DOI: 10.1111/cgf.12353

COMPUTER GRAPHICS *forum*
Volume 00 (2014), number 00 pp. 1–11

PackMerger: A 3D Print Volume Optimizer

J. Vanek[1], J. A. Garcia Galicia[1], B. Benes[1], R. Měch[2], N. Carr[2], O. Stava[2] and G. S. Miller[2]

[1]Computer Graphics Department, Purdue University, West Lafayette, IN, USA
{vanek, garci191, bbenes}@purdue.edu
[2]Adobe Research
{rmech, ncarr, ostava, gmiller}@adobe.com

Abstract

We propose an optimization framework for 3D printing that seeks to save printing time and the support material required to print 3D shapes. Three-dimensional printing technology is rapidly maturing and may revolutionize how we manufacture objects. The total cost of printing, however, is governed by numerous factors which include not only the price of the printer but also the amount of material and time to fabricate the shape. Our PackMerger framework converts the input 3D watertight mesh into a shell by hollowing its inner parts. The shell is then divided into segments. The location of splits is controlled based on several parameters, including the size of the connection areas or volume of each segment. The pieces are then tightly packed using optimization. The optimization attempts to minimize the amount of support material and the bounding box volume of the packed segments while keeping the number of segments minimal. The final packed configuration can be printed with substantial time and material savings, while also allowing printing of objects that would not fit into the printer volume. We have tested our system on three different printers and it shows a reduction of 5–30% of the printing time while simultaneously saving 15–65% of the support material. The optimization time was approximately 1 min. Once the segments are printed, they need to be assembled.

Keywords: Digital Geometry Processing, Geometric Modeling, Computational Geometry Modeling

ACM CCS: Categories and Subject Descriptors I.3.5 [Computer Graphics]: Computational Geometry and Object Modelling, I.3.8 [Computer Graphics]: Applications.

1. Introduction

Additive manufacturing (3D printing) brings digital objects into the real world. A large variety of printing technologies exist, but they all share a few main characteristics. They create the object by adding layers of material that melt or are glued together and this process requires support in order to print overhangs. On any 3D printer, both the cost of the material (main and supporting) and time of printing are proportional to the amount of material used. This makes it prohibitive to print large objects, since the object volume grows with the third power of its size. There is also a limitation on the dimensions of single printed parts for a given printer.

One observation of our approach is that it is usually faster and cheaper to print a thin layer around the object surface (the object shell) rather than the entire volume [WWY*13]. Some printers already support this optimization, and those that do not can be made to do so by converting the input object into a thin shell. A second observation is that an object with thin parts can be divided into segments that can be packed for better printing while decreasing the printing time and saving the amount of support material. Also, objects that exceed the size of the printing tray must be segmented prior to printing [LBRM12]. The printed parts are then reassembled after printing, usually by gluing. As the price of the printing material can be high, the saved volume often outweighs the fact the object is segmented. There are several advantages to converting to thin shell and dividing the model prior to printing. The conversion into thin shells results only in a quadratic increase with scale of the amount of material and indirectly the printing time, making it more feasible to print larger objects at a reasonable cost. Shells or thin parts can be packed together into one printing batch, reducing the time needed for each print run and eventually saving the supporting material. In some cases, proper segmentation and orientation of shells can even make usage of support material unnecessary.

1.1. Related work

Detecting issues with objects intended for 3D printing is a recent topic in 'Computer Graphics'. Bickel *et al.* [BBO*10] used a data-driven model to design and fabricate materials with desired deformation behaviour for multi-material printers. Telea and Jalba

[TJ11] used voxel-based analysis to detect parts of the model unsuitable for printing (e.g. regions thinner than printer resolution). Stava *et al.* [SVB*12] presented a framework for detecting problematic parts of the model, and a method for their automatic correction by thickening, inserting supporting struts and hollowing. The approach of [ZPZ13] detects structural issues by performing approximate analysis and determining the loads based on geometry and material properties only. Hildebrand *et al.* [HBA13] addressed the problem of the directional bias in 3D printing with orthogonal slicing into segments and assigning the best possible printing direction for each segment. Recently, Bacher *et al.* [BBJP12] developed methods to segment articulated figures into fabricatable parts and designed friction joints to make the printed model fully capable of posing. A similar framework is proposed by Cali *et al.* [CCA*12], where special attention is paid to designing joints that do not wear out quickly. Close to our approach is the work of [LBRM12] where objects larger than the printing volume are divided into pieces and then printed separately. However, their work does not solve the problem of saving the printing material and is aimed at printing large objects. Material saving has been addressed by [WWY*13], where the object is converted into a truss skin-frame structure. Prévost *et al.* [PWLSH13] introduced an optimization solution to create 3D printable objects that can stand against gravity. Large data sets required for complex multi-material 3D printing were addressed by [VWRKM13] where a programmable pipeline is used to generate spatially varying 3D material properties. Chen *et al.* [CLD*13] propose a framework for transferring the user-defined model specifications into a material-specific representation, allowing the system to print multi-material objects with desired properties.

