DOI: 10.1111/cgf.13490
Eurographics Symposium on Geometry Processing 2018
T. Ju and A. Vaxman
(Guest Editors)

Volume 37 (2018), Number 5

Packing Irregular Objects in 3D Space via Hybrid Optimization

Y. Ma[1], Z. Chen[†2], W. Hu[1] and W. Wang[1]

[1]Department of Computer Science, The University of Hong Kong, Hong Kong, China
[2]Department of Computer Science, Xiamen University, Xiamen, China

Abstract

Packing problems arise in a wide variety of practical applications. The basic problem is that of placing as many objects as possible in a non-overlapping configuration within a given container. Problems involving irregular shapes are the most challenging cases. In this paper, we consider the most general forms of irregular shape packing problems in 3D space, where both the containers and the objects can be of any shapes, and free rotations of the objects are allowed. We propose a heuristic method for efficiently packing irregular objects by combining continuous optimization and combinatorial optimization. Starting from an initial placement of an appropriate number of objects, we optimize the positions and orientations of the objects using continuous optimization. In combinatorial optimization, we further reduce the gaps between objects by swapping and replacing the deployed objects and inserting new objects. We demonstrate the efficacy of our method with experiments and comparisons.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems

1. Introduction

Packing is a classical problem in computational geometry and operational research. It appears under different names in literature, e.g. shape stock cutting problem, strip packing problem, knapsack problem, or nesting problem [Dyc90]. The basic problem is to make full use of the space to reach a maximum profit. In this paper, we consider the coverage rate of a container's space as the profit and aim to load as big volume sum of objects as possible for a specified container.

Packing algorithms using 3D objects are widely used in practice. Besides the traditional ship loading problem, an emerging application is rapid prototyping. 3D printing is one of these popular manufacturing techniques. It builds up a solid object by laying down successive layers of material. For a powder-based 3D printer, the unsintered material is used to fill the printing volume layer by layer to support the objects built, which costs redundant manufacturing time. It can save much time to pack objects densely in the printing volume and minimize the amount of supporting powder.

There have been extensive methods for 2D packing problems and comprehensive surveys on these methods have been provided in [Dyc90, SP92, DD95, RRS13, ZCT16]. Packing in 3D is much more difficult because of the increasing degrees of freedom in translation and rotation in 3D space. Many previous approaches focus on packing simple geometric objects, like cuboid [CPT12, YSNK00], sphere [Mac62], cylinder [SC09], and ellipsoid [Kal17]. They take advantage of the special geometric properties to pack them densely into a box. But these methods are always hard to be extended to more general situations. To solve the problem of packing objects with more complicated shapes, many studies have proposed algorithms for packing convex [SGS*05] or both convex and concave polyhedrons [LLC*15, ENB09, RBSP18]. They perform well for polyhedrons with few facets and vertices but are not applicable or extremely time-consuming for handing complex 3D models with hundreds or thousands of vertices and facets. Some following works [JW01, DKB13, BTW09] pack irregular objects by voxelizing them. However, such voxel-based representations introduce approximation errors for packing objects bounded by smooth surfaces.

In this paper, we consider the most general forms of irregular shape packing problems in 3D space, where both the containers and the objects can be of any shapes, and free rotations of the objects are allowed. For a given container, we aim to pack as many objects as possible into it to reach a relatively large coverage rate. If the size of a container is flexible, our method can also be used for computing a minimal size of container for given objects. Our algorithm is based on the work about 2D mosaic synthesis [HCP*16], which synthesizes digital surface mosaics with irregularly shaped tiles. However, they pack 2D tiles on curved surfaces and allow tiles to be scaled. Our algorithm packs 3D solid objects in 3D containers and guarantees all the objects have their original sizes in the packing results. Our solution is optimization-based, which combines continuous local optimization and combinatorial optimization. In

† Corresponding author: chenzhonggui@xmu.edu.cn

local optimization, we use a growth-based strategy to iteratively adjust the positions and orientations of packing objects concurrently to achieve tight packing. In combinatorial optimization, we adopt swapping, replacement and hole filling operations to escape from local optima.

The remainder of this paper is organized as follows. We give a brief introduction of prior work on packing problems in Section 2. In Section 3, we provide an overview of the problem description and notations. In Section 4, we present our 3D packing algorithm for irregular objects in detail. Implementation details and experimental results are shown in Section 5. We conclude and discuss the limitations in Section 6.

2. Related Work

Here we only review some closely related packing methods for packing irregular shapes in 3D space. For more comprehensive reviews on packing problems, we refer the readers to [Dyc90, SP92, DD95].

Polyhedron Packing: Packing a set of different polyhedrons into a cuboid for minimal sizes can be concluded as the polyhedron packing problem, which has been studied for decades [CEG89, IBK*97]. The packing problem is NP-complete [CEG89], so it is in general very difficult to solve exactly. General solution methods can be divided into two main categories: one uses heuristics [CKE*14,ROGL16,LZLX10,TGD14] and the other uses mathematical modeling and general optimization [CSR10, Fas13, TJ09, SGPS04]. Egeblad et al. [ENB09] presented an efficient method for packing multi-dimensional polyhedrons in a polyhedron container. It minimizes the overlapping volume between polyhedrons by finding an exact one-dimensional translation. However, this algorithm does not consider free orientations of the objects. Another heuristic constructive algorithm called HAPE3D [LLC*15] is based on the principal of the minimum total potential energy for irregular polyhedrons. This algorithm allows both translations and rotations but is unable achieve high coverage rates. Recently, based on mathematical modeling, Romanova et al. [RBSP18] formulated the packing problem as a nonlinear programming problem. Their algorithm is able to find a local optimum by using quasi phi-functions. It has good performance in packing polyhedrons densely in a cuboid. However, all these polyhedron packing algorithms are applicable for objects with few facets. It is hard or extremely time-consuming for them to process objects with complex surfaces.

Irregular Shape Packing: There are some difficulties [JW01] to solve the packing problem for objects with arbitrary shapes: the difficulty in modeling irregular objects mathematically, the efficiency of overlapping detection between objects, and the difficulty in getting a dense packing efficiently. To solve these difficulties, one method [JW01] is to represent objects with a set of voxels arranged on a regular grid. The representation accuracy will increase with the resolution increasing. It takes the box container as a n^3 matrix and each element stands for one voxel. By this method, it can represent arbitrary shapes and is easy to do overlapping detection. The shortcoming is high memory occupancy and low packing densities with large gaps. Based on this representation, Byholm et al. [BTW09] carried out algorithmic optimization by considering only the outer shells of the objects to save time and memory in collision detection and rotation searching. However, voxel-based methods always have errors in representing objects with smooth surfaces and many details. Even though increasing the resolution of voxelization will decrease the error, the accompanying costs of memory and time are enormous.

