Volume and Complexity Bounded Simplification of Solid Model Represented by Binary Space Partition

Pu Huang Charlie C. L. Wang*
Department of Mechanical and Automation Engineering
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong, China
cwang@mae.cuhk.edu.hk

ABSTRACT

We present a volume and complexity bounded solid simplification of models represented by *Binary Space Partition* (BSP). Depending on the compact and robust representation of a solid model in BSP-tree, the boundary surface of a simplified model is guaranteed to be two-manifold and self-intersection free. Two techniques are investigated in this paper. The volume bounded convex simplification can collapse parts with small volumes on the model into a simple convex volume enclosing the volumetric cells on the input model. The selection of which region to simplify is based on a volume-difference metric, with the help of which the volume difference between the given model and the simplified one is minimized. Another technique is a plane collapse method which reduces the depth of the BSP-tree while still preserving volume bounding. These two techniques are integrated into our solid simplification algorithm to give satisfactory results. Related applications are given at the end of this paper to demonstrate the function of our algorithm.

Categories and Subject Descriptors

I.3.5 [**Computational Geometry and Object Modeling**]: Boundary representations – Curve, surface, solid, and object representations

Keywords

Simplification, Volume Bounded, Complexity Bounded, Binary Space Partition, Solid Model

1. INTRODUCTION

Binary Space Partition (BSP) tree is a binary tree which represents a d-space partitioning by hyper-planes for dimension d, and it can provide an exact representation of arbitrary polyhedral objects in d-dimensional space. A balanced BSP representation of a model allows fast point classification [5, 20], collision detection [13, 17], visible surface detection [9] and Boolean operations [4, 20, 25]. It also generates a natural convex decomposition of a model. However, many applications suffer from the huge data size of a BSP tree when they have to transfer the BSP tree data through network or to perform geometric operations like Boolean operation. Moreover, there are applications that do not necessarily need the exact representation of a model but only require a BSP tree to represent a space partition that bounds the volume of the model loosely (e.g., robot path planning [26], acceleration for collision detection [8,15] and tool path planning for rough machining [29]). Our algorithm aims at reducing the total number of nodes on a BSP tree while preserving the bounds on volume (i.e., ensuring the given model is enclosed by the space partition represented by a simplified BSP tree) and complexity (i.e., retaining or reducing the maximum depth of the tree). We explore a progressive simplification strategy to reduce the size of BSP tree by making use of the natural property of convex decomposition of BSP tree to fulfill the concave features with material. Figure 1 gives a simplification example of a spine model represented by a BSP tree with 530,975 nodes, and Fig.2 shows the mesh simplification results of the same spine model by the algorithm in [10]. The results generated by mesh simplification do not bound the volume of the given model.

1.1 Problem definition

BSP tree provides a compact and robust representation of a solid model, and the boundary surface of a solid represented by BSP tree is guaranteed to be two-manifold and self-intersection free. The data structure of *Binary Space Partition* (BSP) tree is a binary tree, where each non-leaf-node stores a linear equation to specify a hyper-plane to partition the space into two half-spaces. A leaf-node presents a convex region as the intersection of all half-spaces on the path from the tree's root to this leaf node. Every leaf-node is labeled as either *solid* (in black) or *empty* (in white) according to whether or not its corresponding convex region is part of the solid to be represented. Figure 3 shows an example of the BSP tree of a simple polygon.

For a given solid H represented by a BSP tree Γ^0 with $|\Gamma^0|$ nodes, we are going to compute a simplified BSP tree Γ with $|\Gamma| < |\Gamma^0|$. Here, $|\cdots|$ denotes the number of nodes on a BSP tree. However, this is ill-posed as there are many possible such BSP trees. Considering the applications where the simplified tree is employed, we expect that the tree Γ bounds the volume of Γ^0 as tightly as possible. Moreover, the complexity should also be bounded, which means that $\tau_{max}(\Gamma) \leq \tau_{max}(\Gamma^0)$ with $\tau_{max}(\cdots)$ returning the maxi-

*Corresponding Author.

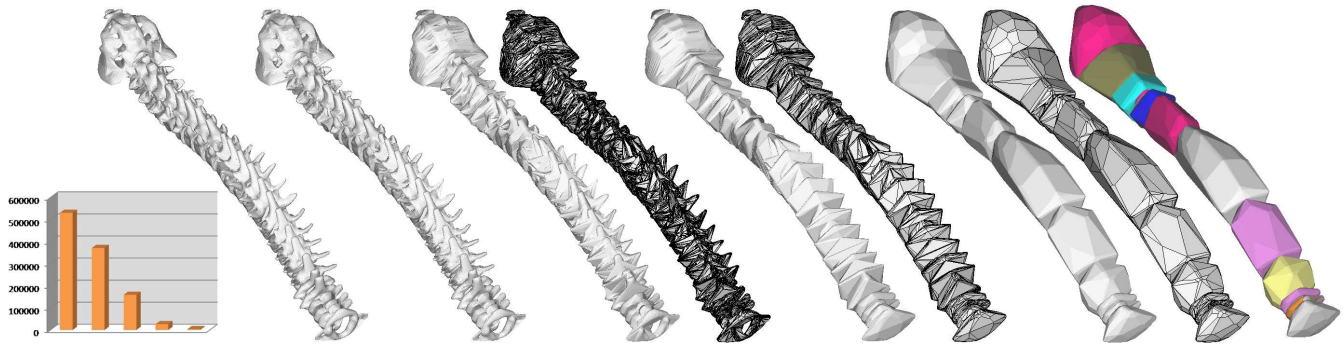

Figure 1: The solid simplification of a spine model. From the left: the input spine solid (with 530k nodes on a BSP tree and the maximum depth, $\tau_{\max}$, of the tree is 124), a slightly simplified solid (with 70% of nodes retained and $\tau_{\max} = 112$), a simplified solid (with 30% of nodes and $\tau_{\max} = 113$), a further simplified solid (with 5% of nodes and $\tau_{\max} = 114$), and a coarser solid (with only 0.5% of nodes and $\tau_{\max} = 114$) – regions in different convex hulls are displayed in different colors. The relative volume errors of the simplified solids are 1.83%, 23.5%, 77.9% and 148.2% of the original model respectively.

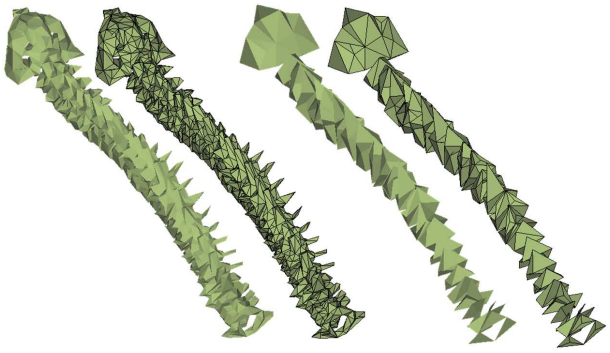

Figure 2: The mesh simplification results of the spine model using the quadric error metrics in [10]: (left) about 5% of vertices are retained, and (right) only 0.5% of vertices of the original model are retained. The simplified models do not bound the volume of the original spine.

