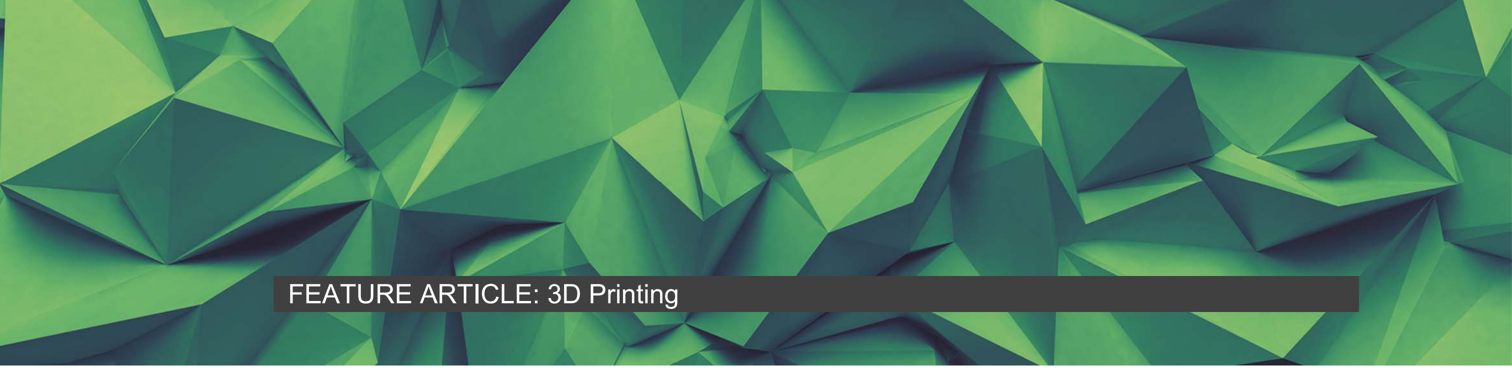


Interactive Partitioning of 3D Models into Printable Parts

Aamir Khan Jadoon
Tsinghua University

Chenming Wu
Tsinghua University

Yong-Jin Liu
Tsinghua University

Ying He
Nanyang Technological University

Charlie C.L. Wang
Delft University of Technology

This article presents an easy, flexible and interactive tool for partitioning a 3D model, which is larger than 3D-printer's working volume, into printable parts in an intuitive way. Our tool is based on the elegant partitioning optimization framework *Chopper*. Our tool aims at improving Chopper by providing users three easy-to-use interactive operations: no-go region painting, cutting plane specification and components reunion. With these operations, we show that (1) exhaustive search in the BSP tree—the most time-consuming step in Chopper—can be avoided, (2) more flexible geometric configurations can be provided, (3) user's design intention is considered naturally and efficiently, and customized 3D partitioning results can be obtained. We test our tool on a wide range of 3D models and observe promising results. A preliminary user study also demonstrates its effectiveness and efficiency.

3D printing, also known as additive manufacturing, has been widely studied in the manufacturing industry. As desktop 3D printers have become mature, more and more people enjoy using this awesome invention to print their daily life 3D models. Desktop 3D printers usually have the limited size of a 3D model at which one can print. This limitation drives the research of 3D-printing-based model partitioning. The desired solution involves partitioning the large 3D model into small parts that can fit into the working volume of the printer. Then these small parts can easily be assembled with the help of connectors, screws, glue or self-interlocking.

There exist a number of 3D-printing-driven model partitioning methods, by considering diverse properties such as printing volume,[1,2] structureness and assemblability,[3] packing and space saving, aesthetics, self-support and connectivity,[4,5] and so on. A representative work is the Chopper,[1] which is an elegant optimization framework that incorporates many partitioning properties

Published by the IEEE Computer Society
0272-1716/18/$33.00 ©2018 IEEE

into an objective function. Chopper is fully automatic. However, its partitioning results heavily depend on the weights in the objective function and users frequently have to adjust these weights by trial and error. For each trial, Chopper is also time-consuming due to the exhaustive searching in a binary space partitioning (BSP) tree. See Figures 1 and 8 for some examples. Furthermore, as fully automatic methods, all of the above methods do not take users' design intentions into account and therefore cannot produce customized partitioning results.

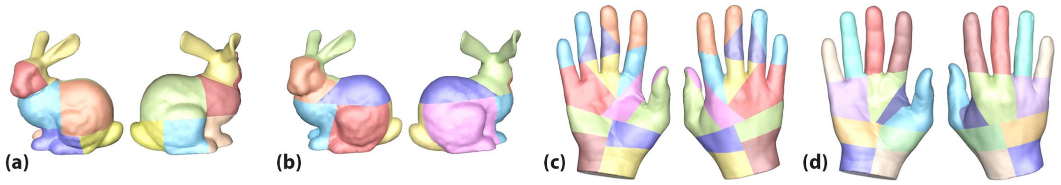


Figure 1. The output of Chopper is sensitive to the weights of the objective function. The default setting (Equation 2) produces good partitioning on the Bunny model, however, it leads to poor results on the Hand model, where all fingers are over-segmented. Our method is not sensitive to different weights. With a few simple interactions from users, it can provide good partitioning quickly. (a) Chopper with default weights : 10 parts, 287 sec. (b) Our method: 7 parts, 119 sec., including interaction. (c) Chopper with default weights: 16 parts, 375 sec. (d) Our method: 12 parts, 173 sec., including interaction.

In this paper, we propose an interactive tool, aiming at improving Chopper by three algorithms based on BSP tree generation and manipulation: 1) constrained tree generation, 2) tree-splitting and 3) sub-tree merging, each of which leads to a simple and intuitive interactive operation, namely, no-go region specification, clipping plane specification and component reunion. With little user interactions, we show that exhaustive search in the BSP tree—the most time-consuming step in Chopper—can be avoided. As a result, our method enables real-time interaction and updating the partition of 3D models with complex geometry and topology. Moreover, by enabling customized partitioning, it can produce more flexible geometric configurations that are not available to Chopper (see Figure 2).

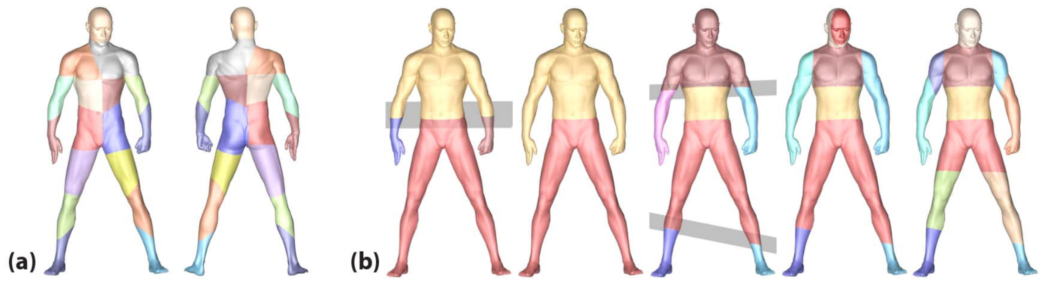


Figure 2. Our method can provide more flexible, better partitioning results than Chopper.[1] The size of the minimum enclosing bounding box of the human model is 21cm × 45cm × 7cm, whereas the 3D printer's working volume is only 18cm × 17cm × 16cm. (a) the front and back views of Chopper's partitioning results with 17 parts, taking more than three minutes in an automatic setting. (b) Leftmost: with our method, users start with specifying a cutting plane, which leads to three disjoint closed cutting curves on the input model. Middle-left: our method merges the undesired parts using the components reunion operator. Middle: partitioning with two more cutting planes. Middle-right: painting the salient region (red region in the face) where no cuts can go through. Rightmost: with the constrained tree generation algorithm, our method produces only 12 parts in less than one minute (including interaction and running the constrained partitioning algorithm).

