# Level-Set-Based Partitioning and Packing Optimization of a Printable Model

Miaojun Yao[*] Zhili Chen[*] Linjie Luo[†] Rui Wang[‡] Huamin Wang[*]
The Ohio State University[*] Snapchat[†] Zhejiang University[‡]

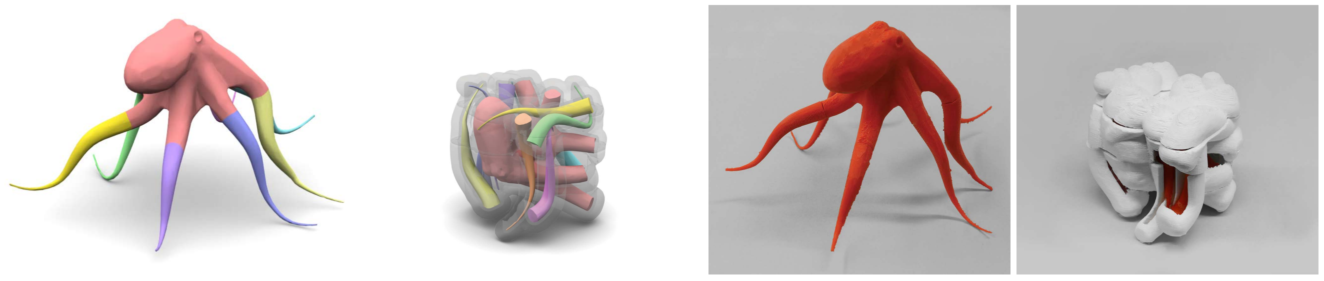


**Figure 1:** *The octopus example. Our constrained partitioning optimization system automatically generates the partitioning of this octopus model, which contains nine pieces as shown in (a) and (c). These nine pieces can be packed into a box container as shown in (b) and (d), which occupies only 10.4 percent of its original container volume.*

## Abstract

As the 3D printing technology starts to revolutionize our daily life and the manufacturing industries, a critical problem is about to emerge: how can we find an automatic way to divide a 3D model into multiple printable pieces, so as to save the space, to reduce the printing time, or to make a large model printable by small printers. In this paper, we present a systematic study on the partitioning and packing of 3D models under the multi-phase level set framework. We first construct analysis tools to evaluate the qualities of a partitioning using six metrics: stress load, surface details, interface area, packed size, printability, and assembling. Based on this analysis, we then formulate level set methods to improve the qualities of the partitioning according to the metrics. These methods are integrated into an automatic system, which repetitively and locally optimizes the partitioning. Given the optimized partitioning result, we further provide a container structure modeling algorithm to facilitate the packing process of the printed pieces. Our experiment shows that the system can generate quality partitioning of various 3D models for space saving and fast production purposes.

**Keywords:** Shape partitioning, mesh segmentation, 3D packing, 3D printing, shape optimization, multi-phase level set.

**CR Categories:** I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling.

[*]e-mail: {yaom, chenzhi, whmin}@cse.ohio-state.edu
[†]e-mail: linjie.luo@gmail.com
[‡]e-mail: rwang@cad.zju.edu.cn

## 1 Introduction

As 3D printers become more versatile and affordable every year, it is expected that soon they will transform our daily life and the manufacturing industries. Interestingly, the 3D printing technology not only revolutionizes how things are made, but also how things are designed. In the past, a designer/engineer/architect/artist must design many parts and their assembling process, through which the parts are put together to form the desired object. Today, the parts are no longer needed in the design of a 3D model, thanks to the fact that 3D printers can now manufacture objects as a whole. This benefit, however, brings new troubles to us when we still need to divide a 3D model and print its pieces separately, for several reasons.

- **Printability.** A model is not printable if it cannot fit into the build volume of a 3D printer. A large printer, such as Objet1000 made by Stratasys, has a large volume, but it is expensive. Small pieces can be built easily by small printers.
- **Fast production.** It takes hours or even days to print a large model. By manufacturing the model pieces separately in parallel and reducing their heights during printing, we can save the printing time, especially for powder-based printers.
- **Space saving.** By arranging the pieces of an object into a packed state, we can reduce their occupied space when the object is not in use. This provides us a cost-effective storage and transportation solution for multiple printed objects.

To achieve these goals, we believe that a good partitioning should have six important qualities.

- **Minimum stress load.** The partitioning should avoid high stress regions within a model. This is to prevent pieces from being easily separable, due to large tension or shear loads.
- **Surface detail alignment.** The pieces may not match perfectly due to the imprecision of a printer. To make seams less noticeable on boundary surfaces, we can align them with surface details, where people expect to see discontinuities.
- **Minimal interface area.** A seam between two pieces can also be reduced by minimizing the contact area. The smaller the area is, the fewer the mismatches are likely to happen.
- **Minimal packed size.** The partitioning should allow the pieces to be tightly organized into a packed state for storage

or transportation. A packed state can also be used to speed up 3D printing, by minimizing the heights of the pieces.

- **Printability.** The partitioning should ensure the printability of each piece. Specifically, each piece should not exceed the build volume of a 3D printer, nor should it contain any thin feature below the printing resolution.
- **Assembling.** There should exist no locking issue preventing the pieces from being assembled into the original state, or the packed state.

We propose to study the partitioning of a 3D model under the level set representation. Compared with other geometric representations, such as meshes or point clouds, the level set representation trivially handles topological changes without remeshing, and it is free of sampling artifacts. There are two additional reasons convincing us the use of the level set representation in this work. First, its underlying signed distance field allows us to perform fast and easy collision handling, which is needed frequently in our simulation-based packing solver. Second, as a volumetric representation, it is more compatible with the layering process commonly used in the 3D printing technology, which successively applies printing material in layers to form the volume of an object. Previous research [Nooruddin and Turk 2003; Telea and Jalba 2011] also indicates that volumetric representations are suitable for geometric processing.

While both shape partitioning and shape packing have been studied in various areas before, they are often handled separately and sequentially in many previous systems [Wu et al. 2014; Vanek et al. 2014]. Doing this limits the packing quality, as the partitioning should still be adjusted to reduce the packed space even further. Unfortunately, it is highly difficult to optimize the partitioning and the packing together using a single objective function, since the partitioning is formulated in the original space while the packing is defined in the packed space. As far as we know, there exists no effective solution to this unique optimization problem.

Instead of optimizing the partitioning and the packing jointly, we propose to process them in a separate yet interleaved fashion. Specifically, we develop a *constrained optimization* system that iteratively refines the qualities of the partitioning for a 3D model in two steps. In the variational step, it improves the intrinsic qualities of the partitioning, including minimum stress load, surface detail alignment, and minimal interface area. In the packing step, it then improves the packing quality of the partitioning to organize the pieces tightly in a packed state. The system runs the two steps iteratively, until both objectives can be achieved sufficiently well. To ensure the usability of the result, we further introduce a constraint enforcement step between the two steps. Once the system detects a constraint violation, it performs local correction to fix it. Since the optimized result is sensitive to packing initialization, we run the whole optimization process multiple times with random initialization seeds, and select the final result to be the one with the highest qualities. Thanks to a number of acceleration approaches, such as multithreading, our system can finalize the partitioning result within 12 minutes. The experiment examples summarized in Table 1 show that our interleaved optimization strategy effectively reduces 12.9 to 44.3 percent more space than optimizing the partitioning and the packing sequentially only once.

Besides the development of this new optimization system, our other technical contributions are:

- **Analysis tools**. We evaluate the qualities of a partitioning by new analysis tools, including collision detector, packing solver, and locking detector. All of these tools are formulated under the multi-phase level set framework.
- **Level set methods**. Based on the analysis, we present level set methods to improve the partitioning qualities. While some are developed using the variational approach, others modify level set functions directly and locally.
- **Container structure**. We present a container structure modeling algorithm to simplify the packing process and protect the pieces from collision damage within a container.

The developed system can be used for different purposes. For space saving, the system arranges the pieces into a packed state, whose volume is only 10.4 percent of the original volume, as Figure 1 shows. For fast production, the system reorganizes the pieces within the volume of a FDM printer, so that the printing time can be reduced by approximately half an hour, as Figure 11 shows.

## 2 Related Work

**Level set methods.** The idea of level set methods was initially developed by Osher and Sethian [1988] as a simple yet versatile tool to track the motion of an interface under advection. Level set methods are known for their flexibility in handling topological changes, such as shape merging or splitting. In computer graphics, level set methods were found to be useful for collision detection and handling as well, since a point can be quickly tested to know whether it is within a volume or not, as Guendelman and collaborators [2003] pointed out. While level set methods have demonstrated its power in many applications, they are often associated with numerical dissipation, especially near high curvature regions. A popular solution to this problem is the particle level set method [Enright et al. 2002]. Fortunately, we do not have to worry too much about it, since our goal is to optimize the internal interfaces rather than the external boundary of a model, and we actually prefer smooth interfaces.