Segmenting and Packing: With the development of 3D printing technologies, some algorithms aim to segment big objects into several parts and tightly pack them in a relatively small printing cuboid to save the supporting materials and total printing time. The PackMerger method [VGB*14] converts 3D objects into shells and breaks them into smaller parts, and then tightly pack them into a smaller volume by optimization. The Dapper method [CZL*15] progressively packs one pile in the printing volume and tries to minimize the height needed for printing. These method are designed specifically for packing several segments of one object in a box. They are a combination of segmenting and packing.

3. Problem Description

For a given arbitrary container and a set of objects with arbitrary shapes, our 3D packing algorithm aims to fill as much space of the container as possible by optimizing the layout of loaded objects. The container denoted by C and the set of n objects denoted by $B = \{B_1, B_2, \ldots, B_n\}$ are all represented as closed manifold triangle mesh surfaces. If the input model is a non-closed surface with holes, we use the surface completion approach [ZGL07] to obtain a watertight model. Let V denote the volume function, and the coverage rate is computed as $R = \sum_{j=1}^{m} V(B_j)/V(C)$, where $m \leq n$ stands for m selected packing objects from total n candidates. The upper bound for R is $min(\sum_{i=1}^{n} V(B_i)/V(C), 1)$. The goal of our algorithm is to obtain the optimal coverage rate using our hybrid optimization techniques.

The state of a deployed object can be expressed as $\phi = [s, P, O]$, where $s \in (0, 1]$ is the current scale factor with respect to the original size of the object, $P = (p_x, p_y, p_z)$ are the centroid coordinates of the object in 3D space, and $O = (o_x, o_y, o_z)$ are the rotating angles about x-, y-, and z-axis, respectively. Our algorithm guarantees that each deployed object keeps its original size in the packing result without overlaps with other objects, and never goes beyond the container boundaries in the whole optimization process.

4. Packing Algorithm

Our general formulation of the packing problem in 3D is made difficult by complex shapes of the packing objects and the large number of different possible packing layouts. We propose an efficient method to obtain sub-optimal solutions by combining continuous optimization and combinatorial optimization. Fig. 1 shows the pipeline of our method. Continuous optimization is used to compute the best position and orientation of each object with respect to its neighboring objects to achieve compact packing, while combinatorial optimization, including object swapping, replacement and hole filling, helps to find a better packing layout of the objects to further improve the packing density. In the following, we will give details of each step.

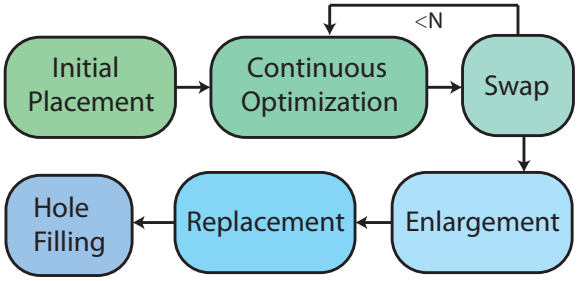


Figure 1: *Pipeline of our 3D packing algorithm.*

4.1. Initial Placement

For a given container C with volume $V(C)$, we randomly select objects from B one by one until the accumulated volume will exceed $r \cdot V(c)$ if add more. Here r is a parameter determining the coverage rate of the initial placement. Because the packing result or the final coverage rate is related to many factors, like the volumes and shapes of objects, the layout of all the objects, and the shape of the container. It is hard to decide the best choice of initial objects. So we use a random selecting strategy. From our experiments, the resulting coverage rates are generally between 0.2-0.5, thus we use $r = 0.3$ as the initial guess of the coverage rate for the initial selection of objects. If the initial coverage rate is too high, many objects cannot grow to their initial sizes and we have to discard them, which will waste some space. If it is too low, we could not make full use the space of the container to load objects. Assuming we have selected m objects, the next step is to divide the container into m subregions according to the volumes of objects. Because objects have different volumes, it would be better to allocate an appropriate space for each object initially. We use the centroidal power diagram method [XLC*16] to compute a partition of C with specified volume constraints that each power cell of the partition has a volume proportional to the volume of its associated object. Then the centroids of the power cells are taken as the initial positions for the selected objects.

There are two potential problems by this initialization. One is that it is possible to cause overlaps between objects, as we divide the container just according to objects' volumes, without considering their shapes. We solve this problem by shrinking all the objects to a small size (usually 10% of the original size in our experiments) in the first step of each round of continuous optimization. The other problem is that it is easy to get trapped in a bad local optimum in the optimization. Because our continuous optimization just locally adjusts the positions and orientations of the objects, the packing result depends heavily on its initial state. We solve this problem by using combinatorial optimization, including object swapping, replacement and hole filling, to reduce the dependence on the initialization.

4.2. Continuous Optimization

This continuous optimization consists of two main steps, shrinking objects to a small size and iteratively adjusting the configuration of objects, including their sizes, positions, and orientations, so that they can grow gradually to their initial size. We use such iterative relaxation scheme to avoid overlaps and achieve a relatively good result of the NP-hard packing problem.

Algorithm 1: Continuous Optimization

Input: C, a container surface mesh
$\{B_1, B_2, \ldots, B_m\}$, a set of objects in C
$\{b_1, b_2, \ldots, b_k | b_i < b_{i+1}\}$, scaling factor barriers
itn_{max}, the maximum iteration number for each b_j
Output: A compact layout for packed objects in C

```
Shrink objects to 10% of their original size;
for each b_i do
    itn = 0;
    while itn < itn_max do
        Compute CAT;
        for each object B_j do
            Apply growth-based optimization (compute the
              largest B_j in its CAT cell with a scaling factor f_j,
              a three dimensional rotation transformation θ_j
              and a translation t_j);
            if scale rate s_j f_j < b_i then
                Apply transformation T(f_j, θ_j, t_j) to B_j;
            else
                Apply transformation T(b_i/s_j, θ_j, t_j) to B_j;
            end
        end
        itn = itn + 1;
    end
end
```

Pseudo-code of this optimization is given in Algorithm. 1. For each object B_j, we compute its largest copy in its associated region (CAT cells in Section 4.2.1) with a scaling factor f_j, a three dimensional rotation transformation θ_j and a translation t_j. To prevent some objects from growing too fast that they hinder other neighboring objects' growth, we define a sequence of thresholds $\{b_1, b_2, \ldots, b_k\}, b_k = 1$ for the scale factor with increasing order to make all the objects grow almost simultaneously. We set the interval between adjacent b_i's as 0.1. For each scale threshold, we iteratively use a growth-based optimization (Section 4.2.2) to find a better configuration.

