Chopper: Partitioning Models into 3D-Printable Parts

Linjie Luo[1,2] Ilya Baran[3] Szymon Rusinkiewicz[1] Wojciech Matusik[4]
[1]Princeton University [2]Disney Research [3]Disney Research Zurich [4]MIT CSAIL

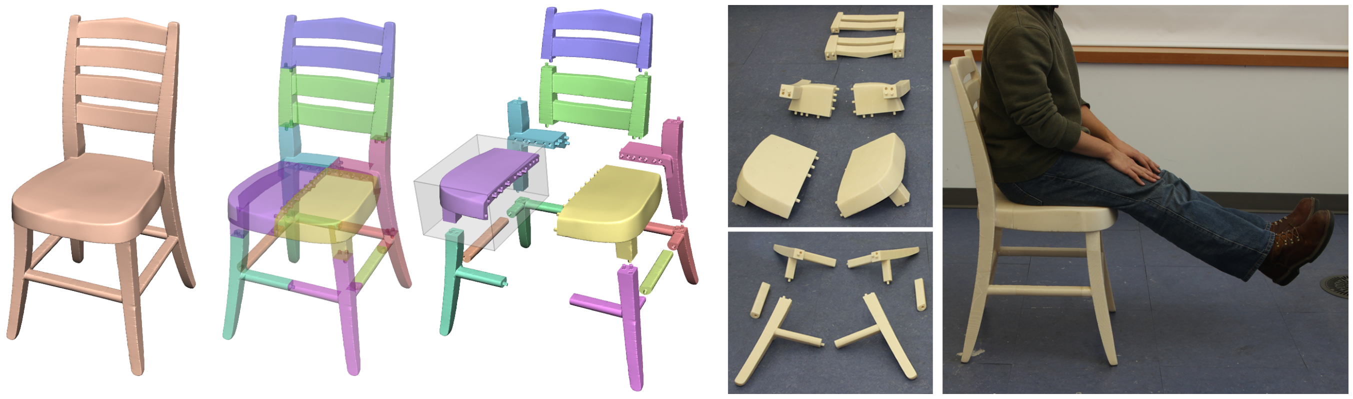


Figure 1: *Chopper partitions a given 3D model into parts that are small enough to be 3D-printed and assembled into the original model. Left to right: the input chair model, Chopper's partition (with a printing volume shown as a reference), printed parts, and assembled chair.*

Abstract

3D printing technology is rapidly maturing and becoming ubiquitous. One of the remaining obstacles to wide-scale adoption is that the object to be printed must fit into the working volume of the 3D printer. We propose a framework, called Chopper, to decompose a large 3D object into smaller parts so that each part fits into the printing volume. These parts can then be assembled to form the original object. We formulate a number of desirable criteria for the partition, including assemblability, having few components, unobtrusiveness of the seams, and structural soundness. Chopper optimizes these criteria and generates a partition either automatically or with user guidance. Our prototype outputs the final decomposed parts with customized connectors on the interfaces. We demonstrate the effectiveness of Chopper on a variety of non-trivial real-world objects.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric Algorithms;

Keywords: 3D printing, mesh segmentation and decomposition

Links: DL PDF

1 Introduction

As 3D printing technology matures, becoming cheaper and simpler to use, printing ever-larger objects becomes feasible. At the same time, the maximum size of an object that a 3D printer can produce in one pass (the *printing volume*) is limited by practical considerations. Larger objects must therefore be printed as separate parts and assembled. Existing commercial systems rely on manual partitioning, but this can be tedious if many parts are needed.

Recent work in the field of computer graphics, however, has investigated computation-assisted fabrication approaches in which the user specifies the desired visual or physical properties of an object to be manufactured, and an optimization is used to determine the precise way in which fabrication devices should be driven to produce an object as close as possible to the goal. The optimization may be formulated to consider visual properties such as scattering [Hašan et al. 2010] or physical properties such as deformation [Bickel et al. 2010], but in this paper we focus purely on 3D shape. We propose that this kind of optimization approach may be applied to the problem of decomposition into print volumes, yielding an automated method that is quicker, easier, and in many instances better than can be accomplished by hand.

This work presents a framework, called *Chopper*, for partitioning objects for 3D printing. Because the number of ways in which an object may be partitioned is large, we seek to find a decomposition that optimizes a number of (sometimes conflicting) objectives:

- **Printability:** the parts must fit inside the working volume.
- **Assemblability:** it must be possible to put parts together (without interference) into a finished model.
- **Efficiency:** the partition should avoid small parts and, in general, minimize the number of required sub-volumes.
- **Connector feasibility:** each interface must be large enough to admit connectors, protrusions, or other aids to assembly.
- **Structural soundness:** parts should not have thin slivers, and seams should be away from areas of high mechanical stress.
- **Aesthetics:** seams should be unobtrusive, detracting from appearance as little as possible, and should follow the natural symmetries of the model.

Formalizing these objectives is not easy, since different use cases have different requirements. Moreover, it is easy to propose definitions for the objective functions that lead to sub-optimal partitions or interact in unwanted ways. We propose specific definitions (Section 3.2) and weightings between different objectives (Section 4) that, based on our testing, typically lead to high-quality results.

Designing a Framework for Optimization: A natural approach to creating printable partitions is to treat this as a covering problem: the object must be completely covered by print volumes. This, however, quickly leads to problems with efficiency and solvability. For example, consider just minimizing the number of parts, under extremely restrictive assumptions: in 2D, if the printing volume is a unit square, it is an open problem whether a square of side length 2.01 can be printed as 6 parts, or if 7 are necessary [Soifer 2006; Januszewski 2009]. Even given a feasible covering, it is difficult to go from this to a concrete set of cuts, which we must do to evaluate some of the objective functions (such as connector feasibility and seam aesthetics). Finding a set of cuts leading to an assemblable partition is difficult (since these cuts may have to be non-planar) and impractical to do every time we evaluate a covering's quality.

Chopper is therefore built upon an alternative approach: searching top-down for a binary tree representing the assembly sequence in which two parts are joined at each step. More specifically, we restrict ourselves to *planar cuts* between any two joined parts, yielding a Binary Space Partitioning or BSP tree [Fuchs et al. 1980]. This is an important design decision, since by giving up the flexibility of arbitrary cuts we gain efficiency. Also BSPs provide a natural recursive framework for optimizing over the objectives, and in fact satisfy two objectives by construction (printability — by recursing down until each part fits inside the working volume — and assemblability — by reversing the order of the cutting recursion and the convexity of each BSP partition).

The search for the best BSP tree can proceed greedily, with the best single plane chosen at each level of recursion, or may consider alternatives that are not necessarily locally optimal in the hope that they lead to a better overall solution. Because of the size of the search space, it is impossible to enumerate all possible partitions exhaustively. We therefore adopt *beam search* [Lowerre 1976], which maintains a set of the b best partitions at each stage of the algorithm. Each step of the search considers all ways of augmenting each existing partition with an additional cut, evaluates the objective functions, and keeps the b best resulting partitions. The search terminates once all parts fit within the working volume. We show that even a relatively small beam width results in modest improvements over greedy search, at the cost of increased running time (by a factor of b). In constrast, we find that depth-first search with backtracking did not lead to better results than greedy search.