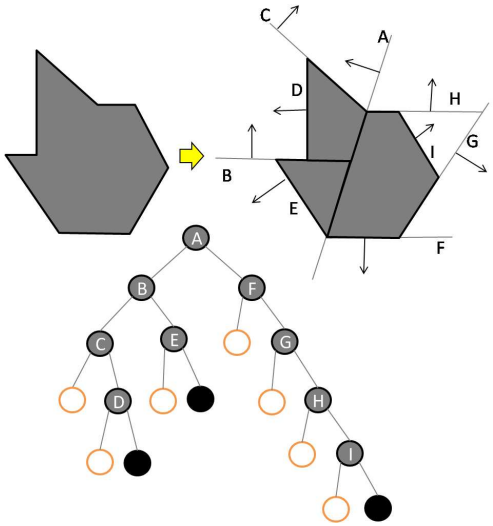

Figure 3: A polygon can be decomposed to be represented by a BSP tree, where non-leaf-nodes are in grey and leaf-nodes are in black or white according to whether its convex region is *solid* or *empty*.

mum depth of a tree. The tightness of volume bounding is measured by the volume difference between the solids represented by the simplified and the original trees as

$$M(\Gamma) = |V(\Gamma) - V(\Gamma^0)|, \tag{1}$$

where $V(\Gamma) = \sum_{s_i} V(s_i)$ is evaluated by summing up the volumes of all *solid* leaf-nodes $s_i \in \Gamma$.

1.2 Related work

We briefly review the work closely related to the conversion between B-rep and BSP tree, the volume simplification, BSP tree based solid modeling and other applications.

At present, the most popular representation for free-form objects is piecewise linear B-rep – polygonal meshes, which is convenient for local manipulation and rendering. However, it inherits the common drawback of B-rep – difficult for space evaluation (e.g., point membership classification and volume computation). The most conventional method to convert polyhedra from B-rep to BSP tree is the approach presented by Thibault and Naylor in [25]. The algorithm is based on repeatedly selecting a planar polygon on the surface of a given model as a clipping plane (i.e., the non-leaf-node on BSP tree) to separate other polygons into its left and right half-spaces. Although the constructed BSP tree can present the boundary surface of a given model exactly, the tree itself is not well balanced. Bajaj et al. introduced a progressive conversion from B-rep to BSP tree for streaming geometric modeling in [1]. They used eigen decomposition of the Euler tensor which represents surface inertia to cut the model in order to get relatively balanced BSP tree. For the BSP trees constructed by different strategies, the simplified solids generated by our method are quite different (see the illustration shown in Fig.4). In our tests, the simplified models obtained from a balanced BSP tree by [1] generally give better visual appearance than the results of the BSP

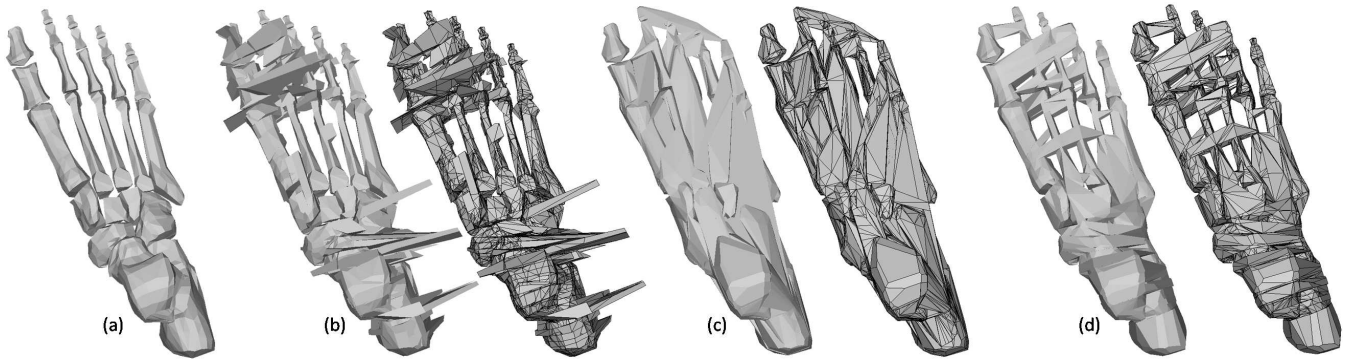

Figure 4: The simplified hand models on different BSP trees: (a) the given solid model with 34,743 nodes, (b) the result of simply truncating the BSP tree at depth 20 – there are 10,986 nodes above this level, (c) the simplified model with 10,317 nodes obtained using a BSP tree constructed by the approach of Thibault and Naylor [25], and (d) the simplified model with 10,403 nodes resulted using a balanced BSP tree generated by [1].

tree generated by [25]. The resultant BSP tree is converted back into a polygonal mesh for some downstream applications (e.g., rendering, FEM and BEM analysis). A widely used strategy which is based on polygon/convex clipping (ref. [2, 20]) is problematic to generate two-manifold boundary surface when using fixed precision arithmetics on the BSP tree models with many nearly co-planar nodes. Therefore, we employ the recently developed algorithm in [27] to compute two-manifold boundary surface from a BSP tree. All results presented in this paper are displayed by mesh surfaces generated by this method. Another B-rep generation method in [7] is based on extending a BSP tree into a topological BSP tree, which however requires much more memory and is impractical to be applied to huge models.

He et al. presented one of the earliest voxel based object simplification algorithms in [12]. They accomplished this by sampling and low-pass filtering the object into multi-resolution volume buffers and using marching cube to generate a triangle-mesh hierarchy. Shekhar et al. [24] proposed a cube-based simplification. This kind of simplification always samples the volume data into an octree and performs adaptive reconstruction using marching cube. However, cube-based algorithms always need a crack patching process to deal with the problem of void space which is not shown on the given models. Wood et al. [28] swept models and constructed a Reeb graph to remove topological errors in isosurface in the form of tiny handles. A skeleton based method was presented in [30] for repairing solid models with the help of octree, and a variant of this was also presented in [14]. Recently, a mesh repair method using BSP tree is presented in [5] to address the problem of removing self-intersection, which is different from our problem. Although many volume simplification approaches exist in literature, they rarely address the tightness of volume bounding. Recently, a topology simplification frame considering both shape and volume errors has been published in [11]. It adopts *Constrained Delaunay Tetrahedralization* (CDT) to fill the concave features on B-rep models. Similar to it, our method also simplifies the given model by filling the concave features, but we do so directly on *Binary Space Partition* (BSP) trees instead.

Employing BSP tree in solid modeling has a very long history (ref. [20, 21, 25]). Recently, fast Boolean operations on solid models represented in BSP tree have been considered in [4] and [18]. However, after repeatedly applying Boolean operations on a model, the resultant BSP tree has many redundancies. The removal of them has not been considered. In addition, a simplified BSP tree with volume and complexity bounded can speed up the computations in applications using BSP trees (e.g., [8, 13, 15, 17, 26, 29]).