4.2.1. 3D Chordal Axis Transform

To construct an individual space for each object to grow, we need to build clapboards between all the objects to separate them. Chordal Axis Transform (CAT) [Pra97] is like the middle clapboards in the vacant space among objects, so we use CAT to divide the container into separated regions and construct a specific cell for each object. We first uniformly sample points on the surfaces of the container and objects. Usually, we directly use the vertices of a mesh surface if they are uniformly distributed. As CAT can be extracted from a Constrained Delaunay Tetrahedralization (CDT) [SG05, Si15] , we use these sampling points as the input to compute the CDT. After building the CDT, we only consider those tetrahedrons whose vertices are not from the same object. They can be classified into four categories (Fig. 2). A container here can be regarded as a spe-

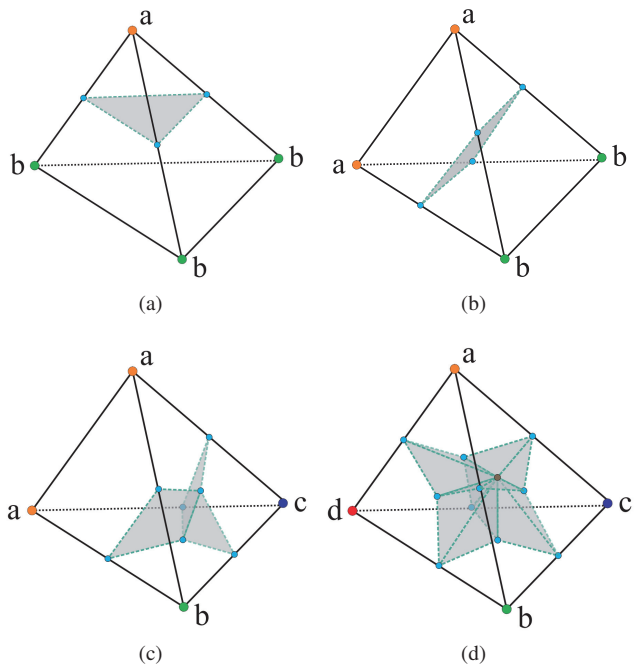


Figure 2: *Four cases for extracting CAT. (a) 1+3 case; (b) 2+2 case; (c) 1+1+2 case; and (d) 1+1+1+1 case. The vertices with different labels in a tetrahedron belong to different objects. Otherwise, they come from the same object. Blue nodes are midpoints of tetrahedron edges or centers of tetrahedron facets. Grey facets are extracted CAT facets.*

cial object. If vertices of one tetrahedron come from different objects, we use different labels and colors to distinguish them. For a tetrahedron, we compute midpoints of edges with two different end labels, and the centers of facets with three different vertex labels. CAT facets can be extracted by connecting related nodes, as shown in Fig. 2. Because every object is enclosed in a set of tetrahedrons, it is also enclosed in a closed CAT cell which is composed of a set of continuously connected CAT facets. Fig. 3 shows the result of extracting CAT cells when objects are shrunk to a small size.

4.2.2. Growth-based Optimization

Note, each object now locates in an independent space which is a CAT cell. We let objects grow in their individual space to guarantee that they never overlap with each other and never cross over the container. Our growth-based optimization is to optimize the position P, orientation O and size s for each object so that they could grow to a big size in their current cells. The transformation is denoted as $T = (f, \theta, t)$, where f is the scale factor for object's current size, $\theta = (\theta_x, \theta_y, \theta_z)$ represents the rotation angle about x-, y-, and z-axis, and $t = (t_x, t_y, t_z)$ indicates the translation along x-, y-, and z-axis.

We solve the transformation in a local coordinate system for every object. Then the transformation in homogeneous coordinates can be defined as follows:

$$T = \begin{pmatrix} R & t \\ 0 & f \end{pmatrix}, \quad R = R_x R_y R_z,$$

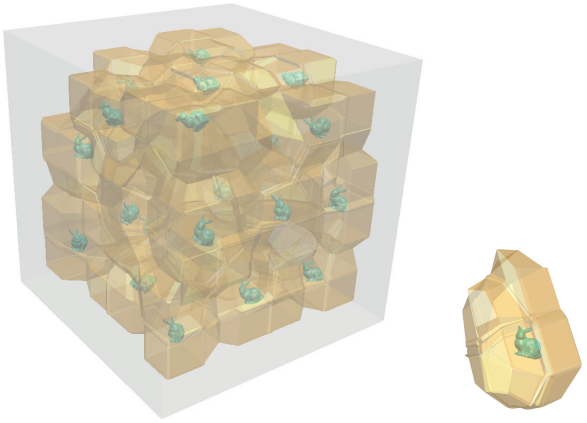


Figure 3: *Extracting CAT cells. (a) CAT cells; and (b) an object and its associated CAT cell.*

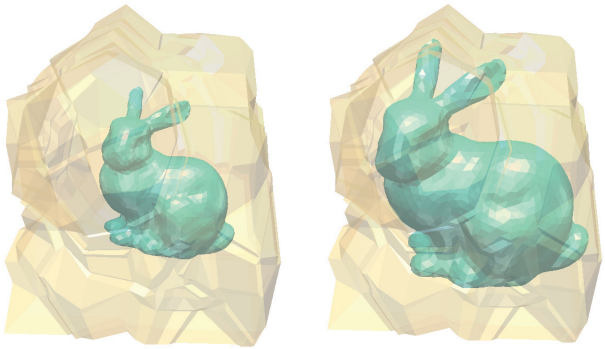
where

$$R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & \sin\theta_x \\ 0 & -\sin\theta_x & \cos\theta_x \end{pmatrix}, R_y = \begin{pmatrix} \cos\theta_y & 0 & -\sin\theta_y \\ 0 & 1 & 0 \\ \sin\theta_y & 0 & \cos\theta_y \end{pmatrix},$$
$$R_z = \begin{pmatrix} \cos\theta_z & \sin\theta_z & 0 \\ -\sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

The container and objects can have arbitrary shapes. We formulate this problem as an optimization with nonlinear constraints. Each vertex v_i of an object is related to a set of constrained Delaunay tetrahedrons and then it will correspond to a set of CAT facets F_i of its CAT cell. We take F_i as a group of constraints for v_i to limit its scope of movement to prevent the object from leaving the CAT cell. Assuming the object has n vertices, we have $\sum_{i=1}^{n} w(v_i)$ constraints in total for this object, where $w(v_i)$ is the size of F_i. The growth-based optimization can be formulated as follows.

$$\begin{aligned} &\max \quad F(f, \theta_x, \theta_y, \theta_z, t_x, t_y, t_z) = f, \\ &s.t. \frac{(T \cdot v_i - q_j) \cdot \vec{n}_j}{\|\vec{n}_j\|} \geq 0, i = 1, 2, \ldots, n, j = 1, 2, \ldots, w(v_i), \\ &f, t_x, t_y, t_z \in R^+, \\ &\theta_x, \theta_y, \theta_z \in [-\frac{\pi}{12}, \frac{\pi}{12}], \end{aligned}$$

where q_j denotes a certain point in the jth facet of F_i, $\vec{n}_j$ represents the normal of the jth facet of F_i which points to the side of v_i. We limit the changing amount of the rotation θ to a small range in each optimization, because the constraints for each vertex is in a local area and may lose the control when the rotation angle is large. Fig. 4 shows the result after executing the constrained growth optimization for a bunny. The scale, position, and orientation all have changed, and at the same time the bunny still locates inside the CAT cell. For each scaling factor barrier b_i in the configuration optimization, we run the growth-based optimization for several times (itn_{max}) to make all the objects approach the barrier as much as



