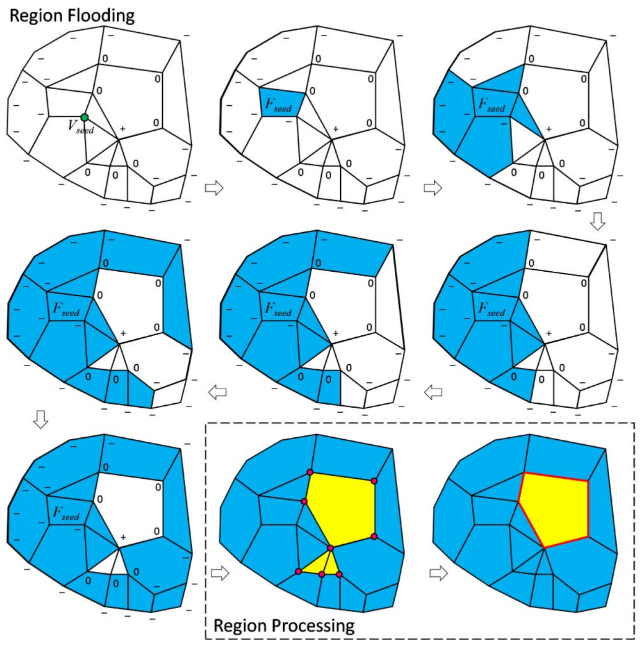


Fig. 7. Region flooding and region processing are performed to ensure that the intersection curve homeomorphic to a circle, which guarantees that the resultant cells generated by our clipping algorithm have genus zero topology.

region processing, selecting any other separated region will also give a result with two nearly convex cells. We select the region with maximal area according to the heuristic such that the "above" status of such a region is less likely to be affected by numerical errors.

In some degenerate cases caused by round-off error, there may be no region found by the first step highlighted above. However, as described in Definition 2, every clipping plane generated by a *valid* node of the BSP tree must separate the region into two parts, the clipping on the cell $C'$ should be enforced as follows: Among all the faces, the face with a maximal signed-distance from its center to $P$ is enforced to be "above" the clipping plane by assigning the "+" flag.

The surface of $C'$ is separated by the curve $\varsigma$ into two open mesh surfaces $M^+$ (composed of all "+" faces) and $M^-$ (composed of all "$-$" faces). As the topology of $\varsigma$ is simple, a loose face $F_\varsigma$ using $\varsigma$ as the boundary is constructed to separate $C$ into $C^+$ and $C^-$. Note that we only construct one face for the two convex polyhedra, $C^+ = M^+ \cup F_\varsigma$ and $C^- = M^- \cup F_\varsigma$, which can be stored in the nonmanifold data structure presented in Section 4.4.

## 4.3 Analysis

The properties of our algorithm are analyzed as follows:

**Remark.** The clipping algorithm for a genus zero cell $C$ with a plane $P$ only generates cells with genus zero topology.

**Proof.** The topology of the resultant cells is guaranteed by the following factors:

- The input cell $C$ is two-manifold and has genus zero. Thus, the cell $C'$ computed after edge splitting and face clipping is still a two-manifold and has genus zero.
- The boundary of each processed region has a simple topology homeomorphic to a circle—this property is satisfied by our method of cell separation. Therefore, the separating curve $\varsigma$ also has a simple topology. That is to say, the common boundary between $M^+$ and $M^-$ is a simple curve.

Without loss of generality, let us assume that $M^+$ has $n.V^+$ vertices, $n.E^+$ edges and $n.F^+$ faces, while $M^-$ has $n.V^-$, $n.E^-$ and $n.F^-$ vertices, edges and faces, respectively. The formula of Euler characteristic gives

$$(n.V^+ + n.V^- - n.V^s) - (n.E^+ + n.E^- - n.E^s) \\ + (n.F^+ + n.F^-) - 2 = -2G,$$

where $G$ is the genus of $C'$ composed by $M^+$ and $M^-$, and $n.V^s$ and $n.E^s$ correspond to the number of shared vertices and edges between $M^+$ and $M^-$. Again, based on the Euler characteristic, the genus $G^+$ of $C^+$ is $G^+ = 1 - (n.V^+ - n.E^+ + (n.F^+ + 1))/2$, and similarly the genus $G^-$ of $C^-$ is

$$G^- = 1 - (n.V^- - n.E^- + (n.F^- + 1))/2,$$

where the additional face comes from the filled face $F_\varsigma$. Adding $G^+$ and $G^-$ together gives $G = G^+ + G^- - (n.E^s - n.V^s)/2$. The topology of the intersecting curve $\varsigma$ is simple, and $n.E^s = n.V^s$. This implies that

$$G = G^+ + G^- \equiv 0.$$

As the genus cannot be negative for a two-manifold polyhedron, we get $G^+ = G^- = 0$. □

As long as the neighboring information between the nearly convex cells is stored in a nonmanifold data structure (which will be detailed in Section 4.4), the boundary surface of a solid defined by the union of those solid cells can be obtained. Specifically, the boundary surface consists of the loose faces between the neighboring solid and empty cells. No numerical predicate is involved in the surface extraction (i.e., Step 9 of Procedure *BSPtoBRep*).

For a given BSP solid $H$, it is possible that the topology of surface $M_H$ generated by our BSP => B-rep algorithm may be different from the boundary surface of $H$. However, our experimental results indicate that the Hausdorff distance between $M_H$ and the exact solution is typically less than $10^{-6}$.

Notice that, the theoretical error bounds on the topology and geometric representation of the resulting B-rep depend on several factors including the number of times the clipping operations are performed, the orientation of clipping planes and the size of models relative to the precision of floating-point arithmetic. For example, if a polytope is defined using many planes that are nearly parallel to each other, the clipping operations performed by our algorithm could result in a shape that may have a large Hausdorff distance due to round-off errors. However, such extreme cases are rare in practice and do not occur if we select the clipping plane defined in a BSP node perpendicular to the plane used in its parent node during the construction of BSP trees.