2 Related Work

Partitioning models has been studied in different contexts outside rapid prototyping, therefore ignoring the concerns specific to 3D printing (part size, assemblability, etc.) on which we focus. For example, decomposing polyhedra into convex parts [Chazelle 1981] — even approximately [Lien and Amato 2007] — is a well-known challenging problem in computational geometry. Generating a volumetric mesh of limited-size tetrahedra (e.g., via constrained Delaunay tetrahedralization [Shewchuk 1998]) is a vital preprocess for physical simulation. Mesh segmentation [Shamir 2008; Chen et al. 2009; Attene et al. 2006] typically focuses on decomposing the *surface* of an object into semantically meaningful parts, using heuristics based on geometric properties such as curvature and shape diameter. Our work shares a few objectives with these techniques (e.g., minimizing seam length).

A number of design tools focus on creating physical objects using a variety of fabrication techniques. These objects include plush toys [Mori and Igarashi 2007], chairs [Saul et al. 2011], furniture

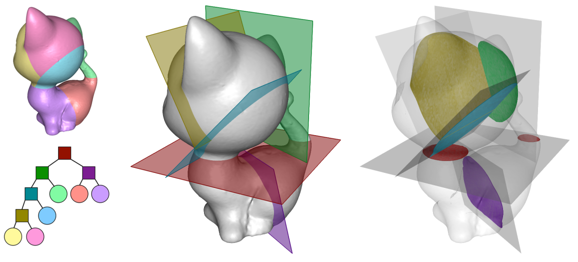


Figure 2: An illustration of a 3D object partitioned by a BSP tree into six parts. Left: the partitioned parts and the BSP tree. Planes are denoted as squares and parts as circles with corresponding colors. Middle: planes in the BSP tree, with colors corresponding to interior nodes (squares) in the tree at left. Right: cross sections of the planes and the object. Note that the cross-sections may have multiple connected components, e.g., on the red plane.

in general [Lau et al. 2011], buildings [Whiting et al. 2009], garments [Umetani et al. 2011], Burr puzzles [Xin et al. 2011], planar sections [McCrae et al. 2011; Hildebrand et al. 2012], etc. Recently, Stava et al. [2012] proposed a tool that performs a structural analysis on a 3D model and then corrects the model using hollowing, thickening, and strut insertion. Our work falls broadly into this category of research, and uses a similar constrained optimization framework to many of these approaches, though the details are specific to our application. The *assembly* of objects from parts has also been investigated [Agrawala et al. 2003].

Several methods focus on determining the optimal 3D orientation at which a single part should be printed [Alexander et al. 1998; Thrimurthulu et al. 2004], considering factors such as support material cost, printing time, and surface roughness. Research has also focused on techniques for packing multiple parts into the printing volume [Ikonen et al. 1998; Dickinson and Knopf 1998; Egeblad et al. 2009].

An automatic curvature-based method for partitioning models for 3D printing has been recently proposed [Hao et al. 2011]. This method makes strong assumptions that make it suitable for only a small class of models. Among its limitations are:

- It assumes that potential cuts are restricted to planar sharp or filleted features in the model.
- To detect these features, it relies on properties of the triangulation such as dihedral angles and perimeter ratios of adjacent triangles.
- Because the method may choose a partial planar cut, there is no guarantee that the resulting parts can be assembled or even that the cut actually partitions the model into two.
- The method does not consider printing parts in different orientations to make better use of the print volume.

In our dataset, only the bearing model has the properties needed to be successfully processed by the algorithm of Hao et al.

To our knowledge, the only other automatic methods for partitioning models for 3D printing are based on partitions on a regular grid [Medellin et al. 2007]. This is a greatly reduced space of possible partitions that are likely to use too many cuts, ignore object features, and make poor use of the printing volume.

3 Approach

For a user-provided model, Chopper employs *beam search* (Section 3.1) to find a BSP tree that partitions the model into a number of *parts*, guided by the *objective functions* that evaluate various aspects of the BSP, e.g., the number of parts, seam length, symmetry (Section 3.2). Each plane of the BSP tree forms a *cut*. (Note that each part is not necessarily a single connected component.) The *cross-section* of a cut is the polygon set that is the intersection of the object interior and the portion of the cut plane inside its BSP region. This polygon set may comprise multiple disjoint polygons, which may contain holes, and defines the *connected components* of the cut. Figure 2 illustrates these concepts.

Each part partitioned by the output BSP tree should fit into the *printing volume* of the target 3D printing device. Low-cost printers may have a printing volume as small as 10 cm $\times$10 cm $\times$10 cm, while higher-end devices may go up to 50 cm $\times$ 40 cm $\times$20 cm.

Chopper places *connectors*, according to a user-provided pattern, on every cross-section (Section 3.3). Depending on their design, the connectors may provide sufficient structural strength to hold parts together, or may merely serve as guides for assembly, with glue used to permanently attach parts. In our experiments we use connectors in the shape of pentagonal prisms, providing a sturdy connection that prevents parts from rotating.

3.1 Search Framework

Algorithm 1: Chopper search

```
input  : $O$ is the object, $b$ is beam width
output : $\mathcal{T}$ is the BSP tree that partitions the object

function $\mathcal{T}$ = BeamSearch($O$, $b$)
    currentBSPs ← ∅
    while not AllAtGoal(currentBSPs)
        newBSPs ← ∅
        foreach $\mathcal{T} \in$ NotAtGoalSet (currentBSPs)
            currentBSPs −= $\mathcal{T}$
            $P$ ← LargestPart($O$, $\mathcal{T}$)
            newBSPs += EvalCuts($\mathcal{T}$, $P$)
        while |currentBSPs| < $b$
            currentBSPs += HighestRanked(newBSPs)
    return HighestRanked(currentBSPs)

function ResultSet = EvalCuts ($\mathcal{T}$, $P$)
    $\mathcal{N}$ ← UniformNormals()
    $\mathcal{N}'$ ← AuxiliaryNormals($P$)
    parallel foreach $\mathbf{n}_i \in \mathcal{N} \cup \mathcal{N}'$
        foreach objective function $f_k$
            foreach plane $\pi_{ij} = (\mathbf{n}_i, d_j)$ intersecting $P$
                $\mathcal{T}'$ ← AddToBSP($\mathcal{T}$, $\pi_{ij}$)
                $f(\mathcal{T}')$ += $\alpha_k f_k(\mathcal{T}')$
    ResultSet ← ∅
    foreach $\mathcal{T}$ sorted by $f(\mathcal{T})$
        if SufficientlyDifferent($\mathcal{T}$, ResultSet)
            ResultSet += $\mathcal{T}$
    return ResultSet
```

Chopper uses beam search to partition the model (Algorithm 1). The algorithm begins with an empty BSP, and at each step it selects the most promising BSPs from the previous step to extend with an additional cut. The maximum number of BSPs selected at each step is the *beam width*: it determines the breadth that the algorithm explores into the combinatorial search space. All results in this paper use a beam width of $b = 4$, unless otherwise stated.