(a) (b)

Figure 4: *Object grows in a CAT cell. (a) Before growth. (b) After growth.*

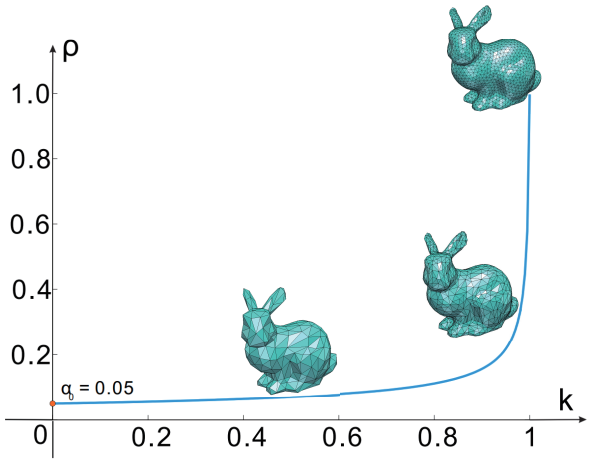


Figure 5: *The plot of* $\rho(k)$ *with* $\alpha = 0.05$ *and* $\beta = 0.5$. *From left to right, the bunnies indicate* $k = 0.49$ *with 139 vertices,* $k = 0.95$ *with 435 vertices, and* $k = 1$ *with original 1998 vertices, respectively.*

possible. We update the CAT cells for objects after each time of growing.

4.2.3. Adaptive Mesh Sampling

The run time will increase with the number of sampling points of objects increasing. One reason is that the CDT computation and CAT extraction are related to the amount of sampling points. The other one is that the number of constraints in the constrained growth optimization depends on the number of CAT facets. When objects have small sizes during the growth period, it is applicable to use simplified mesh surfaces instead of the original meshes. Because at that time, it has little chance to collide with other objects. Then we use the vertices of object surfaces as the sampling points. To speed up our algorithm, we adopt an adaptive mesh sampling method to simplify the input for each iteration of configuration optimization.

The mesh simplification method [LT98, LT99] we use is boundary and volume preserving. We define the simplified level adaptively according to the scale factor of objects. We use the following function to compute appropriate stop condition for simplification.

$$\rho(k) = \alpha(1 + \alpha^{\frac{1}{\beta}} - k)^{-\beta},$$

where k is the current scale factor of objects, α is a small number, and β is a constant in $(0,1)$ ($\alpha = 0.05$, $\beta = 0.5$ in our experiments). $\rho(k)$ is a percentage number and locates in $(0,1]$, which stands for that the simplification processing stops when the edges of mesh surface have been simplified to $\rho(k)$ times of the original number. When k is approaching 1, the value of $\rho(k)$ will increase rapidly. Otherwise, the simplification will keep at a high level. It meets our requirement that uses coarse representation when growing but accurate representation when approaching original scale. Fig. 5 shows the functional figure of the simplification level and the scale factor of an object. Fig. 6 illustrates the run time performance against the average number of vertices of packing objects with the machine configuration in Section 5.

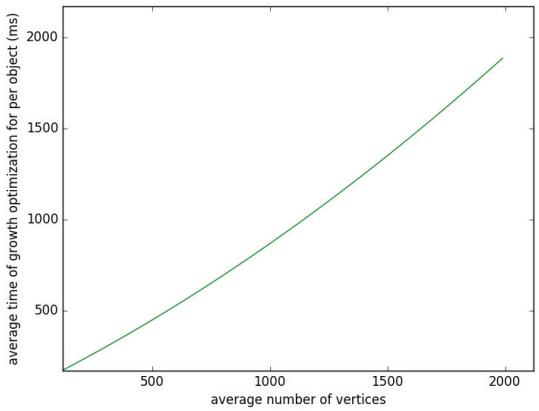


Figure 6: *Relationship between the average run time of the growth-based optimization for each object and the average number of vertices of objects.*

4.3. Object Swapping

The continuous optimization is a kind of local optimization, which highly depends on the initialization. The object adjusts their configuration in the nearby area of the initial place. It is easy to fall into a local optimum. Packing problem is not only related to the configuration of an object itself, but also related to the permutation of objects. So we use some heuristic methods to help our algorithm escape from the strong dependence to initialization. The available growing space for an object is much related to its surrounding objects. If an object matches the shape and volume of its corresponding cell well, it will make full use of the cell space to grow as much as possible. So we adopt swapping strategy to help objects find more appropriate cells and at the same time to help cells find more suitable objects. It is the first method we use to break the limitation of initialization.

In this section, we use the CDT cell, which is a closed surface

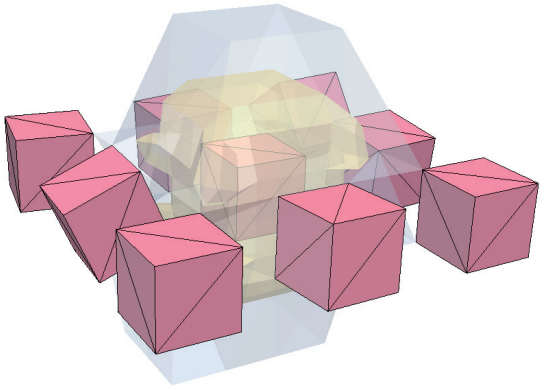


Figure 7: *Comparison between the CAT cell and the CDT cell. For the middle pink object, the blue transparent cell is the CDT cell and the inside yellow transparent cell is the CAT cell.*

extracted from all the related constrained Delaunay tetrahedrons for one object. It's bigger than the corresponding CAT cell and the CAT cell is inside it. The example in Fig. 7 shows the comparison and relationship between CAT cell and CDT cell for one object. We use Algorithm. 2 to compute the suitability score for one object and one CDT cell. At first, we shrink object and translate it to the center of CDT cell. We randomly test 20 orientation as the initial orientation and process the growth optimization. The final maximal scale factor of an object after iterative growth in CDT cell is taken as the suitability between the object and the CDT cell. Because the CDT cell may contain some facets of other objects, we shrink the target object after growth by 0.99 to avoid the connection with other objects if the object grows to a large size. The reason why we use scaling factor barriers is to avoid the situation that growing rapidly will cause blocks in partial areas of CDT cell. Since rotation changes are limited to a small range for growth optimization, increasing the times of this optimization will help objects find a more appropriate configuration with big size. For each CDT cell, we compute the suitability scores for all the objects and build a bipartite graph. Then, we find a match with the highest sum of suitability scores by one assignment algorithm [Mun57].