Multiple level set functions can be used to represent multiple phases within a 3D volume. An interesting question is how to fix overlapping or vacuum regions, when these functions are evolved independently. Zhao and colleagues [1996] established a theoretical variational formulation to remove these regions using Lagrangian multipliers. A faster solution [Ruuth 1998; Smith et al. 2002; Losasso et al. 2006] is to directly modify the function values in problematic regions by projection steps. Instead of using multiple functions, Kim [2010] and Saye and Sethian [2011] proposed to represent multiple phases by an indicator map and a single level set function. In this work, we still use multiple level set functions, since 3D printing needs high accuracy.

**Printable shape design.** Due to the importance of the 3D printing technology, researchers have studied a variety of problems relevant to printable shapes in recent years, including strength improvement [Stava et al. 2012; Wang et al. 2013; Zhou et al. 2013; Lu et al. 2014], articulated objects [Bächer et al. 2012; Calì et al. 2012], masonry models [Panozzo et al. 2013; Whiting et al. 2012], inverse shape design [Chen et al. 2014], support structures [Dumas et al. 2014], spinnable shapes [Bacher et al. 2014], and stackable shapes [Li et al. 2012]. Among them, shape partitioning is a relatively less studied topic. Xin and collaborators [2011] and Song and colleagues [2012] studied the design of a printable puzzle, by splitting a model into interlocking parts. Lau and collaborators [2011] proposed to automatically divide a furniture model into pieces with connectors. Hu and colleagues [2014] developed a novel method to decompose a model into pyramidal shapes for optimal layered 3D printing. Given a self-supporting structure and its partitioning, Deuss and collaborators [2014] investigated the physical construction process by establishing a temporal assembling sequence. Luo and colleagues [2012] used planar partitioning to fit the pieces of a large model into the build volume of a small printer. Recently, Vanek and colleagues [2014] developed a system to reduce the printing time of a 3D model. In their system, partitioning and packing are processed sequentially, so the result can contain more pieces

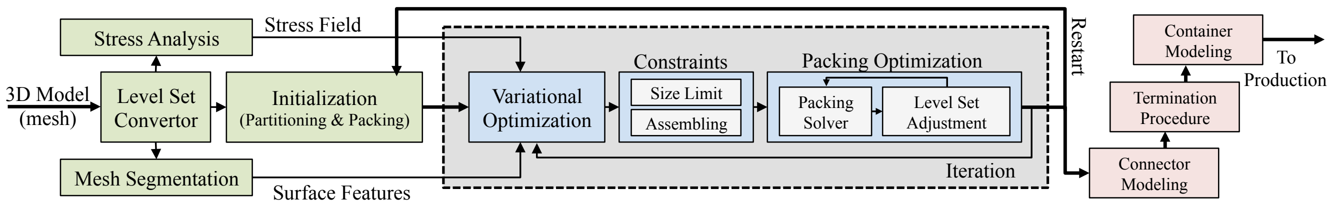


**Figure 2:** *The system pipeline. The key component of this system is the partitioning optimization process, which uses three steps to iteratively optimize the partitioning of a 3D model. The whole system is formulated under the multi-phase level set framework.*

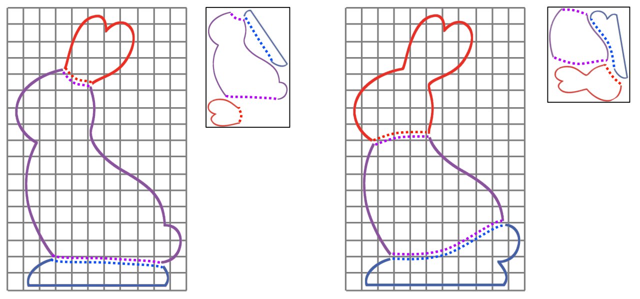
than necessary. Their system also uses a height-field-based representation, which is difficult to be generalized for other purposes, such as space saving.

**Image and mesh segmentation.** Computer vision researchers have extensively studied image and volume segmentation, and they have developed a variety of techniques, including the ones using level set methods [Vese and Chan 2002; Gibou and Fedkiw 2005]. Since image segmentation is typically used to segment objects from an image for easier recognition purposes, its goal is different from ours. Many image segmentation techniques have been borrowed by graphics researchers to handle the partitioning of a polygonal mesh. A recent survey on mesh segmentation can be found in [Shamir 2008]. Although the goals of mesh segmentation are different from ours, we can make our result more meaningful, by incorporating mesh segmentation results into partitioning optimization.

**Bin packing.** Bin packing is an NP-hard problem. Compared with bin packing of spheres and convex polygons, 2D and 3D packing of irregular shapes is even more challenging. Many existing irregular shape packing algorithms are designed for 2D cases, including a hybrid method that combines simulated annealing with linear programming [Gomes and Oliveira 2006], genetic algorithms and heuristics [Junior et al. 2013], an iterative local search algorithm [Imamichi et al. 2009], and a heuristic approach [Yang and Huang 2013] designed for exhibiting features and relationships among similar objects. Unfortunately, extending these algorithms to handle 3D cases is not so straightforward. Egeblad and collaborators [2009] developed an algorithm to pack arbitrary $d$-dimensional polytopes with translational motions only. Dickinson and Knopf [1998] proposed a fast serial approach to pack arbitrary 3D models into a fixed container one after another. Imamichi and Nagamochi [2007] used spheres to represent irregular polyhedra for collision detection and solved the packing problem by minimizing the collisions among spheres and a fixed container. By predetermining a packing sequence and object orientations, Wu and colleagues [2014] studied bottom-left-front heuristics for packing objects into the build volume of a 3D printer. In our work, we study both fixed and adjustable container cases.

## 3 System Overview

The pipeline of our system is shown in Figure 2. The input to this system is a watertight 3D polygonal model. The system first performs a pre-computation step on it, including stress analysis, mesh segmentation, and initial partitioning generation. Given an initial partitioning as shown in Figure 3a, it then starts the optimization process to iteratively optimize the partitioning in three steps. In Step 1, the system uses a variational approach to improve the intrinsic partitioning qualities, including minimal stress load, detail alignment, and minimal interface area. In Step 2, it detects constraint violations and fix them locally in each level set. In Step 3, it runs a packing solver and adjusts the partitioning, to pack the pieces more tightly in the packed state. Since the optimization result is sensitive to packing initialization, the system randomly initializes



(a) Initial partitioning (b) Optimized partitioning

**Figure 3:** *A bunny example. We use multiple level set functions (in different colors, discretized over a regular grid) to represent the pieces of this bunny model. The goal of partitioning optimization is to improve the intrinsic qualities and to reduce the packed space simultaneously, by adjusting the interfaces (in dotted lines).*

the packed state and runs the optimization process multiple times, to obtain a high-quality partitioning result. Finally, the system builds auxiliary structures, including connectors and a container structure, and outputs the surface mesh of each piece.

**Termination procedure.** While each optimization step modifies level set functions to improve one or more partitioning qualities, it also compromises other qualities that were previously improved by other steps. For example, the variational optimization step can trigger new collisions in the packed state. Meanwhile, the packing optimization step can create sharp interface features and downgrade the intrinsic partitioning qualities. Both steps can cause new constraint violations that make the partitioning unusable. This is why the system runs the steps iteratively, to improve all of the partitioning qualities through their balance.

A critical question is: *how can we know a balance is reached*? In other words, how can we determine when to terminate the optimization process? Since the process does not have a single objective function, we cannot use the convergence of that function to define the termination condition. Our solution here is to prioritize the steps according to their importance. First, constraints are important and they must be strictly satisfied for usability purposes. Note that constraint violation is rare and its enforcement step does not affect the intrinsic qualities much. Second, the pieces must be strictly collision-free in the packed state, to make sure it is still valid. Finally, we claim that the intrinsic qualities are more important than the packing quality, because they directly affect the interfaces. Based on these assumptions, we propose to run and terminate the optimization process as follows. We first disable the packing optimization step and improve the intrinsic qualities alone. After the variational optimization step converges, we turn packing optimization on and run all of the three steps iteratively. Once the packing optimization step fails to remove the remaining collisions, we terminate the whole partitioning optimization process, restore the shape of

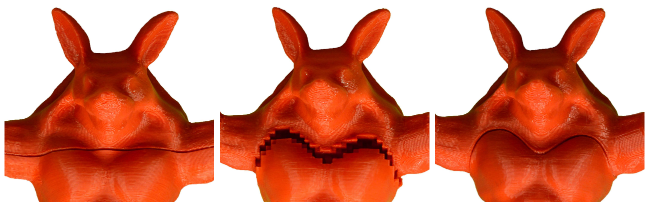

(a) Planes (b) Voxels (c) Level sets

**Figure 4:** *The printed results using different representations. The imprecision of a 3D printer causes inevitable seams among the pieces. The use of the level set representation makes the seams less visible, by aligning them with surface details as shown in (c).*

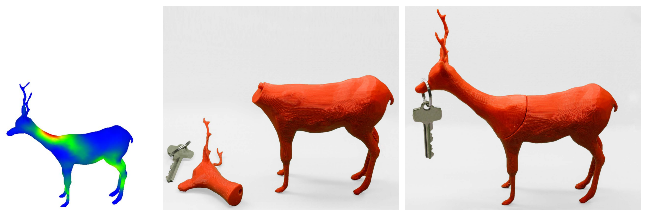

(a) Stress (norm) (b) Not using stress (c) Using stress

**Figure 5:** *A deer example. The deer head falls as (b) shows, if the connector is loose and partitioning optimization does not consider the stress. Once we add the stress field into the objective function, the new partitioning keeps the head still as (c) shows.*

each piece to the one before packing optimization, and finally use the same packing solver described in Subsection 5.2 to relax the pieces and expand the container, until the collisions are gone. By slightly sacrificing the packing quality, our approach ensures that the constraints are satisfied and the packed state is collision-free. We note that if the packing quality is more important, we can adjust the termination procedure to sacrifice the intrinsic qualities instead.