Evaluating Candidate Cuts: Given a partial BSP, we consider all ways of augmenting it by cutting the *largest* part — i.e., the part with the largest number-of-subvolumes estimate (Section 3.2.1). In addition to making progress towards the ultimate goal, we find that evaluating the largest parts first improves our estimate of the quality of a BSP tree: the biggest parts are usually the ones with the greatest overestimations in the objective functions.

We consider cutting the part with all possible planes $\pi_{ij} = (\mathbf{n}_i, d_j)$, where $\mathbf{n}_i$ and d_j are plane normals and offsets, respectively (and the plane is defined by $\mathbf{n}_i \cdot \mathbf{x} - d_j = 0$). To ensure uniform sampling, we take our set of plane normals $\mathcal{N} = \{\mathbf{n}_i\}$ to be the vertices of a thrice-subdivided regular octahedron. This results in 129 uniformly distributed directions and proves to be a good tradeoff between result quality and running time. (Only one of each pair of opposite directions is taken.) We augment the set of candidate normals with $\mathcal{N}'$, the axes of each part's minimum oriented bounding box, in order to provide additional opportunities for selecting an aesthetically-pleasing cut. The plane offsets $\{d_j\}$ are sampled uniformly (at intervals of 0.5 cm in our examples) over the range for which the planes intersect the object, for each $\mathbf{n}_i$.

We observe that evaluating the objective functions for each candidate cut is naturally organized as a set of nested loops (per-cut-plane-orientation, per-cut-plane-position, per-objective). In order to minimize computation time, Chopper pushes as much computation as possible into the outer loops. In addition, we exchange the two innermost loops relative to their natural order, evaluating each objective on a batch of parallel planar cuts (i.e., looping over objectives *outside* the loop over plane positions). This batching provides optimization opportunities for most objectives. For example, computing the cross-sections for k single planes independently takes $O(kn + c)$ time (where n is the number of mesh vertices and c is the total size of all the cross-sections), but doing this computation batched takes $O(k + n + c)$ time.

Selecting Cuts: After evaluating all the candidate augmented BSPs, we would ideally like to retain the b best ones. However, for relatively small (hence practical) values of b, there is a danger of degrading the diversity of the selection if too many similar BSPs are selected. Therefore, the new BSPs resulting from each parent are first pruned by sorting the planes according to their objective scores, then walking along this list from lowest to highest. BSP trees that are sufficiently close (in RMS distance) to already-selected ones are discarded — a typical value for the threshold is 0.1 of the diagonal of the printing volume. The surviving trees resulting from all parents are combined, and the b best ones are kept.

Termination: The search terminates when all BSPs in the beam reach their goal states, which are decompositions in which all parts fit into the target printing volume. The best BSP, as evaluated by the objective functions, is returned as the final solution. As a post-process, Chopper places connectors on the cross-sections so that they do not interfere with each other (Section 3.3).

3.2 Objective Functions

Given a set of candidate BSPs, we evaluate a set of objective functions that rank these BSPs in terms of the number of resulting parts, connector feasibility, structural soundness, and seam unobtrusiveness. The final score is computed as a linear combination of the separate scores of all objective functions of a BSP $\mathcal{T}$:

$$f(\mathcal{T}) = \sum_k \alpha_k f_k(\mathcal{T}). \tag{1}$$

The details of the objective functions will be discussed in the following subsections. We motivate each objective by demonstrating a result obtained when that objective is not used. The objectives are all unitless so that a single set of weights can work with a large range of objects.

Note that due to the nature of the beam search algorithm (Section 3.1), the BSPs being evaluated are all extended from existing ones, which enables fast accumulative computation of most of the objective functions from the previous results.

An objective function may return infinity if the cut is completely infeasible according to that objective. In that case, subsequent objective function evaluations are skipped for that cut.

3.2.1 Number of Parts

Estimating the minimum number of boxes (assuming that a print volume is a box) to cover an arbitrary object is NP-hard even in 2D and disallowing rotations [Aupperle et al. 1988]. We do not attempt to find the minimum number, but we do need to guarantee that eventually the algorithm will terminate. To this end, we estimate an upper bound, $\Theta(P)$, on the number of parts for P neglecting all other objectives. As this bound, we use the number of print volumes that (in a regular grid) tile the minimum Oriented Bounding Box (OBB) of P. The minimum OBB is computed approximately by iteratively optimizing two axes at a time using rotating calipers [Toussaint 1983] or gets its orientation from the parent part, whichever leads to a smaller Θ (breaking ties by volume).

Given a BSP $\mathcal{T}$, we would like to minimize $\Theta(P)$ on all parts P in $\mathcal{T}$: (Figure 3):

$$f_{\text{part}}(\mathcal{T}) = \frac{1}{\Theta_0} \sum_{P \in \mathcal{T}} \Theta(P), \tag{2}$$

where Θ_0 is the upper bound estimate of the initial input model.

Because the granularity of f_{part} is entire working volumes, we wish to add a finer-level distinction to penalize parts that do not use the print volume efficiently (Figure 4). Let V_P be the volume of a part P in $\mathcal{T}$ and let V be the printing volume. Then we compute a *utilization* objective:

$$f_{\text{util}}(\mathcal{T}) = \max_{P \in \mathcal{T}} \left(1 - \frac{V_P}{\Theta(P)V} \right). \tag{3}$$

3.2.2 Connector Feasibility

We force each connected component of the cross-section of each cut to have one or more connectors to ensure the object does not fall apart. While this condition is not strictly necessary (a donut cut in half needs only one connector for the two connected components of the cross-section), we found that it helps the structural soundness of the part without being overly restrictive. A single connector on a large cross-section, however, may not be desirable: allowing for several connectors results in a more robust connection. Moreover, several connectors in a straight line are not as robust as the same number of connectors spread out in 2D. Therefore, we formulate an objective that seeks to find cuts maximizing the potential quality of connector placement (Figure 5).

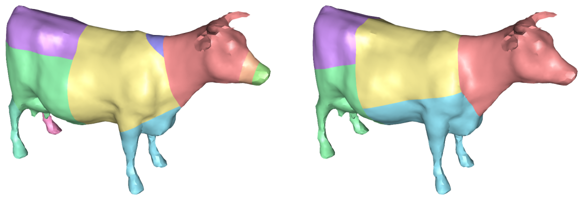


Figure 3: *Without the number-of-parts objective, Chopper chooses a partition (left) with more cuts than necessary (right).*

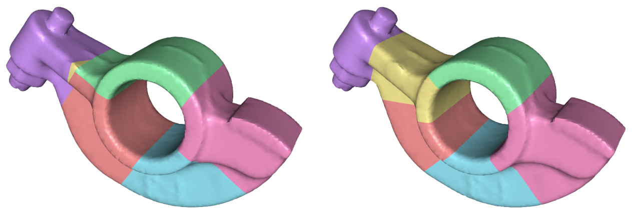


Figure 4: *Without the utilization objective (left), Chopper produces a small part for this rocker arm (yellow). The utilization objective forces larger parts and results in one fewer cut (right).*

During the execution of Algorithm 1, we maintain a point set of *potential* (female) connector locations on both sides of every cut, initially sampled from the cross-section with a uniform grid (see Figure 13, left). When choosing a new cut, we must consider the quality of both the connector locations on the new cut and connectors on previously chosen cuts, some of which may intersect the new cut (Figure 6).