Each object has a set of neighbors, some of whose vertices share the same CDT tetrahedrons with the object's vertices. Because the CDT cells have overlaps between any two objects with neighborhood. To avoid adjacent objects after swapping and growing have overlaps, we divide all the objects inside the container into some groups. Any pair of members in one group do not have neighboring relationship. Then for each group, we use the method above to swap the positions for its objects to get a better permutation and then renew the CDT cells for next group. We repeat the continuous optimization and swapping process for N times ($N = 8$ in our experiments).

4.4. Object Enlargement

A CAT cell contains the facets that divide the space between the object and its neighboring objects relatively evenly. That is to say,

Algorithm 2: Suitability Score Computation

Input: D, the CDT cell surface
B, the target object
$\{b_1, b_2, \ldots, b_k | b_i < b_{i+1}, b_k = 1\}$, scaling factor barriers
Output: S, the suitability score
for $n < 20$ **do**
  Copy B to B';
  Apply an random orientation to B';
  Shrink object B' and translate it to the center of D;
  for *each* b_i **do**
    Apply growth-based optimization in CDT cells (Get (f, θ, t));
    if *scale rate* $sf < b_i$ **then**
      Apply transformation $T(f, \theta, t)$ to B';
      Shrink object B' by 0.99;
    else
      Apply transformation $T(\frac{b_i}{s}, \theta, t)$ to B';
    end
  end
  n = n+1;
end
S = s;

the space in a CAT cell is smaller than the actual room where the object could grow. However, the facets of a CDT cell to some extent stand for the maximal available space the object could use. Because objects grow in their CAT cells in the optimization phases above, some of them may still do not reach their original size. In this enlargement step, we let them grow in CDT cells. Fig. 8 shows an example of enlarging some objects. The enlargement strategy performs well to improve the size of objects, especially for those ones whose CDT cells are obviously bigger than their CAT cells.

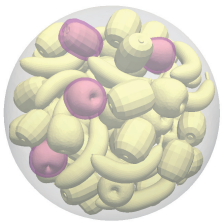
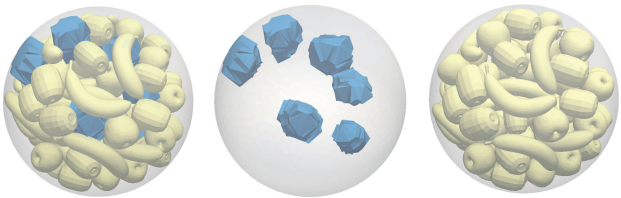


Figure 8: *Objects after enlargement are drawn in semitransparent red.*

For the set of objects that do not reach their original size, we sort these objects in decreasing order of their scaling factors. Then, we use the same method as the suitable score computation to enlarge them in CDT cells sequentially. We have mentioned in the last section that CDT cells of two neighboring objects have overlaps. We can enlarge non-adjacent objects simultaneously to speed up the algorithm.

4.5. Object Replacement

Similar to the reason of swapping strategy, the shape of a CDT cell is much related to the shapes and configurations of its neighboring objects and it can hinder the growth of inside object if the shape of a CDT cell does not match that of the object well. The difference between replacement and swapping is we use new unpacked object to replace some unsuitable objects rather than exchanging inside. After previous operations, for these objects whose scaling factors are still smaller than 1, we search in the rest unpacked objects for one



(a) (b) (c)

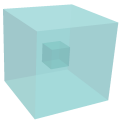
Figure 9: *Hole filling. (a) Detect holes between packed objects. (b) Detected holes (CDT cells). (c) Fill holes with appropriate objects and apply growth optimization.*

with maximal suitability score, and use it to replace the deployed object. If the maximal value of suitability score is smaller than 1, we do continuous optimization to the new object and its neighbors to further make the new object approach its original size. We just shrink them to 0.9 times of their previous scaling factor at the beginning of the continuous optimization to avoid large changes to its neighbors.

In this process, we do not want to sacrifice the neighbors to expand the alternative object. So if the local configuration optimization causes the shrink of any neighbor or the new object still does not reach its original size, we recover neighbors to configurations before new alternative object coming and delete the old object directly.

4.6. Hole Filling

To make full use of the space of the container and improve the coverage rate, we detect holes of the rest space and find appropriate objects from the rest unpacked objects to fill them.



We take the container and already deployed objects as a whole, denoted as W. The container is still the outside surface of W and these packed objects are inside surface of W. Previous vacant space inside the container is the main body of W currently. Like the right figure, assuming the outside box is the container and the inside box is the only object, W contains the space that the container minus the object. We know that the diameter of the medial sphere of medial axis [DZ04] in 3D denotes the thickness in the local area around the center of the medial sphere. We take advantage of this concept to find available space for filling new objects in W.

The CDT among packed objects and the container are the inner tetrahedrons of W. We compute the circumspheres for all the tetrahedrons and let M represent the set of circumspheres. M is just the set of approximated medial spheres and we select these spheres from M, which are not covered or crossed by other bigger medial spheres and whose radius is bigger than a threshold, as available vacant spherical space inside the container. For each spherical space, we put a sphere with half-length radius of corresponding medial sphere on the center. And then, we compute the CDT cells for them. Then, we select objects with similar volumes to these cells and place them in the center of the corresponding cells. Shrink new added objects and let them iteratively grow in local CDT cells. We abandon some of them, which could not reach their original size after growth. Fig. 9 shows one example of holes filling.

The circumspheres of CDT will infinitely approach the medial spheres when the sampling points used for constructing CDT are infinite. To improve the accuracy of estimation for the size of available holes, we use densely and evenly sampling points on the surface of objects and the container to compute CDT in this phase.