## 4.4 More Implementation Details

The data structure employed in our approach is introduced below. The whole model has four lists: cell list, face list, edge

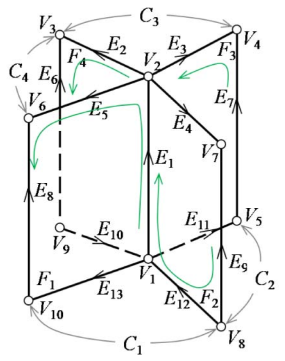


Fig. 8. An example of four cells $C_1$, $C_2$, $C_3$, and $C_4$ sharing the same edge $E_1$ and the same two vertices $V_1$ and $V_2$, where the edge $E_1$ will be nonmanifold if $C_1$ and $C_3$ are *empty* and $C_2$ and $C_4$ are *solid*.

list, and vertex list. The most important issue in the representation is the information about how an entity in dimension $d$ is shared by neighboring entities in one dimension higher as well as the orientations of these entities. In our data structure for BSP => B-rep conversion, an edge entity consists of two links pointing to its starting and ending vertices (e.g., $V_1$ and $V_2$ linked by $E_1$ in Fig. 8). At every vertex $V$, the list of adjacent edges is also stored. A face entity $F$ contains a list of edges forming its boundary in the counterclockwise order. Moreover, the direction of an edge in the face is specified by a sign of "+" or "−," and the links pointing to the faces adjacent to an edge $E$ are also stored in a face-link list of $E$. A cell $C$ is a closed two-manifold mesh surface composed by loose faces. The orientation of a face in $C$ is recorded as "+" if it's normal is pointing outwards and "−" for inwards. In addition, a face $F$ also stores the information about the cells above and below as $C_{F^+}$ and $C_{F^-}$, respectively. Note that the orientations of newly created entities are decided by topological information stored in the data structure and are not affected by the round-off errors.

By using this data structure during an edge split, the edge list of all these faces must be updated. When clipping a face $F$, the cells $C_{F^+}$ and $C_{F^-}$ located on both sides of $F$ must also update their face list although the polyhedron clipping is only performed on one side of $F$. The updating operations of this data structure have the complexity linear to the adjacency—according to our experimental tests, which is always less than 20. Processing the complex cells in this way can effectively avoid the face gluing step (as discussed in [36]). The clipping algorithm presented in the Section 4.2 can be easily implemented with the help of this data structure. Our current version of this data structure may have some redundant information, a more compact one can be used [38].

The boundary mesh surface for $H$ is extracted from the faces between the solid and the empty cells, called boundary faces. The vertices and edges adjacent to the boundary faces are also copied to the resultant mesh $M_H$. When an edge $E$ is shared by interlaced empty and solid cells after the comprehensive polyhedron clipping, $E$ will become a nonmanifold entity (e.g., $E_1$ in Fig. 8). Similarly, nonmanifold vertex $V$ will be generated when its adjacent cells are interlaced as empty and solid. According to our formulation of nearly convex cells, there is another nonmanifold case that some cell may only have two faces. However, all above nonmanifold entities can be easily eliminated during mesh construction, which is a separate problem and has been addressed in prior work (e.g., [39], [40], and [41]).

## 5 Experimental Results and Discussion

We have implemented our algorithm in C++ and used it for different geometric processing applications. All the performance results shown in this paper are generated on a PC with Intel Core 2 Quad CPU Q6600 2.4 GHz and 4 GB RAM running the 32-bit Windows Vista operating system.

In order to demonstrate the performance advantage of our approach, we implemented an exact BSP => B-rep algorithm based on [15]. The various components of the exact boundary algorithm are as follow:

- Since using exact arithmetic to represent the coordinates of vertices on a complex model results in a high memory overhead, we use plane-based representations [3], [9].
- The boundary surface is extracted from the set of convex polytopes generated and stored by the algorithm proposed by Bajaj and Pascussi [15].
- The orientation predicate is implemented by following the method of [3], which is similar to the method proposed by Shewchuk [42] but with a single stage filter due to simplicity. More specifically, when the absolute value of determinant is less than $10^{-10}$, the exact arithmetic routine provided by the GMP library [43] is used.

Table 1 shows the comparison of performance of our B-rep extraction algorithm with that of exact B-rep extraction algorithm highlighted above. In practice, we observe that our algorithm is 5 to 8 times faster, and has a speed that can be employed in the interactive applications (i.e., less than 5 s on moderate size models). To provide a fair comparison, we used IEEE double precision arithmetic to convert the plane-based representation into the point-based representation on the resultant mesh surfaces. This approach may not be robust when the three planes used to define the position of a vertex are nearly parallel to each other. A more robust way is to use exact arithmetic or *Singular Value Decomposition* (SVD), which will take more time, as compared to the timings reported in Table 1. Table 2 reports the peak memory usage of both the exact approach and our method. We see that memory costs are similar. This is because, when models are stored by plane-based representations [3], [9], the exact arithmetic is only temporarily used in computation which does not take too much memory.

We use 64 bits (i.e., standard double precision variables) for the coefficients in our implementation of exact BSP => B-rep algorithm since our point-based approach only uses 64 bits for the coordinates of vertices. However, the orientation predicates based on plane-based representation require more bits to compute an exact answer (as mentioned in [42]). When the required intermediate computation exceeds the hardware limit (e.g., 64 bits), the computation becomes expensive. This is the major reason why our approach using only IEEE double precision arithmetic can generate the results much quickly. The running time for 3D orientation predicates shown in Table 1 demonstrate the speedups. Although the speedups depend on the number of bits employed to represent the input,