Evaluating the robustness of a potential connector placement is intractable for the large number of candidates we wish to process, so we approach the problem analogously to computing a support polygon in rigid-body mechanics. We use the convex hull of the potential connector locations as a measure of the quality of a potential connector placement on a connected component G of a cross-section C. Intuitively, a large convex hull makes it less likely that a large torque will be applied.

We compute the area a_G of this convex hull, offset outwards by the connector radius. We also compute the area A_G of the relevant connected component of the cross-section. The connector feasibility objective is then defined as

$$f_{\text{connector}}(\mathcal{T}) = \max \left(\max_{G \in C \in \mathcal{T}} A_G / a_G - T_c,\ 0 \right), \tag{4}$$

which is appropriately infinity if some connected component has no feasible connector locations and therefore $a_G = 0$. We clamp A_G/a_G to $T_c = 10$ from below to avoid this objective's having an undesirably large influence on cuts with sufficiently good connector placement potential.

To determine if the female connector fits at a location in the volume, we approximate its geometry conservatively as a union of a few spheres. We also maintain a distance field inside the object. We sample the distance field at the locations of potential sphere centers and compare the distances to the sphere radii to determine if the connector fits. In this phase, we ignore the possibility that two female connectors from different cross-sections may overlap; we

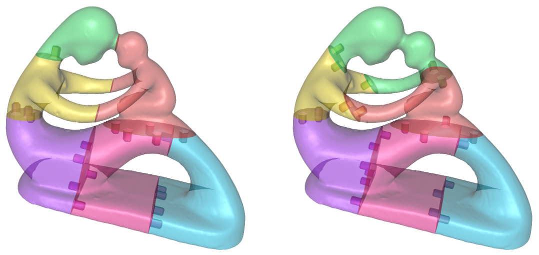


Figure 5: *Without considering connector feasibility (left), Chopper selects a cutting plane that results in cross-sections too small to install connectors (between the hands and heads). The connector feasibility objective ensures each connected component of the cross sections is sufficient for connector placement (right).*

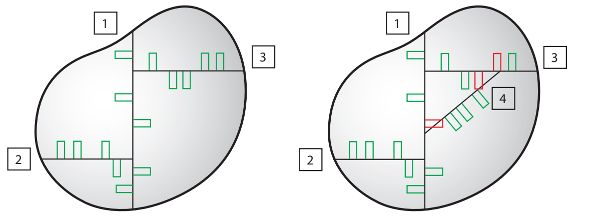


Figure 6: *Left: An object with three cuts and potential female connector locations (green) on the cross-sections. Right: when a new cut is introduced, some of the connector locations become invalidated (red) due to intersection.*

resolve such conflicts when we actually place the connectors (after the BSP tree has been completely generated — see Section 3.3).

Chopper can support a number of possible connector designs. We have considered connectors that snap together, require screws, or even glue. For our experiments, we simply use pentagonal prisms, with a male prism extruding from the surface of one part and fitting into a female prism on another part. Supporting glue would be strictly easier, requiring no modifications to geometry.

3.2.3 Structural Soundness

We use two methods for evaluating the structural soundness of a proposed BSP. The first attempts to avoid cuts through high-stress areas of the model, given a user-specified upright orientation (or gravity direction). Because this method requires a comparatively slow finite-elements structural analysis, as well as user input about the intended orientation of the object, it is not always desirable. Therefore, we also include a second fragility heuristic that avoids creating thin "fins" and "bridges" in the parts.

Structural Analysis: Our structural analysis begins by voxelizing the model on a regular grid, and forming a tetrahedral mesh with six tetrahedra per voxel. We then apply a positional constraint to the lowest 5 percent of exterior tetrahedral vertices (with a minimum of 20 vertices), according to the user-supplied gravity direction. Finally, we apply gravity as an external force, with user-specified density and modulus.

Using a standard finite-elements structural analysis, we compute the stress tensor σ throughout the volume. Then, at run time, we evaluate each cut in $\mathcal{T}$ for soundness by integrating the tension T_t and shear T_s computed from σ and the normal of that plane (compression is ignored, since the resistance of most 3D-printing materials to compression is high):

$$f_{\text{structure}}(\mathcal{T}) = \sum_{C\in\mathcal{T}} \frac{1}{A_C} \int_C \big(\alpha_t\, T_t(x) + \alpha_s\, T_s(x)\big)\, dA, \tag{5}$$

where A_C is the area of the cut. The weights applied to this objective depends on the material's tensile and shear strength.

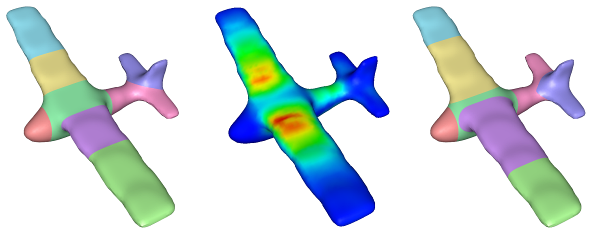


Figure 7: *A naive partition may place cuts in structurally unsound areas (left). A structural analysis reveals portions of the model under high tension (middle), leading Chopper to select cuts away from the highest-stress areas (right).*

Fragility: Even if cuts do not pass through high-stress areas of the original model, the resulting parts may still be fragile due to thin fins or bridges that connect two larger parts (Figure 8). We assume that the initial object is structurally sound and try to avoid creating structures that introduce fragility. The necessary condition for this structure to result from a cut is the existence of one point on the structure whose tangential plane is nearly parallel to the cut plane π and whose distance to π is sufficiently small. To eliminate these cases, we identify the set $\mathcal{S}$ of vertices on the part surface that satisfy at least one of two conditions. The first condition is that the vertex normal n_p is sufficiently parallel to the cut plane's normal n: i.e., $n_p \cdot n > T_n$, where we use $T_n = 0.95$. To evaluate the second condition (primarily for parts with bad normals), we check which one-ring edges point away from the normal and which point towards the normal. If either set of edges is not contiguous, or is empty, we add the vertex to $\mathcal{S}$.

Then we identify a subset of points $\mathcal{S}_{\text{fragility}} \subset \mathcal{S}$ that lead to high fragility. From each point in $\mathcal{S}$, we shoot a ray towards the plane. If it does not intersect the mesh before hitting the plane and if the distance to the plane is below a manually chosen threshold $\tau_{\text{fragility}}$, then we add the point to $\mathcal{S}_{\text{fragility}}$. We use 1.5 times the connector diameter as the threshold for all test cases. Tying the threshold to the connector diameter ensures that there is enough room in subsequent cross sections to place a connector as a side effect of the fragility objective. Note that $\mathcal{S}$ only needs to be computed once per plane direction instead of once per plane. The final fragility objective is:

$$f_{\text{fragility}}(\mathcal{T}) = \begin{cases} 0 & \text{if } \mathcal{S}_{\text{fragility}} = \varnothing, \\ \infty & \text{otherwise.} \end{cases} \tag{6}$$