Our interactive tool is easy to use. Instead of the full manual partitioning by applying existing CAD modeling/CSG tools, our tool works in either automatic or interactive mode, or a mix of both. For example, our tool can make use of an initial partitioning automatically generated from Chopper, and then users can simply and quickly modify a few cutting planes (leafs in the BSP tree) in the semi-automatic mode. Thanks to the high interaction performance, our tool can update the partitioning result immediately. Users can repeat this procedure until obtaining a satisfactory result. We tested our interactive tool on a wide range of 3D models and observed promising results. A preliminary user study also demonstrates its effectiveness and efficiency.

RELATED WORK

As an active field, 3D segmentation has been studied extensively in the last decade. Many methods have been proposed to segment a 3D model into meaningful parts, providing semantic information about the model for shape understanding and recognition.[6,11,12] However, most existing methods are not suitable for 3D printing, since they do not address some critical issues, such as printing time, supporting material, printer volume, assemblability, structure soundness, aesthetics, etc.

Given a limited printing volume, one has to decompose a large 3D model into smaller printable subparts. Decomposing polyhedra into convex parts or even approximately is a well-known challenging problem in computational geometry. In manufacturing industry, Medellin and colleagues[7] proposed a regular-grid-based decomposition designed for both rapid prototyping and assembly. A drawback of this method is that it always adds but never subtracts 3D units to solve potential manufacturing problems and then it leads to too large 3D units. Hao and colleagues[8] decomposed a large complex model into simpler 3D parts by using curvature-based partitioning.

A pyramid is a shape that has a flat base with the remaining boundary forming a height function over the base. Pyramidal shapes are optimal for 3D printing since they do not need any supporting structures. Hu and colleagues[10] proposed an elegant method for approximate pyramidal shape decomposition using a clustering scheme. Luo and colleagues[1] proposed the Chopper, which optimizes a number of desired 3D printing criteria in an elegant framework. In addition to considering a single decomposed subpart that fits the printer volume, some researchers had considered the problem of packing multiple parts into the printing volume. Recently, Chen and colleagues[4] and Yao and colleagues[5] proposed optimization methods to address both the segmentation and packing issues.

To the best of our knowledge, none of the existing methods incorporate users' inputs into model partitioning to reflect users' special intention. The interactive tool proposed in this paper aims at addressing this issue.

PRELIMINARY ON CHOPPER

Chopper[1] partitions a 3D model using a set of cutting planes, which are determined by an exhaustive search of all possible combinations of candidate planes and evaluated by an objective function. It adopts a BSP tree to represent the set of cutting planes.

Candidate planes. A plane is represented by a 2-tuple $\pi_{ij} = (\mathbf{n}_i, d_j)$, where $\mathbf{n}_i$ is the plane normal and d_j is the offset. Chopper chooses a planar normal from a set $N = \{\mathbf{n}_i\}$, which are 129 uniformly distributed points on a unit sphere, determined by the vertices of a thrice-subdivided regular octahedron. The plane offsets $\{d_j\}$ are sampled uniformly at intervals of 0.5 cm over the range within which the planes intersect the model, for each $\mathbf{n}_i$.

Objective functions. For a BSP tree τ, Chopper defines six objective functions for the number of parts f_{part}, efficient utilization of the printing volume f_{util}, connector feasibility $f_{connector}$, structural soundness by finite element analysis $f_{structure}$ and by avoidance to introducing fragile parts $f_{fragility}$, aesthetics by seam unobtrusiveness f_{seam} and by symmetry $f_{symmetry}$. Then it minimizes the combined objective function

$$f(\tau) = \sum_k w_k f_k(\tau) \qquad (1)$$

where w_k are non-negative weights. Chopper is fully automatic. However, its results heavily depend on the weights. Luo and colleagues[1] suggested the following weights:

$$\mathrm{w}_{\mathrm{part}} = 1, w_{util} = 0.05, w_{connector} = 1, w_{fragility} = 1, w_{seam} = 0.1, w_{symmetry} = 0.25 \qquad (2)$$

Beam search. Chopper uses a beam search to partition the model. Starting with an empty BSP tree, at each step, Chopper extends the BSP trees from the previous step by adding one more cutting plane and the beam search selects the most promising BSP trees (evaluated by Equation 1) so far. The maximum number of BSP selections at each step is the beam width *b* and Chopper uses *b* = 4. The search terminates when all leafs in BSP trees can be fit in the target printing volume. The best BSP tree (having the minimal score in Equation 1) is the solution.

OUR INTERACTIVE METHOD

We introduce three simple, intuitive interaction operations into Chopper. With them, a user can indicate some constraints, specify a particular cutting plane and merge two subparts. To incorporate users' input constraints in an interactive environment, we present a simple yet efficient constrained tree generation algorithm to quickly update the BSP trees. Given a particular cutting plane specified by a user according to some design intention, the BSP tree maintained by the system is quickly updated by a tree-splitting algorithm. The planar cuts indicated by a BSP tree can only achieve some limited geometric configurations. We provide an interactive component reunion operation with a sub-tree merging algorithm so that more flexible geometric configurations can be obtained. Experiments and the user study show that a user can use our interactive tool to obtain customized, user satisfied 3D printable partitioning in a short time.

Throughout this paper, we use the terms of the leaf node in a BSP tree and its corresponding subpart in a 3D model interchangeably.

Three interactive operations

To allow a user to input his/her design intention simply and intuitively, we propose three interactive operations as follows.

No-go region painting with the aid of stress analysis. In a preprocessing step, our method analyzes the stress distribution of the model using the finite element method. During interaction, the stress analysis is visualized with a color map and a user can paint on the object surface with strokes. A scribble-like interface is used for painting since it is intuitive and requires no expertise. Our method computes the minimum bounding boxes of each disjoint painted region if its volume is less than the printer volume. Cutting planes on the model surface are not allowed to pass through the painted regions and we call them *no-go regions*. Two design intentions can be reflected by no-go regions (see Figure 3a): the first is the aesthetically salient regions that can be user customized; and the second is the structurally fragile regions. This interactive operation is realized by Algorithm 1.

Noting that Chopper[1] also allows the user to specify regions to avoid cutting through, our method has two major differences on no-go region specification. First, our method treats three interactive operations as an integrated framework in an interactive system. Second, Chopper computes a penalty for running the seam through painted regions and it evaluates the cost of the seam on a cut as the normalized integral of penalty along the seam. Due to the other objective functions, unobtrusiveness objective function weight and painted region dimensions, Chopper cannot strictly guarantee that painted regions would always be avoided by cuts.