TABLE 1
Comparison with an Exact Implementation Using Plane-Based Representation and Exact Predicates

| Model | BSP node Number$^{\triangle}$ | Total Time (sec.) | | Region Flooding* Time (sec.) | Time on Orient. Predicate (sec.) | |
|---|---|---|---|---|---|---|
| | | Exact Approach† | Our Method‡ | | Filtered Exact Predicate | Our Method |
| Spider | 69k (68) | 18.3 | 3.74 (×4.9) | 0.389 | 14.029 | 0.079 (×178) |
| Turtle | 54k (34) | 14.2 | 2.62 (×5.4) | 0.391 | 10.929 | 0.048 (×228) |
| David-egea | 114k (43) | 32.3 | 4.38 (×7.4) | 0.871 | 25.799 | 0.093 (×277) |
| Oil-rig | 127k (57) | 33.3 | 6.08 (×5.5) | 0.880 | 26.937 | 0.096 (×280) |
| Dutch-ship | 75k (70) | 22.8 | 3.45 (×6.6) | 0.554 | 18.225 | 0.059 (×308) |
| Harp | 73k (54) | 19.6 | 3.98 (×4.9) | 0.517 | 15.227 | 0.061 (×249) |
| Femur ∩ Truss | 101k (38) | 25.8 | 3.14 (×8.2) | 0.682 | 20.559 | 0.083 (×247) |
| Pig \ Helix | 176k (61) | 52.9 | 6.86 (×7.7) | 1.35 | 42.615 | 0.133 (×320) |
| Bunny | 163k (266) | 60.4 | 10.6 (×5.7) | 1.682 | 46.447 | 0.046 (×1,010) |

$^{\triangle}$ *The numbers shown in the bracket of the second column are the depths of the BSP trees.*
† *The exact approach is based on [15] and is improved by using plane-based representation and the single stage filtered exact predicates.*
‡ *Our method uses the standard IEEE double precision arithmetic supported in current processors.*
* *The time spent in the region flooding step of our algorithm to enhance robustness takes only less than 25 percent of the total running time.*

using too few bits to represent the input BSP tree may not work; and there is no consensus on how many bits are really needed to represent the input. Moreover, during our experimental tests, when 32 bits are used to represent the coefficients of plane equations on an input BSP tree, our implementation of the exact approach runs faster than given a 64 bits input. Finally, the time spent in the region flooding step of our clipping algorithm is also reported in Table 1. This part takes less than 25 percent of the total running time on all the examples.

Generally, plane-based representations [3] together with exact predicates have an expensive overhead in terms of running time. To reduce the cost, Campen and Kobbelt [9] only compute BSP trees locally in a few regions where intersections occur. In order to compare our approach with theirs, we tested the intersection removal implementation presented in Section 6.3 on the cow model with 6K faces, as used in [9]. A similar result is computed by our algorithm (see Fig. 10) in 2.7 s, while the authors of [9] reported that their algorithm takes 1.2 s on the same model but using a more powerful PC. The main reason our method takes longer is because of the iterative computation performed on all the cells of the BSP tree to determine their solidity by using [10]. More specifically, when using Murali and Funkhouser's algorithm [10] for model repair, the entire BSP solid must be converted into connected volumetric cells that lead to much higher memory requirements and more time is spent on memory management.

The shape approximation errors between the models generated by our BSP => B-rep approach and the models produced by the exact method are measured using the publicly available Metro tool [44]. Both the average $L^2$-norm and the Hausdorff distance with respect to the diagonal length of models' bounding boxes are checked. Among all models shown in Table 1, only the Dutch-ship and the Harp models have Hausdorff distances greater than $10^{-6}$ (i.e., $3 \times 10^{-6}$ and $1 \times 10^{-6}$ respectively). For the average $L^2$-norm, all models have errors less than $10^{-6}$. Note that $10^{-6}$ is the minimal error that can be reported by the Metro tool. This study about shape approximation error shows that our approach can reconstruct the two-manifold boundary surfaces very accurately.

**Limitations.** Although the experiments show that the results generated by our BSP => B-rep algorithm are accurate, the shape approximation error is actually affected by several issues.

- The source of the shape approximation error is the inaccurate positions computed for the newly inserted vertices as well as the epsilon-tweaking-based point orientation predicate. Both come from the edge splitting step. Positional errors depend on the orientation of the clipping plane compared to the direction of line segments to be clipped. Analysis shows that if they are nearly parallel to each other, it tends to generate larger round-off errors. In short, it depends on how the clipping planes are defined during the BSP tree construction. A good heuristic is to select a clipping plane which is nearly perpendicular to the previous clipping plane (see [45]).
- After edge splitting, the operations in the face clipping and the cell separation steps may slightly

TABLE 2
Statistics of Peak Memory Usage

| Model | BSP node Number | Peak Memory Usage | |
|---|---|---|---|
| | | Exact Approach | Our Method |
| Spider | 69k | 140 MB | 165 MB |
| Turtle | 54k | 117 MB | 141 MB |
| David-egea | 114k | 347 MB | 346 MB |
| Oil-rig | 127k | 359 MB | 406 MB |
| Dutch-ship | 75k | 171 MB | 202 MB |
| Harp | 73k | 185 MB | 182 MB |
| Femur ∩ Truss | 101k | 166 MB | 197 MB |
| Pig \ Helix | 176k | 435 MB | 445 MB |
| Bunny | 163k | 423 MB | 428 MB |

TABLE 3
Performance of Our Boolean Computation

| Models | Femur ∩ Truss | Pig \ Helix |
|---|---|---|
| Face # | 5.00k & 39.3k | 1.28k & 74.0k |
| BSP Construction | 0.432 sec. | 0.181 sec. |
| Boolean Operation | 1.24 sec. | 1.50 sec. |
| B-rep Extraction | 3.14 sec. | 6.86 sec. |
| ACIS R15 | 26.38 sec. | 48.10 sec. |
| CGAL | Failed | 64.23 sec. |

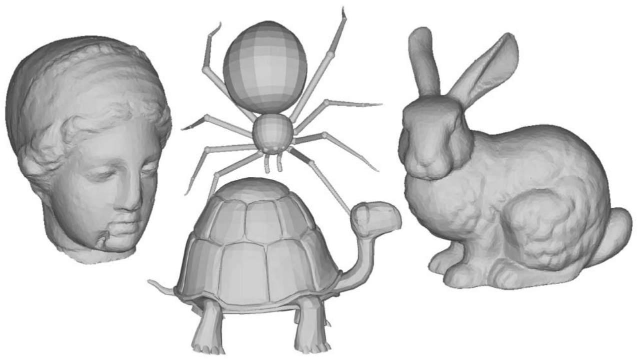


Fig. 9. Models used in comparing our method with an exact implementation using plane-based representation.