3.2.4 Aesthetics

Seam Unobtrusiveness: We would like seams to run through parts of the surface where they are least likely to be visible or distracting. For every point on the part surface, we compute a penalty p for running the seam through that point. This penalty is by default based on ambient occlusion (we want seams in self-occluded areas — see Figure 10), but we can also use texture edges if the part is textured, or user input painted onto the model (Figure 11). In all cases, we compute the cost $\varepsilon(C)$ of the seam S on a cut C as the normalized integral of p along the seam:

$$\varepsilon(C) = \frac{1}{d_0} \int_{\partial C} p\, dx, \tag{7}$$

where d_0 is the diagonal length of the input model's OBB. The seam objective is defined as the sum of the the seam costs of each existing C in $\mathcal{T}$ plus the estimated seam costs for each unfinished part P whose $\Theta(P) > 1$:

$$f_{\text{seam}} = \frac{1}{\Theta_0} \Big(\sum_{C\in\mathcal{T}} \varepsilon(C) + \sum_{P\in\mathcal{T}} \big(\Theta(P) - 1\big)\, \hat{\varepsilon}(P) \Big), \tag{8}$$

where $\hat{\varepsilon}(P)$ is the estimated seam cost for each future cut on P. We found that the ε of the most recent seam on P is a good estimate of $\hat{\varepsilon}(P)$ in practice.

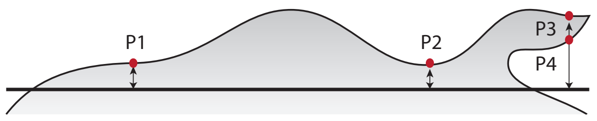


Figure 8: *Fins (e.g., P1) and bridges (e.g., P2) are locations of potential structural weakness. Our objective distinguishes P1 and P2 from locations such as P3, which is excluded from consideration because it hits P4 before it hits the cutting plane.*

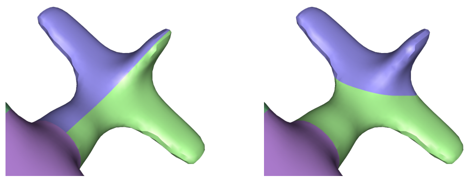


Figure 9: *Without taking part fragility into account, the yellow part has a thin fin near the tail (left). The fragility objective shifts the cutting plane so that the tail is solidly attached (right).*

One danger with this seam objective is that it favors planes that cut off only tiny parts. Thus, it is important that its weight be balanced against the utilization objective (Equation 3), which bears the responsibility of preventing such small cuts.

Symmetry: To encourage symmetric cuts (Figure 12), we first detect the model's dominant reflective symmetry by computing the Hausdorff distances between the model and its reflections with respect to uniformly sampled planes (500 directions) passing through the model's center of mass. To improve robustness to tessellation, the distance measure is based on a uniform sampling of 10,000 points on the model's surface. The plane resulting in the smallest symmetry distance is considered the symmetry plane π_{symm}, as long as that distance is below 0.1 times the bounding box diagonal.

The symmetry objective is computed by sampling a set of points $\mathcal{P}$ on all the planes of a candidate BSP tree, and computing the root-mean-square (RMS) distance between each point and the closest point under reflection about the symmetry plane. This is normalized by the diagonal of the initial bounding box d_0:

$$f_{\text{symmetry}} = \frac{1}{d_0} \underset{p \in \mathcal{P}}{\text{RMS}} \left(\min_{q \in \mathcal{P}} \left\| p - \text{reflect}(q, \pi_{\text{symm}}) \right\| \right). \tag{9}$$

3.3 Connector Placement

After the beam search, we are left with a number of potential connector locations on each cross-section. Some of them may interfere with each other and not all are necessary: having too many connectors can make the part hard to clean (e.g., of support material used in manufacturing). We use simulated annealing [Kirkpatrick et al. 1983] to place the connectors, minimizing the following objective:

$$w_I \sum_{i,j} I_{i,j} + \sum_i \frac{C_i}{\varepsilon + \max(0, c_i)}, \tag{10}$$

where $w_I = 10^{10}$ and $I_{i,j}$ is 1 if connectors i and j interfere and 0 otherwise. C_i is the area of the cross-section connected component i and c_i is the area "covered" by the connectors on that connected component, computed as follows: each connector covers a certain area πr_C^2. For large cross-sections, we use r_C equal to 20 times the connector radius, but to place more connectors on smaller cross-sections, we clamp r_C to be at most $0.5\sqrt{C_i}$. For any two connectors at distance d from each other with $d < 2r_C$ of each other, we subtract $2\pi(r_C - d/2)^2$ from c_i. This ensures that connectors are spaced roughly r_C apart. We set ε to 10^{-5} to have a high penalty when a cross-section connected component has no connectors. For some i, c_i can be below zero, and for each such c_i we add $-c_i/C_i$ to the objective, to guide the optimization towards a better solution.

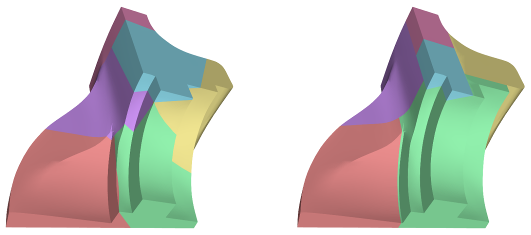


Figure 10: *If seam cost is not penalized, Chopper prefers larger cut cross-sections due to the connector objective (left). With the seam cost penalty, less obtrusive cuts are chosen (right).*

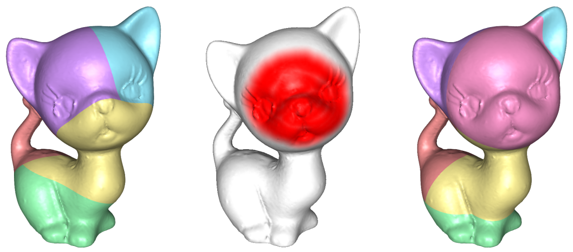


Figure 11: *Without user guidance, Chopper may cut through salient areas (left). The user may indicate an area that should not be cut (middle), and f_{seam} encourages partitions to avoid it (right).*

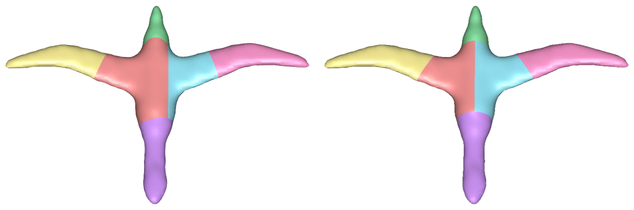


Figure 12: *Taking into account the dominant reflective symmetry of a model discourages off-center cuts (left) and results in a more pleasing partition (right).*

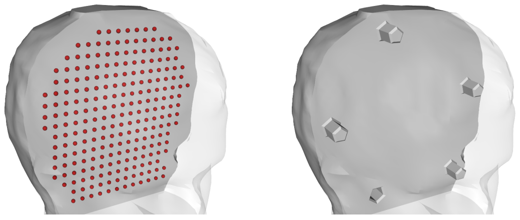


Figure 13: *Left: sampled potential connector locations on a cross-section. Right: final optimized connector placement.*

A state during simulated annealing is simply a subset of all possible connectors. A random mutation performs one of the following actions (with equal probability): (1) picks a connector from among all possible ones and toggles its presence in the state, or (2) removes a random existing connector and adds a different one. We run 15,000 iterations at zero temperature to initialize, and then use a linear cooling schedule and 300,000 iterations for the annealing to improve the placement. A sample result is shown in Figure 13.