1.3 Main results

We present a progressive simplification method for BSP tree, where the priority of simplification is controlled by the volume difference between the given model and the solid represented by the simplified BSP tree (i.e., the metric in Eq.(1)). Starting from Γ^0, we progressively compute a sequence of simplified BSP trees Γ^i $(i = 1, 2, \ldots, l)$ with $|\Gamma^i| > |\Gamma^{i+1}|$ and $M(\Gamma^i) \leq M(\Gamma^{i+1})$. Not only the number of nodes and the volume but also the spaces $\Omega(\cdots)$ occupied by Γ^i are embedded, i.e.,

$$\Omega(\Gamma^0) \subseteq \Omega(\Gamma^1) \subseteq \cdots \Omega(\Gamma^i) \subseteq \Omega(\Gamma^{i+1}) \subseteq \cdots \Omega(\Gamma^l).$$

The complexities of trees are also bounded by preserving $\tau_{\max}(\Gamma^i) \leq \tau_{\max}(\Gamma^0)$.

Two techniques are investigated in this paper. The volume bounded convex simplification introduced in section 3 can merge parts with small volumes on the model into a simple convex volume enclosing the volumetric cells on the input model. The plane collapse technique presented in section 4 can reduce the depth of a BSP tree while still preserving volume bounding.

2. ALGORITHM OVERVIEW

The overall algorithm of our solid simplification is introduced in this section. We adopt a bottom-up incremental strategy here.

Starting from every solid leaf-node on the given BSP tree Γ^m for a solid H, we walk up k levels and reach a node, the sub-tree under which is named as *granule*. Here, k is a parameter that can be assigned by users to specify the

granularity of simplification in each step; $k = 6$ is selected in all our examples in this paper. We wish to approximate the solid inside a granule γ by a simplified BSP sub-tree $\hat{c}_\gamma$ that holds fewer nodes (i.e., half-spaces). Also, the depth of $\hat{c}_\gamma$ should satisfy $\tau_{\max}(\hat{c}_\gamma) \leq \tau_{\max}(\gamma)$. Details about how to obtain $\hat{c}_\gamma$ are presented in sections 3 and 4.

When replacing γ by the half-spaces in $\hat{c}_\gamma$, the volume error added to the new tree is $(V(\hat{c}_\gamma) - V(\gamma))$. Therefore, in our algorithm, all candidate granules are inserted into a priority queue keyed by this volume error, which can guarantee that the granule with the minimal volume error is always simplified first. After a simplification step, a new granule is identified and inserted into the queue.

The simplification procedure keeps running until the volume difference $M(\Gamma^n)$ of the current new tree Γ^n or the reduced number of nodes exceeds the given thresholds. The tree Γ is then reported as the simplification result. The pseudo-code is presented in **Algorithm** *BSPSolidSimplification*.

Algorithm 1 BSPSolidSimplification

1: $V_{err} = 0$ and $n_{red} = 0$;
2: Duplicate a BSP tree Γ from the input Γ^0;
3: Initialize the priority queue Θ;
4: Find all granules starting from the left deepest solid leaf-nodes; {*We visit the tree in the LMR order*}
5: **for** all granules γ **do**
6: Compute the simplified BSP sub-tree $\hat{c}_\gamma$;
7: Insert γ into Θ by the weight $(V(\hat{c}_\gamma) - V(\gamma))$;
8: **end for**
9: **while** $\Theta \neq \phi$ **do**
10: Remove the granule γ_{cur} from the top of Θ;
11: On the BSP tree Γ, replace γ_{cur} by $\hat{c}_{\gamma_{cur}}$;
12: $V_{err} = V_{err} + (V(\hat{c}_{\gamma_{cur}}) - V(\gamma_{cur}))$;
13: $n_{red} = n_{red} + \Delta_{\gamma_{cur}}$;
{ $\Delta_{\gamma_{cur}}$ *reporting the number of nodes reduced*}
14: Update the queue Θ by adding the new granule;
{*Note that the new granule is determined by the new solid leaf-node on* γ_{cur}}
15: **if** ($V_{err} > threshold$) OR ($n_{red} > threshold$) **then**
16: **break**;
17: **end if**
18: **end while**
19: **return** Γ^n;

3. VOLUME BOUNDED CONVEX SIMPLIFICATION

The method for computing a volume bounded simplification from a given BSP tree Γ^m is presented in this section. For a given granule represented by a BSP sub-tree γ, we wish to approximate the solid inside it by a simplified BSP tree with fewer nodes (i.e., half-spaces). A BSP tree representation provides a natural convex decomposition for a model. The number of solid convex hulls equals to the number of black nodes. For example in Fig.3, the granule of node B contains two convex hulls, where the one below D is surrounded by A^+, B^+, C^- and D^-, and the one below E is formed by A^+, B^- and E^-. Here the signs specify the portion of their corresponding half-spaces.

Proposition 1 For a set of convex hulls $\{c_i\} \in \Re^3$, if the vertices of these convex hulls are $\{\mathbf{v_i}\}$, the convex hull of these vertices C_v is coincident with the convex hull C_c of these convex regions.

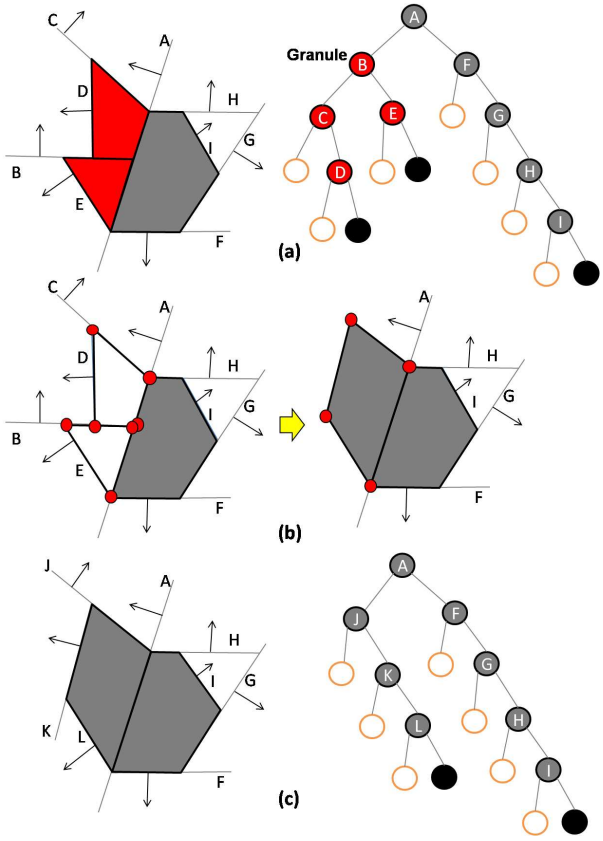

Figure 5: Illustration of the volume bounded solid simplification on a sub-tree: (a) the sub-tree γ under the granule to be simplified, (b) the convex hull C_c is computed from the vertices of the convex regions for every solid leaf-nodes, and (c) the replaced sub-tree spans the convex hull C_c of the granule.

Proposition 2 For a set of non-intersected convex hulls $\{c_i\} \in \Re^3$ where each is bounded by n_i^p planes, the number of planes forming the the convex hull C_c of $\{c_i\}$, n_C^p, has the property of $n_C^p \leq \sum_i n_i^p$.