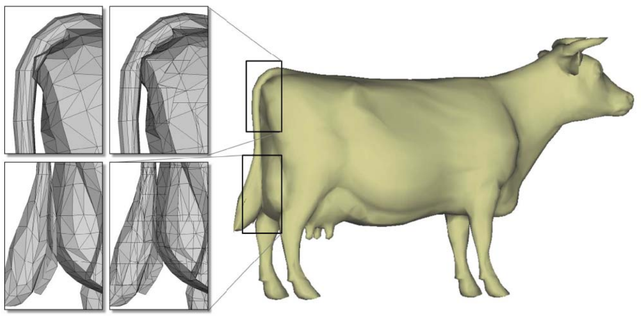


Fig. 10. Self-intersection removal on the cow model.

amplify the errors while enforcing the topology of clipping results.

- Finally, as the resultant inexact cells will be further clipped into smaller ones, errors embedded on the cells can be further propagated. Therefore, the final error also depends on the depth of the given BSP tree. In short, a much deeper BSP tree will likely lead to higher shape approximation error.

On the other hand, although the clipping algorithm is resistant to degeneracies, the BSP => B-rep algorithm does not have theoretical error bounds on its results. However, the geometric error could be small (e.g., less than $10^{-5}$ in our examples) if we tend to select the clipping plane defined in a BSP node perpendicular to the plane used in its parent node when constructing BSP trees. The resulting mesh surface $M_H$ generated by our algorithm is a good approximation of the real boundary surface $\partial H$ of $H$.

## 6 Applications

In this section, we highlight several applications of this BSP => B-rep algorithm, including Boolean operations, model repair and mesh reconstruction.

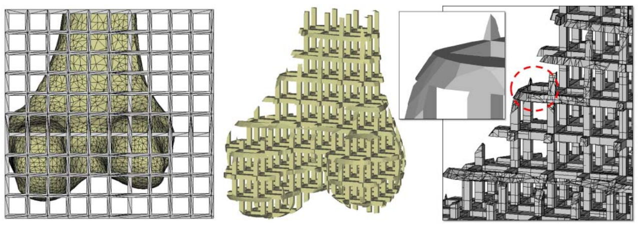


Fig. 12. Our result of Boolean operation on the truss model and the femur model. The small and sharp features are preserved by our polygonization algorithm. The model is very complex in terms of shape and topology, however it still can be successfully reconstructed by our method.

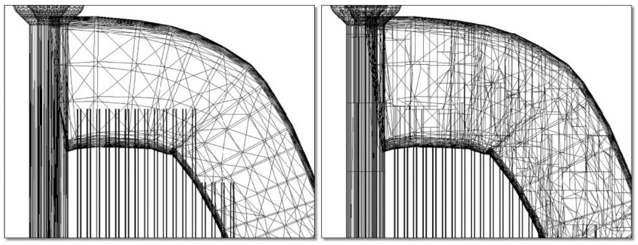


Fig. 13. Self-intersection removal on the harp model (previously shown in Fig. 3): (left) the problematic regions corresponding to small features—the strings, and (right) our repaired result which preserves the small features.

### 6.1 Boolean Operations

Our computation of Boolean operations is based on the approach described in [4], where linear programming is used to check the feasibility of regions represented by leaf-nodes of the BSP tree. The resultant BSP tree corresponding to the Booleans is obtained by removing all the infeasible regions, and the B-rep is generated by our extraction method. The first example shown in Fig. 12 comes from the tissue engineering. The goal is to generate the scaffold of a femur by performing an intersection operation between a truss model and the femur. The second example corresponds to subtracting a helix from a pig model, and Fig. 11 shows the surface generation result. We have also compared the performance of our algorithm with ACIS R15 [14] and CGAL [20], where the CGAL implementation uses the kernel with exact predicates and exact constructions. The performance in Table 3 shows that our algorithm is more efficient and stable.

### 6.2 Model Repair

BSP trees can be used for model repair [9], [10]. After constructing the connected complex cells, the algorithm described in [10] can be adopted to assign the solidity of each cell in terms of global optimization. However, when this method is applied to a self-intersecting closed mesh

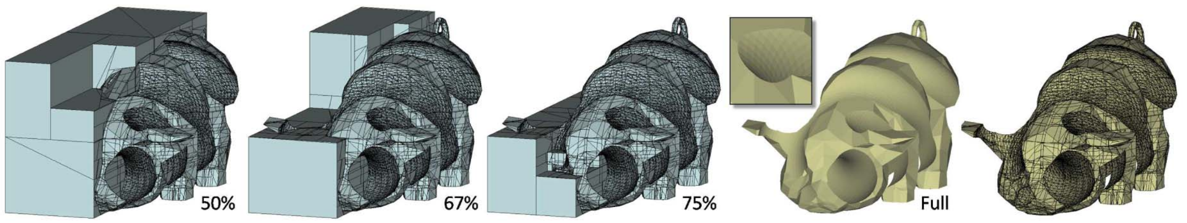


Fig. 11. The procedure of generating the resultant surface from Boolean difference operation (Pig\Helix). Smooth and sharp features are successfully reconstructed.