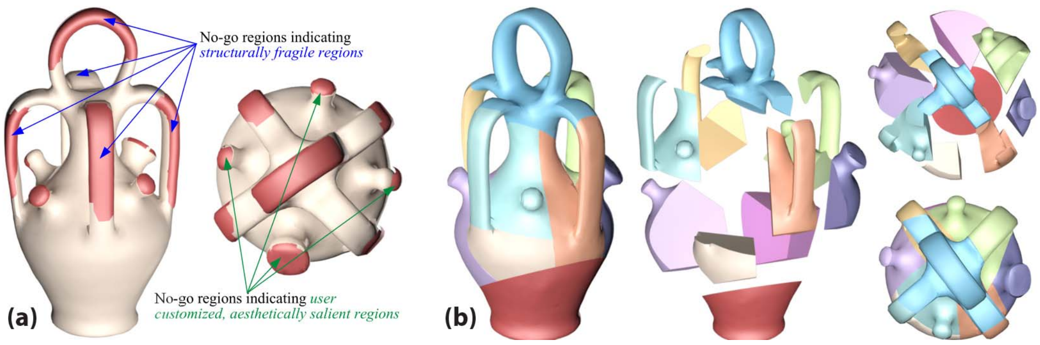

No-go regions indicating
structurally fragile regions
No-go regions indicating user customized, aesthetically salient regions
(a)
(b)

Figure 3. (a) Two types of no-go regions: one is for user-customized, aesthetically salient regions and the other is for structurally fragile regions. (b) The result in our method by running Algorithm 1 with constraints of no-go regions. The minimum enclosing bounding box of the whole model is 19.65cm × 39.69cm × 19.26cm and the printer volume is 18cm × 18cm × 18cm. The number of parts in the final partitioning result is 10.

Cutting plane specification. A user can simply draw free-hand strokes to partition the model at the desired locations. Our method fits a cutting plane to the set of points in the stroke by principal component analysis (PCA) and performs the cut in real-time interaction. During the interactive partitioning, our method only considers the visible parts (the user has an option on the GUI to show/hide any sub-parts). If the fitted plane intersects any visible part, our method partitions that part and add sub-parts to the BSP tree. See Figure 4 for an example. This interactive operation is realized by Algorithm 2.

```
Input: An unprintable BSP tree $\tau$, the set of no-go regions $R$, the set of candidate planes $P$ and the
       beam width $b$
Output: A constrained optimal BSP Tree, with which cutting planes do not pass through any no-go
       regions and the volume of each leaf node is less than the printer volume
foreach no-go region $r \in R$ do
    if there is a cutting plane passing through $r$ then
        Merging two leaf nodes in $\tau$ whose regions share the boundary formed by the cutting line;
        Update the tree $\tau$ using Algorithm 3;
    end
end
$currentBSP \leftarrow \tau$;
while all the trees in currentBSP are unprintable do
    $newBSPs \leftarrow \tau$;
    foreach tree $\tau' \in currentBSP$ do
        $currentBSP = currentBSP \setminus \tau'$;
        foreach plane $p \in P$ that does not pass through any no-go regions do
            foreach non-printable leaf node $l \in \tau'$ do
                if $p$ intersects $l$ then
                    Cut $l$ using $p$ and put the new BSP tree into newBSPs;
                end
            end
        end
    end
    Put the top $b - 1$ ranked BSP trees in newBSPs (evaluated by the Eq. (1)) into currentBSP;
    Randomly choose one among the rest trees in newBSPs and add it into currentBSP;
end
Return the printable BSP tree with the highest rank in currentBSP;
```

Algorithm 1. Constrained tree generation.

Component reunion. Planar cuts by unbounded planes can sometime prevent desired partitions. For example, any cutting plane partitions the neck of the Kitty model will inevitably cut the middle of the tail part simultaneously. Component reunion is an operation in which a user can merge two adjacent parts into one part. Together with the cutting plane specification, flexible geometric configurations can be obtained. For example, if a user wants to cut the Kitten model into three parts (head, body and tail), he/she can first cut the head from its body, then cut the tail from the head and the body, and finally reunite the two sub-parts of the tails together (Figure 6). This interactive operation is realized by Algorithm 3.

Constrained tree generation

Our method works either in an automatic mode or in a semi-automatic mode. A BSP tree is called *unprintable* if either (1) it is empty or (2) there exists one leaf node whose volume is larger than the printer volume or there exist a cutting plane passing through no-go regions; otherwise it is *printable*. In semi-automatic mode, a user can specify a few constraints and we propose an algorithm (Algorithm 1) that outputs a constrained optimized printable BSP tree compatible with no-go region constraints. The other constraints are handled in Algorithms 2 and 3.

Algorithm 1 is a constrained tree generation algorithm, whose input can be either empty (for automatic mode) or an unprintable BSP tree with no-go regions (for semi-automatic mode). Note that the input unprintable BSP tree can result from some interactive operations. Given an initial unprintable BSP tree, Algorithm 1 first examines whether each cutting plane in this tree intersects no-go regions or not. If a cutting plane violates this nonintersection constraint, it is removed from the tree by merging two corresponding sub-trees. We examine all possible situations of sub-tree merging in a BSP tree and classify them into three configurations. We present these configurations and the sub-tree merging algorithms later in this article Secondly, Algorithm 1 uses a constrained beam search to partition the non-printable leaf nodes in a pool of BSP trees using cutting planes. The set of candidate planes P are the same as those in Chopper. The search space P is further constrained by no-go regions.

Algorithm 1 has a linear complexity $O(b|\tau||R||P|)$, where $|\tau|$ is the number of leaf nodes in τ, $|R|$ and $|P|$ are the cardinality of R and P respectively. There are three key differences between our constrained tree generation and the Chopper's beam search. The first is that our method merges the neighboring leaf nodes (i.e., neighboring subparts) whose boundary (i.e., the cutting line) violate the no-go regions and maintain a value binary tree by local updating. The second is that the no-go regions help to quickly filter a large number of unnecessary candidate planes in P before performing the time-consuming intersection and cutting operations. The third is that rather than choosing all top b ranked BSP trees in *newBSPs*, we randomly pick up a BSP tree into *newBSPs*. We observe that this operation improves the performance of local beam search in Chopper that frequently suffers from a lack of diversity, since all the top b ranked BSP trees can quickly become concentrated in a narrow region in the search space. The latter two differences Algorithm 1 performs in real time for user interaction in our practice.

BSP tree splitting with user-specified cutting planes

It is often desired by users to partition a model at some places that have functional or semantic meaning. Sometimes these places (e.g., the waist and arms of the Armadillo model in Figure 4 and the human model in Figure 2b) are error-prone by automatically detection using low-level geometric features. In our method, a user can interactively specify a cutting plane by drawing a free-hand stroke. It is worth noting that a user does not need to use both no-go regions and cutting plane specifications. If a user has some experiences to specify good initial cutting planes, no-go regions are no longer needed. On the other hand, if a novice user only has some fuzzy requirements on salient regions to be preserved, she/he can use no-go regions alone in the semi-automatic mode for partitioning.