**Level set representation.** Our system is built upon the multiple level set representation. Let $i = 1, 2, ..., n$ be a partitioned piece of a 3D model. Its level set function $\phi_i$ defines the signed distance field to its boundary, such that: $\phi_i(\mathbf{x}) = 0$ means $\mathbf{x}$ is on the boundary; $\phi_i(\mathbf{x}) < 0$ means $\mathbf{x}$ is inside; and $\phi_i(\mathbf{x}) > 0$ means $\mathbf{x}$ is outside. Given these level set functions $\{\phi_1, \phi_2, ..., \phi_n\}$, we have $\min(\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), ..., \phi_n(\mathbf{x})) = \phi_0(\mathbf{x})$ for any $\mathbf{x}$ satisfying $\phi_0(\mathbf{x}) > 0$, in which $\phi_0$ is the level set function of the original model. This is because the original model is the union of its pieces and the distance from an outer point to the boundary of the model must be the same as the distance from that point to the pieces. We use a regular grid to discretize each level set function, as shown in Figure 3.

We choose the level set representation rather than others, because it can represent interface details for high-quality partitioning results. In comparison, planar cuts leave unnatural seams on the boundary surface of an armadillo model as Figure 4a shows, and they do not allow easy local partitioning adjustment. The use of voxels makes the result even worse, since voxelized pieces cannot be closely aligned due to printer errors, as shown in Figure 4b. In contrast, the smooth interfaces represented by the level set functions effectively reduce the visibility of the seams, as Figure 4c shows. Although meshes and particles can achieve similar results, they have difficulty in addressing topological change, collision detection, or sampling issues, as discussed in Section 1.

## 4 Precomputation and Initial Partitioning

The initialization process in our system contains three steps: stress analysis, mesh segmentation, and initial partitioning.

**Stress analysis.** To begin with, we tessellate the surface mesh input into a tetrahedral mesh and use the finite element method to perform stress analysis. In this analysis, we fix the bottom mesh vertices as boundary conditions and apply the gravity force as the only load. If needed, we can also use yield criteria, additional loads, or other analysis tools. After stress analysis, we build a Cauchy stress tensor field on the regular grid defining the level set function $\phi_0$ of the external surface. Using this field, we can identify high stress regions, where partitioning should be avoided. Figure 5a and 6a visualize the stress fields on the surfaces of two examples.

**Mesh segmentation.** Our system offers several automatic mesh segmentation options to users, including shape diameter, random walk, random cuts, normalized cuts, fitting primitives, k-means, and core extraction, as listed in [Chen et al. 2009]. We apply size constraints on each segmentation to ensure that no piece is too small or too large. Any segmentation that violates the size constraint will be discarded. In addition, our system provides geometric metrics, including interface area, cut perimeter, convexity and compactness, to help users evaluate each segmentation. If automatic segmentation results are not satisfactory, we allow users to manually segment the mesh by adding or editing surface curves.

Given a mesh segmentation result, we compute the mean curvature and the geodesic distance to the segmentation for every surface vertex, using the methods described in [Novotni and Klein 2002]. We sum these two values up and use extrapolation to form a scalar field within a narrow band around the boundary surface $|\phi_0| \leq b$, in which $b$ is the bandwidth. This field will be used in variational optimization to specify desired surface partitioning locations. In practice, we consider only negative mean curvatures, since partitioning at bumps may cause thin features over the surface of a model piece. Figure 6b shows the segmentation field of an airplane example.

**Level set initialization.** Our last initialization step is to convert mesh segmentation into an initial partitioning of the 3D model, by calculating the level set functions of its pieces. Given the segmented surface mesh, we assign each segmented region with an ID and label each triangle with its regional ID. To introduce some randomness, our system allows users to randomly merge two regions, to randomly split one region into two by a new cutting plane, or to randomly switch the ID of a triangle. After that, we compute the distance from each grid cell to its closest surface triangle and assign the grid cell with the ID of that triangle. Using these IDs, we now effectively segment the whole volume into multiple regions and we build their level set functions $\{\phi_1, ..., \phi_n\}$.

To simplify the constraint enforcement step, we use the same detection methods described in Subsection 5.3 to check whether a partitioning violates size limit or assembling constraints. If the size of a piece is beyond the limit, we use planes to divide it. If a cluster of pieces are found to be locked, we select an axial direction for separation and swap the regional IDs near the interfaces in that direction, until the locking issue is resolved. Note that both issues are rare in practice, especially if mesh segmentation is smooth.

## 5 Partitioning Optimization

The system uses the constrained optimization process to improve the qualities of the partitioning in three steps, each of which is responsible for one or more qualities. These steps are executed iteratively, until the packed space cannot be reduced any further.

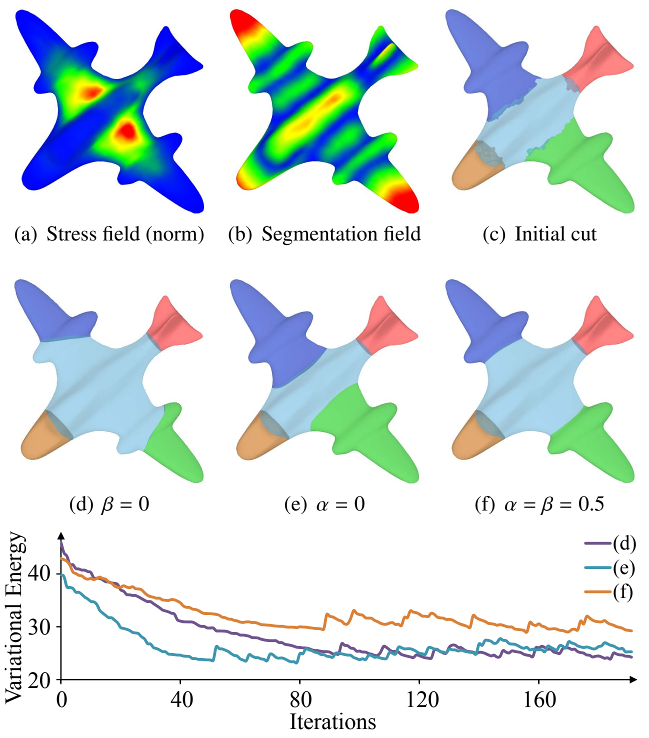


**Figure 6:** *An airplane example. By changing the parameters in Equation 4, we can use variational optimization to achieve different partitioning results. For example, the result in (d) ignores the segmentation field by setting $\beta = 0$, while the result in (e) ignores the stress field, by setting $\alpha = 0$. In practice, we consider both fields, so variational optimization can move the interfaces to places where stress is low and surface segmentation is preferred, as in (f).*

## 5.1 Variational Optimization

We use variational optimization to improve the intrinsic qualities of the partitioning, including minimal stress load, surface detail alignment, and minimal interface area. Intuitively, it tries to smooth the interfaces, and move them to low stress and preferred segmentation regions. Given the stress tensor field $\sigma(\mathbf{x})$ obtained from stress analysis, we first compute the separation load $f(\mathbf{x})$ at a point $\mathbf{x}$ of the $i$-th piece by:

$$f(\mathbf{x}) = \max(\mathbf{f}(\mathbf{x}) \cdot \nabla\phi_i(\mathbf{x}), 0) + (\|\sigma(\mathbf{x})\|_F - \|\mathbf{f}(\mathbf{x})\|_2), \quad (1)$$

where $\mathbf{f}(\mathbf{x}) = \sigma(\mathbf{x})\nabla\phi_i(\mathbf{x})$ is the tension force density at $\mathbf{x}$. In Equation 1, the first term measures the tension load and the second term measures the shear load. Compression should not cause separation, in which case $\mathbf{f}(\mathbf{x}) \cdot \nabla\phi_i(\mathbf{x}) < 0$. The use of $f(\mathbf{x})$ is to prevent the pieces from becoming easily separable due to the use of a loose connector, as shown in Figure 5. Let $g(\mathbf{x})$ be the scalar field defining the segmentation preference on the model surface, and $\alpha$ and $\beta$ be two control variables. We define the following surface energy functional over the whole grid volume $D$ under the level set framework:

$$E = \sum_{i=1}^{n} \iiint_D \delta(\phi_i(\mathbf{x})) \|\nabla\phi_i(\mathbf{x})\| (\alpha f(\mathbf{x}) + \beta g(\mathbf{x}) + 1)\, d\mathbf{x}, \quad (2)$$

subject to the following constraints:

$$\begin{cases} \sum_{i=1}^{n} H(-\phi_i(\mathbf{x})) \equiv 1, & \text{for any } \mathbf{x} : \phi_0(\mathbf{x}) < 0; \\ \sum_{i=1}^{n} H(-\phi_i(\mathbf{x})) \equiv 0, & \text{for any } \mathbf{x} : \phi_0(\mathbf{x}) \geq 0, \end{cases} \quad (3)$$