4 Results

We have evaluated Chopper on a number of models, including mechanical parts, man-made art objects, and organic forms. Figures 1 and 14 show results printed on different 3D printers, ranging from commercial to hobbyist-grade. The chair and kitten were printed on a Fortus 400mc, the helmet and fertility models on an Objet Connex500, and the armadillo on a Bits from Bytes BFB-3000.

We found that the commercial printers were able to produce parts that fit together reasonably well, but the Bits from Bytes printer produced significant distortion in the printed parts, due to uneven cooling during printing and deformation of parts under their own

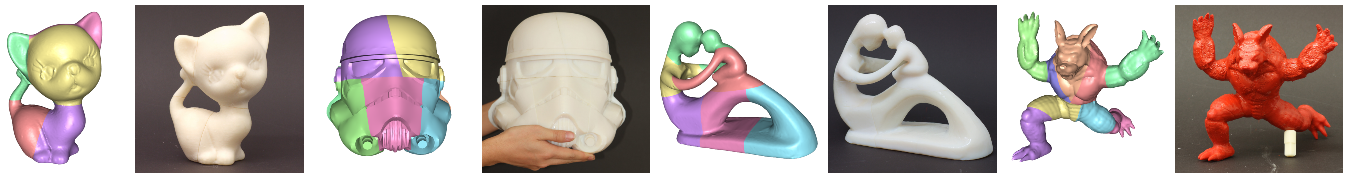


Figure 14: *4 Models partitioned by Chopper, then 3D-printed and assembled.*

weight. This required the use of a smaller effective working volume than the maximum of which the device was capable, further motivating the need for a tool such as Chopper. We also attempted to produce results using an extremely low-cost printer (Makerbot Thing-O-Matic), but found that the printed parts had sufficient deformation to be unusable. We expect that changes to the driver software of such printers could result in more usable prints.

Figure 18 shows decompositions and statistics for 16 models. We report wall-clock timings for an Intel Core i7-920 (2.66 GHz). Parallelism is achieved by evaluating batches of cuts with different normals simultaneously (see Algorithm 1); this yields an average speedup of 5.9 across eight hardware threads (four cores). All results in this table were produced with the following weights:

$$\begin{array}{llllll}\alpha_{\text{part}} & = 1 & \alpha_{\text{util}} & = 0.05 & \alpha_{\text{connector}} & = 1 \\ \alpha_{\text{fragility}} & = 1 & \alpha_{\text{seam}} & = 0.1 & \alpha_{\text{symmetry}} & = 0.25\end{array}$$

Note that only α_{util}, α_{seam}, and α_{symmetry} need to be carefully adjusted relative to $\alpha_{\text{part}} = 1$ for the application: $f_{\text{fragility}}$ is either 0 or ∞ and $f_{\text{connector}}$ primarily influences the result when it is ∞. There are a few other constants, such as T_n for the fragility objective that can be tuned independently (although we found no need to do this) and others, such as the number of planes that control the tradeoff between the result quality and performance. For our tests, we have manually selected an appropriate scale for each model, though Chopper can of course be applied for any ratio of model size to working volume (Figure 15).

Figure 16 shows running time and the resulting objective value, as a function of beam width. The running time is roughly linear in beam width, as confirmed by the slope near 1 in this log/log plot (red line). The objective value typically decreases little in absolute terms (black line), but even this modest numerical improvement results in noticeably better partitions.

5 User Study

To test the plausibility of the partitions produced by Chopper, we have compared its output to the partitions produced by humans. The users were constrained to BSP-tree partitions, as is Chopper, and were given a tool allowing them to add or remove cuts, as well as see information about the current partition. The information provided to the users included the values of all the objective functions used by Chopper, though we observed that in practice users only referred to the number-of-parts estimates. Users were not given any explicit instructions about what kinds of partitions were to be preferred: they were only instructed to partition the model into pieces that fit within the provided printing volume.

There were 6 participants in our user study. Each was given 2 models on which to practice, then partitioned a further 4 or 5 models (kitten, armadillo, chair, fertility, ant). Figure 17 shows results on one model, and full results are provided as supplemental material.

Chopper was consistently able to produce partitions with equal or lower numbers of parts than did humans, always achieving lower total values of its objective functions. This suggests that the optimization itself (as opposed to the design of objective functions) is achieving its goals: matching any "better" cuts produced by humans would likely require changing the objective functions, not the beam-search optimization. Though we have not conducted a formal evaluation of whether Chopper's partitions are generally preferred to the users', or vice versa, informally we observe that there are few cases in which the users' results are obviously better, and those cases require significantly more parts (e.g., compare User #4's 7-part armadillo to Chopper's 5-part result).

6 Conclusion, Limitations, and Future Work

We have explored an automatic and practical method capable of partitioning a wide range of models for 3D printing. Experimental results confirm that the partitions Chopper produces can be printed on a variety of 3D printers and assembled.

Our investigation opens up many avenues for future exploration. For example, Chopper currently does handle connectors of different types and sizes for a single object. In our results, the connector size for a few models with thin features was reduced from the default, but Chopper could be extended to automatically consider multiple connector types and sizes and pick the best ones based on the cross-section, surrounding geometry, and structural considerations.

A more challenging problem is to relax the restriction that our partitions be BSP trees. While any partition can be represented as a tree, our restriction that cuts must be planar can prevent desirable partitions. The user study indicated that some users feel hampered by this restriction in placing their cuts: for example, they want to cut the kitten's head off independently from its tail, or to place more natural-looking cuts that follow curved surface features.

Any algorithm accommodating non-planar cuts will have to satisfy the same objectives we have identified, in particular guaranteeing printability and assemblability. It is possible to test for these properties during the search, though in some cases the check is more complex than for BSP trees (e.g., testing for assemblability requires checking that all normals of the cutting surface lie within in a hemisphere, rather than being guaranteed by construction). A more serious problem is organizing the search over possible cutting surfaces in an efficient way. There are many more possibilities to explore if cutting surfaces may be non-planar, and enumerating plausible candidates is a challenging avenue for future work.

Although partitioning models is necessary for objects larger than the printing volume, it can be beneficial for smaller objects as well, even if they could in principle be printed as a single part. For example printing a hollow object as a single part is impossible on most printers because support material cannot be removed. Using partitioning to allow hollow printing, minimizing support material and other objectives are intriguing directions for future research.