One of the key components of our approach is 3D model segmentation. An overview of segmenting techniques was presented by [Sha08]. Most of the techniques for 3D segmentation seek to segment the model in an intuitive way, for example, segment the model of a human body into meaningful parts like head, hands, legs and the rest. For example, the method of Katz *et al.* [KT03, KLT05] segments a mesh hierarchically using graph cuts with fuzzy segment boundaries which are refined to obtain smooth cuts, and this approach can also be used to extract the object skeleton. Another approach based on hierarchical segmentation was used by Attene *et al.* [AFS06], where the segmented clusters are approximated by basic primitives such as cubes, cones or spheres. Cohen-Steiner *et al.* [CSAD04] showed how a mesh can be segmented and a surface approximated according to shape properties. Robust segmentation algorithm was presented in [SSCO08] which introduced a shape-diameter function that measured the diameter of an object's volume for each surface point and segmentation is based on thresholding these diameter values. A segmentation can be done also with random cuts, as demonstrated by Golovinskiy and Funkhouser [GF08]. An advantage of random-based segmentation techniques is their speed and suitability for real-time applications, as shown by Lai *et al.* [LHMR08] who segmented the mesh with the use of random walks. Best segmentation results, comparable to the segmentation created by humans, are achieved with the use of machine-based learning. Such methods can also label created segments based on the collection of training meshes, as demonstrated by [KHS10]. The above-described methods provide very good results for segmentation into meaningful parts. However, segmentation for 3D print ing requires different criteria such as constraints on the size of the segments, or making the cross-sectional area suitable for future assembly, and these methods cannot be used directly.

Packing is an important problem related to our work and due to its nature it is a nondeterministic polynomial time-hard problem. Various heuristics have been used such as simulated annealing [Dow93], Monte Carlo optimization [HKSY25] and genetic algorithms [WC10]. Most of the previous work deals with the special case of packing orthogonal boxes such as [MPV00]. The unused space in bounding boxes makes this approach not suitable for our purposes as we are packing arbitrary shapes. A notable exception is [CSR10] where they define a mathematical framework based on phi-functions to pack arbitrary shapes. Levy *et al.* [LPRM02] introduced a different, height field–based approach and demonstrated how to use it to store and compress meshes as geometry images. This approach was later extended by Sander *et al.* [SWG*03] to generate texture atlases. Apart from computer graphic applications, packing is relevant in very large scale integration circuit design, where the objective is to minimize the chip volume [PF09].

1.2. Contributions

We introduce *PackMerger*, a novel algorithm that improves the efficiency of printing 3D objects by converting them into shells and breaking them into smaller parts. The shells are generated by hollowing out the volume of the object and are then divided into a set of small segments. The segments are merged and packed together according to a cost function that evaluates the efficiency with which the parts can be packed into a smaller volume. Further optimizations attempt to minimize the cross-sectional area of each pair of segments and the number of segments. The result is a small volume with tightly packed segments representing the input object. This packed configuration is printed while saving the printing material and decreasing the printing time.

An example in Figure 1 shows an object that was first printed as a single object on an Objet30 printer in 13.5 h using 351 g of support material. Our framework reduced the printing time to 9.5 h and the packing used 229 g of support material yielding a 30% time savings and a 35% material savings. The optimization took 1.5 min and the object was glued together in 15 min.

Our framework can be applied to printing a single object, printing multiple objects or printing objects that do not fit the printing area and would have to be printed in multiple passes using existing techniques [LBRM12].

2. System Overview

The input of our algorithm (Figure 2) is a 3D polygonal watertight mesh, as only meshes with a well-defined and closed volume can be 3D printed (non-closed meshes with overlaps would lead to ambiguities and failures). The output of our algorithm is a tightly packed set of meshes (segments) suitable for 3D printing.

During *pre-processing* the input is converted into a thin shell by hollowing. This step saves the printing material and allows for better packing than the original full volume. The boundary shell is then converted into a set of tetrahedral cells using tetrahedralization [Si10].

Figure 1: *A 3D object was automatically divided into a set of similarly sized segments that can be easily assembled together and printed. The overall printing time was reduced from 13.5 h to 9.5 h (saving 30%) and the amount of support material was reduced from 351 g to 229 g (saving 34%). The optimization was completed in approximately 1 min and the final object assembly took about 15 min.*