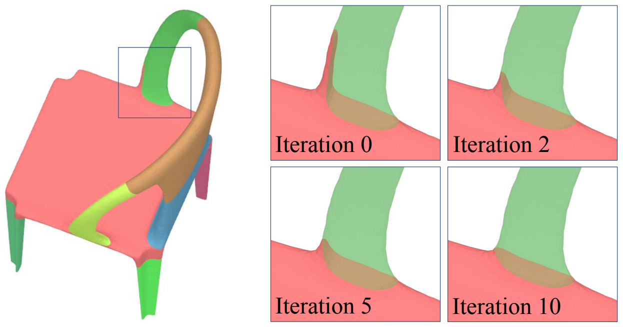


**Figure 7:** *The removal of a thin feature. The variational optimization step automatically removes a thin feature of a partitioned piece through the smoothing process, within just 10 iterations as shown on the right side.*

in which $H$ is the Heaviside function and $\delta(x) = \frac{dH(x)}{dx}$ is the Dirac delta function. The energy in Equation 2 contains three surface energies, corresponding to the separation load $f$, the surface segmentation field $g$, and the surface area. Without considering Equation 3 and the dependency of $f(\mathbf{x})$ on $\phi_i(\mathbf{x})$, we can use the gradient projection method to formulate the time evolution function of $\phi_i$:

$$\frac{d\phi_i}{dt} = \|\nabla\phi_i\| \left( (\alpha f + \beta g + 1)\, \kappa_i + (\alpha\nabla f + \beta\nabla g) \cdot \frac{\nabla\phi_i}{\|\nabla\phi_i\|} \right), \quad (4)$$

in which the first term gives the mean curvature flow that smooths the boundary surface of the $i$-th piece. To prevent it from constantly affecting other terms, we perform smoothing only when the mean curvature magnitude is above a threshold $\kappa_i^c$:

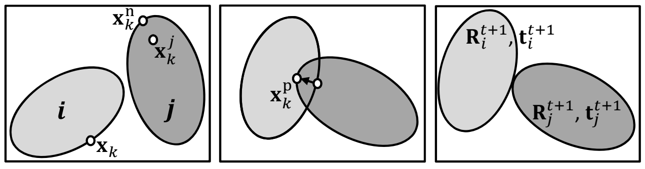
$$\kappa_i = \max\left(\left|\kappa_i^0\right| - \kappa_i^c, 0\right) \kappa_i^0 \Big/ \left|\kappa_i^0\right|, \quad (5)$$

in which $\kappa_i^0 = \nabla \cdot \left(\frac{\nabla\phi_i}{\|\nabla\phi_i\|}\right)$ is the original mean curvature.

Figure 6 demonstrates how the surface energy defined in Equation 2 changes, when the partitioning optimization process handles an airplane example using different control variables. Before starting packing optimization, the system uses variational optimization to reduce nearly one third of the energy within 50 to 90 iterations. After packing optimization starts, variational optimization keeps the energy at a low level, although the energy does not necessarily decrease. To decide the time step, we set the CFL condition to $C = 0.3$. In other words, the interface cannot move more than $0.3h$ within a time step, in which $h$ is the size of a grid cell.

One advantage of our variational optimization step is that it can automatically remove unprintable thin features near the boundary surface of a partitioned piece, as shown in Figure 7. This is because thin features have high curvatures and they will be gradually eliminated by the smoothing process described in Equation 4. We note that this step only affects the interfaces defined by the pieces. If the original model itself contains thin features on its boundary surface, we require users to fix it ahead of time by existing techniques.

**Level set correction.** A level set function may no longer be a valid signed distance field, once it gets updated. Meanwhile, there can be gaps or overlaps among level set functions, after they are updated separately. To solve these two issues, we must correct the distances of each grid cell. Here we use the projection method proposed by Losasso and colleagues [2006]. The basic idea is to find the two smallest level set function values of each grid cell around the boundaries, and then shift their average to zero. This ensures that the grid cell has the same distance to the zero surfaces specified by the two level set functions. Since it is possible to have gaps or



$\mathbf{x}_k^{\mathrm{n}}$
$\mathbf{x}_k^j$
$i$
$j$
$\mathbf{x}_k$
$\mathbf{x}_k^{\mathrm{p}}$
$\mathbf{R}_i^{t+1}, \mathbf{t}_i^{t+1}$
$\mathbf{R}_j^{t+1}, \mathbf{t}_j^{t+1}$

(a) Original space (b) Packed space (c) Packed space (after)

**Figure 8:** *A collision example. Two separate pieces in the original space (a) are in collision in the packed space (b). We remove this collision by correcting the sample points and then enforcing the rigidity of the pieces, until no collision occurs as (c) shows.*

overlaps between a piece and the external boundary of the original model, we consider $-\phi_0$ in the above process as well. If $-\phi_0$ gives one of the smallest level set values, we modify the value of the other level set function only. Doing this prevents the shape and the topology of the original model from being modified by optimization. Once we remove gaps and overlaps, we use the fast marching method to recalculate the distances of other cells.

We perform level set correction at the end of the variational optimization step. Since level set functions can also be changed by other steps, they must be corrected again. We will discuss how to do so later in this paper.

## 5.2 Packing Optimization

In this subsection, we will discuss how to incorporate 3D packing into the partitioning optimization process. Our idea is to reduce the size of a container first, and then try to remove all of the collisions by iteratively running two sub-steps: *packing solver*, which updates the positions and the orientations of the pieces to eliminate their collisions, and *level set adjustment*, which modifies the level set functions of the pieces to remove the colliding regions.

**Packing solver.** Given the level set functions $\{\phi_1, ..., \phi_n\}$ representing the pieces of a 3D model, we would like to find the optimal configuration for them to be tightly packed within a specified container. Inspired by Imamichi and Nagamochi [2007]'s work on spherical packing, we formulate a packing solver using a rigid body simulator based on shape matching [Müller et al. 2005]. Let $< \mathbf{R}_i, \mathbf{t}_i >$ be the rotation matrix and the translation vector describing the rigid transformation of the $i$-th piece from the original space to the packed space. The packing solver iteratively updates $< \mathbf{R}_i, \mathbf{t}_i >$, to remove both piece-piece collisions and piece-container collisions.

Let $\mathbf{x}_k$ be a grid cell sample belonging to the $i$-th piece in the original space, such that $\phi_i(\mathbf{x}_k) \in [-h, 0]$, in which $h$ is the size of a grid cell, as shown in Figure 8a. We use the following condition to test whether it is in collision with the $j$-th piece:

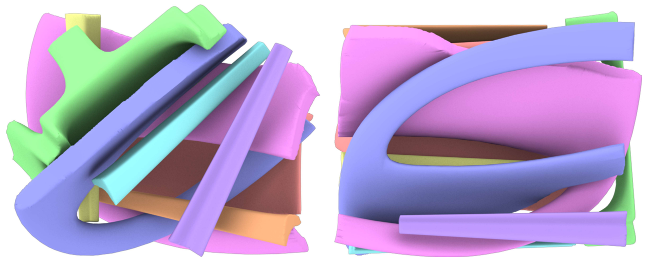
$$d_k = -\phi_i(\mathbf{x}_k) - \phi_j(\mathbf{x}_k^j) + D > 0, \tag{6}$$

where $d_k$ is the penetration depth, $D$ is a safety threshold, $\mathbf{x}_k^j$ is the corresponding location of $\mathbf{x}_k$ in the original space of the $j$-th piece:

$$\mathbf{x}_k^j = \mathbf{R}_j^{\mathsf{T}} \left( \mathbf{R}_i(\mathbf{x}_k - \mathbf{t}_i^{\mathrm{o}}) + \mathbf{t}_i - \mathbf{t}_j \right) + \mathbf{t}_j^{\mathrm{o}}. \tag{7}$$

Here $\mathbf{t}_i^{\mathrm{o}}$ and $\mathbf{t}_j^{\mathrm{o}}$ are the mass centers of the two pieces in the original space. Equation 6 means the distance from $\mathbf{x}_k$ to the boundary of the $i$-th piece must be shorter than the distance from $\mathbf{x}_k^j$ to the boundary of the $j$-th piece. If not, the two pieces must be in collision. We can fix this collision, if we move $\mathbf{x}_k^j$ to $\mathbf{x}_k^{\mathrm{n}}$ in the original space:

$$\mathbf{x}_k^{\mathrm{n}} = \mathbf{x}_k^j + d_k \nabla \phi_j(\mathbf{x}_k^j), \tag{8}$$



(a) Without preferred orientations (b) With preferred orientations

**Figure 9:** *The packed state of a chair model. By using preferred orientations during the packing process, our packing solver can produce a more organized result, as shown in (b).*

which can then be converted to $\mathbf{x}_k^{\mathrm{p}}$ in the packed space, as Figure 8b shows. Once we obtain $\mathbf{x}_k^{\mathrm{p}}$ for all of the colliding points, we use shape matching [Müller et al. 2005] to update $\mathbf{R}_i$ and $\mathbf{t}_i$, by minimizing $\sum_k m_k \left\| \mathbf{R}_i(\mathbf{x}_k - \mathbf{t}_i^{\mathrm{o}}) + \mathbf{t}_i - \mathbf{x}_k^{\mathrm{p}} \right\|^2$, where $m_k$ is the mass of $\mathbf{x}_k$. This can be quickly solved through mass center alignment and polar decomposition. Since some collisions may not be removed immediately, we update $\mathbf{R}_i$ and $\mathbf{t}_i$ several times, before resorting to the level set adjustment process. In average, we use approximately 1500 samples for each piece and we do not use collision culling currently. The collision between a piece and a container can be handled in the same way, by formulating the container boundary as a signed distance field as well.