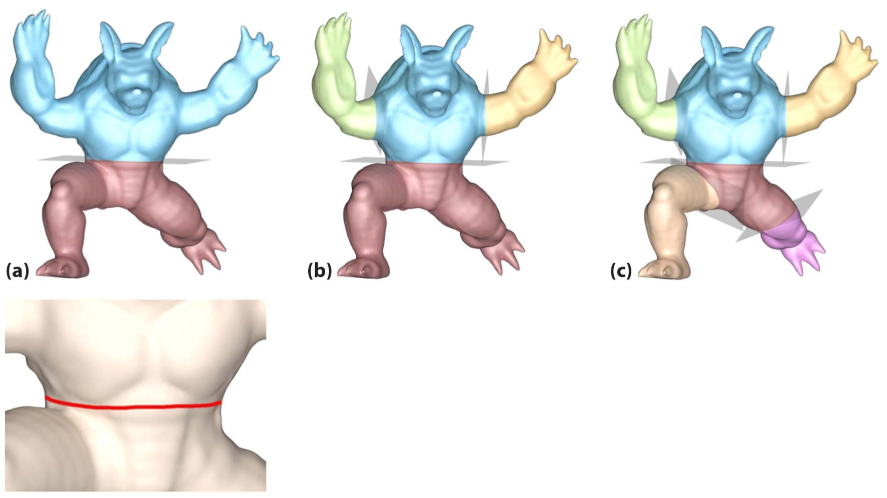


Figure 4. Cutting the waist and arms of the Armadillo model to obtain a clear semantic segmentation. The final result in our method is obtained by running the BSP tree splitting with a user-specified cutting plane, as in Algorithm 2. (a) Cutting at the waist of the Armadillo model. (b) More cutting at arms. (c) Final result.

For specifying a cutting plane, a user can rotate the model until a best viewpoint is reached. By fixing this viewpoint, the user draw a freehand stroke and a straight line is fit with stroke points by a least-squares method. A cutting plane is determined which passes through the fitting line and is perpendicular to the screen plane. Then the BSP tree τ maintained by our method is updated by adding this cutting plane. The pseudo-codes are presented in Algorithm 2. Note that the input BSP tree in Algorithm 2 may be empty and the model partitioning by a single plane may lead to an unprintable BSP tree. In this case, Algorithm 1 (with an empty set of no-go regions) is applied to obtain a printable model partitioning.

```
Input: A BSP tree $\tau$ and a point array of free-hand stoke
Output: An updated printable BSP Tree, whose partition contains a boundary formed by the
        user-specified cutting plane $p$
In the screen plane, fit a line $\pi$ to the point array by a least-squares method;
Build a cutting plane $p$ which passes through $\pi$ and is perpendicular to the screen plane;
$currentBSP \leftarrow \varnothing$ ;
foreach leaf node $l$ in $\tau$ do
    if $l$ is visible and $p$ intersects $l$ then
        Cut $l$ using $p$;
        if the resulted tree $\tilde{\tau}$ is unprintable then
            Update $\tilde{\tau}$ into a printable one using Algorithm 1 with an empty set of no-go regions;
        end
        $currentBSP \leftarrow \tilde{\tau}$ ;
    end
end
$newBSPs \leftarrow \varnothing$ ;
while currentBSP is not empty do
    $newBSPs \leftarrow \tau$;
    foreach tree $\tau' \in currentBSP$ do
        $currentBSP = currentBSP \setminus \tau'$ ;
        $newBSPs = newBSPs \cup \tau'$ ;
        Find all mergeable nodes in $\tau'$ ;
        if there exist any mergeable nodes then
            Merge mergeable nodes and update the tree $\tau'$ into $\tau''$ using Algorithm 3 in Section 4.4;
            $currentBSP = currentBSP \cup \tau''$ ;
        end
    end
end
Evaluate all the trees in newBSPs and return the highest ranked one;
```

Algorithm 2. BSP tree splitting with a user-specified cutting plane.

The user-specified plane may also cut a printable leaf node in the input BSP tree and sometimes result in very small subparts. We further optimize the BSP tree by considering possible merge of leaf nodes. Two leaf nodes in a BSP tree are called *mergeable* if (1) they share a common boundary cutting plane not specified by users and (2) the merged node has a printable volume. Then we recursively merge all mergeable nodes and for merging each pair of mergeable nodes a new BSP tree is generated by Algorithm 3 in Section 4.4. Finally. we chose the highest ranked BSP tree from the pool of newly generated BSP trees.

A linear scan of all leaf nodes in a BSP tree τ can identify all the mergeable nodes. The leaf nodes of a tree built from merged nodes can be further merged. Let n be the number of leaf nodes in τ. There are at most O(log$n$) hierarchical mergeable nodes. Then Algorithm 2 has the worst-case time complexity of O($n$log$n$). In all our experimental tests, Algorithm 2 runs in real time for user interaction by observing that there are only O(1) hierarchical mergeable nodes in real models, leading to a linear empirical time complexity.

Flexible geometric configurations by component reunion operations

If a 3D model has complex geometry or has a complex topology of high genus, a cutting plane may intersect the 3D model with more than one disjoint closed curve (e.g., the cutting at the waist of the human model in Figure 2(b)). Then some desired partitioning may not be accessible by a BSP tree built from the top-down manner in Chopper in which two parts are separated by a cutting plane at each step. One more example is shown in Figure 5, in which a cutting plane at the neck of the Kitten model will inevitably cut the tail into two parts and Chopper can never recover the whole tail part.

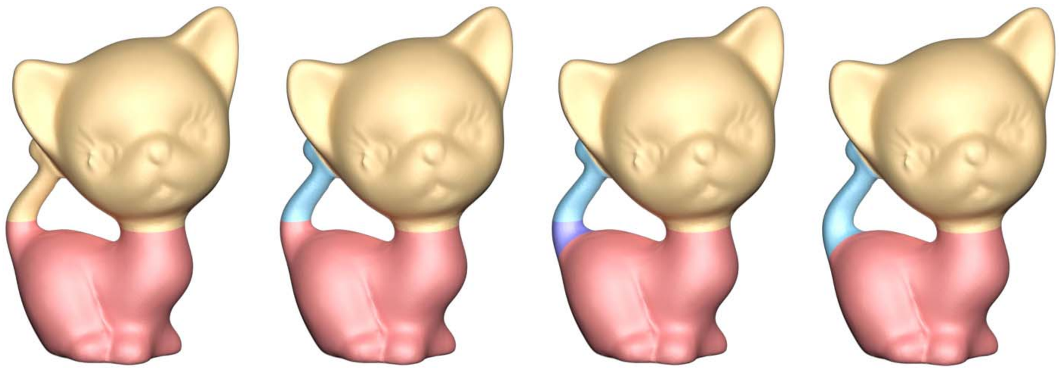


Figure 5. With the component reunion function, our method can provide the partitioning geometric configuration containing the whole tail part. (a) A cutting plane at the neck of the Kitten model will inevitably cut the tail into two parts. Chopper can never recover the whole tail part. (b) A cutting plane separates the head and tail. (c) A cutting plane separates the body and tail. (d) Reuniting the two parts of the tail in our method.