In our experience with Chopper, it has produced many good partitions, but some have room for improvement. Because the precise desires of a user are difficult to express as objective weights, a reasonable solution would be to put the user in the loop in a more fundamental way. In our prototype, the user can provide guidance by marking rough desirable partition regions on the surface, but Chopper makes the final decisions. Instead, the algorithms in

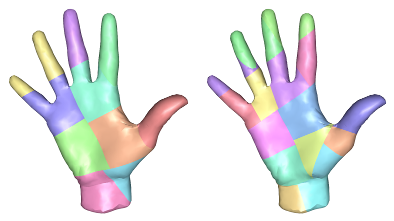


Figure 15: *Reducing the working volume causes this hand model to be partitioned into 10 and 18 parts (cf. original in Figure 18, with 7 parts).*

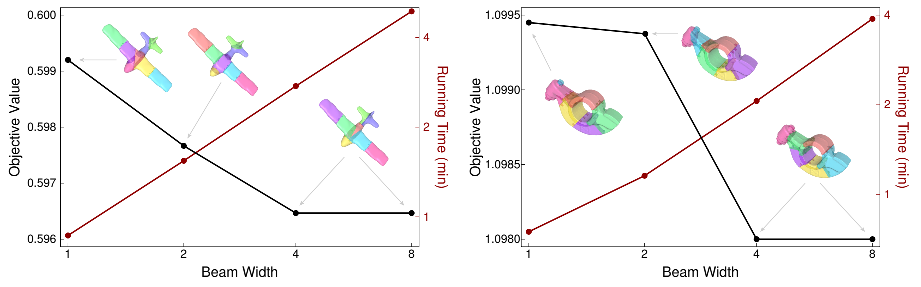


Figure 16: *Increasing beam width results in decreased objective values, and hence better partitions, at the cost of increased running time.*

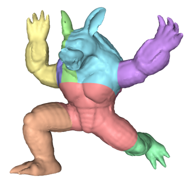
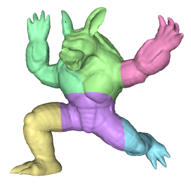
Chopper	User #1	User #2	User #3	User #4	User #5	User #6
						
5 parts, 139 s	8 parts, 504 s	8 parts, 728 s	6 parts, 346 s	7 parts, 267 s	9 parts, 540 s	7 parts, 252 s

Figure 17: *Sample results from a user study comparing Chopper to humans. Full results are provided as supplemental material.*

Chopper could interactively provide suggestions for cutting planes and feedback (such as min-cover or printing time estimates), but the user would be responsible for the final cutting plane. Which solution is ultimately superior is a subject for future investigation.

Acknowledgments

We thank AIM@SHAPE for providing the models and Jonathan Shewchuk for Triangle [Shewchuk 1996] that we used to tessellate the cut surfaces. We thank the members of the Princeton Graphics Group, especially Sema Berkiten, Vladimir Kim, Tianqiang Liu, Jingwan Lu, and Thiago Pereira, as well as Sebastian Koch, Pitchaya Sitthi-Amorn and Justin Lan. This work is partially supported by NSF grant CCF-1012147, IIS-1116296 and ISTC-VC.

References

AGRAWALA, M., PHAN, D., HEISER, J., HAYMAKER, J., KLINGNER, J., HANRAHAN, P., AND TVERSKY, B. 2003. Designing effective step-by-step assembly instructions. *ACM Trans. Graphics (Proc. SIGGRAPH) 22*, 3, 828–837.

ALEXANDER, P., ALLEN, S., AND DUTTA, D. 1998. Part orientation and build cost determination in layered manufacturing. *Computer-aided Design 30*, 5, 343–356.

ATTENE, M., KATZ, S., MORTARA, M., PATANE, G., SPAGNUOLO, M., AND TAL, A. 2006. Mesh segmentation - a comparative study. In *Proceedings of the IEEE International Conference on Shape Modeling and Applications 2006*, 7–.

AUPPERLE, L., CONN, H., KEIL, J., AND O'ROURKE, J. 1988. Covering orthogonal polygons with squares. In *Proc. Communication, Control and Computing*, 97–106.

BICKEL, B., BÄCHER, M., OTADUY, M. A., LEE, H. R., PFISTER, H., GROSS, M., AND MATUSIK, W. 2010. Design and fabrication of materials with desired deformation behavior. *ACM Trans. Graphics (Proc. SIGGRAPH) 29*, 3, 63:1–63:10.

CHAZELLE, B. 1981. Convex decompositions of polyhedra. In *Proc. ACM Symposium on Theory of Computing*, 70–79.

CHEN, X., GOLOVINSKIY, A., AND FUNKHOUSER, T. 2009. A benchmark for 3D mesh segmentation. *ACM Trans. Graphics (Proc. SIGGRAPH) 28*, 3, 73:1–73:12.

DICKINSON, J., AND KNOPF, G. 1998. Serial packing of arbitrary 3D objects for optimizing layered manufacturing. In *Proc. SPIE*, vol. 3522, 130–138.

EGEBLAD, J., NIELSEN, B. K., AND BRAZIL, M. 2009. Translational packing of arbitrary polytopes. *Computational Geometry 42*, 4, 269 – 288.

FUCHS, H., KEDEM, Z. M., AND NAYLOR, B. F. 1980. On visible surface generation by a priori tree structures. In *Computer Graphics (Proc. SIGGRAPH)*, vol. 14, 124–133.

HAO, J., FANG, L., AND WILLIAMS, R. 2011. An efficient curvature-based partitioning of large-scale stl models. *Rapid Prototyping Journal 17*, 2, 116–127.

HAŠAN, M., FUCHS, M., MATUSIK, W., PFISTER, H., AND RUSINKIEWICZ, S. 2010. Physical reproduction of materials with specified subsurface scattering. *ACM Trans. Graphics (Proc. SIGGRAPH) 29*, 3, 61:1–61:10.

HILDEBRAND, K., BICKEL, B., AND ALEXA, M. 2012. crdbrd: Shape fabrication by sliding planar slices. In *Computer Graphics Forum (Proc. Eurographics)*, vol. 31, 583–592.

IKONEN, I., BILES, W., LEWIS, J., KUMAR, A., AND RAGADE, R. 1998. GARP: Genetic algorithm for part packing in a rapid prototyping machine. In *Proc. SPIE*, vol. 3517, 54.

JANUSZEWSKI, J. 2009. A note on covering a square of side length $2+\varepsilon$. *American Mathematical Monthly 116*, 2, 174–178.

KIRKPATRICK, S., GELATT JR, C., VECCHI, M., AND MCCOY, A. 1983. Optimization by simulated annealing. *Science 220*, 4598, 671–679.

LAU, M., OHGAWARA, A., MITANI, J., AND IGARASHI, T. 2011. Converting 3D furniture models to fabricatable parts and connectors. *ACM Trans. Graphics (Proc. SIGGRAPH) 30*, 4, 85:1–85:6.