If a model has obvious intrinsic orientations, its packed result can appear to be messy as Figure 9a shows. To make the result more organized and more visually appealing, our system offers an option for users to specify preferred orientations. For example, 24 axis-aligned orientations can be set as the preferred ones for the chair example. In each iteration, the packing solver gradually rotates each piece toward the closest preferred orientation. Figure 9b shows that the solver can make most pieces aligned with their preferred orientations. Although doing this increases the occupied volume by approximately 5 percent, we think it is worthwhile, since the result simplifies the packing process in the real world.

**Level set adjustment and correction.** If the packing solver cannot remove all of the collisions after a number of sub-iterations, the system adjusts the level set function of each piece to remove the colliding region directly, so that the pieces can be packed even further. Let $\mathbf{x}_k$ be a point sample of the $i$-th piece close to its boundary: $\phi_i(\mathbf{x}_k) \in [-h, 0]$. If it is in collision with another piece: $d_k > 0$, we modify $\phi_i$ by removing a sphere around $\mathbf{x}_k$ with radius $d_k$:

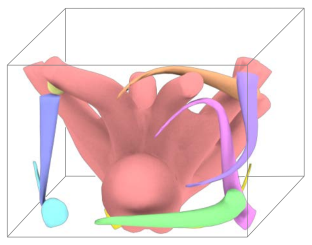
$$\phi_i^{\mathtt{new}}(\mathbf{x}) = \max\left(\phi_i(\mathbf{x}), d_k - \|\mathbf{x} - \mathbf{x}_k\|\right). \tag{9}$$

Doing this prevents the same collision from happening at $\mathbf{x}_k$ again. If $\mathbf{x}_k$ is in collision with the container whose level set function is defined as $\phi_{\mathrm{b}}$, we remove the colliding region from $\phi_i$:

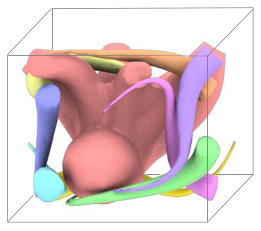
$$\phi_i^{\mathtt{new}}(\mathbf{x}) = \max\left(\phi_i(\mathbf{x}), \phi_{\mathrm{b}}(\mathbf{x})\right). \tag{10}$$

Once we update $\phi_i$, we use the projection method described in Subsection 5.1 to fill the vacuum regions by other pieces. Since the projection method also repairs the gap between the pieces and the external boundary of the original model, we do not need to process level set adjustment differently near the external boundary. Finally, we use the fast marching method to re-distance the updated level set functions, and restart the packing solver to detect and handle newly emerged collisions.

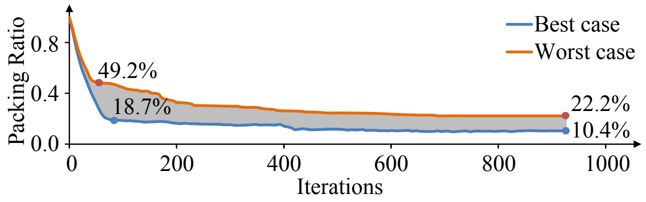
The system repeats the packing solver and the level set adjustment multiple times. If all of the collisions are removed within a fixed



(a) Before adjustment

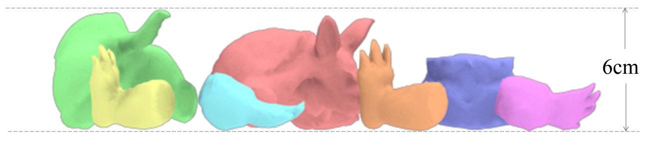


(b) After adjustment

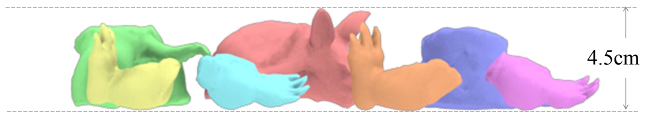


(c) Convergence plot

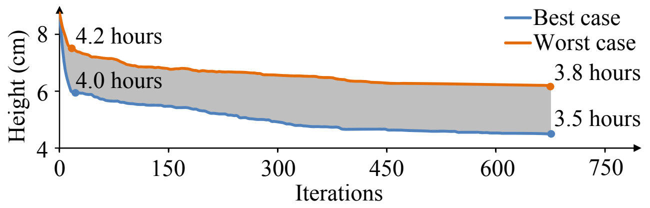
**Figure 10:** *A volume reduction example using the octopus model. Enabling the level set adjustment allows packing optimization to further reduce the volume in the packed state, from 18.7 percent of the original volume to 10.4 percent of the original volume.*



(a) Before adjustment



(b) After adjustment



(c) Convergence plot

**Figure 11:** *A height reduction example using the armadillo model. The packing optimization step reduces the maximum height of the pieces from 6cm to 4.5cm and the printing time from 4.0 hours to 3.5 hours using a FDM 3D printer.*

number of sub-iterations, it ends the current optimization iteration and starts another. If there are still collisions, the system terminates the whole partitioning optimization process and finalizes the partitioning result using the procedure described in Section 3.

**Applications.** We use packing optimization for multiple purposes, depending on how the container shrinks. If the goal is to reduce the volume of a box container, we shrink the container by a small amount along an axis at the beginning of each packing optimization step, and we end the process when the container cannot be compressed in any of the three axes without causing collisions. The final shape of the container is typically similar to a cube, as the octopus example shows in Figure 10b. This example also demonstrates the importance of adjusting the partitioning in packing optimization: the packing ratio[1] is reduced from 18.7 percent without level set adjustment to 10.4 percent with level set adjustment. For fast production, we can use the same system to reduce the maximum height of the pieces, by decreasing the height slightly in each iteration. Figure 11 shows that the system reduces the maximum height of an armadillo model from 6.0cm to 4.5cm, and its printing time from 4.0 hours to 3.5 hours using a FDM 3D printer.

When using packing optimization to reduce the maximum height for fast printing, we can also optimize the orientation of each piece, to reduce the use of support structure due to large overhanging features. To do that, we uniformly sample 124 orientations, evaluate each of them for every piece, and rotate the piece to the best orientation at the beginning of each iteration. An even better solution is to formulate the avoidance of overhanging features as a partitioning quality as well, and optimize the partitioning to remove these features. We will investigate this idea further in the future.

**Packing initialization.** Unlike spheres or convex polyhedra, the pieces in irregular shapes may get locked in the packed state, if we allow them to randomly swap their positions as did in [Imamichi and Nagamochi 2007]. So instead, we initialize the packed state to be locking-free by making the pieces well separated, and we allow the packing solver to naturally avoid the locking issue afterwards. Doing this makes our result sensitive to the initialization and we need multiple random initializations to ensure the packing quality of the final result. Figure 10c and 11c visualize the best and the worst packing results using different initializations. Their difference demonstrates the importance of using multiple initializations. Our experiment shows that 100 initializations are typically sufficient for the system to obtain an acceptable result.

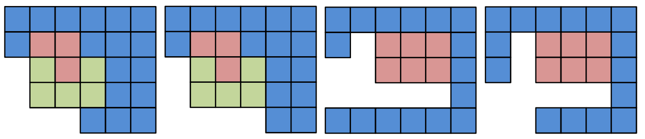
[1]We define the packing ratio as the ratio of the packed volume to the bounding box volume of the original model.

## 5.3 Constraint Enforcement

We enforce two types of constraints to ensure the usability of the partitioning result: the size limit of each piece and the ability to get the pieces assembled. Since the system ends the whole partitioning optimization process by packing optimization as discussed in Subsection 3, we place the constraint enforcement step between variational optimization and packing optimization. Our system guarantees that the constraints are strictly satisfied.

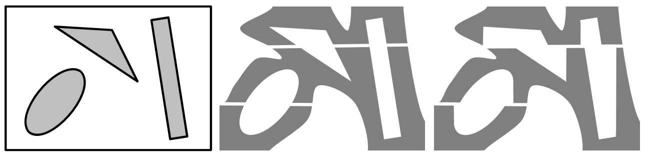
**Size limit.** A printable piece must fit into the build volume of a 3D printer, which is in a box shape typically. To know if a piece can fit into the box, we simply define it as a container and use the packing solver to eliminate their collisions, as in Subsection 5.2. If the piece does not fit, we use the same level set adjustment method to remove the collisions and then correct the level set functions.