The proofs of these two propositions are given in *Appendix*.

By these propositions, we can make the following conclusion. For the solid regions $\{H(d_j)\}$ represented by the solid leaf-nodes d_j on the BSP sub-tree γ of a granule, the convex hull of $\{H(d_j)\}$, denoted by c_γ, can be represented by fewer half-spaces than the number of nodes in γ plus the nodes on the way from the granule to the root of the given BSP tree Γ^m. Meanwhile, the space occupied by c_γ bounds the solid regions $\{H(d_j)\}$.

To compute the convex cell C_c of the solid leaf-nodes on the sub-tree γ, we first compute the vertices of the convex hull c_i for each of such node. The convex region c_i defined by a solid leaf-node here is the intersection of all half-spaces on

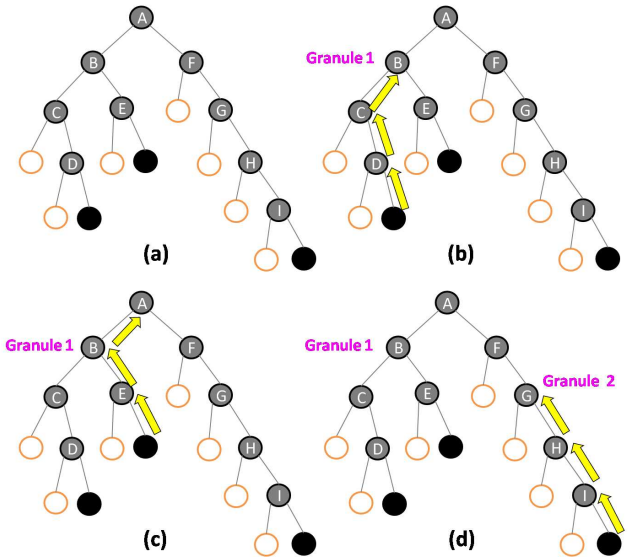

Figure 6: An example of finding candidate granules: (a) the given BSP tree, (b) a granule (no.1) is found by the search starting from the left deepest solid leaf-node, (c) other solid leaf-nodes lower than granule 1 are excluded, and (d) granule 2 is found by another solid leaf-node.

the path from the root of the BSP tree to this leaf-node. The vertices of a convex hull represented by a sequence of half-spaces are computed by the intersection algorithm introduced in [22]. The intersection algorithm requires an interior point in the convex region to compute the coordinates of vertices. To obtain an interior point of the convex hull c_i represented by a set of half-spaces, we first use the linear programming method [23] to compute a point $\mathbf{p}_{\max}$ with the maximum coordinate in $x-$, $y-$ or $z-$axis. Then, a ray not on the boundary of c_i, passing $\mathbf{p}_{\max}$ and intersecting c_i with two distinct points, is searched. The middle point of these two distinct intersection points is served as the interior point to compute the vertices of c_i (ref. [22]). With a given interior point $\mathbf{p}_{int}$, the algorithm translates the half-spaces by $(-\mathbf{p}_{int})$ to make $\mathbf{p}_{int}$ as the origin. After that, the dual polyhedron, which is the convex hull of the points dual to the original planes of the half-spaces, is generated. Finally, the resultant polyhedron for the convex hull that is dual to the dual polyhedron is computed and translated by the vector $\mathbf{p}_{int}$.

After getting the vertices $\{\mathbf{v}_i\}$ of the convex regions c_i defined by the solid leaf-nodes on γ, the convex hull C_c of $\{c_i\}$ can be obtained from $\{\mathbf{v}_i\}$ by the quick-hull algorithm [3]. We then build a new BSP sub-tree γ_c by the planes of faces on C_c to replace the sub-tree γ of a granule. Note that those planes on C_c but coplanar to the planes defined by nodes above the granule in the given BSP tree Γ^0 must be neglected. Figure 5 gives an illustration for the operations in solid convex simplification.

At the beginning of our algorithm, we start searching the granule from the left deepest solid leaf-node. However, inserting the granules from all the solid leaf-nodes into the priority queue is inefficient. Therefore, we conduct a study to find a way to reduce the length of the priority queue but while not affecting the computing results.

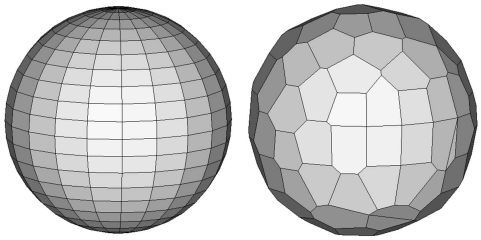

Figure 7: An example of applying the *PlaneCollapse* algorithm to a sphere model (left) with 800 facet planes. By using the threshold $t = 0.04$, we obtain a resultant model with 108 facet planes (right). The resultant model bounds the volume of the given sphere.

Observation 1 For two solid leaf-nodes d_i and d_j located at different levels i and j of a BSP tree Γ^0 with $i > j$ and d_j is on the sub-tree γ_{d_i} of d_i's granule, the convex hulls of their granules satisfy $c_{\gamma(d_i)} \subseteq c_{\gamma(d_j)}$. Therefore, the volume error introduced by $c_{\gamma(d_j)}$ should not be smaller than that of $c_{\gamma(d_i)}$.

To shorten the length of the priority queue, once a granule is found, all solid leaf-nodes under the sub-tree γ of this granule are excluded from the further search of granules. Figure 6 gives an example of such an exclusion.

4. COMPLEXITY SIMPLIFICATION

The convex hull c_γ of the solid regions under the granule's sub-tree γ can be represented by a number of half-spaces which does not exceed the number of nodes below the granule (including the granule itself), and c_γ gives a good shape approximation of these solid regions. However, when intrinsically presenting c_γ by a sequence of half-planes on the sub-tree below γ, the maximum depth $\tau_{\max}$ of the new tree may be greater than that of the given BSP tree Γ^0 (i.e., the complexity is not bounded). A collapse technique is exploited in this section to approximate c_γ by a similar convex hull $\hat{c}_\gamma$ but with fewer half-spaces. The number of half-spaces in $\hat{c}_\gamma$ must be ensured so that $\tau_{\max}$ on the updated tree is not greater than $\tau_{\max}(\Gamma^0)$.

After obtaining the convex hull c_γ, we check whether the number of half-spaces on it makes the new BSP tree's maximum depth exceed $\tau_{\max}(\Gamma^0)$. If it does, the **Algorithm** *PlaneCollapse* is conducted to merge the planes on c_γ that are similar to each other. Here, as the planes are a convex hull, their similarity can simply be measured by the difference between normal vectors. A threshold t is assigned for detecting similar planes. If the merging procedure generates more planes than the maximum number of planes allowed to bound $\tau_{\max}$ on the new tree, we increase the threshold t to further merge the planes until $\tau_{\max}$ is bounded. Each of these new planes generated by merging is translated to the position that all vertices of the convex hull c_γ are not above the plane; therefore, volume bounding can still be preserved.