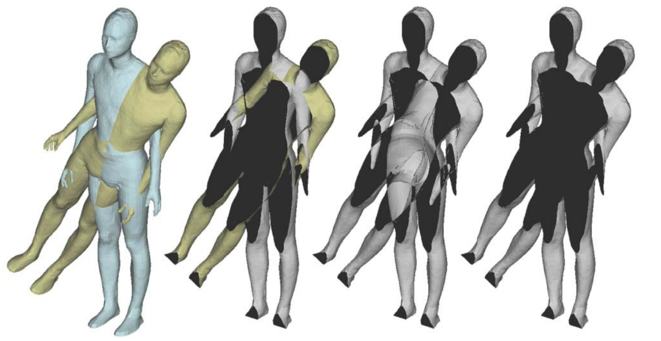


Fig. 14. Example of self-intersection removal. From left to right, the given model with self-intersection, inside of the given model, the processed result generated by Murali and Funkhouser [10] with interlaced solid and empty shells, and the final model generated by our algorithm.

surface, a solid with interlaced solid and empty shells will be generated (see the middle of Fig. 14). Given such a solid with interlaced shells, we apply a flooding step to the connected cells similar to [9]. The inside of the model is defined as the volume that is not reachable from infinity (i.e., the unbounded cells) without crossing the surface. The results of our approach are shown in Figs. 3, 9, 10, 13, 14, and 15.

Our BSP => B-rep algorithm is fast, which gives it the benefit of providing a very good model repair function (by using [10]) in an interactive speed. Table 4 shows the statistics of the model repair implementation using our BSP => B-rep algorithm. The repair on most models can be completed in less than 5 s on a moderate level PC, where the reported time includes all three steps of the model repair: 1) BSP construction, 2) solidity correction [10], and 3) B-rep extraction.

## 6.3 Mesh Reconstruction

As shown in Fig. 16, the above mesh repair technique is also employed to fix the mesh surface that is reconstructed from point cloud using a parallel local triangulation method (e.g., [47]), which can generate the mesh surfaces in an interactive speed. The mesh surfaces generated by this local triangulation-based method usually have the problem of nonmanifold entities, holes, as well as inconsistent orientations (see Fig. 16). These models can be repaired into closed two-manifold mesh surfaces by our approach.