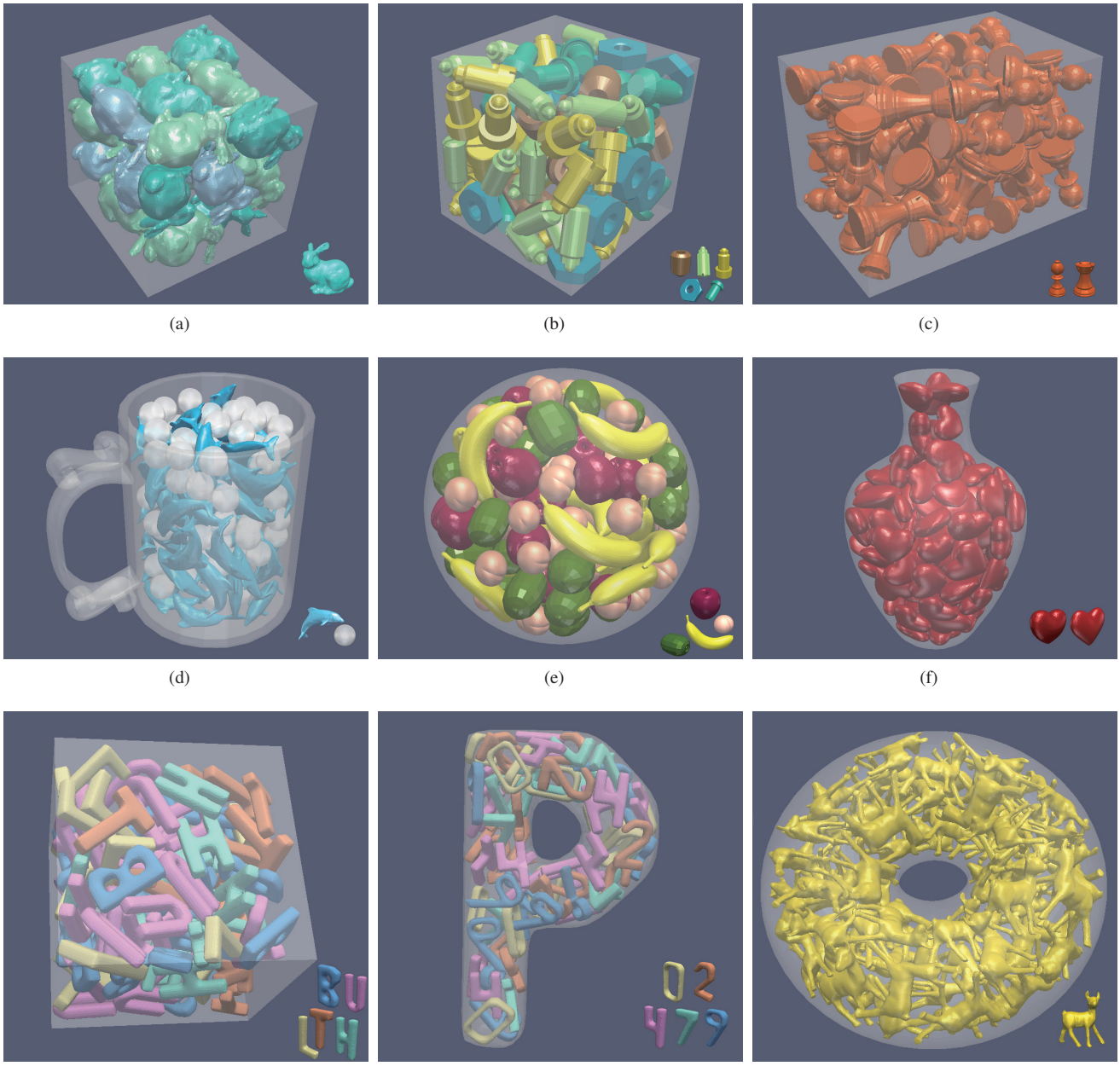
5. Results

In this section, we highlight the performance of our algorithm on packing heterogeneous 3D objects in arbitrary containers. We implemented the algorithm in C++ and conducted all experiments on a computer with Intel i7-6700 CPU and 8GB RAM. Although the most time-consuming steps of our algorithm, including continuous optimization and suitability score computation, can be easily parallelized on multiple cores. All the results in this paper were generated by using a single CPU core.

Fig. 10 shows nine results of our algorithm. For each example, we illustrate all the kinds of packing objects on the right-down corner. We use arbitrary objects and containers, including convex and non-convex shapes, to show our algorithm's ability in packing different kinds of models in irregular containers. In 3D, the coverage rate 45% means that objects have covered by average 76.6% of the space along x-, y-, and z-axis. They are relatively high packing densities for irregular packing. In Fig. 10(g-i), we use highly concave objects, whose bounding boxes are different from themselves, like the letters, the numbers and the dog with spindlelegs. We show some zoom-in views of some details in Fig. 11. We can see that objects can get into other concave objects to use available spaces by our algorithm.

To show the performance of our algorithm, we divide a cube, regarded as the original cube, into $4 \times 4 \times 4$ small cubes and throw them with rotation disturbance into a container, which is also a cube but a little bigger than original cube (1.04 times bigger in our experiment). Fig. 12 shows the packing result by our algorithm. Every small cube has grown to their initial size. The reason why we do not use the original cube as the container is that each two objects could not touch each other in our algorithm. If using the original cube as the container, small cubes inside could not grow to their initial scale.

As for the comparison experiments, there are three main areas mentioned in Section 2 similar to our work. However, the optimized goal of segmenting and packing is different from ours. The voxel-based arbitrary shapes packing is not applicable for mesh represented objects packing because of big approximation errors. The most similar work is the polyhedron packing algorithms. We compare our algorithm with the latest paper [RBSP18] and the result is shown in Fig. 5. Using the same dataset and container, our algorithm uses 3.98% computational time of [RBSP18] with comparable result. When dealing with mesh represented complex objects, the superiority of our algorithm in efficiency could be much more obvious. In addition, the method in [RBSP18] applies to cuboid containers only.

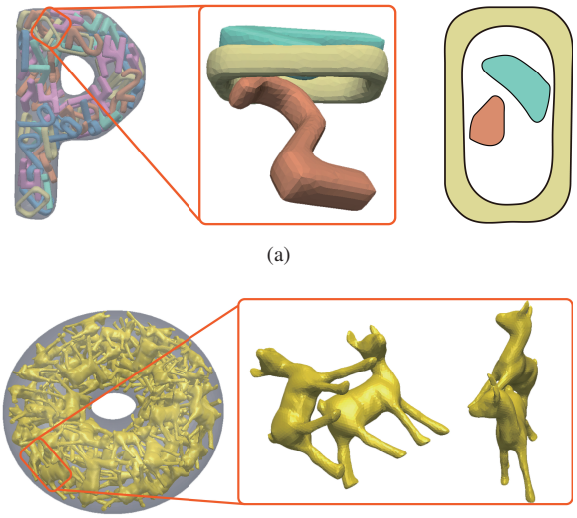


(a) (b) (c)

(d) (e) (f)

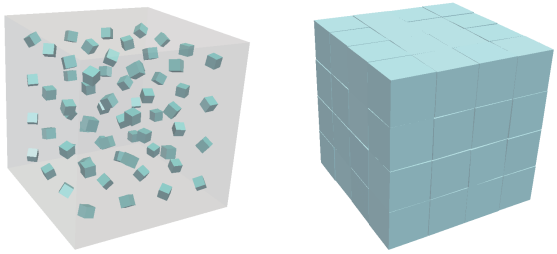
(g) (h) (i)