**Assembling.** The partitioning is usable only if the pieces can be assembled into both the original state and the packed state. Our simulation-based packing solver naturally allows the pieces to be organized into the packed state, as long as the pieces are well separated initially. So here we just need to know whether the pieces can be assembled into the original state. In other words, whether the pieces are locked as Figure 12a shows. To answer this question, we need to find a separation order, using which each piece can be separated from the rest. For example, the pieces in Figure 12b are separable in the green, red, blue order. Asking whether a piece is separable is equivalent to asking whether a separation path exists, through which the piece can be moved away from others without collision. This is a highly difficult problem, since the search space is large and the path may change its direction, as Figure 12c shows.



(a) Locked (b) Separable (c) Separable (d) Detachable

**Figure 12:** *Locked and separable cases. For simplicity, we test whether a piece is detachable rather than separable from others.*



(a) The packed space (b) Chopped structure (c) Final structure

**Figure 13:** *The construction of a container structure. Given the packed state in (a), we perform a set of steps to create a container structure in (c), which secures the pieces within a container.*

For simplicity, we choose to evaluate the detachable condition instead, by testing whether a piece can be disconnected from others. To know if a piece is detachable in a direction, we test whether the normals on the contact interface are consistent with that direction. Here we sample the spherical coordinates into a number of directions, and check the detachable condition for each of them. Without loss of generality, let $\phi_i$ be the level set function of the $i$-th piece and $\min_{j \neq i} \phi_j$ be the union of the rest. We select the grid cell $\mathbf{x}_k$ as a sample near the interface between the $i$-th piece and the rest, if $|\phi_i(\mathbf{x}_k)| < h$ and $\left|\min_{j \neq i} \phi_j(\mathbf{x}_k)\right| < h$, in which $h$ is the grid cell size. To know if the piece is detachable in the direction $\mathbf{d}$, we test: $\mathbf{d} \cdot \nabla\phi_i(\mathbf{x}_k) > 0$. If this is true for all $\mathbf{x}_k$, the surface normals are consistent with $\mathbf{d}$ and the $i$-th piece is detachable. We then remove it and test if another piece is detachable from the rest. We continue this process, until all of the pieces are removed, or a subset of pieces are found to be locked. If a locking issue is found, we simply restore the level set values of the involved grid cells, given the fact that the partitioning has no locking issue previously.

We note that our approach does not guarantee the detection of every locking issue, since a detachable piece may still be trapped as shown in Figure 12d. Fortunately, those "traps" are rare in practice, because they should be naturally eliminated by variational optimization through the smoothing process in Subsection 5.1. It is possible that this problem can become more serious, if there are "traps" on the external boundary of the model that cannot be smoothed. In that case, we can still use more sophisticated approaches, which do not seem to be necessary at this time.

## 6 Container Structure

Here we present a container structure modeling algorithm to facilitate the packing process and secure the pieces within a container. Our first step is to convert the level set functions from the original space to the packed space, as Figure 13a shows. Let $\varphi_1, \varphi_2, ..., \varphi_n$ be the level set functions in the packed space. A naïve idea to define the container structure is to simply use the complement of the level sets: $\epsilon - \min(\varphi_1, \varphi_2, ..., \varphi_n)$, in which $\epsilon$ is a buffering variable to prevent the pieces from being too tightly packed. However, there are two issues associated with this idea. The first issue is that it uses too much printing material, especially if the packed state contains large holes that need to be filled as part of the structure now. To solve this issue, we formulate the level set function of the structure as: $\left|\epsilon - \min(\varphi_1, \varphi_2, ..., \varphi_n) + \frac{B}{2}\right| - \frac{B}{2}$, in which $B$ is a thickness variable. Intuitively, this defines the structure as a shell with thickness $B$ around the pieces. The second issue is that the pieces may be locked within the structure. This problem is closely related to locking detection discussed in Subsection 5.3. For simplicity, we consider the vertical separation direction only, which is a reasonable assumption since most containers can be opened only from the top. Our idea is to chop the container structure by a set of horizontal planes across the centers of the pieces, so that every piece can be reached by at least one plane. For example, two cutting planes are used in Figure 13b. We then eliminate the locking issue of each piece by projecting its level set function vertically onto its cutting plane and carving the container structure accordingly. The final result allows each piece to be separable in the vertical direction, as shown in Figure 13c.

| Name (#Pieces) | Original Resolution | Time (min) | Initial Ratio | Final Ratio | Pack Merger |
|---|---|---|---|---|---|
| Tree (4) | $96 \times 152 \times 54$ | 6.0 | 30.7% | 23.5% | 39.0% |
| Tree (8) | $96 \times 152 \times 54$ | 7.4 | 20.1% | 15.2% | 22.5% |
| Chair (8) | $78 \times 180 \times 94$ | 9.8 | 32.4% | 24.9% | 38.5% |
| Chair (10) | $78 \times 180 \times 94$ | 10.7 | 25.3% | 19.4% | 24.1% |
| Armadillo (7) | $94 \times 118 \times 78$ | 8.6 | 36.4% | 31.7% | 45.4% |
| Octopus (9) | $98 \times 68 \times 107$ | 7.2 | 18.7% | 10.4% | 22.0% |
| Deer (9) | $39 \times 149 \times 148$ | 8.6 | 28.7% | 23.1% | 29.5% |
| Sphere (6) | $89 \times 90 \times 90$ | 6.4 | 50.7% | 43.0% | 55.8% |
| Bunny (8) | $105 \times 105 \times 81$ | 8.4 | 60.3% | 52.8% | 70.9% |
| Airplane (8) | $104 \times 34 \times 198$ | 6.9 | 33.5% | 27.3% | 30.1% |
| Sculpture (10) | $58 \times 70 \times 149$ | 6.8 | 50.4% | 42.6% | 64.4% |
| Vase (10) | $6.5 \times 167 \times 92$ | 11.2 | 65.1% | 57.0% | 78.5% |
| Wheel (9) | $80 \times 122 \times 122$ | 10.2 | 62.6% | 53.1% | 55.7% |
| Creature (8) | $108 \times 134 \times 88$ | 11.6 | 37.7% | 21.4% | 44.0% |
| Knots (9) | $107 \times 106 \times 50$ | 4.9 | 38.7% | 29.1% | 40.1% |
| Trophy (9) | $81 \times 144 \times 84$ | 8.3 | 58.8% | 46.7% | 62.0% |
| Scorpion (9) | $126 \times 79 \times 111$ | 9.7 | 22.7% | 13.3% | 28.7% |
| Hand (8) | $96 \times 84 \times 150$ | 9.9 | 18.7% | 14.8% | 17.1% |

**Table 1:** *Statistics showing the number of pieces, the grid resolution, the running time, and the packing ratios of using different methods, including before and after using the level set adjustment.*

## 7 Results and Discussions

We tested the performance of our system using an Intel Core i7-4790K 3.6GHz processor. Our examples, listed in Table 1, cover a wide range of 3D printable models. For all of the examples, we choose the grid cell size $h = 1$mm. We set the collision safety threshold $D$ to 1.5mm, the buffer variable $\epsilon$ to 0.5mm, and the thickness variable $B$ to 4mm. These variables allow the container structure to have sufficient strength, even though they downgrade the packing quality slightly. We use a Makerbot Replicator 2× 3D printer to build some of the models for evaluation purposes.

It is not surprising to see from Table 1 that the more pieces we create, the more tightly we can pack the pieces. The packing quality relies on the original shape of the model as well. In general, it is likely to get higher packing quality if a model has hollow regions, such as the tree example and the octopus example. Table 1 also shows that our system outperforms Packmerger [Vanek et al. 2014], which offers only a limited number of orientations for the pieces in the packed state and does not change partitioning after packing.

**Fragment removal.** Small disconnected fragments are rare, but they may still occur due to numerical instability at the contact front where three pieces meet, or level set correction near thin features on the external boundary. To remove these fragments, we perform connected component search after every level set update and use a

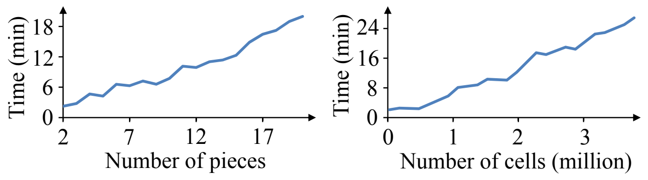


**Figure 14:** *Running time plots. These plots show how the number of pieces and the number of cells can affect the running time.*

volume threshold to identify these fragments. We then merge them with large pieces nearby and recompute the signed distances locally using the fast marching method.

**Connectors.** To prevent the pieces from movement in the original state, we create male and female cylindrical connectors on every interface between two pieces, as Luo and colleagues [2012] did. To avoid new locking issues, we align the connectors with the detaching direction detected in Subsection 5.3. We add these connectors before the termination procedure (described in Section 3) starts, so we do not have to worry about new collisions in the final packed state. It is possible that a connector cannot be added when both pieces are thin. Fortunately, it is rare in practice and we did not notice it happening in our experiment.