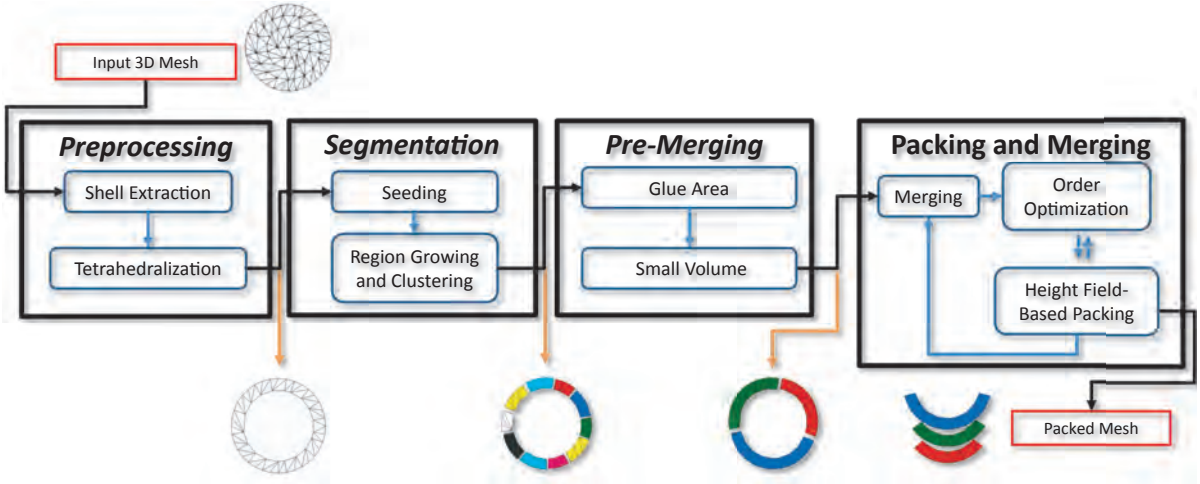

Figure 2: *System overview. The input 3D mesh is pre-processed and converted into a volumetric shell. The shell is divided into a set of segments that are pre-merged, if they would cause problems during final assembly (small volume or small cross-sectional area). The updated segments are then tightly packed and merged together, resulting in a printing configuration that saves supporting material and speeds up the printing.*

In the second step, the *segmentation* converts the shell into a large number of segments. It first chooses a certain number of tetrahedra to act as seeds that are then expanded into segments by clustering. In all examples in this paper we used a fixed number of 100 segments. However, the smallest size of a segment could be an individual tetrahedra.

The over-segmented mesh could be directly optimized but the large number of segments would be difficult to assemble after printing. Therefore, our framework connects some segments in the *pre-merging* step using several criteria. First, it is desirable to avoid small cross-sectional areas between segments that would be difficult to glue. Therefore, we connect neighbouring segments with a small cross-sectional area. Second, tiny volumes may be difficult to assemble, so we merge small segments to larger ones. After pre-segmentation, mesh boundaries are smoothed to improve the visual appearance and to simplify assembly.

The previous steps create segments that have similar volumes and can be packed in the *packing and merging* stage. The optimization uses a height field-based packing. As the order in which the segments are accommodated affects the resulting packing, the packing is tightly coupled with the tabu search heuristic [CPT09] that optimizes it. Also, during the packing, further merging occurs that reduces the number of segments. The packing and merging phases attempt to minimize by optimization of the unused space, total bounding volume and the number of segments. The output is a set of packed segments that are ready for printing.

Our framework can be used to directly pack a set of objects without segmentation and multiple objects can be segmented and packed at once.

3. Pre-Processing

The input watertight, two-manifold boundary mesh is converted into a thin inner shell using uniform mesh re-sampling. The re-sampling builds a uniform volumetric voxel representation of the inner volume, with each voxel containing its signed distance from the original surface.

The width of the shell was set to a fixed value of 3 mm in all our experiments. This value provided stable structures for all objects in

this paper. If better structural stability would be needed, the width could be increased, truss shells could be inserted [WWY*13] or the objects could be made structurally sound, for example by using the method of [SVB*12].

The inner surface of the shell is reconstructed using the Marching Cubes algorithm. Although this does not provide a smooth mesh and better techniques exist, the inner shell is invisible after the object is assembled, so the quality of the inner shell surface is not a concern. From the surface shell, a tetrahedral volumetric mesh is generated using constrained Delaunay tetrahedralization [Si10].

4. Segmentation

The input of the segmentation is the object O that is represented as a volumetric, tetrahedral cell-complex. The output is a set of segments $S = \{S_1, S_2, \ldots, S_n\}$, where $n = |S|$ is the number of segments. The set S exactly covers O. The volume of O is denoted by V_o and its bounding box volume by V_{bo}. The object's volume is calculated as the sum of volumes of all tetrahedra and it remains constant during processing.

The objective of the segmentation is to generate segments S of approximately equal volume that is much smaller than the overall available printing volume. Moreover, the neighbouring segments should have cross-sectional areas larger than some predefined minimal value. Both criteria simplify the final object assembly. The segmentation consists of two steps: seeding and clustering. We use parallel k-means clustering of [Sha08] on the tetrahedral mesh.

The objective of seeding is to find cluster centroids distributed equally along the tetrahedral structure. Seed locations are found in an iterative manner by locating one seed at a time. The seeding first randomly places a single seed and grows it until the desired volume is reached (1% of the total object volume V_o in our examples). Seeds are then placed sequentially into random, unassigned tetrahedra and they grow into the maximum volume as well. The process stops when there is not enough available space to grow any other seed. The results of seeding are locations of cluster centroids c_i that cover the input set of tetrahedra (see Figure 3a).