Figure 10: *Results. In the bottom right of each subfigure, we show all the kinds of objects we use in the corresponding experiment. The set of objects B we are going to pack contains the same number of copies of them. (a) Packing 25 bunnies in a box with a coverage rate* 45% *and computational time 28 min. (b) Packing 77 tools in a box with a coverage rate* 47.5% *and computational time 33 min. (c) Packing 60 chess pieces in a box with a coverage rate* 32.5% *and computational time 40 min. (d) Packing 129 dolphins and bubbles in a cup with a coverage rate* 30% *and computational time 46 min. (e) Packing 108 fruits in a sphere with a coverage rate* 35.7% *and computational time 37 min. (f) Packing 125 hearts in a vase with a coverage rate* 40.3% *and computational time 29 min. (g) Packing 104 letters in a cube container with a coverage rate* 26.2% *and computational time 48 min. (h) Packing 66 numbers in a P-shape container with a coverage rate* 23.5% *and computational time 46 min. (i) Packing 67 dogs in a torus container with a coverage rate* 19.6% *and computational time 58 min.*



(a) (b)

Figure 11: *Zoom-in views of Fig. 10(h) and Fig. 10(i). (a) From left to right is the original figure, selected three objects, and the cross section of the objects. (b) From left to right is the original figure, and two views of selected two objects.*



(a) (b)

Figure 12: *Packing $4\times4\times4$ rotated cubes in a cube container with 1.04 times of the volume of total small cubes. (a) Initial placement. (b) Packing result with 50 min.*

6. Conclusion and Limitation

We present a novel heuristic algorithm for packing irregular objects into an arbitrary-shaped container. We guarantee that there are no overlaps during the processing by separating individual space for each object and also guarantee that all the objects keep their original sizes in the final result. Our algorithm combines continuous optimization and combinatorial optimization to make the packing result has a large coverage rate. Parallel computation can be used in our algorithm to improve the efficiency. To our knowledge, our algorithm is the first that can be used to pack complex mesh represented objects into arbitrarily shaped containers.

One limitation for our algorithm is that we must fix the container at first and let a set of objects grow inside simultaneously. For the problem of searching the minimal height of container, we have to shrink or enlarge the height by executing several times of our algorithm according to bisection method, which may be less efficient than previous methods that put objects into the container one by

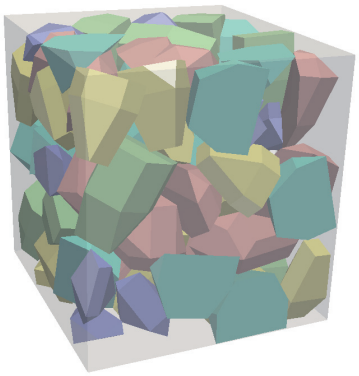
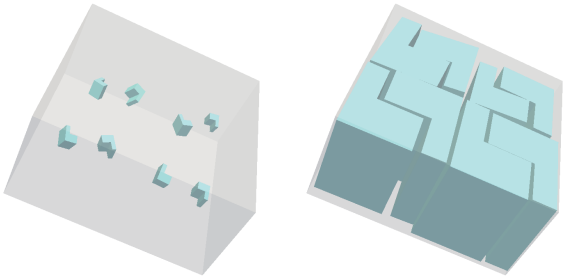
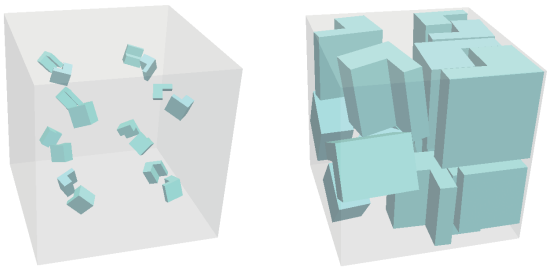


Figure 13: *Comparison with the Example 5(a) in the polyhedron packing paper [RBSP18]. They pack 80 polyhedrons (16 for each type of polyhedron in their dataset) with 18.85 hours and a coverage rate 53.9%. Using the same dataset, we pack 77 polyhedrons (15 for the first three types and 16 for the last two types of polyhedrons) with 45 min. and a coverage rate 51.3% .*



(a) (b)

Figure 14: *We divide a cuboid (half of a cube) container into 2×2 small cubes and each small cube is divided into two L-shape objects. We throw these L-shape objects with rotations into the original cuboid and then pack them by our algorithm. We keep the objects which do not grow back to their original volumes. (a) Initial placement. (b) Packing result with a coverage rate 69.9% and computational time 21 min.*



(a) (b)

Figure 15: *We divide a cube container into $2\times2\times2$ small cubes and repeat the experiment as done in Fig. 14. (a) Initial placement. (b) Packing result with a coverage rate 41.2% and computational time 43 min.*

one. Another limitation is that our algorithm is more about local optimization and hard to reach the global optimum. The ground truth of the packing result of Fig. 14 is that every two L-shape objects constitute a small cube and 4 small cubes form the container which is half of a cube. We shrink all the L-shape objects and add rotation perturbations to them in the initialization. All the pairs of L-shape objects are locked with each other and almost constitute small cubes by our algorithm. However, we still could not reach the global optimum finally. The result will become worse when we increase the number of L-shape objects. The ground truth of the packing result of Fig. 15 is to constitute a big cube. Such highly concave objects have strict requirements of orientations. With the number of objects increasing, the surrounding space of some objects becomes more complex. It becomes harder for them to find the optimal state.

Acknowledgements

The research of Zhonggui Chen was supported by the National Natural Science Foundation of China (No. 61472332) and the Natural Science Foundation of Fujian Province of China (No. 2018J01104). The research of Wenping Wang was partially supported by the Research Grant Council of Hong Kong (17208214).

References

[BTW09] BYHOLM T., TOIVAKKA M., WESTERHOLM J.: Effective packing of 3-dimensional voxel-based arbitrarily shaped particles. *Powder Technology 196*, 2 (2009), 139–146. 1, 2

[CEG89] CHAZELLE B., EDELSBRUNNER H., GUIBAS L. J.: The complexity of cutting complexes. *Discrete & Computational Geometry 4*, 2 (1989), 139–181. 2

[CKE*14] CHEN E. R., KLOTSA D., ENGEL M., DAMASCENO P. F., GLOTZER S. C.: Complexity in surfaces of densest packings for families of polyhedra. *Physical Review X 4*, 1 (2014), 011024. 2

[CPT12] CRAINIC T. G., PERBOLI G., TADEI R.: Recent advances in multi-dimensional packing problems. In *New technologies-trends, innovations and research*. InTech, 2012. 1

[CSR10] CHERNOV N., STOYAN Y., ROMANOVA T.: Mathematical model and efficient algorithms for object packing problem. *Computational Geometry 43*, 5 (2010), 535–553. 2

[CZL*15] CHEN X., ZHANG H., LIN J., HU R., LU L., HUANG Q.-X., BENES B., COHEN-OR D., CHEN B.: Dapper: decompose-and-pack for 3D printing. *ACM Trans. Graph. 34*, 6 (2015), 213:1–213:12. 2