**Surface mesh reconstruction.** Although the level set functions provide sufficient details on the interfaces, they may not carry the same external surface details as the original input mesh does. Because of this, we do not use them to construct the surface meshes of the pieces. Instead, we first extend the interfaces slightly and construct the interface meshes by a multi-phase marching cubes algorithm. After that, we calculate the intersections between the interface meshes and the original input mesh, the latter of which is then segmented into parts. Finally, we stitch the segmented mesh and the interface meshes to form the surface mesh of each piece.

**Performance acceleration.** Although each iteration of the partitioning optimization process is computationally inexpensive, the system needs to run 600 to 800 iterations each time and restart the whole process 100 times with new packing initialization seeds. So the overall cost of the system can still be large. Besides the use of multi-threading, we apply three methods to improve the system performance. First, we define the level sets in two resolutions. We run the optimization in low resolution first, and then use the up-sampled result to initialize partitioning optimization in high resolution. This method alone can reduce nearly 70 percent of the computational cost. Second, we terminate the optimization process, if an early result is far from satisfactory. Specifically, we run the optimization without the level set adjustment sub-step in packing optimization first. If the packing quality of the current result is far below that of the best result we have obtained, its final result is unlikely to be the best and there is no need to optimize it even further. Finally, we incorporate velocities and repulsion forces into our packing solver and handle them as in a rigid body simulator, to reduce the number of needed sub-iterations. Table 1 shows that these methods effectively reduce the total computational cost from three hours to less than 12 minutes, given a single initial partitioning input.

**Running time analysis.** In general, the running time is linearly proportional to the number of pieces and the number of grid cells, as shown in Figure 14. This is not surprising, since the number of pieces determines the number of level set functions and the number of grid cells controls the cost spent on the level set function. The running time can be affected by other factors as well, such as model shape and packing initialization. However, these relationships are not so straightforward.

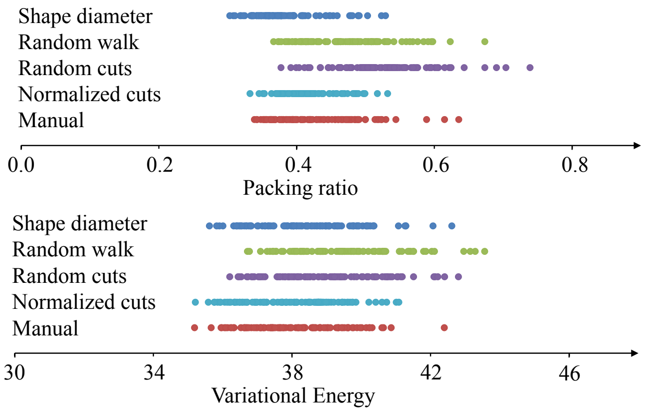


**Figure 15:** *The results of using five partitioning initialization methods. Each dot represents the result of a packing initialization seed.*

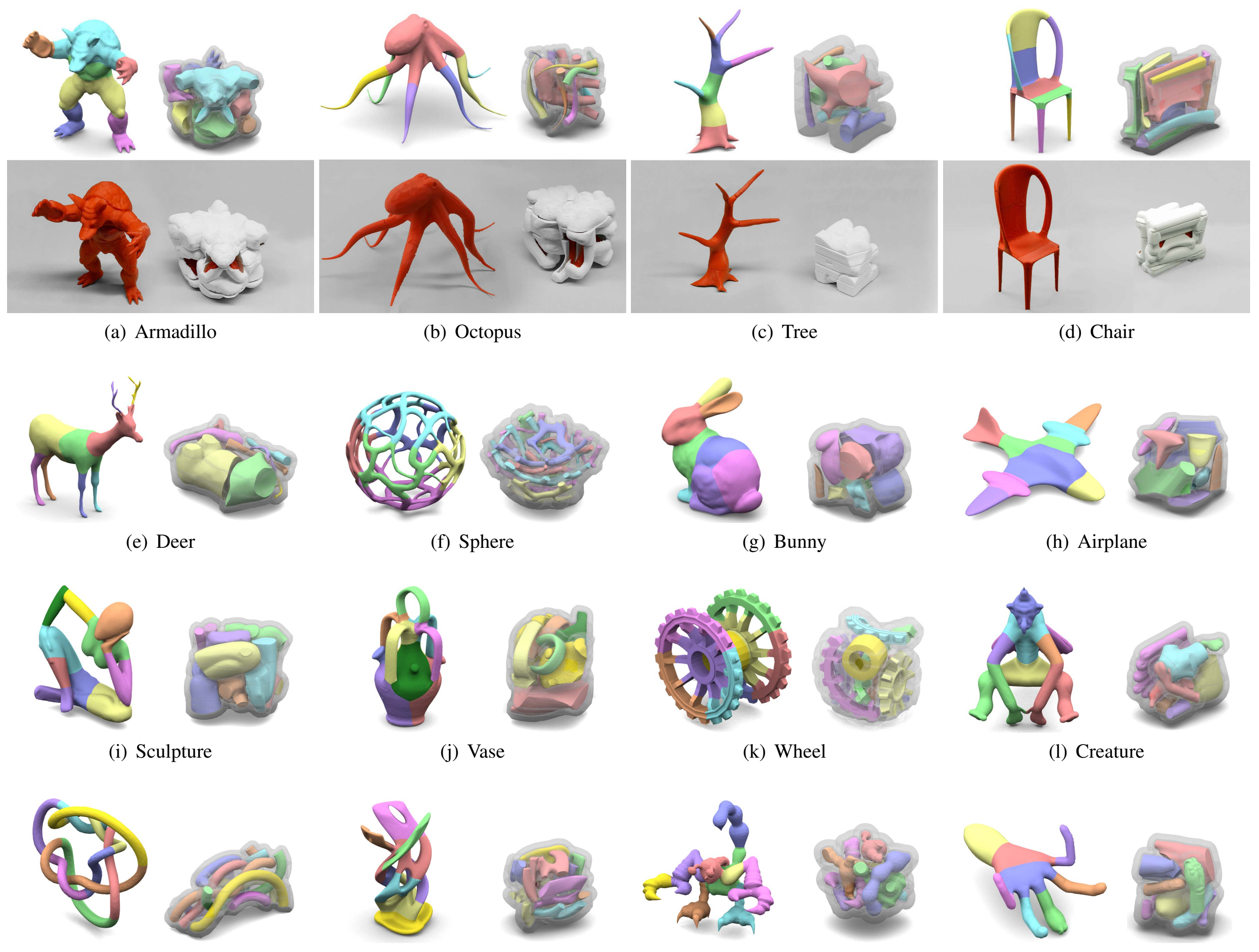
**Partitioning initialization.** So far we have discussed packing initialization and its influence on the system performance. Since our system offers several ways to initialize the partitioning as well in Section 4, it is interesting to know how they affect the final result. Figure 15 demonstrates the results using five partitioning initialization methods, each of which is tested by 100 packing initialization seeds. In our experiment, we typically combine the shape diameter method with manual editing to initialize the partitioning.

**Limitations.** Our system needs to run the optimization process multiple times using random packing initialization seeds, which is a dominant factor of the system performance. The system needs large computational and memory costs to handle thin features. Fortunately, this is not a big issue since 3D printers have difficulty in building these details anyway. Our printability metric considers the size limit only. In practice, 3D printers may have other printability criteria and they must be addressed additionally. Our assembling tester evaluates whether a piece is detachable, rather than separable from others. As a result, it cannot guarantee the detection of every locking issue. The container structure modeled by our algorithm may contain unprintable thin features, or overhanging features that require additional material for its support structure. The container structure has not been optimized or tested by stress analysis, so it is not clear how much protection it can offer, especially under large impact. Finally, the number of pieces are pre-determined in partitioning initialization and the system does not change it afterwards.

## 8 Conclusions and Future Work

We present a level-set-based system to automatically generate the partitioning of a 3D model for printing and packing purposes. Compared with mesh-based and particle-based representations, the level set representation can more conveniently and flexibly handle partitioning optimization, as shown in this work. We believe it is suitable for solving other printable shape optimization problems as well.

In the future, we plan to improve our system in many ways. Our immediate plan is to make the packing solver even faster, by applying collision culling techniques. When using the system to reduce the maximum height of the pieces for fast 3D printing, we are interested in developing new metrics to minimize the use of support structure material as well. We also would like to study the locking issue further, by finding ways to detect and fix sophisticated interlocking cases. Finally, we will improve connector and container structure modeling algorithms, to make the pieces more useful in both the original state and the packed state.



(a) Armadillo (b) Octopus (c) Tree (d) Chair

(e) Deer (f) Sphere (g) Bunny (h) Airplane

(i) Sculpture (j) Vase (k) Wheel (l) Creature

(m) Knots (n) Trophy (o) Scorpion (p) Hand

**Figure 16:** *Our examples and results. We enlarge some of the packed states to make them more visible. Some of the models are obtained from Autodesk 123D, under an Attribution-NonCommercial-ShareAlike license.*

## 9 Acknowledgments

This work was supported by NSF grant IIS-1524992 and in part by the open project program of the State Key Lab of CAD&CG at Zhejiang University. We also thank ACCAD at OSU, Nvidia and Adobe for additional support through funding and equipment.