The next step is clustering, attempting to segment the mesh into equally sized connected parts. The clustering takes the set of cluster centroids c_i as the initial seeds. The output of the clustering is a set of IDs for each tetrahedron assigning them to a segment S_i. We use the Euclidean norm to measure distances in the dual graph between connected vertices. A parallel growth process is performed, expanding each cluster by adding adjoining unassigned tetrahedra.

Our algorithm ensures that all clusters form a single connected component throughout the growth process as a tetrahedron can be added to the cluster only if there is a path from the seed through the cluster to the tetrahedron. Growth stops when all tetrahedra have been assigned a cluster.

All clusters are then re-grown again to reflect changes in the centroids that serve as new seeds. The clustering ends when centroids stop changing their location significantly (we use $\epsilon = 10^{-3}$ as the minimum distance between c_i in two consecutive iterations) or a maximum number of iterations is reached (50 iterations in our implementation). In most cases, the cluster centres stop changing their position before 10 iterations were reached. The result of clustering is the set of segments S covering O (see Figure 3b).

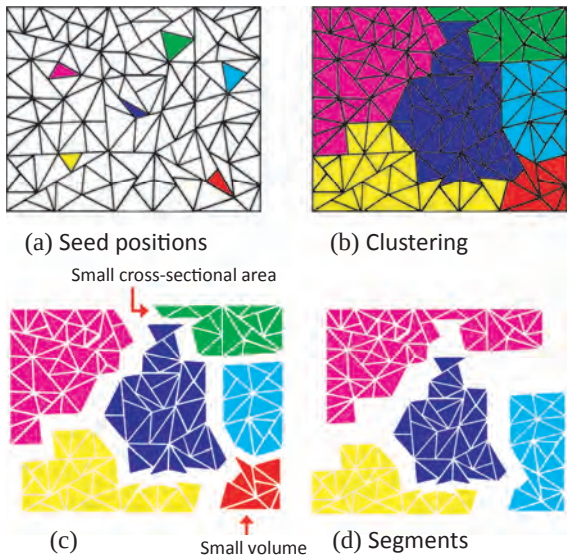

Figure 3: *Segmentation first finds suitable seed positions (a) that are used to cluster the tetrahedra to near-to-equal segments (b). In the pre-merging step small segments are merged (c) resulting in a set of segments covering the input object (d).*

Speed of the segmentation is a linear function of the model complexity represented by the number of tetrahedra and it takes seconds for a medium-complexity model (100k tetrahedra). Segmentation runs only once and the created segments are used throughout the rest of the optimization pipeline.

5. Pre-Merging

The segmentation creates a set of segments with similar volume that can be merged later. We over-segment the object to allow more aggressive merging in the next steps. The segmentation may produce segments unsuitable for the final assembly of the printed object such as segments connected with a small cross-sectional area that are difficult to glue or small segments (see Figure 3c). Pre-merging addresses these problems by combining segments based on two heuristics: the cross-sectional area should allow good gluing and segment volume should be larger than a minimum value. We also impose the restriction that parts cannot be merged if their bounding box volume would exceed the maximum printing volume.

The segment connectivity is stored in a weighted graph $G(V, E)$, with vertices $v_i \in V$ representing segments S_i and edges $e_{ij} \in E$ representing two neighbouring segments v_i and v_j sharing a common boundary. A weight $a_{ij} : E \to \mathbb{R}$ is assigned to each edge e_{ij}, where a_{ij} is the cross-sectional area of the segments v_i and v_j. The graph G is represented by its adjacency matrix M. Merging two segments v_i and v_j corresponds to contraction of the edge e_{ij}.

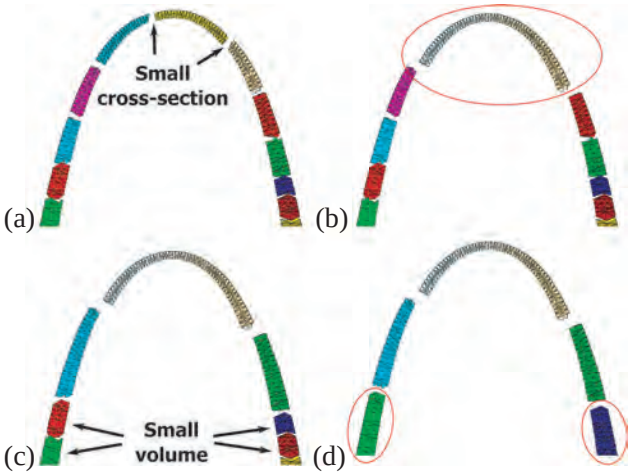

Figure 4: *Pre-merging connects segments that would be difficult to assemble such as tiny cross-sections and small volumes: (a) shows the initial segments; (b–d) show segments merged while increasing the smallest segment volume and cross-sectional area.*