More than that, only the planes which do not belong to the nodes above the granule on the BSP tree Γ^m are processed by the collapse algorithm to be merged. Pseudo-code of **Algorithm** *PlaneCollapse* is listed below. Figure 7 shows the result of applying this algorithm to a sphere.

Algorithm 2 PlaneCollapse

Require: the threshold t, the list of planes to be collapsed L_I, and the convex hull c_γ
Ensure: the output list of planes L_O

```
while L_I is NOT empty do
  Randomly remove a plane ϱ from L_I;
  Insert ϱ into L_t;
  for all ϖ ∈ L_I do
    if the normals have ||n_ϖ − n_ϱ||^2 < t then
      Remove ϖ from L_I and insert it into L_t;
    end if
  end for
  Calculate the average normal n_avg of all planes in L_t;
  Clear up L_t;
  Find a vertex v ∈ c_γ that maximizes (v · n_avg);
  Determine a new plane ϱ_new by n_avg and v;
  Insert ϱ_new into L_O;
end while
return L_O;
```

5. RESULTS

We have already implemented the proposed algorithm in a C++ program. The examples shown in this paper are all tested on a PC with Intel Core 2 Quad CPU Q6600 2.4GHz.

5.1 Experimental tests and discussion

The results of experimental tests on our algorithm are encouraging. The first example is a spine model with more than 530k nodes on a BSP tree, which is shown in Fig.1. Our solid simplification algorithm can significantly reduce the number of nodes on the BSP tree. The original spine model is guaranteed to be enclosed by the simplified solids, and the maximum depth of the simplified BSP tree is bounded by the maximum depth of the given tree.

The second example is a chair model in Fig.8. Together with the bump model shown in Fig.9, we study the performance of our solid simplification algorithm with and without the plane collapse step. An interesting result we observed is that, when stopping with a similar number of retained nodes, the simplification procedure without the plane collapse step fills up more void spaces. This is because more nodes are reduced after consolidating a granule if the plane collapse step is taken. Therefore, when reaching a similar number of reduced nodes, fewer granules are consolidated, which means that less void volume is filled if the plane collapse step is taken. However, the shape of convex hull c_γ without taking the plane collapse bounds the shape of the given model more tightly than $\hat{c}_\gamma$, the one with fewer nodes on the sub-tree.

Two more examples from biomedical applications are shown in Fig.10 and 11. The computational statistics of our algorithm are listed in Table 1. It shows that our algorithm is very fast when being applied to models with a moderate size. The thresholds chosen in the **Algorithm** *BSPSolidSimplification* are specified by users according to their applications.

Table 1: Computational Statistics

Model	Fig.	Given BSP Tree		Resultant BSP Tree*		
		Nodes	$\tau_{\max}$	Nodes	$\tau_{\max}$	Time
Spine	1	530,975	124	3,787	114	7.31 min.
Chair	8	247,979	140	3,331	129	2.27 min.
Bump	9	202,621	308	4,293	301	2.25 min.
Donna	10	1,520,949	188	6,831	163	39.7 min.
Hand	11	38,535	46	723	43	12.5 sec.
Twirl	14	23,461	35	481	31	5.38 sec.
Venus	15	11,673	79	389	43	3.28 sec.

*Note that we report the full simplification with both the volume and the complexity bounded.

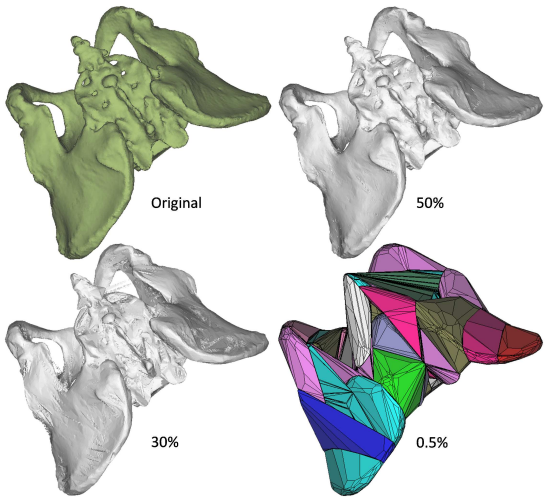

Figure 10: The simplification results of the donna model (the BSP tree has 1,520,949 nodes and $\tau_{\max} = 188$) obtained by retaining different numbers of nodes.

Choosing thresholds with very few nodes and very large allowed volume error eventually makes the results of our algorithm converge to a loose convex hull[1] of the given model (see Fig.12). Our algorithm proposed in this paper can be employed in many applications. Several are demonstrated below.

5.2 Application I: post-processing for Boolean operation

The first application is the post-processing of the BSP tree generated by repeatedly applying Boolean operations to a model. For the BSP tree obtained in such a scenario, there are lots of redundancies retained. For example, the model shown in Fig.13 is produced by simulating shaping operation (using Boolean operation) on a piece of stock material. Although the shape of the resultant model is simple, its corresponding BSP tree generated by the approach in [18] contains 787,683 nodes. To remove the redundancy, we apply our solid simplification approach to this BSP tree with an extremely small volume threshold (e.g., 10^{-8}). The number

[1]When neglecting the plane collapse step, the algorithm converges to the tight convex hull.

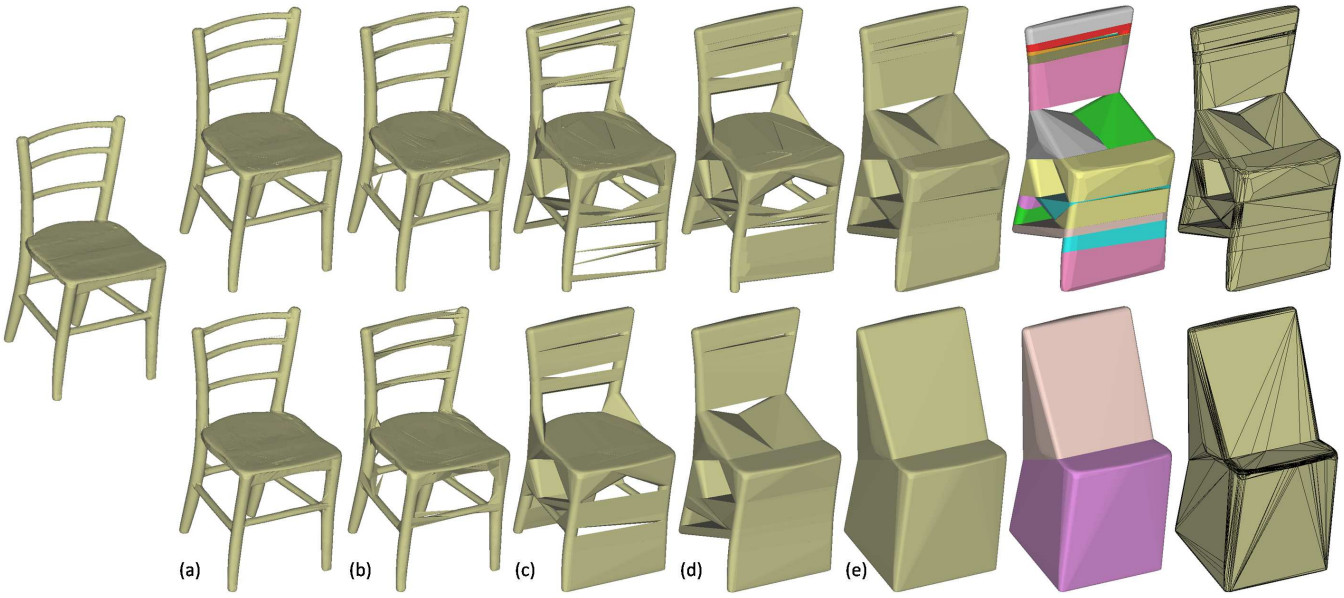

Figure 8: A simplification example of a chair model: (top row) the results obtained from full solid simplification algorithm, and (bottom row) the complexity simplification is neglected. Simplification stops at different numbers of retained nodes compared with the given BSP tree: (a) 50%, (b) 30%, (c) 10%, (d) 5% and (e) 1.5%.