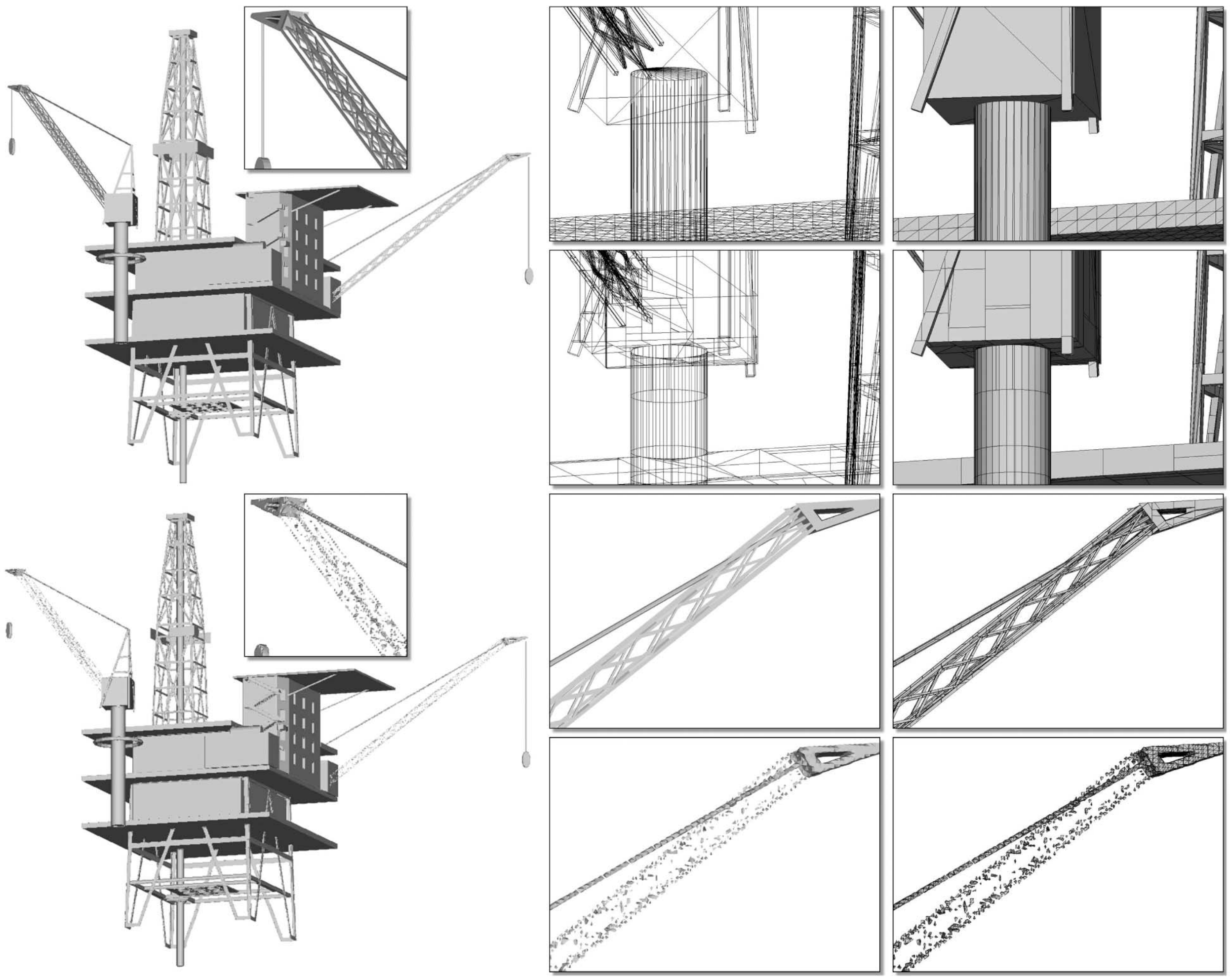


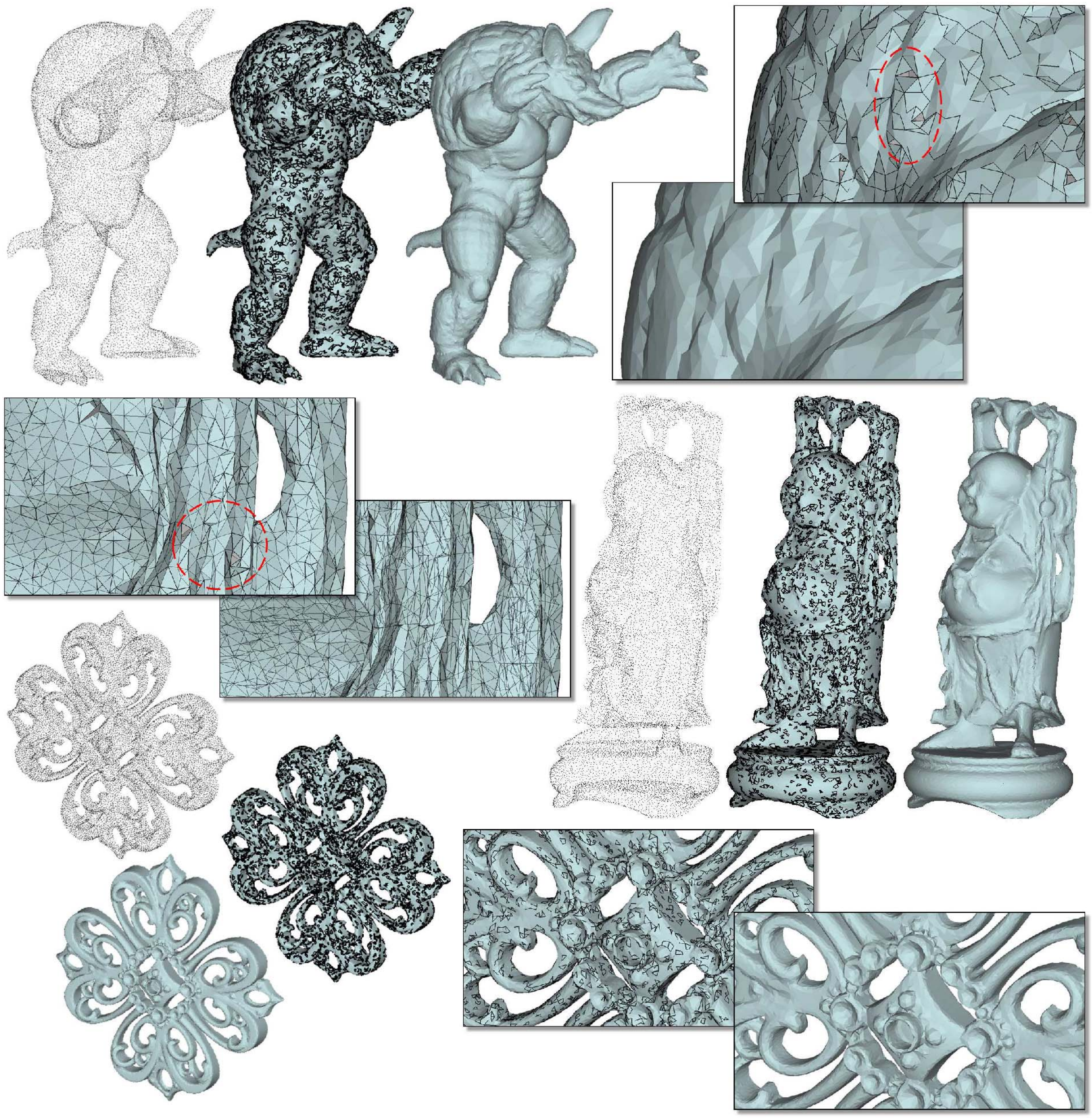
Fig. 15. Example of the oil-rig model with self-intersection removed by our approach (top) versus the mesh repair result from the PolyMender algorithm of [46] (bottom). The zoomed views from top to bottom show: the given mesh model with self-intersections, the repaired mesh model generated by our method, the repaired mesh model with preserved small features by our method, the small features not preserved by the octree-based PolyMender algorithm [46].

TABLE 4
Statistics of Model Repair

| Models | Fig. | Input f# | Time (sec.) | Result f# |
|---|---|---|---|---|
| Dutch-ship | 3 | 15,812 | 4.91 | 45,984 |
| Harp | 3 | 10,652 | 4.67 | 45,853 |
| Spider | 9 | 4,999 | 4.68 | 36,317 |
| Turtle | 9 | 5,499 | 3.57 | 29,386 |
| David-egea | 9 | 16,532 | 6.27 | 62,521 |
| Cow | 10 | 5,804 | 2.86 | 25,988 |
| Oil-rig | 15 | 45,326 | 7.90 | 83,672 |

## 7 Conclusions and Future Work

We present an efficient polygonization algorithm for solids represented using BSP trees. The main contribution is a novel clipping algorithm for degenerate convex polyhedra using IEEE double precision arithmetic. In our algorithm, the clipping operations combinatorially ensure the output of two objects with genus zero topology when using standard IEEE double precision arithmetic—the benefit of hardware support is retained. Based on this novel clipping scheme, we have developed a stable and efficient B-rep extraction method for BSP solids. The initial results are



| Model | Point # | Triangle # | Repair Time (Sec.) | Resultant Face # |
|---|---|---|---|---|
| Armadillo | 25k | 108,241 | 25.1 | 231,322 |
| Buddha | 25k | 105,578 | 23.9 | 238,119 |
| Filigree | 25k | 106,018 | 31.4 | 246,027 |

Fig. 16. Mesh repair for surfaces reconstructed from point clouds. From left to right, point clouds, mesh surfaces with problems (where the problematic regions are displayed by bolded lines), and repaired two-manifold mesh surfaces. The reported time includes the time for 1) BSP construction, 2) solidity repair [10], and 3) boundary surface extraction.

quite promising and our algorithm can accurately reconstruct sharp features.

There are many avenues for future work. We would like to further improve our algorithm to provide bounds on geometric and topological errors. We can also combine the algorithm with octrees (similar to [9]) and further improve its performance. In the approach presented in this paper, we basically assume that the BSP tree is constructed "exactly" and try to compute the corresponding B-rep. Our future work would take into account the error in constructing the BSP tree and its impact on the final B-rep. Another possible area for future work is to design efficient and specialized algorithms for "BSP => CSG => B-rep conversion." Finally, as our algorithm proceeds in a divide-and-conquer manner, it may be possible to develop out-of-core implementation to handle very large and complex models by using a method similar to [48].