Our method supports to merge two neighboring subparts into one by a simple components reunion operation (Figure 2d). The core to this component reunion operation is a merging algorithm. A key observation is that to merge two parts (represented by leafs in the BSP tree) which are physically adjacent, there are only three configurations in BSP tree:

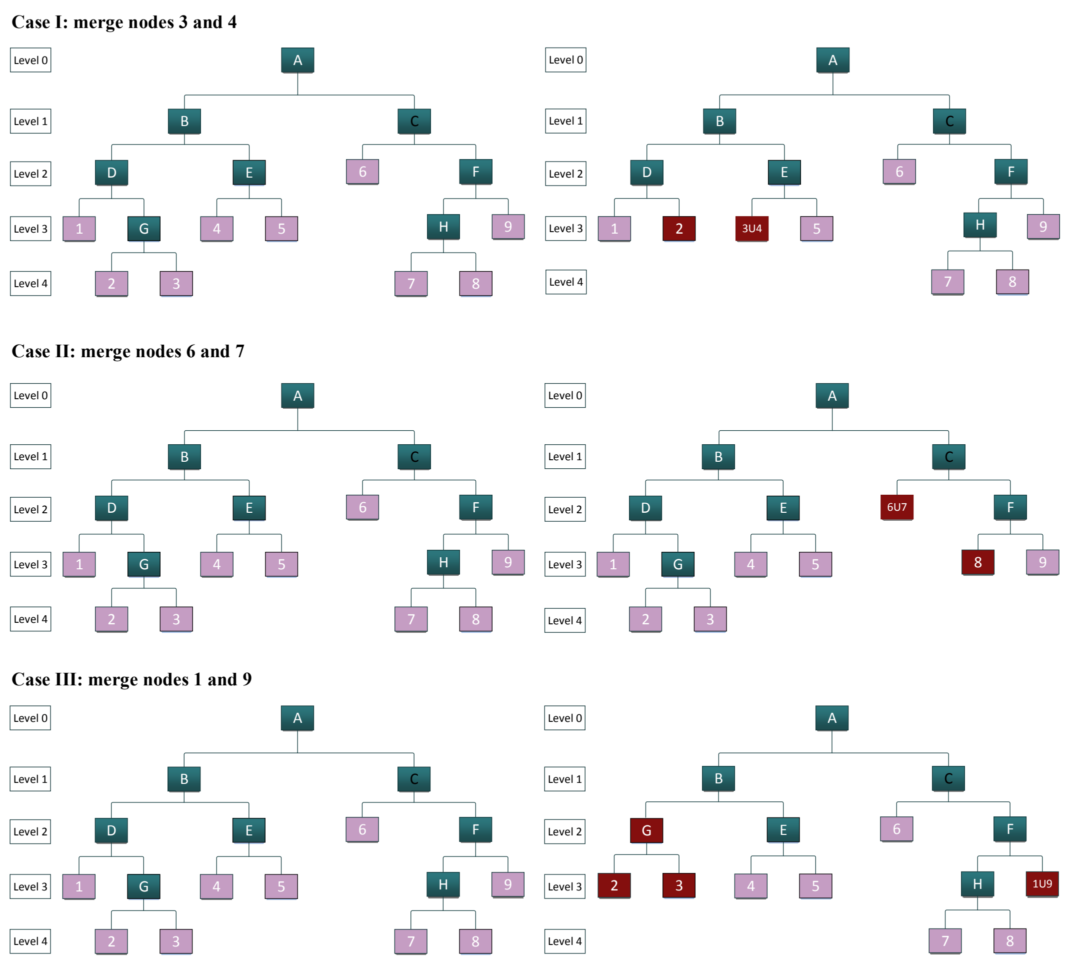

Case I: merge nodes 3 and 4
Case II: merge nodes 6 and 7
Case III: merge nodes 1 and 9
Level 0
Level 1
Level 2
Level 3
Level 4
A
B
C
D
E
F
G
H
1
2
3
4
5
6
7
8
9
3U4
6U7
1U9

Figure 6. There are three different configurations in Algorithm 3. Case I: Siblings of both nodes to be merged are leaf nodes. Case II: The sibling of one node to be merged is a leaf node and the sibling of the other is a non-leaf node. Case III: Siblings of both nodes to be merged are non-leaf nodes.

- Case I: Siblings of both nodes to be merged are leaf nodes (Figure 6, Case I).
- Case II: The sibling of one node to be merged is a leaf node and the sibling of the other is a non-leaf node (Figure 6, Case II).
- Case III: Siblings of both nodes to be merged are non-leaf nodes (Figure 6, Case II).

The detailed operations for merging two leaf-nodes in each of these three cases include 1) assigning one leaf-node to another leaf-node's parent and 2) reducing the sub-tree of the node that has been merged into others. During the merging, to simplify the implementation, we prefer to merge the leaf-node the sibling of which is a leaf-node into the one whose sibling is a non-leaf-node.

```
Input: A BSP tree $\tau$ and two nodes (which share a common boundary) to be merged
Output: The updated binary tree after merging
Denote the two nodes to be merged as $i$ and $j$, where $i$ is visited before $j$ in the depth-first traversal of $\tau$.
if siblings of both nodes $i$ and $j$ are leaf nodes then
    $j \leftarrow i \cup j$;
    Move the sibling of $i$ to the parent of $i$ (Figure 6 Case I);
else if the sibling of one node to be merged is a leaf node and the sibling of the other is a non-leaf node
then
    Let $m$ be the node to be merged (one of $i$ and $j$) whose sibling is a non-leaf node;
    Let $n$ be the node to be merged whose sibling is a leaf node;
    $m \leftarrow m \cup n$;
    Move the sibling of $n$ to the parent of $n$ (Figre 6 Case II);
else
    $j \leftarrow i \cup j$;
    Set the parent of $i$ to be the union of the parent of $i$ and the sibling of $i$ (Figure 7 Case III);
end
```

Algorithm 3. Sub-tree merging.

The pseudo-codes of components reunion operation are presented in Algorithm 3. Since there are only three cases in merging two neighboring subparts, it is easy to check that Algorithm 3 guarantees to output a binary tree with the time complexity O(1). Accordingly, we can always traverse the resulted tree recursively in depth-first fashion, which plays an important role in Algorithms 1 and 2. It is worth noting that, although the merged tree is always binary, merging deviate resulted binary tree from the basic definition of BSP tree in which convexity exists by nature and that satisfies the assemblability objective by its construction. To deal with possible assemblability issue, we analyzed all printable pieces on the Gaussian map for assemblability by the common regions of half-spaces according to the motion restrictions formulated in [9]. If merging two neighboring subparts results in a binary tree that cannot pass the assemblability check, we reject this merging operation and try another merging. In terms of implementation, our merging operation needs store multiple, disjoint cutting plane segments at some nodes in the merged tree. Fortunately, the Chopper's objective function (1) is still applicable, given that all leaf nodes pass the assemblability check.

As a short summary, the components reunion operation in our method provides more flexible geometric configurations than those in Chopper. Two examples are shown in Figure 2b and Figure 5.