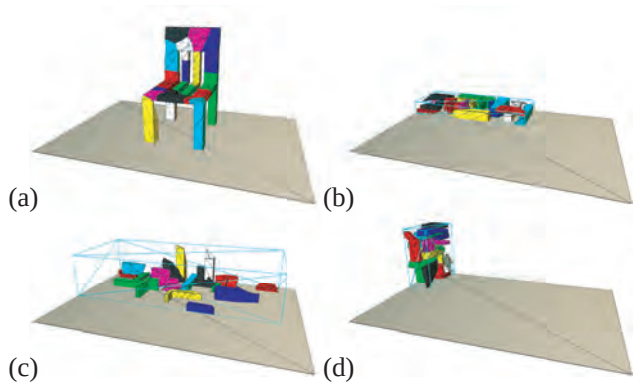

Figure 5: *The effect of the parameter w from Equation (1) on the configuration. (a) Initial segments, (b) $w = 0.5$, (c) $w = 0$ (minimal support) and (d) $w = 1$ (minimal volume).*

'Cross-sectional area merging' assures better segment gluing by maximizing a_{ij}. The pre-merging iteratively finds the smallest a_{ij} and the corresponding segments are merged. This process continues until the smallest a_{ij} is above the critical threshold (10 mm^2 in our experiments).

'Volume merging' connects segments with small volumes together. We select the segment with the smallest volume and merge it with its neighbour which has also the smallest volume. This phase stops when the smallest segment has its volume above the minimum limit (5% of V_o in our implementation).

The previous steps can generate jagged-edge boundaries between clusters. This is an unwanted effect both visually and structurally, as such boundaries negatively affect the appearance of the segmented object and their uneven surface can lead to difficulties during gluing. We smooth each segment's boundary (inner and outer) by using Laplacian smoothing [HDZ05]. Each vertex on the boundary is averaged with its neighbours resulting in the smoother boundary with better adhesion properties. Figure 4 shows an example of pre-merging.

6. Packing and Merging

The goal of packing is to arrange segments in S tightly together into a packed configuration P with an optional additional merging. The volume of P is equal to V_o (the optimization does not change the volume of the object) and the packed configuration bounding volume is denoted by V_{bp}.

The support volume V_s of an object is the additional volume necessary to print objects with overhangs. It is defined as the volume under all facets of the object with normals pointing down, that is, if the angle α_s between the horizontal plane and the normal lies in the range $[0^\circ, -180^\circ]$. The angle α_s depends on the actual type of the printer. For example, printers using 'Fused Deposition Modelling' have this angle more relaxed and are able to print overhangs up to $\alpha_s = 45^\circ$ while stereo-lithography–based printers insert support under whole faces with $\alpha_s = 0^\circ$. Although the supports do not have to fully cover the facet due to printer-specific optimization, we account for the full volume in our optimization in order to provide comparative results.

The objective of packing and merging (see Figure 2) is to minimize the bounding box volume V_{bp}, the support volume V_s and the number of segments n in packed configuration P. The optimization assigns to each segment S_i a translation, represented by the coordinates of the centre of mass $(x_{c_i}, y_{c_i}, z_{c_i})$, and a rotation, represented by corresponding Euler angles $(\alpha_{c_i}, \beta_{c_i}, \gamma_{c_i})$. Our system attempts to minimize the following cost function:

$$f(P) = w\ V_{bp} + (1 - w)\ V_s, \tag{1}$$

where V_{bp} is the volume of the axis-aligned bounding box of the packing configuration, V_s is the total support volume and $w \in [0, 1]$ is the weight between the bounding box volume and support volume optimization.

Note that neither the cross-sectional area nor number of parts is included in Equation (1). The reason is that the cross-sectional area has already been optimized in the pre-merging phase, where the goal was to connect parts with area too small to be glued together. The additional merging attempts to optimize the number of parts using the order optimization and the height field–based packing as independent blocks.

The user-defined w helps to optimize packing for specific types of printers. Figure 5 shows different objectives for the packing optimization. Figure 5(a) shows the initial segment position, (b) the packing optimizes V_{bp} and V_s with equal priority $w = 0.5$, (c) packing only optimizes the support volume $w = 0$ and (d) packing optimizes only the bounding volume $w = 1$. In our experiments, we use $w = 0.75$ as it was giving good results in minimizing both V_{bp} and V_s.

The optimization consists of three modules (see Figure 2):

(1) *Merging* finds segments to merge and performs the operation. Input is the list of segments S^k and output is a new list S^{k+1} where segments $S_i, S_j \in S^k$ were merged into $S_r \in S^{k+1}$.
(2) *Order optimization* sorts the list of segments $L : S \to \mathbb{N}$. Since the next stage packs one segment at a time, different permutations of S significantly affect the final result. The input is the list S and the output is the ordered list $L(S)$.
(3) *Height field–based packing* takes segments from $L(S)$ one by one and calculates their position and orientation minimizing the function (1). Input is the ordered list L and outputs are the translations and rotations of each segment.

The order optimization uses the height field–based packing as a subroutine to search and test for the optimal order of the list (see Figure 2). After the optimal packing is found, new merging is done and the process is repeated. This way each list of segments S^i has an optimal packing associated with it. The final result is the best packing.