[DD95] DOWSLAND K. A., DOWSLAND W. B.: Solution approaches to irregular nesting problems. *European Journal of Operational Research 84*, 3 (1995), 506–521. 1, 2

[DKB13] DE KORTE A., BROUWERS H.: Random packing of digitized particles. *Powder technology 233* (2013), 319–324. 1

[Dyc90] DYCKHOFF H.: A typology of cutting and packing problems. *European Journal of Operational Research 44*, 2 (1990), 145–159. 1, 2

[DZ04] DEY T. K., ZHAO W.: Approximate medial axis as a Voronoi subcomplex. *Computer-Aided Design 36*, 2 (2004), 195–202. 7

[ENB09] EGEBLAD J., NIELSEN B. K., BRAZIL M.: Translational packing of arbitrary polytopes. *Computational Geometry 42*, 4 (2009), 269–288. 1, 2

[Fas13] FASANO G.: A global optimization point of view to handle non-standard object packing problems. *Journal of Global Optimization 55*, 2 (2013), 279–299. 2

[HCP*16] HU W., CHEN Z., PAN H., YU Y., GRINSPUN E., WANG W.: Surface mosaic synthesis with irregular tiles. *IEEE transactions on visualization and computer graphics 22*, 3 (2016), 1302–1313. 1

[IBK*97] IKONEN I., BILES W. E., KUMAR A., WISSEL J. C., RAGADE R. K.: A genetic algorithm for packing three-dimensional non-convex objects having cavities and holes. In *ICGA* (1997), pp. 591–598. 2

[JW01] JIA X., WILLIAMS R.: A packing algorithm for particles of arbitrary shapes. *Powder technology 120*, 3 (2001), 175–186. 1, 2

[Kal17] KALLRATH J.: Packing ellipsoids into volume-minimizing rectangular boxes. *Journal of Global Optimization 67*, 1-2 (2017), 151–185. 1

[LLC*15] LIU X., LIU J.-M., CAO A.-X., ET AL.: HAPE3D-a new constructive algorithm for the 3D irregular packing problem. *Frontiers of Information Technology & Electronic Engineering 16*, 5 (2015), 380–390. 1, 2

[LT98] LINDSTROM P., TURK G.: Fast and memory efficient polygonal simplification. In *Visualization'98. Proceedings* (1998), IEEE, pp. 279–286. 5

[LT99] LINDSTROM P., TURK G.: Evaluation of memoryless simplification. *IEEE Transactions on Visualization and Computer Graphics 5*, 2 (1999), 98–115. 5

[LZLX10] LI S., ZHAO J., LU P., XIE Y.: Maximum packing densities of basic 3D objects. *Chinese Science Bulletin 55*, 2 (2010), 114–119. 2

[Mac62] MACKAY A.: A dense non-crystallographic packing of equal spheres. *Acta Crystallographica 15*, 9 (1962), 916–918. 1

[Mun57] MUNKRES J.: Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics 5*, 1 (1957), 32–38. 6

[Pra97] PRASAD L.: Morphological analysis of shapes. *CNLS newsletter 139*, 1 (1997), 1997–07. 3

[RBSP18] ROMANOVA T., BENNELL J., STOYAN Y., PANKRATOV A.: Packing of concave polyhedra with continuous rotations using nonlinear optimisation. *European Journal of Operational Research 268*, 1 (2018), 37–53. 1, 2, 7, 9

[ROGL16] RAMOS A. G., OLIVEIRA J. F., GONÇALVES J. F., LOPES M. P.: A container loading algorithm with static mechanical equilibrium stability constraints. *Transportation Research Part B: Methodological 91* (2016), 565–581. 2

[RRS13] REINERT B., RITSCHEL T., SEIDEL H.-P.: Interactive by-example design of artistic packing layouts. *ACM Transactions on Graphics (TOG) 32*, 6 (2013), 218:1–218:7. 1

[SC09] STOYAN Y., CHUGAY A.: Packing cylinders and rectangular parallelepipeds with distances between them into a given region. *European Journal of Operational Research 197*, 2 (2009), 446–455. 1

[SG05] SI H., GÄRTNER K.: Meshing piecewise linear complexes by constrained delaunay tetrahedralizations. In *Proceedings of the 14th international meshing roundtable* (2005), Springer, pp. 147–163. 3

[SGPS04] STOYAN Y., GIL M., PANKRATOV A., SCHEITHAUER G.: Packing non-convex polytopes into a parallelepiped. *Preprint MATH-NM-06-2004: Technische Universität of Dresden* (2004). 2

[SGS*05] STOYAN Y., GIL N., SCHEITHAUER G., PANKRATOV A., MAGDALINA I.: Packing of convex polytopes into a parallelepiped. *Optimization 54*, 2 (2005), 215–235. 1

[Si15] SI H.: Tetgen, a delaunay-based quality tetrahedral mesh generator. *ACM Transactions on Mathematical Software (TOMS) 41*, 2 (2015), 11. 3

[SP92] SWEENEY P. E., PATERNOSTER E. R.: Cutting and packing problems: a categorized, application-orientated research bibliography. *Journal of the Operational Research Society 43*, 7 (1992), 691–706. 1, 2

[TGD14] TASIOS N., GANTAPARA A. P., DIJKSTRA M.: Glassy dynamics of convex polyhedra. *The Journal of chemical physics 141*, 22 (2014), 224502. 2

[TJ09] TORQUATO S., JIAO Y.: Dense packings of the platonic and archimedean solids. *Nature 460*, 7257 (2009), 876. 2

[VGB*14] VANEK J., GALICIA J., BENES B., MĚCH R., CARR N., STAVA O., MILLER G.: Packmerger: A 3D print volume optimizer. In *Computer Graphics Forum* (2014), vol. 33, Wiley Online Library, pp. 322–332. 2

[XLC*16] XIN S.-Q., LÉVY B., CHEN Z., CHU L., YU Y., TU C., WANG W.: Centroidal power diagrams with capacity constraints: Computation, applications, and extension. *ACM Transactions on Graphics (TOG) 35*, 6 (2016), 244. 3

[YSNK00] YAMAZAKI H., SAKANUSHI K., NAKATAKE S., KAJITANI Y.: The 3D-packing by meta data structure and packing heuristics. *IEICE transactions on fundamentals of electronics, communications and computer sciences 83*, 4 (2000), 639–645. 1

[ZCT16] ZEHNDER J., COROS S., THOMASZEWSKI B.: Designing structurally-sound ornamental curve networks. *ACM Transactions on Graphics (TOG) 35*, 4 (2016), 99. 1

[ZGL07] ZHAO W., GAO S., LIN H.: A robust hole-filling algorithm for triangular mesh. *The Visual Computer 23*, 12 (2007), 987–997. 2