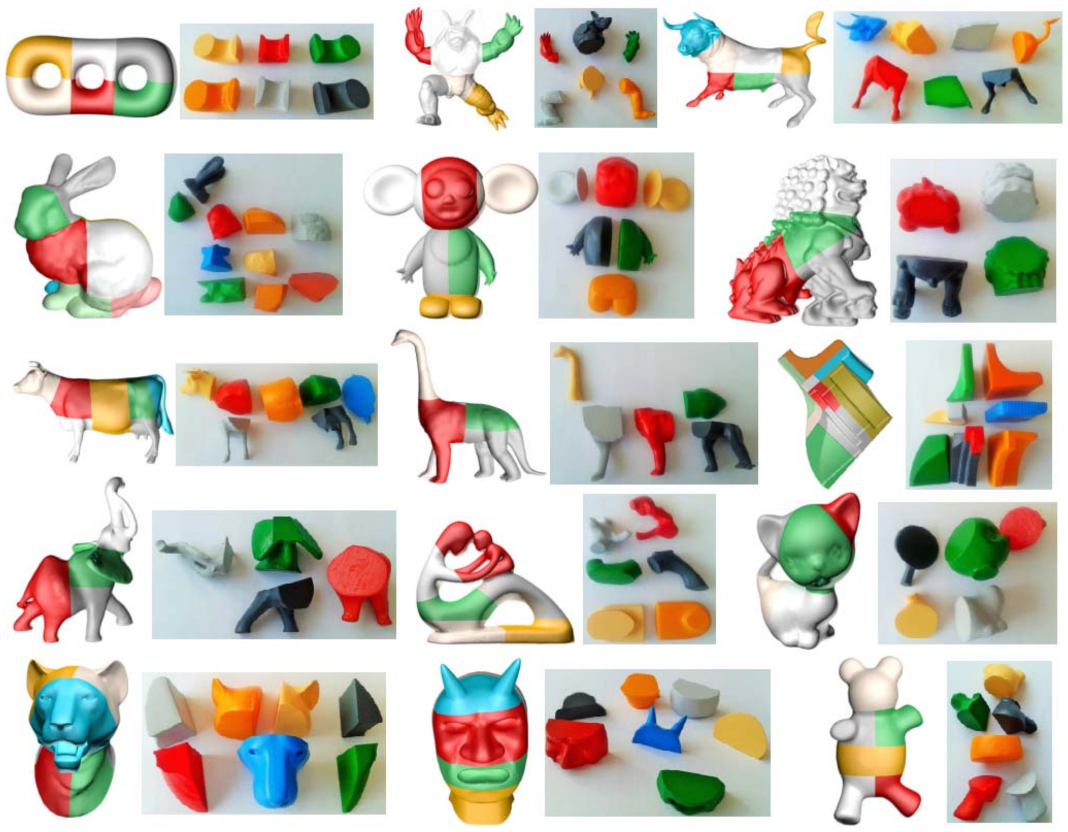


Figure 7. Some partitioning results by users with a few interactions using our proposed tool. For each model, the user's partitioning result and its printed parts are shown. The model volumes are 3Holes (8066.9cm^3), Armadillo (2288.82cm^3), Bull (2281.62cm^3), Bunny (6003.38cm^3), Cheburashka (3288.39cm^3), Chinese_Lion (6001.62cm^3), Cow (3277.76cm^3), Dinosaur (1299.09cm^3), Elephant (2232.42cm^3), FanDisk (5726.12cm^3), Fertility (3195.51cm^3), Kitten (6581.03cm^3), LionHead (4257.72cm^3), Monster (7330.87cm^3), Toy (4594.35cm^3), and the printer volume is 18.0cm $\times$ 17.0cm $\times$ 16.0cm.

EXPERIMENTS AND USER STUDY

We have implemented the proposed interactive tool in C++ and tested it on a PC with an Intel i7-860 CPU (2.80GHz) and 8GB RAM. Users can easily and quickly obtain a desired partitioning of a large model fitting into a 3D printer's volume by a few interactions using our tool. Partitioning examples generated by users are shown in Figures 7 and 9, in which the volumes of models range from 1299.09 cm^3 to 8066.9 cm^3. The volume of 3D printer used in our test is 18.0cm $\times$ 17.0cm $\times$ 16.0cm.

The experiments and user study consist of two parts. First, we made a qualitative comparison between our method and Chopper.[1] Ten participants who were all college students (three females and seven males) and had common computer skills were invited. They were instructed to read built-in help for clear understanding of the interactive operations in our method. Note that Chopper is fully automatic and there is no need to manually operate it. Six large 3D models were provided and each participant was requested to partition using our method. We recorded the time it took for them to complete their operations (including both interactive operations and the running time of applying Algorithms 1-3)

Table 1 summarizes the users' performance we collected in the user study. The table shows the partitioning time of each model for each participant using our method. As a comparison, the running time of Chopper is also presented in Table 1. The results showed that our method (1-3 minutes for partitioning a model) is more efficient than Chopper (3-6 minutes for partitioning a model) by using less partitioning time. This is because Chopper evaluated all possible partitions (even those invalid partitions) by an exhaustive search of all possible combinations of candidate planes, while with user's interaction in our method, the user specified constraints (either no-go regions or specific cutting plane) greatly reduced the search space. Furthermore, after partitioning, all participants evaluated that the partitioning results obtained in our method are better than those in Chopper, since our method efficiently incorporates user's intention.

Table 1. The partitioning time of six models by each of ten users using our interactive method. As a comparison, the partitioning time of Chopper is also presented. All running times were measured in seconds and were collected on a PC with an Intel I7-860 CPU (2.80GHz) and 8GB RAM.

Models	**Running time in seconds**										
	Chop-per	**Our Interactive Method**									
		User 1	**User 2**	**User 3**	**User 4**	**User 5**	**User 6**	**User 7**	**User 8**	**User 9**	**User 10**
Elk	327	70	70	92	50	101	125	83	128	99	112
Cheburashka	245	81	73	85	60	69	89	98	102	77	104
Cow	203	86	94	70	76	121	115	91	78	85	125
Bunny	287	133	92	140	72	140	147	89	115	158	174
Monster	357	91	79	132	80	135	90	102	78	87	134
Hand	295	77	89	90	115	126	70	101	81	70	135

Note that the partitioning results by Chopper heavily depend on the weights (Equation) in the objective function (Figure 1). By imposing users' constraints, our method is not sensitive to different weights. One example is shown in Figure 8.

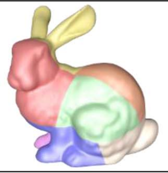
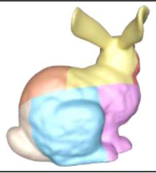
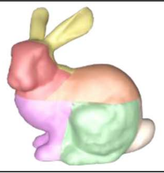
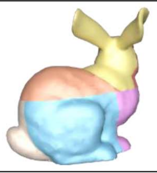
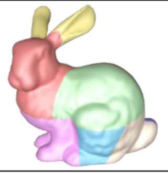
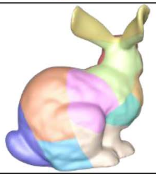
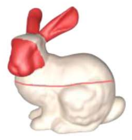
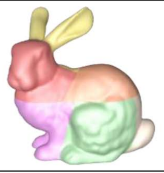
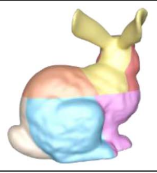
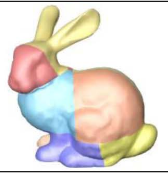
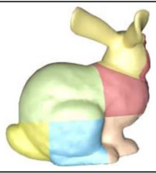
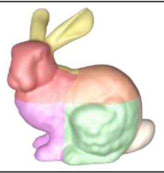
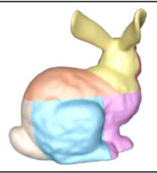
Chopper			Our method			
Front view	Back view	Weights	User input	Front view	Back view	Weights
		(1.0, 0.05, 1.0, 0.25, 1.0, 0.1)				(1.0, 0.05, 1.0, 0.25, 1.0, 0.1)
		(1.0, 0.5, 1.0, 0.25, 1.0, 0.0)	 No-go regions and a cutting plane painted by a user			(1.0, 0.5, 1.0, 0.25, 1.0, 0.0)
		(1.0, 0.5, 0.0, 0.25, 1.0, 0.1)				(1.0, 0.5, 0.0, 0.25, 1.0, 0.1)

Figure 8. Left: for the bunny model, the partitioning results by Chopper heavily depend on the weights (Equation 1) in the objective function. Right: Given user specified no-go regions and a cutting plane, our method outputs similar partitioning results for different weights. The weights are $\left(w_{\text{part}}, w_{util}, w_{fragility}, w_{symmetry}, w_{connector}, w_{seam}\right)$.