Several bin-pack approaches exist, but they either do not work with arbitrary shapes in 3D space, or are prohibitively slow. Although there have been mathematical models for how to describe general shapes by [CSR10], to the best of our knowledge, no implementation of such packing in 3D space has been developed yet. Therefore, we decided to use a simpler approach with modified height field–based packing with order optimization that can be adjusted to better suit our needs and allows for fast GPU-oriented implementation.

6.1. Height field–based packing

At each step of the optimization, a packing configuration P and its cost Equation (1) are calculated. Various packing strategies exist (see Section 1.1) and in our work we have extended the *height field–based packing* (a.k.a 'the Tetris packing') of [SWG*03] for packing 3D texture atlases.

The packing starts with an ordered list of segments $L(S)$ and attempts to pack each $S_i \in L$ one by one in the given order. The height field itself is defined as a discrete 2D array that stores the upper horizon of all contained objects defining the distance from the printing tray (see Figure 6a). In order to find the optimal position $T(S_i)$ of the segment S_i, we iterate by moving S_i by the fixed step defined by the grid resolution (1–10 mm in our experiments) through the 2D height field and aligning the bottom of the segment with the top of the height field (e.g. dropping the part on the top of the height field). When the object location is found, the amount of wasted space (support volume) is evaluated as the difference between the height field and the lower part of S_i (Figure 6b). The part position with minimal wasted space (or minimal bounding volume, depending on parameter w) is selected as the best and the part is added to the height field.

We do not count the volume of the empty space (if the object is hollow) towards the wasted space; we assume that segmentation splits the object into parts without holes or highly concave areas. If in any case the part would be hollow, it will have no influence on the object position in the height field as the volume of the hole does not add to the wasted space.

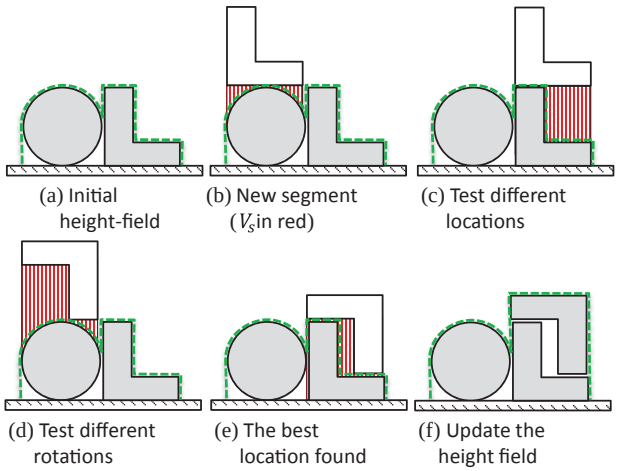

Figure 6: *Overview of steps in height field–based packing. The main goal is to find the position of the incoming part that minimizes the wasted space V_S (in red).*

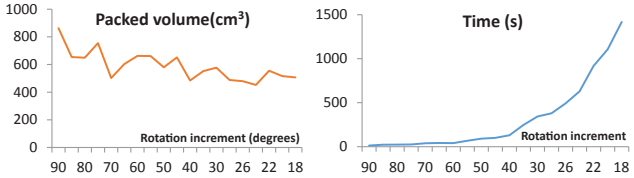

Figure 7: *A smaller rotation angle leads to better packing, but the computation time increases significantly.*

We also rotate the segment in each location in order to improve packing and minimize the wasted space in the packed configuration. The segment is rotated around all three main axes in fixed increments. Figure 7 shows the effect of the rotation angle on the packing and the calculation time. Lowering the angle improves packing quality. Unfortunately the complexity of packing is exponential. We found a good value for the rotation angle to be around 30°. This provides a good compromise between the calculation time and packing quality.

To avoid very tight packing, we insert a small gap between segments so that the configuration can be printed and the support material removed. We use a gap of 1 mm. Although the thickness of the glue required to connect two parts could be also accounted in this gap. However, in practice the glue thickness is much smaller than the printer resolution and is thus considered negligible.

6.2. Order optimization

The result of the previous step is a packing configuration P of S for the given order of segments. However, the order of segments can lead to significantly different packing configurations with different costs.

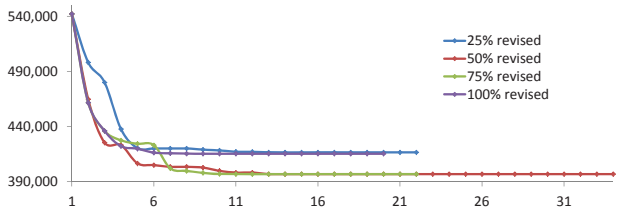

Figure 8: *The convergence of the tabu search for different percentage of revisions. The x-axis shows the number of iterations and the y-axis the cost function.*

Having a list $L(S)$, there are $n!$ possibilities for sorting it. We use the *tabu search* optimization [KS98] that has polynomial complexity to find a near to optimal solution and that has been used with success for packing problems [CPT09].