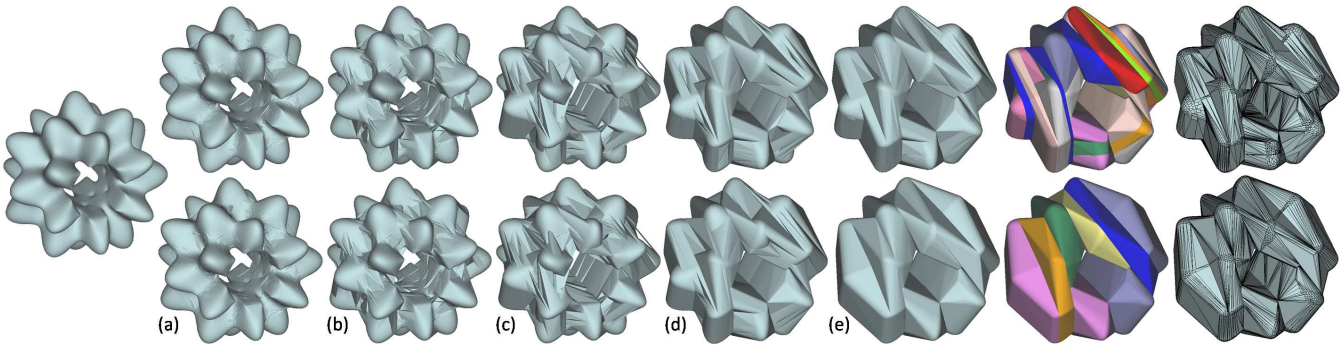

Figure 9: A simplification example of a bump model: (top row) the results obtained from full solid simplification algorithm, and (bottom row) the complexity simplification is neglected. Simplification stops at different numbers of retained nodes compared with the given BSP tree: (a) 70%, (b) 50%, (c) 30%, (d) 10% and (e) 5%.

of nodes on the tree can be reduced from 787,683 to 410,575 in 9.953 seconds.

5.3 Application II: collision detection

The BSP tree representation of solid models has been employed to accelerate collision detection in many applications (ref. [8,15]). A common strategy to speed up collision detection algorithm is to develop some methods to fast exclude those collision-free cases, while not missing any of those possibly collided cases. Meanwhile, there are also some applications, where physical simulation involving collision detection is only conducted on a coarser level of geometry than the shape in visual rendering (e.g., games). Our solid simplification algorithm presented in this paper satisfies the requirements of a coarse collision detection. First, the simplified BSP tree with fewer nodes can be adopted to detect those collision-free cases efficiently. Second, our simplification results tightly bound the volume of the given models, which guarantees that the collided cases will never be missed. In addition, the tight volume bounding could also have more collision-free cases being quickly detected. Moreover, as the complexity of the simplified BSP tree is also bounded, the simplified BSP tree usually gives better speed performance. See the examples of simplified models shown in Fig.14 and 15.

5.4 Application III: rough material preparation

The last application demonstrated in this paper is the generation of rough material prepared for further subtractive machining. In product manufacturing, the final part can only be produced by subtractive machining; thus, the rough material prepared should bound the volume of the final product – our method can ensure this. Although the convex hull of the designed part can always satisfy this requirement, its shape may have a large volume difference compared with

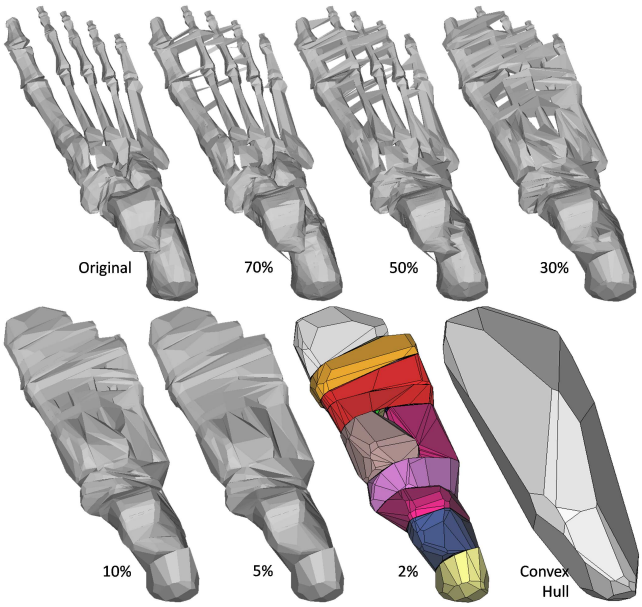

Figure 11: The results of simplifying the hand model (the BSP tree has 38,535 nodes and $\tau_{\max} = 46$) to the BSP trees with different percentages of nodes on the given model.

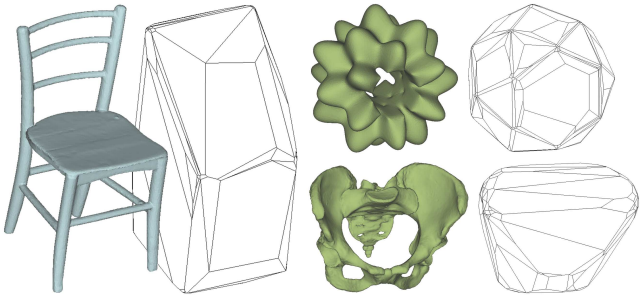

Figure 12: The result of our simplification algorithm converges to a loose convex hull of the given model.

the designed model, which leads to a long machining time. Therefore, some coarser shapes which still bound the given model is wanted (e.g., the model with volume simplified as shown in Fig.16). A more practical strategy for solving this problem may be the feature-based approach like [16]; however, our algorithm provides a general geometric tool for implementing feature-based methods.