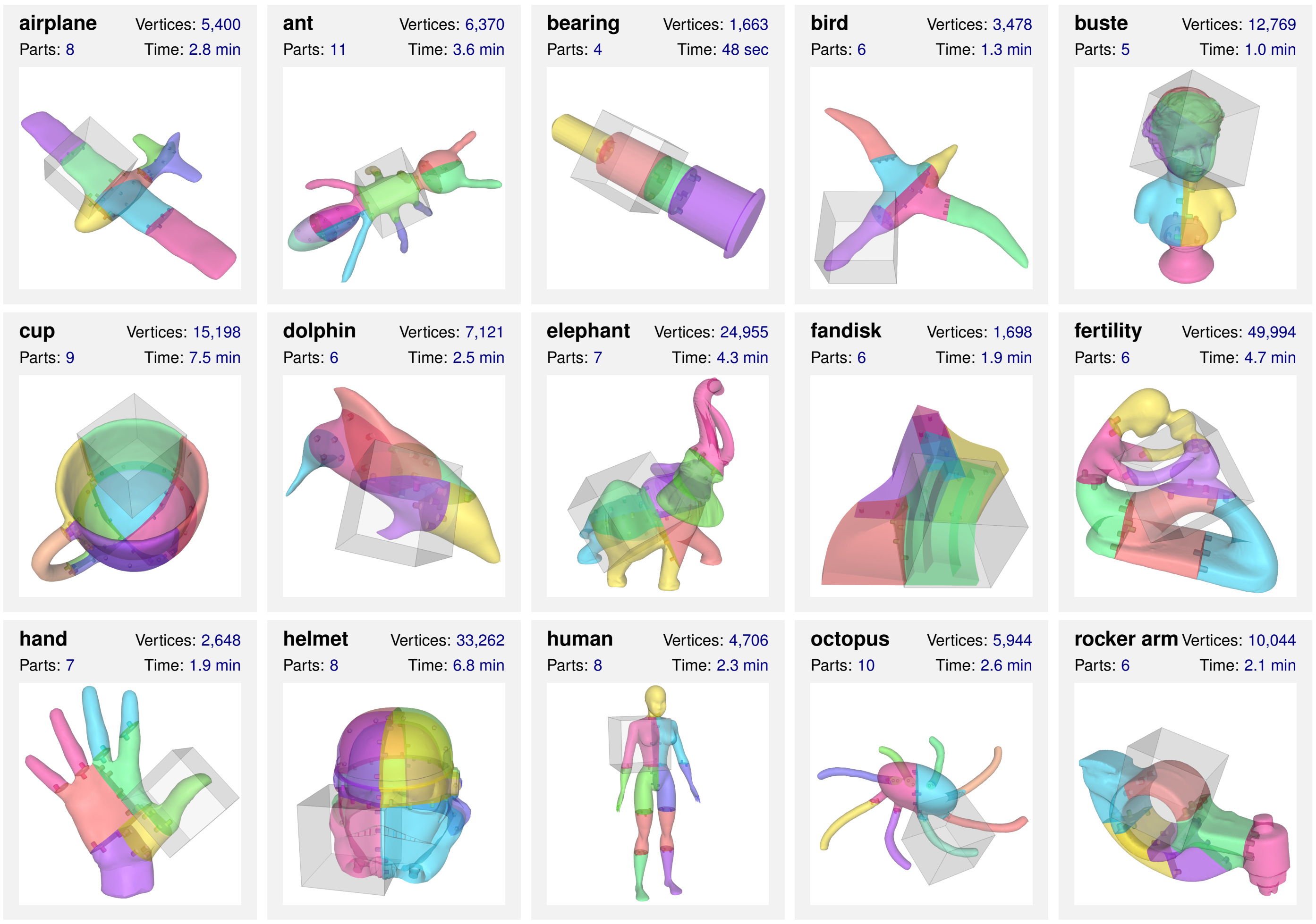

airplane Vertices: 5,400 Parts: 8 Time: 2.8 min
ant Vertices: 6,370 Parts: 11 Time: 3.6 min
bearing Vertices: 1,663 Parts: 4 Time: 48 sec
bird Vertices: 3,478 Parts: 6 Time: 1.3 min
buste Vertices: 12,769 Parts: 5 Time: 1.0 min
cup Vertices: 15,198 Parts: 9 Time: 7.5 min
dolphin Vertices: 7,121 Parts: 6 Time: 2.5 min
elephant Vertices: 24,955 Parts: 7 Time: 4.3 min
fandisk Vertices: 1,698 Parts: 6 Time: 1.9 min
fertility Vertices: 49,994 Parts: 6 Time: 4.7 min
hand Vertices: 2,648 Parts: 7 Time: 1.9 min
helmet Vertices: 33,262 Parts: 8 Time: 6.8 min
human Vertices: 4,706 Parts: 8 Time: 2.3 min
octopus Vertices: 5,944 Parts: 10 Time: 2.6 min
rocker arm Vertices: 10,044 Parts: 6 Time: 2.1 min

Figure 18: *Results of Chopper on a variety of models (with target working volume shown in gray). Note the connectors on each cross-section.*

LIEN, J., AND AMATO, N. 2007. Approximate convex decomposition of polyhedra. In *Proc. ACM Symposium on Solid and Physical Modeling*, 121–131.

LOWERRE, B. T. 1976. *The harpy speech recognition system*. PhD thesis, Carnegie Mellon University.

MCCRAE, J., SINGH, K., AND MITRA, N. J. 2011. Slices: A shape-proxy based on planar sections. *ACM Trans. Graphics (Proc. SIGGRAPH Asia) 30*, 6, 168:1–168:12.

MEDELLIN, H., LIM, T., CORNEY, J., RITCHIE, J., AND DAVIES, J. 2007. Automatic subdivision and refinement of large components for rapid prototyping production. *Journal of Computing and Information Science in Engineering 7*, 3, 249–258.

MORI, Y., AND IGARASHI, T. 2007. Plushie: An interactive design system for plush toys. *ACM Trans. Graphics (Proc. SIGGRAPH) 26*, 3, 45:1–45:8.

SAUL, G., LAU, M., MITANI, J., AND IGARASHI, T. 2011. SketchChair: An all-in-one chair design system for end users. In *Tangible, Embedded, and Embodied Interaction*, 73–80.

SHAMIR, A. 2008. A survey on mesh segmentation techniques. *Computer Graphics Forum 27*, 6, 1539–1556.

SHEWCHUK, J. R. 1996. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*. 203–222.

SHEWCHUK, J. 1998. Tetrahedral mesh generation by Delaunay refinement. In *Proc. Symposium on Computational Geometry*, 86–95.

SOIFER, A. 2006. Covering a square of side $n + \varepsilon$ with unit squares. *Journal of Combinatorial Theory A 113*, 2, 380–383.

STAVA, O., VANEK, J., BENES, B., CARR, N., AND MECH, R. 2012. Stress relief: Improving structural strength of 3d printable objects. *ACM Trans. Graphics (Proc. SIGGRAPH) 31*, 4, 1–8.

THRIMURTHULU, K., PANDEY, P. M., AND REDDY, N. V. 2004. Optimum part deposition orientation in fused deposition modeling. *Machine Tools and Manufacture 44*, 6, 585 – 594.

TOUSSAINT, G. 1983. Solving geometric problems with the rotating calipers. In *Proc. IEEE Melecon*, vol. 83, A10.

UMETANI, N., KAUFMAN, D. M., IGARASHI, T., AND GRINSPUN, E. 2011. Sensitive couture for interactive garment editing and modeling. *ACM Trans. Graphics (Proc. SIGGRAPH) 30*, 4, 90:1–90:12.

WHITING, E., OCHSENDORF, J., AND DURAND, F. 2009. Procedural modeling of structurally-sound masonry buildings. *ACM Trans. Graphics (Proc. SIGGRAPH Asia) 28*, 5, 112:1–112:9.

XIN, S., LAI, C.-F., FU, C.-W., WONG, T.-T., HE, Y., AND COHEN-OR, D. 2011. Making burr puzzles from 3D models. *ACM Trans. Graphics (Proc. SIGGRAPH) 30*, 4, 97:1–97:8.