Each step of the tabu search is defined by a permutation L_i and it has associated cost Equation (1) that is the result of the height field–based optimization. The neighbourhood $N(L_i)$ is defined as all possible swaps of the elements of L_i and a *move* is another permutation $L_{i+1} \in N(L_i)$.

Once the *move* is made it is placed into a list of forbidden moves called the *tabu list*, for a certain number of iterations m (called *tabu memory*). This helps the optimization to escape from local minima. In our experiments we fix $m = 3$.

At each step the optimization keeps track of the best permutation L_i that has been found so far. The search starts with random permutation L_0 and searches $N(L_0)$ for a move that minimizes the function Equation (1) and that is not in the tabu list. Then it moves to that permutation L_{i+1} and updates the tabu list. If necessary, it also updates the global minimum L. The optimization ends if L has not been updated for a certain number of iterations (10 in our implementation). Figure 8 shows the convergence of the tabu search in our application. The x-axis shows the number of iterations and the y-axis shows the cost function.

The size of neighbourhood $N(L)$ grows rapidly with the number of segments ($|N(L)| = O(n^2)$). Instead of exploring all moves, we use random sampling and we review only a fraction of the neighbourhood. This saves calculation time and permits certain variations that improve the search further. The effect of this selection is shown in Figure 8. We see that a random sample performs better than exhaustive search because not taking the greedy choice gives the algorithm an additional way to escape from local minima.

6.3. Additional merging

For easier assembly, it is desirable to finish with as few parts as possible while still obtaining the lowest volume possible. Based on this observation, we insert additional merging into the optimization pipeline. This merging is implicit in the objective function Equation (1) since merging two segments tends to decrease the support volume V_s and the bounding volume V_{bp}. The reason for this is that even though it could happen that the packing would put the segments next to each other (a rare case for arbitrary shapes), there would still be a gap between them. Merged segments also reduce the gluing area, contributing to simpler assembly. The additional merging is part of the packing optimization and therefore differs from the pre-merging step described in Section 5.

Every time a merge is performed, the optimization loop between order optimization and height field–based packing is repeated (see Figure 2). Therefore we select a random edge e_{ij} from the adjacency graph. We use the weight between the bounding box and the support volume optimization $w(e)$ as a merging strategy (see Figure 5). This strategy is repeated until further merging is impossible, for example, until we have merged all parts into a single one or the merged parts exceed the available printing volume. After exploring different merged configurations we select the one with minimal volume. Typically, we end up with no more than 10 segments after this step.

7. Implementation and Results

The system is implemented in C++ and uses OpenGL for visualization. All tests were run on an Intel i7 CPU clocked at 3.2 GHz with 16 GB of memory. The object's shell is generated as a uniform offset from the original mesh using MeshLab [mes13] and the Tetgen library [Si13] generates a tetrahedral representation of the shell. All original input objects are considered as solid.

The height field–based packing implementation makes extensive use of the GPU. The height field is obtained by drawing the packing configuration in the top-down orthogonal projection and height field values are read from the z-buffer. The z-buffer is also used when the position and rotation of the new segment is examined, by rendering the back side of the object into a separate z-buffer. The support volume is obtained by subtracting these two buffers (Figure 6b). To reduce memory transfers between the CPU and GPU, an array of copies of each segment is created. Segments are placed next to each other and the support volume calculations are performed on each chunk of z-buffer during each copy step, so that multiple configurations are examined in parallel. The GPU solution also makes the packing speed invariant to the model complexity. Although the total processing time is linear with the number of segments, it is influenced by the bounding volume of the packed configuration—a larger volume means more traversing over the height field and longer calculations.

7.1. Printed examples

We used three 3D printers representing different printing technologies: stereo-lithography with photo-polymers (Stratasys Objet30), fused deposition modelling (FDM) with PLA plastic (Makerbot Replicator) and FDM with polypropylene-like material and soluble support (Stratasys uPrint). Table 1 summarizes printing times and used support material for examples in the paper.

Table 1 also contains the time needed to run the optimization. Computational time depends on the number of segments, but in most cases optimization took less than 1 min. The most time-consuming part (80%) is the order optimization, as it runs the height-based

Table 1: *Printing times and used support material savings for our experiments. Original models are solid before optimization.*

Objects	Printer	Print time (orig.)	Print time (packed)	$\mathbf{V}_s$ (orig.)	$\mathbf{V}_s$ (packed)	Time saved	$\mathbf{V}_s$ saved	Opt. time	# of parts
Arch (Figure 9)	uPrint	Did not fit	14 h 18 min	Did not fit	60.6 cm^3	N/A	N/A	124 s	11
Bunny (Figure 1)	Objet30	13 h 43 min	9 h 30 min	351 g	229 g	4 h 13 min (31%)	122 g (35%)	97 s	13
Molecule (Figure 11)	Objet30	15 h 55 min	12 h 03 min	264 g	160 g	3 h 52 min (24%)	156 g (39%)	56 s	12
Sphere	Objet30	6 h 23 min	4 h 13 min	110 g	39 g	2 h 10 min (34%)	71 g (65%)	28 s	6
(Figure 10)	Replicator	2 h 43 min	2 h 18 min	17 g	11 g	0 h 25 min (18%)	6 g (35%)		
Table (Figure 12)	Replicator	1 h 05 min	1 h 02 min	16 g	16 g	0 h 03 min (5%)	0 g (0%)	14 s	4
Group (Figure 13)	Replicator	6 h 18 min	6 h 14 min	53 g	52 g	0 h 04 min (1%)	1 g (2%)	41 s	5