Second, we evaluated and compared our method with the Chopper[1] and a state-of-the-art level set method,[5] quantitatively and qualitatively. Note that the level set method[5] consists of two parts:

- an initial mesh surface segmentation: we implemented the shape diameter segmentation which is one of recommended automatic mesh segmentations by Yao and colleagues,[5] and
- an iterative, constrained variational optimization.

Both parts are time-consuming. Six models, Elk, Cheburashka, Cow, Monster, Bunny and Hand, were partitioned into printable pieces using three methods, Chopper and the level set method and our method. The partitioning results of six models, as well as the number of partitioning pieces and running time, are summarized in Figure 9. These results showed that with a few user interactions, our method can obtain fewer pieces of printable components and less performance time than Chopper and the level set method. Note that compared with the model partitioning by curved surfaces as in the work of Yao and colleagues,[5] an advantage of model partitioning by cutting planes is that connectors can be easily placed on the cross sections.

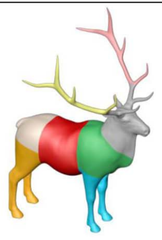
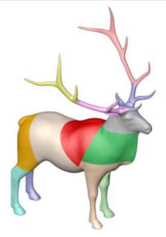
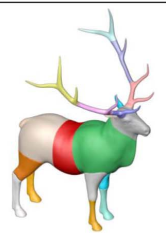
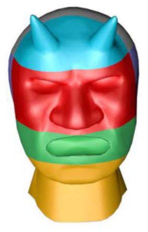
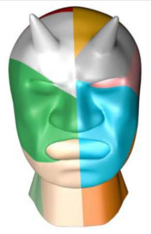
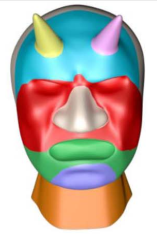
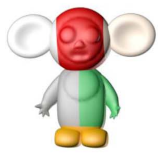
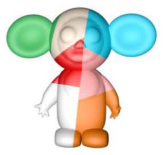
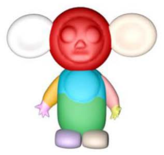
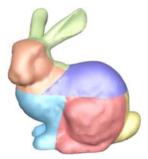
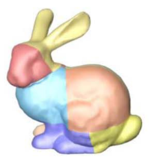
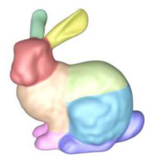
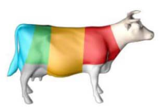
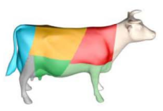
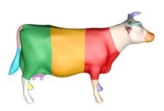
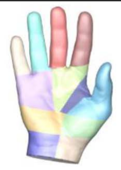
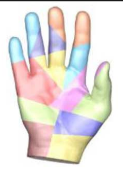
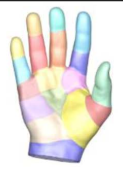
Our method	Chopper	Level set method	Our method	Chopper	Level set method
					
8 parts, 189 sec	13 parts, 437 sec	21 parts, 296+312 sec	7 parts, 60 sec sec	13 parts, 357 sec	9 parts, 83+231
					
6 parts, 59 sec	7 parts, 245 sec	9 parts,72+197 sec	7 parts, 119 sec	10 parts, 287 sec	8 parts, 80+285 sec
					
7 parts, 74 sec	8 parts, 203 sec	16 parts, 77+204 sec	12 parts, 173 sec sec	16 parts, 375 sec	17 parts, 132+303

Figure 9. Some partitioning results obtained by our method, Chopper and the level set method. The performance time in our method includes all interaction time and the time running Algorithms 1-3. The time for the level set method includes two parts: an initial shape diameter mesh segmentation plus an iterative constrained variational optimization. The model volumes are Elk ($5832cm^3$), Monster ($7330.87cm^3$), Cheburashka ($3288.39cm^3$), Bunny ($6003.38cm^3$), Cow ($3277.76cm^3$), Hand ($3668cm^3$), and the printer volume is $18.0cm \times 17.0cm \times 16.0cm$. The printed Elk models are shown in Figure 10.

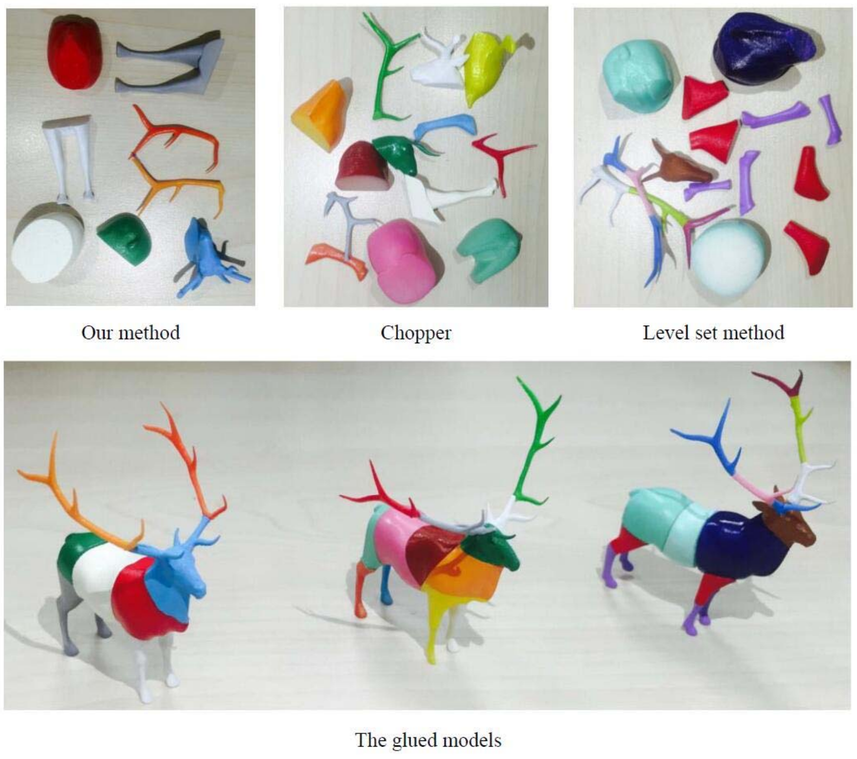


Figure 10. The printed Elk models partitioned by our method, Chopper, and the level set method.