## Acknowledgments

The authors would like to thank the valuable comments given by anonymous reviewers. This work was partially supported by HKSAR Research Grants Council (RGC) General Research Fund (GRF) CUHK/417508 and CUHK/417109, and it was also supported by US National Science Foundation (NSF) grants 1000579 and 1117127. The authors would like to thank Pu Huang and Hoi Sheung who help implement the algorithm of [4] and [47].

## References

[1] R.A. Schumacker, R. Brand, M. Gilliland, and W. Sharp, "Study for Applying Computer-Generated Images to Visual Simulation," Technical Report AFHRL-TR-69-14, U.S. Air Force Human Resources Laboratory, 1969.

[2] H. Fuchs, Z.M. Kedem, and B.F. Naylor, "On Visible Surface Generation by a Priori Tree Structures," *Proc. ACM SIGGRAPH '80*, pp. 124-133, 1980.

[3] G. Bernstein and D. Fussell, "Fast, Exact, Linear Booleans," *Computer Graphics Forum*, vol. 28, no. 5, pp. 1269-1278, 2009.

[4] M. Lysenko, R. D'Souza, and C.-K. Shene, "Improved Binary Space Partition Merging," *Computer-Aided Design*, vol. 40, no. 12, pp. 1113-1120, 2008.

[5] A. Paoluzzi, V. Pascucci, and G. Scorzelli, "Progressive Dimension-Independent Boolean Operations," *Proc. Ninth ACM Symp. Solid Modeling and Applications (SM '04)*, pp. 203-211, 2004.

[6] R. Banerjee and J. Rossignac, "Topologically Exact Evaluation of Polyhedra Defined in CSG with Loose Primitives," *Computer Graphics Forum*, vol. 15, no. 4, pp. 205-217, 1996.

[7] B.F. Naylor, J. Amanatides, and W. Thibault, "Merging BSP Trees Yields Polyhedral Set Operations," *Proc. ACM SIGGRAPH '90*, pp. 115-124, 1990.

[8] W.C. Thibault and B.F. Naylor, "Set Operations on Polyhedra Using Binary Space Partitioning Trees," *SIGGRAPH Computer Graphics*, vol. 21, no. 4, pp. 153-162, 1987.

[9] M. Campen and L. Kobbelt, "Exact and Robust Self-Intersections for Polygonal Meshes," *Computer Graphics Forum*, vol. 29, no. 2, pp. 397-406, 2010.

[10] T.M. Murali and T.A. Funkhouser, "Consistent Solid and Boundary Representations from Arbitrary Polygonal Data," *Proc. ACM Symp. Interactive 3D Graphics (I3D '97)*, pp. 155-162, 1997.

[11] P. Huang and C.C.L. Wang, "Volume and Complexity Bounded Simplification of Solid Model Represented by Binary Space Partition," *Proc. ACM Symp. Solid and Physical Modeling*, 2010.

[12] Rhino3d, *Rhinoceros ver 4.0*, http://www.rhino3d.com, 2009.

[13] Autodesk, *Maya*, http://www.autodesk.com, 2009.

[14] Spatial, *3D ACIS Modeler R15*, http://www.spatial.com, 2008.

[15] C.L. Bajaj and V. Pascucci, "Splitting a Complex of Convex Polytopes in Any Dimension," *Proc. ACM 12th Ann. Symp. Computational Geometry (SCG)*, pp. 88-97, 1996.

[16] C.L. Bajaj, A. Paoluzzi, and G. Scorzelli, "Progressive Conversion from B-Rep to BSP for Streaming Geometric Modeling," *Computer-Aided Design and Applications*, vol. 3, no. 5, pp. 577-586, 2006.

[17] W.C. Thibault, "Application of Binary Space Partitioning Trees to Geometric Modeling and Ray-Tracing," PhD thesis, Director-Naylor, Bruce F, 1987.

[18] Supplementary Document, "Exact versus Approximate Predicates and Point-Based versus Plane-Based Representation," 2010.

[19] A.A.G. Requicha and H.B. Voelcker, "Boolean Operations in Solid Modeling: Boundary Evaluation and Merging Algorithms," *Proc. IEEE*, vol. 73, no. 1, pp. 30-44, Jan. 1985.

[20] CGAL, *CGAL - Computational Geometry Algorithms Library*, http://www.cgal.org/, 2010.

[21] J. Rossignac, "Blist: A Boolean List Formulation of CSG Trees," 1998.

[22] A. Rappoport and S. Spitz, "Interactive Boolean Operations for Conceptual Design of 3-D Solids," *Proc. ACM SIGGRAPH '97*, pp. 269-278, 1997.

[23] J. Hable and J. Rossignac, "Blister: GPU-Based Rendering of Boolean Combinations of Free-Form Triangulated Shapes," *Proc. ACM SIGGRAPH '05*, pp. 1024-1031, 2005.

[24] J. Hable and J. Rossignac, "CST: Constructive Solid Trimming for Rendering BReps and CSG," *IEEE Trans. on Visual and Computer Graphics*, vol. 13, no. 5, pp. 1004-1014, Sept./Oct. 2007.

[25] B.R. Vatti, "A Generic Solution to Polygon Clipping," *Comm. ACM*, vol. 35, no. 7, pp. 56-63, 1992.

[26] G. Greiner and K. Hormann, "Efficient Clipping of Arbitrary Polygons," *ACM Trans. Graphics*, vol. 17, no. 2, pp. 71-83, 1998.

[27] D. Pavic, M. Campen, and L. Kobbelt, "Hybrid Booleans," *Computer Graphics Forum*, vol. 29, pp. 75-87, 2010.