6. CONCLUSIONS AND DISCUSSION

In this paper, we present a general solid simplification algorithm that works on solids represented by *Binary Space Partition* (BSP) trees. The volume and the complexity of given models are bounded in our simplification algorithm. Depending on the compact and robust representation of a solid model in BSP-tree, boundary surfaces of the simplified models are guaranteed to be two-manifold and self-intersection free. Two techniques have been investigated in this paper. The volume bounded convex simplification can collapse parts with small volumes on the model into a simple convex volume enclosing the volumetric cells on the input

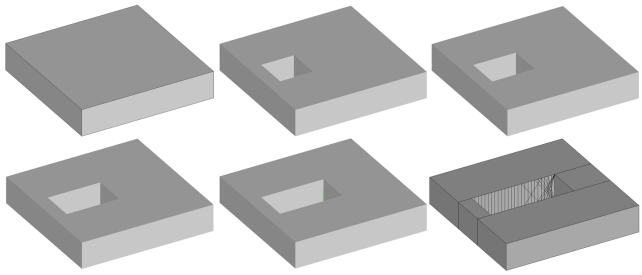

Figure 13: The results of repeatedly applying Boolean operations to a box model. The redundancy on the resultant BSP tree can be removed by our solid simplification algorithm.

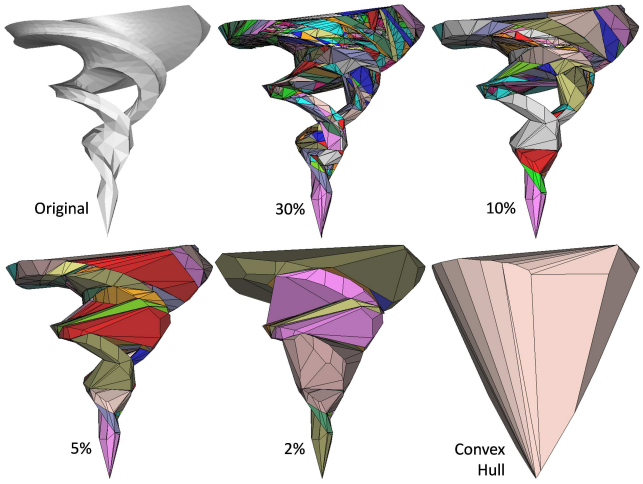

Figure 14: The twirl model (with 23,461 nodes on the BSP tree and $\tau_{\max} = 35$) has been simplified into solids represented by BSP trees with different numbers of nodes compared with the given BSP tree. The convex hull contains 69 half-spaces. The resultant BSP trees form several convex hulls which are displayed in different colors. The maximum depths of the simplified BSP trees are: 28, 28, 28, 31 and 35 respectively.

model. The selection of which region to simplify is based on a volume-difference metric, which minimizes the volume difference between the given model and the simplified one is minimized. The plane collapse approach can reduce the depth of a BSP tree while still preserving volume bounding. The effectiveness of our approach has been proved by several experimental tests.

Two problems of the current approach need to be solved in our future research. First, the plane collapse approach somewhat loose the tightness of volume bound. A *zero* volume error approach is planned to be developed from a combinatorial algorithm. Another alternative is to adopt the Lloyd algorithm based shape approximation like [6]. Second, the shape of a simplified solid does depend on the way the BSP tree is constructed (see the illustration in Fig.4). We plan to develop some feature-based approach to construct BSP trees for applications like feature-based multi-

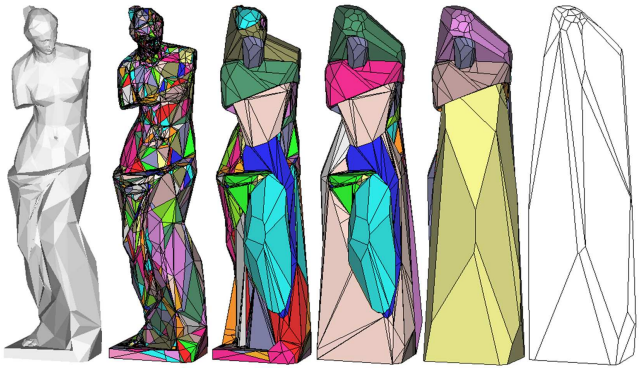

Figure 15: The results of simplifying the Venus model (with 11,673 nodes on the BSP tree and $\tau_{\max} = 79$) into trees with different numbers of nodes (50%, 30%, 10% and 5%) and convex hulls. The last convex hull of the given model has 109 half-spaces only. The maximum depths of the simplified BSP trees are: 68, 41, 72, 43 and 55 respectively.

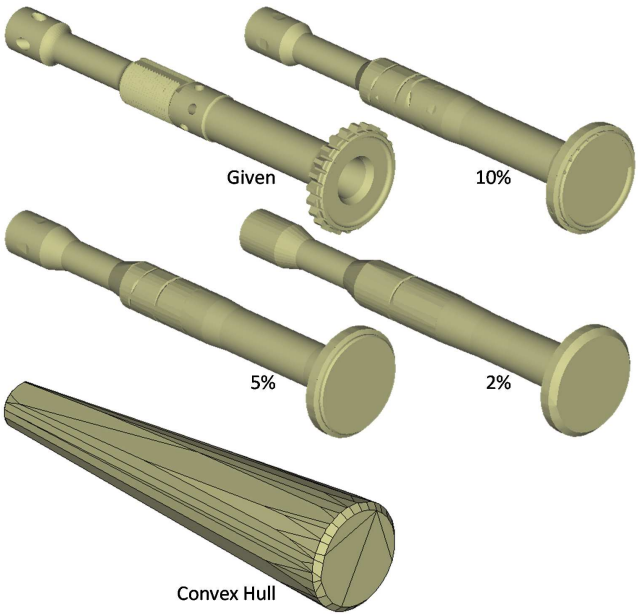

Figure 16: The example application of our solid simplification algorithm – rough material preparation.

resolution solids (ref. [16]) in the near future.

7. ACKNOWLEDGMENTS

The research presented in this paper is partially supported by the Hong Kong Research Grants Council (RGC) General Research Fund (GRF): CUHK/417508 and CUHK/417109.

8. REFERENCES

[1] C. Bajaj, A. Paoluzzi, and G. Scorzelli. Progressive conversion from b-rep to bsp for streaming geometric modeling. *Computer-Aided Design and Applications*, 3(5):577–586, 2006.

[2] C. L. Bajaj and V. Pascucci. Splitting a complex of convex polytopes in any dimension. In *SCG '96: Proceedings of the twelfth annual symposium on Computational geometry*, pages 88–97, New York, NY, USA, 1996. ACM.

[3] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.*, 22(4):469–483, 1996.

[4] G. Bernstein and D. Fussell. Fast, exact, linear booleans. *Computer Graphics Forum*, 28(5):1269–1278, 2009.

[5] M. Campen and L. Kobbelt. Exact and robust (self-)intersections for polygonal meshes. *Computer Graphics Forum*, 29(2), 2010.

[6] D. Cohen-Steiner, P. Alliez, and M. Desbrun. Variational shape approximation. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 905–914, New York, NY, USA, 2004. ACM.

[7] J. Comba and B. Naylor. Conversion of binary space partitioning trees to boundary representation. In *Proceedings of Theory and Practice of Geometric Modeling*, Tuebingen, Germany, 1996.

[8] S. A. Ehmann and M. C. Lin. Accurate and fast proximity queries between polyhedra using convex surface decomposition. *Computer Graphics Forum*, pages 500–510, 2001.