## References

BÄCHER, M., BICKEL, B., JAMES, D. L., AND PFISTER, H. 2012. Fabricating articulated characters from skinned meshes. *ACM Trans. Graph. (SIGGRAPH) 31*, 4 (July), 47:1–47:9.

BACHER, M., WHITING, E., BICKEL, B., AND SORKINE-HORNUNG, O. 2014. Spin-It: Optimizing moment of inertia for spinnable objects. *ACM Trans. Graph. (SIGGRAPH) 33*, 4 (August), 96:1–96:10.

CALÌ, J., CALIAN, D. A., AMATI, C., KLEINBERGER, R., STEED, A., KAUTZ, J., AND WEYRICH, T. 2012. 3D-printing of non-assembly, articulated models. *ACM Trans. Graph. (SIGGRAPH Asia) 31*, 6 (Nov.), 130:1–130:8.

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

CHEN, X., ZHENG, C., XU, W., AND ZHOU, K. 2014. An asymptotic numerical method for inverse elastic shape design. *ACM Trans. Graph. (SIGGRAPH) 33*, 4 (August), 95:1–95:11.

DEUSS, M., PANOZZO, D., WHITING, E., LIU, Y., BLOCK, P., SORKINE-HORNUNG, O., AND PAULY, M. 2014. Assembling self-supporting structures. *ACM Trans. Graph. (SIGGRAPH Asia) 33*, 6 (Nov.), 214:1–214:10.

DICKINSON, J. K., AND KNOPF, G. K. 1998. Serial packing of arbitrary 3D objects for optimizing layered manufacturing. In *Procedings of SPIE 3522, Intelligent Robots and Computer Vision XVII: Algorithms, Techniques, and Active Vision*.

DUMAS, J., HERGEL, J., AND LEFEBVRE, S. 2014. Bridging the gap: Automated steady scaffoldings for 3D printing. *ACM Trans. Graph. (SIGGRAPH) 33*, 4 (August), 98:1–98:10.

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

ENRIGHT, D., FEDKIW, R., FERZIGER, J., AND MITCHELL, I. 2002. A hybrid particle level set method for improved interface capturing. *J. Comput. Phys. 183*, 1 (Nov.), 83–116.

GIBOU, F., AND FEDKIW, R. 2005. A fast hybrid k-means level set algorithm for segmentation. In *4th Annual Hawaii International Conference on Statistics and Mathematics*, 281–291.

GOMES, A. M., AND OLIVEIRA, J. F. 2006. Solving irregular strip packing problems by hybridising simulated annealing and linear programming. *European Journal of Operational Research 171*, 3, 811 – 829.

GUENDELMAN, E., BRIDSON, R., AND FEDKIW, R. 2003. Nonconvex rigid bodies with stacking. *ACM Trans. Graph. (SIGGRAPH) 22*, 3 (July), 871–878.

HU, R., LI, H., ZHANG, H., AND COHEN-OR, D. 2014. Approximate pyramidal shape decomposition. *ACM Trans. Graph. (SIGGRAPH Asia) 33*, 6 (Nov.), 213:1–213:12.

IMAMICHI, T., AND NAGAMOCHI, H. 2007. A multi-sphere scheme for 2D and 3D packing problems. In *Proc. of SLS*, 207–211.

IMAMICHI, T., YAGIURA, M., AND NAGAMOCHI, H. 2009. An iterated local search algorithm based on nonlinear programming for the irregular strip packing problem. *Discrete Optimization 6*, 4, 345 – 361.

JUNIOR, B., PINHEIRO, P., AND SARAIVA, R. 2013. Tackling the irregular strip packing problem by hybridizing genetic algorithm and bottom-left heuristic. In *2013 IEEE Congress on Evolutionary Computation*, 3012–3018.

KIM, B. 2010. Multi-phase fluid simulations using regional level sets. *ACM Trans. Graph. (SIGGRAPH) 29*, 6 (Dec.), 175:1–175:8.

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

LI, H., ALHASHIM, I., ZHANG, H., SHAMIR, A., AND COHEN-OR, D. 2012. Stackabilization. *ACM Trans. Graph. (SIGGRAPH Asia) 31*, 6 (Nov.), 158:1–158:9.

LOSASSO, F., SHINAR, T., SELLE, A., AND FEDKIW, R. 2006. Multiple interacting liquids. *ACM Trans. Graph. (SIGGRAPH) 25*, 3 (July), 812–819.

LU, L., SHARF, A., ZHAO, H., WEI, Y., FAN, Q., CHEN, X., SAVOYE, Y., TU, C., COHEN-OR, D., AND CHEN, B. 2014. Build-to-last: Strength to weight 3D printed objects. *ACM Trans. Graph. (SIGGRAPH) 33*, 4 (August), 97:1–97:10.

LUO, L., BARAN, I., RUSINKIEWICZ, S., AND MATUSIK, W. 2012. Chopper: Partitioning models into 3D-printable parts. *ACM Trans. Graph. (SIGGRAPH Asia) 31*, 6 (Nov.), 129:1–129:9.

MÜLLER, M., HEIDELBERGER, B., TESCHNER, M., AND GROSS, M. 2005. Meshless deformations based on shape matching. *ACM Trans. Graph. (SIGGRAPH) 24*, 3 (July), 471–478.

NOORUDDIN, F. S., AND TURK, G. 2003. Simplification and repair of polygonal models using volumetric techniques. *IEEE Transactions on Visualization and Computer Graphics 9*, 2, 191–205.

NOVOTNI, M., AND KLEIN, R. 2002. Computing geodesic distances on triangular meshes. In *Proc. of WSCG*, 341–347.

OSHER, S., AND SETHIAN, J. A. 1988. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *J. Comput. Phys. 79*, 1 (Nov.), 12–49.

PANOZZO, D., BLOCK, P., AND SORKINE-HORNUNG, O. 2013. Designing unreinforced masonry models. *ACM Trans. Graph. (SIGGRAPH) 32*, 4 (July), 91:1–91:12.

RUUTH, S. J. 1998. A diffusion-generated approach to multiphase motion. *J. Comput. Phys. 145*, 1 (Sept.), 166–192.

SAYE, R. I., AND SETHIAN, J. A. 2011. The Voronoi implicit interface method for computing multiphase physics. *Proceedings of the National Academy of Sciences 108*, 49, 19498–19503.

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

SMITH, K., SOLIS, F., AND CHOPP, D. 2002. A projection method for motion of triple junctions by level sets. *Interfaces and Free Boundaries 4*, 3.

SONG, P., FU, C.-W., AND COHEN-OR, D. 2012. Recursive interlocking puzzles. *ACM Trans. Graph. (SIGGRAPH) 31*, 6 (Nov.), 128:1–128:10.

STAVA, O., VANEK, J., BENES, B., CARR, N., AND MĚCH, R. 2012. Stress relief: Improving structural strength of 3D printable objects. *ACM Trans. Graph. (SIGGRAPH) 31*, 4 (July), 48:1–48:11.

TELEA, A., AND JALBA, A. 2011. Voxel-based assessment of printability of 3D shapes. In *Proc. of ISMM*, 393–404.

VANEK, J., GALICIA, J. A. G., BENES, B., MCH, R., CARR, N., STAVA, O., AND MILLER, G. S. 2014. Packmerger: A 3D print volume optimizer. *Computer Graphics Forum 33*, 6 (Sept.), 322–332.

VESE, L. A., AND CHAN, T. F. 2002. A multiphase level set framework for image segmentation using the Mumford and Shah model. *Int. J. Comput. Vision 50*, 3 (Dec.), 271–293.

WANG, W., WANG, T. Y., YANG, Z., LIU, L., TONG, X., TONG, W., DENG, J., CHEN, F., AND LIU, X. 2013. Cost-effective printing of 3D objects with skin-frame structures. *ACM Trans. Graph. (SIGGRAPH Asia) 32*, 6 (Nov.), 177:1–177:10.

WHITING, E., SHIN, H., WANG, R., OCHSENDORF, J., AND DURAND, F. 2012. Structural optimization of 3D masonry buildings. *ACM Trans. Graph (SIGGRAPH Asia). 31*, 6 (Nov.), 159:1–159:11.

WU, S., KAY, M., KING, R. E., VILA-PARRISH, A., AND WARSING, D. 2014. Multi-objective optimization of 3D packing problem in additive manufacturing. In *Proceedings of the 2014 Industrial and Systems Engineering Research Conference*.

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. Graph. (SIGGRAPH) 30*, 4 (July), 97:1–97:8.

YANG, Y.-L., AND HUANG, Q.-X. 2013. Traygen: Arranging objects for exhibition and packaging. *Comput. Graph. Forum (Pacific Graphics) 32*, 7, 187–195.

ZHAO, H.-K., CHAN, T., MERRIMAN, B., AND OSHER, S. 1996. A variational level set approach to multiphase motion. *J. Comput. Phys. 127*, 1 (Aug.), 179–195.

ZHOU, Q., PANETTA, J., AND ZORIN, D. 2013. Worst-case structural analysis. *ACM Trans. Graph. (SIGGRAPH) 32*, 4 (July), 137:1–137:12.