[28] M. Kazhdan, A. Klein, K. Dalal, and H. Hoppe, "Unconstrained Isosurface Extraction on Arbitrary Octrees," *Proc. Fifth Eurographics Symp. Geometry Processing (SGP '07)*, pp. 125-133, 2007.

[29] S. Bischoff, D. Pavic, and L. Kobbelt, "Automatic Restoration of Polygon Models," *ACM Trans. Graphics*, vol. 24, no. 4, pp. 1332-1352, 2005.

[30] T. Ju, F. Losasso, S. Schaefer, and J. Warren, "Dual Contouring of Hermite Data," *Proc. ACM SIGGRAPH '02*, pp. 339-346, 2002.

[31] P. Brunet and I. Navazo, "Solid Representation and Operation Using Extended Octrees," *ACM Trans. Graphics*, vol. 9, no. 2, pp. 170-197, 1990.

[32] C. Wojtan, N. Thürey, M. Gross, and G. Turk, "Physics-Inspired Topology Changes for Thin Fluid Features," *ACM Trans. Graphics*, vol. 29, no. 4, pp. 1-8, 2010.

[33] C.M. Hoffmann, "Robustness in Geometric Computations," *ASME J. Computing and Information Science in Eng.*, vol. 1, pp. 143-156, 2001.

[34] G. Barequet, "Using Geometric Hashing to Repair CAD Objects," *IEEE Computational Science and Eng.*, vol. 4, no. 4, pp. 22-28, Oct. 1997.

[35] C. DeCoro and N. Tatarchuk, "Real-Time Mesh Simplification Using the GPU," *Proc. ACM Symp. Interactive 3D graphics and Games (I3D '07)*, pp. 161-166, 2007.

[36] J. Comba and B. Naylor, "Conversion of Binary Space Partitioning Trees to Boundary Representation," *Proc. Theory and Practice of Geometric Modelling*, 1996.

[37] Qhull "Qhull 2010.1," UIUC Geometry Center, QHull Computational Geometry Package, 2010.

[38] L. de Floriani and A. Hui, "A Scalable Data Structure for Three-Dimensional Non-Manifold Objects," *Proc. Eurographics Symp. Geometry Processing*, pp. 72-82, 2003.

[39] J. Rossignac and D. Cardoze, "Matchmaker: Manifold BReps for Non-Manifold R-Sets," *Proc. Fifth Symp. Solid Modeling*, pp. 31-41, 1999.

[40] T.K. Dey and S. Goswami, "Tight Cocone: A Water-Tight Surface Reconstructor," *Proc. Eighth ACM Symp. Solid Modeling and Applications (SM '03)*, pp. 127-134, 2003.

[41] C.C.L. Wang, "Approximate Boolean Operations on Large Polyhedral Solids with Partial Mesh Reconstruction," *IEEE Trans. Visualization and Computer Graphics*, vol. 17, no. 6, pp. 836-849, June 2010.

[42] J.R. Shewchuk, "Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates," *Discrete and Computational Geometry*, vol. 18, pp. 305-363, 1997.

[43] GMP, *The GNU Multiple Precision Arithmetic Library,* http://gmplib.org/, 2010.

[44] P. Cignoni, C. Rocchini, and R. Scopigno, "Metro: Measuring Error on Simplified Surfaces," *Computer Graphics Forum,* vol. 17, no. 2, pp. 167-174, 1998.

[45] E. Langetepe and G. Zachmann, *Geometric Data Structures for Computer Graphics.* A K Peters, Ltd., 2006.

[46] T. Ju, "Robust Repair of Polygonal Models," *ACM Trans. Graphics,* vol. 23, no. 3, pp. 888-895, 2004.

[47] C. Buchart, D. Borro, and A. Amundarain, "GPU Local Triangulation: An Interpolating Surface Reconstruction Algorithm," *Computer Graphics Forum,* vol. 27, no. 3, pp. 807-814, 2008.

[48] T.K. Dey, J.A. Levine, and A. Slatton, "Localized Delaunay Refinement for Sampling and Meshing," *Computer Graphics Forum,* vol. 29, no. 5, pp. 1723-1732, 2010.

**Charlie C.L. Wang** received the BEng degree in mechatronics engineering from Huazhong University of Science and Technology, in 1998, and the MPhil and PhD degrees in mechanical engineering from the Hong Kong University of Science and Technology, in 2000 and 2002, respectively. Currently, he is working as an associate professor at the Department of Mechanical and Automation Engineering, the Chinese University of Hong Kong, where he began his academic career in 2003. His current research interests include geometric modeling in computer-aided design and manufacturing, biomedical engineering and computer graphics, as well as computational physics in virtual reality. He has received a few awards including the ASME CIE Young Engineer Award (2009), the CUHK Young Researcher Award (2009), the CUHK Vice-Chancellor's Exemplary Teaching Award (2008), the Best Paper Awards of ASME CIE Conferences (in 2008 and 2001), and the Prakash Krishnaswami CAPPD Best Paper Award of ASME CIE Conference in 2011. He is a member of the ASME and the IEEE, and chairman of Technical Committee on Computer-Aided Product and Process Development (CAPPD) of ASME.

**Dinesh Manocha** received the PhD degree in computer science at the University of California, Berkeley, in 1992. Currently, he is working as a Phi Delta Theta/Mason distinguished professor of computer science at the University of North Carolina at Chapel Hill. He has published more than 330 papers in computer graphics, geometric computation, robotics and many-core computing and received 12 best-paper awards. Some of the software systems developed by his group on collision and geometric computations, interactive rendering, crowd simulation, and GPU-based algorithms have been downloaded by more than 100K users and widely licensed by commercial vendors. He has served in the program committees of more than 100 leading conferences and in the editorial board of more than 10 leading journals. He has won many awards including US National Science Foundation (NSF) Career Award, ONR Young Investigator Award, Sloan Fellowship, IBM Fellowship, SIGMOD IndySort Winner, Honda Research Award, and UNC Hettleman Prize. He is a fellow of the ACM, the AAAS, and the IEEE and received Distinguished Alumni Award from Indian Institute of Technology, Delhi.