All ten participants in the first user study watched the partitioning results of six models (shown in Figure 9) by three methods (ours, Chopper, level set method) in computer as well as the printed physical models (Figure 10). Then each participant was asked to compare and rank these partitioning results by choosing one mark from five levels: excellent (5), good (4), fair (3), bad (2), very bad (1). Finally they filled a questionnaire with Yes/No choices.

The average rank of each partitioning result evaluated by ten participants was summarized in Table 2. The results showed that the subjective ranks of partitioning results obtained by our method were consistently higher than the subjective ranks of results obtained by Chopper and the level set method.

Table 2. The average subjective ranks from ten participants for the partitioning results of six models shown in Figure 9 using three methods (our method, Chopper, the level set method).

Model name	**Average subjective rank**		
	Our method	**Chopper**	**Level set method**
Elk	4.5	3.0	3.7
Cheburashka	4.4	2.0	4.2
Cow	4.6	3.1	4.0
Bunny	4.5	3.4	3.8
Monster	4.0	2.4	3.8
Hand	4.8	2.5	3.8
Overall	**4.5**	**2.7**	**3.9**

The post-test questionnaire filed by participants evaluated three characteristics of our method. Each characteristic had five questions. A Yes answer to each question gave one score and then the full score to each characteristic is five. The three characteristics and their average scores by participants are as follows.

Usefulness. The five questions are: (1) Does it give more control to the user over the partitioning process? (2) Does it make the partitioning process easier? (3) Does it save time? (4) Does it meet the needs of 3D-model partitioning? (5) Does it have everything that you would expect it to do? The average score given by participants is 4.4.

Ease of use. The five questions are: (1) Is it easy to use? (2) Is it user friendly? (3) Does it require the fewest steps possible to accomplish partition? (4) Is it flexible? (5) Can you use it without noticing any inconsistencies? The average score given by participants is 4.2.

Satisfaction. The five questions are: (1) Have you learned to use it quickly? (2) Have you easily remembered how to use it? (3) Have you quickly became skillful with it? (4) Are you satisfied with it? (5) Does it work the way you want? The average score given by participants is 4.2.

The above results show that our method can offer a simple and easy-to-use interactive tool for 3D-printing-oriented model partitioning to users.

CONCLUSION

We presented an efficient interactive tool to partition large 3D models into components that fit into the volume of a given 3D printer. Three interaction operations, i.e., no-go region painting, cutting plane specification and stitching, are realized, based on three algorithms, including (1) constrained tree generation, (2) tree-splitting and (3) sub-tree merging. These interaction operations are simple and intuitive to use. With the help of these operations, users can efficiently partition large models into smaller printable components in good quality. Experiments and a preliminary user study have been conducted to verify the functionality of our interactive tool.

ACKNOWLEDGEMENTS

A.K. Jadoon and C. Wu contributed equally to this paper. Y.-J. Liu is the corresponding author. Models used in this paper are courtesy of Stanford University and AIM@SHAPE Shape Repository. This work was partially supported by the National Key Research and Development Plan (2016YFB1001202), the Natural Science Foundation of China (61432003, 61322206, 61272228, 61661130156), Hong Kong RGC GRF Grant (CUHK/14207414) and the Royal Society-Newton Advanced Fellowship.

REFERENCES

1. L. Luo et al., "Chopper: Partitioning models into 3D-printable parts," *ACM Transactions on Graphics*, vol. 31, no. 6, 2012, p. 129:1.
2. J. Vanek et al., "Packmerger: A 3D print volume optimizer," *Computer Graphics Forum*, vol. 33, no. 6, 2014, pp. 322–332.
3. Q. Zhou, J. Panetta, and D. Zorin, "Worst-case structural analysis," *ACM Transactions on Graphics*, vol. 32, no. 4, 2013, p. 137:1.
4. X. Chen et al., "Dapper: decompose-and-pack for 3D printing," *ACM Transactions on Graphics*, vol. 34, no. 6, 2015, p. 213:1.
5. M. Yao et al., "Level-set-based partitioning and packing optimization of a printable model," *ACM Transactions on Graphics*, vol. 34, no. 6, 2015, p. 214:1.
6. Z. Xie et al., "3D shape segmentation and labeling via extreme learning machine," *Computer Graphics Forum*, vol. 33, no. 5, 2014, pp. 85–95.

7. H. Medellin et al., “Automatic subdivision and refinement of large components for rapid prototyping production,” *Journal of Computing and Information Science in Engineering*, vol. 7, no. 3, 2007, pp. 249–258.
8. J. Hao, L. Fang, and R.E. Williams, “An efficient curvature-based partitioning of large-scale STL models,” *Rapid Prototyping Journal*, vol. 17, no. 2, 2011, pp. 116–127.
9. X. Zhang et al., “Computing stable contact interface for customized surgical jigs,” *IEEE International Conference on Robotics and Automation* (ICRA 15), 2015, pp. 6160–6166.
10. R. Hu et al., “Approximate pyramidal shape decomposition,” *ACM Transactions on Graphics*, vol. 33, no. 6, 2014, p. 213:1.
11. T.A. Funkhouser et al., “Modeling by example,” *ACM Transactions on Graphics*, vol. 23, no. 3, 2004, pp. 652–663.
12. Y. Lee et al., “Intelligent Mesh Scissoring Using 3D Snakes,” *Proceedings of the 12th Pacific Conference on Computer Graphics and Applications* (PG 04), 2004, pp. 279–287.

ABOUT THE AUTHORS

Aamir Khan Jadoon is a master’s student in advanced computing at Tsinghua University, Beijing, China. He received a BSc degree in computer science from the University of Peshawar and MSc in computer science from Quaid-e-Azam University, Islamabad Pakistan. He is also a software analyst and programmer with over 10 years of industry experience occupying a number of seniority and leadership positions. His main research interests include computer graphics, geometric modeling and processing, interactive techniques and virtual reality. Contact him at qinl14@mails.tsinghua.edu.cn.

Chenming Wu is a master’s student with TNList in the Department of Computer Science and Technology at Tsinghua University. He received his B.Eng. degree from Beijing University of Technology. His research interests include 3D printing and computer graphics. Contact him at wcm15@mails.tsinghua.edu.cn.

Yong-Jin Liu received a BEng from Tianjin University and a PhD from the Hong Kong University of Science and Technology. He is currently an associate professor TNList in the Department of Computer Science and Technology at Tsinghua University. His research interests include computational geometry, computer graphics and computer-aided design. He is a senior member of the IEEE and a member of ACM. Contact him at liuyongjin@tsinghua.edu.cn.

Ying He received BS and MS degrees in electrical engineering from Tsinghua University and a PhD in computer science from Stony Brook University. He is an associate professor in the School of Computer Engineering at Nanyang Technological University. He is interested in the problems that require geometric computing and analysis. Contact him at YHe@ntu.edu.sg.

Charlie C.L. Wang received a BEng in mechatronics engineering from the Huazhong University of Science and Technology, and MPhil and PhD degrees in mechanical engineering from The Hong Kong University of Science and Technology. He is now a professor and Chair of Advanced Manufacturing, in the Department of Design Engineering at Delft University of Technology. He is a Fellow of the American Society of Mechanical Engineers and a senior member of IEEE. Contact him at c.c.wang@tudelft.nl.