[9] H. Fuchs, Z. M. Kedem, and B. F. Naylor. On visible surface generation by a priori tree structures. In *SIGGRAPH '80: Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, pages 124–133, New York, NY, USA, 1980. ACM.

[10] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 209–216, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[11] N. Hagbi and J. El-Sana. Carving for topology simplification of polygonal meshes. *Comput. Aided Des.*, 42(1):67–75, 2010.

[12] T. He, L. Hong, A. Kaufman, A. Varshney, and S. Wang. Voxel based object simplification. In *VIS '95: Proceedings of the 6th conference on Visualization '95*, page 296, Washington, DC, USA, 1995. IEEE Computer Society.

[13] P. M. Hubbard. Collision detection for interactive graphics applications. *IEEE Transactions on Visualization and Computer Graphics*, 1(3):218–230, 1995.

[14] T. Ju, Q.-Y. Zhou, and S.-M. Hu. Editing the topology of 3d models by sketching. *ACM Trans. Graph.*, 26(3):42, 2007.

[15] J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):21–36, 1998.

[16] S. H. Lee. Feature-based multiresolution modeling of solids. *ACM Trans. Graph.*, 24(4):1417–1441, 2005.

[17] R. G. Luque, a. L. D. Comba, Jo and C. M. D. S. Freitas. Broad-phase collision detection using

semi-adjusting bsp-trees. In *I3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 179–186, New York, NY, USA, 2005. ACM.

[18] M. Lysenko, R. D'Souza, and C.-K. Shene. Improved binary space partition merging. *Computer-Aided Design*, 40(12):1113–1120, 2008.

[19] M. Mortenson. *Geometric Modeling*. Wiley, New York, 1997.

[20] B. Naylor, J. Amanatides, and W. Thibault. Merging bsp trees yields polyhedral set operations. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 115–124, New York, NY, USA, 1990. ACM.

[21] A. Paoluzzi, V. Pascucci, and G. Scorzelli. Progressive dimension-independent boolean operations. In *SM '04: Proceedings of the ninth ACM symposium on Solid modeling and applications*, pages 203–211, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association.

[22] F. Preparata and M. Shamos. *Computational Geometry*. Springer-Verlag, New York, 1985.

[23] R. Seidel. Linear programming and convex hulls made easy. In *SCG '90: Proceedings of the sixth annual symposium on Computational geometry*, pages 211–215, New York, NY, USA, 1990. ACM.

[24] R. Shekhar, E. Fayyad, R. Yagel, and J. F. Cornhill. Octree-based decimation of marching cubes surfaces. In *VIS '96: Proceedings of the 7th conference on Visualization '96*, pages 335–ff., Los Alamitos, CA, USA, 1996. IEEE Computer Society Press.

[25] W. C. Thibault and B. F. Naylor. Set operations on polyhedra using binary space partitioning trees. *SIGGRAPH Comput. Graph.*, 21(4):153–162, 1987.

[26] A. Tokuta. Motion planning using binary space partitioning. In *Proceedings of Intelligent Robots and Systems '91*, pages 86–90. IEEE, 1991.

[27] C. Wang. Topology preserved polygon clipping: a robust algorithm to convert BSP tree to B-rep. *to appear*, 2010.

[28] Z. Wood, H. Hoppe, M. Desbrun, and P. Schröder. Removing excess topology from isosurfaces. *ACM Trans. Graph.*, 23(2):190–208, 2004.

[29] Z. Yin. Rough and finish tool-path generation for nc machining of freeform surfaces based on a multiresolution method. *Computer-Aided Design*, 36(12):1231–1239, 2004.

[30] Q.-Y. Zhou, T. Ju, and S.-M. Hu. Topology repair of solid models using skeletons. *IEEE Transactions on Visualization and Computer Graphics*, 13(4):675–685, 2007.

APPENDIX

A. PROOF OF PROPOSITIONS

Proposition 1 For a set of convex hulls $\{c_i\} \in \Re^3$, the vertices of these convex hulls are $\{\mathbf{v_i}\}$, the convex hull C_v of these vertices is coincident with the convex hull C_c of these convex regions.

PROOF. Without loss of generality, let s_i represent the point set of the convex hull c_i and M_i denote the polyhedron of c_i. C_c is the convex hull of all these point sets $\{s_i\}$. Here, we actually need to prove that all the vertices on C_c come from the vertices of the polyhedra $\{M_i\}$, which can be proved by using reduction of absurdity.

Assume that, for one of these point sets, s_r, there is an interior point $\mathbf{p}$ of M_r (also s_r) which becomes a vertex on the convex hull C_c. Based on the property of convex hull, we have

$$\mathbf{p} = \sum_{k=1}^{n_r} \alpha_k \mathbf{v}_k^r,$$

where $\mathbf{v}_k^r$ is the k-th vertex of M_r, n_r is the number of vertices of M_r, and $\sum_{k=1}^{n_r} \alpha_k = 1$.

For a vertex $\mathbf{v}_k^r \in M_r$, it is either a vertex on C_c or an interior point of C_c. When it is an interior point, we have

$$\mathbf{v}_k^r = \beta_m \mathbf{p} + \sum_{j=1}^{m-1} \beta_j \mathbf{v}_j^c,$$

where $\mathbf{v}_j^c$ is the j-th vertex of C_c, m is the number of vertices on C_c, and $\sum_{j=1}^{m} \beta_j = 1$. Note that here $\mathbf{p}$ is one of the vertices on C_c based on the assumption stated before. When the vertex $\mathbf{v}_r$ is a vertex on C_c, we have

$$\mathbf{v}_k^r = \mathbf{v}_j^c.$$

Based on these formulas, we could have

$$\mathbf{p} = \mu_m \mathbf{p} + \sum_{j=1}^{m-1} \mu_j \mathbf{v}_j^c, \quad \Rightarrow \quad \mathbf{p} = \sum_{j=1}^{m-1} \frac{\mu_j}{1-\mu_m} \mathbf{v}_j^c.$$

This means that $\mathbf{p}$ can be written as a linear combination of the remaining $m-1$ vertices of C_c. This conflicts with the basic property of linear independency among all the vertices of a convex hull. Therefore, the assumption we made at the very beginning of this proof is wrong. We can thus prove that all the vertices of C_c come from the vertices of $\{M_i\}$. □

Proposition 2 For a set of non-intersected convex hulls $\{c_i\} \in \Re^3$ where each is bounded by n_i^p planes, the number of planes forming the convex hull C_c of $\{c_i\}$, n_C^p, has the property that $n_C^p \leq \sum_i n_i^p$.

PROOF. By the Euler-Poincaré formula (ref. [19]), for a 3D convex hull with n_i^v vertices, it has $n_i^p = 2n_i^v$ faces. From the property of convex hull, we know that the number of vertices on the convex hull of $\{c_i\}$, is smaller than the total number of vertices on the non-intersected convex hulls $\{c_i\}$ in most cases, and equals to the total number in some extreme cases. As the relationship between the numbers of vertices and faces $n_i^p = 2n_i^v$ is always kept on the convex hull (the model with genus number zero), this proposition is proved. □