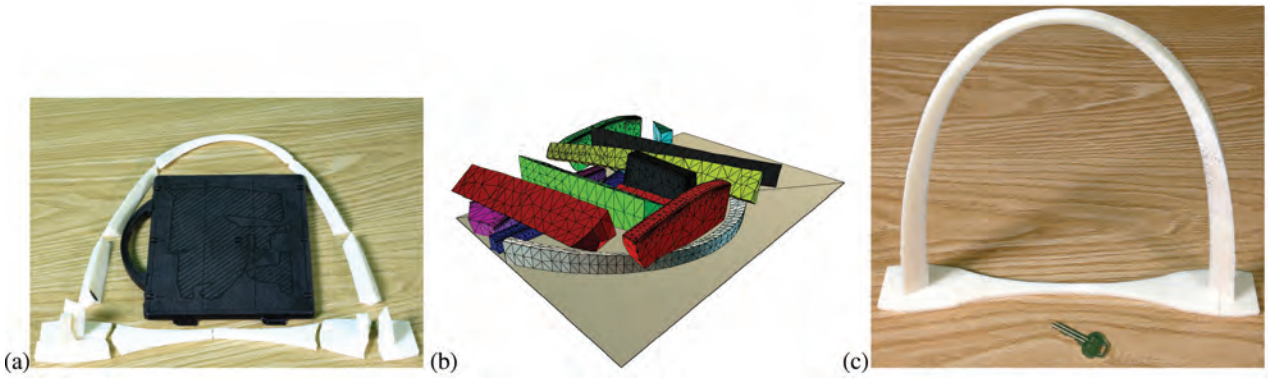

Figure 9: *An object larger than the printing tray (shown in the figure) (a) was automatically segmented into pieces that fit the tray (b) and it was printed in a single print (b). It was then assembled in less than 5 min (c).*

packing multiple times. The height field–based packing takes less than 1 s on the GPU.

The time required to remove the support material is important as well. With FDM printers, support structures use the same primary material, and must be removed by breaking them out of the model which can result in damage to the model. Precise removal can take tens of minutes, especially with large objects. Stereolithography printers use different support material that is weaker, making removal of it much easier—a strong stream of water effectively removes all support material within minutes and keeps surfaces relatively intact. A soluble support material leaves the surface of the original object perfectly clean but dissolving it takes hours.

An object larger than the printing tray is shown in Figure 9(a). It has been automatically divided into 11 segments that were packed in the printing tray as shown in Figure 9(b). The overall printing time was 14 h on the uPrint printer and it used 60 cm^3 of support material and 167 cm^3 of the primary material. The object was assembled in less than 5 min. As the object could not be printed in one piece we cannot report the estimated time or compare it to the un-optimized object.

'The Stanford bunny' in Figure 1 was automatically segmented into 13 pieces. The overall printing time was reduced from 13.5 h to 9.5 resulting in a 30% saving and 35% of the support material was saved by using 229 g instead of 351 g. The final object was glued together in 15 min.

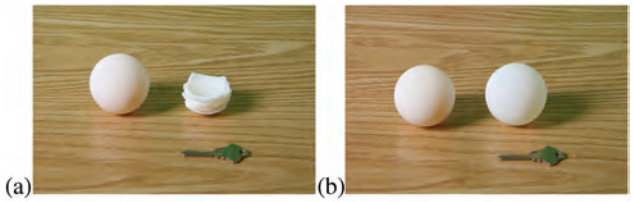

Figure 10: *(a) Tight packing was achieved for a sphere saving 65% of the support material on the Objet30 printer; (b) shows the solid and the assembled sphere.*

'The Sphere' in Figure 10 is an example of the best case for our algorithm. The sphere was segmented into six parts and the printing time on the Objet30 printer was reduced by 34% from 6.5 h to 4 h. Material savings were even more significant as 65% of the support material was saved by using only 39 g instead of 110 g. The time savings were 18% for the FDM printer (see Table 1).

'A molecular' model was first segmented with no cross-sectional area in mind as shown in Figure 11(a). The final model was printed and savings of time and material were substantial (60%). However, it was impossible to assemble it because of the very small cross-sectional area for gluing. Then the cross-sectional area was considered during pre-merging. The model was segmented again, the new model was printed in Figure 11(b), and assembled as shown in Figure 11(c). Figure 11(d) shows the packing configuration for

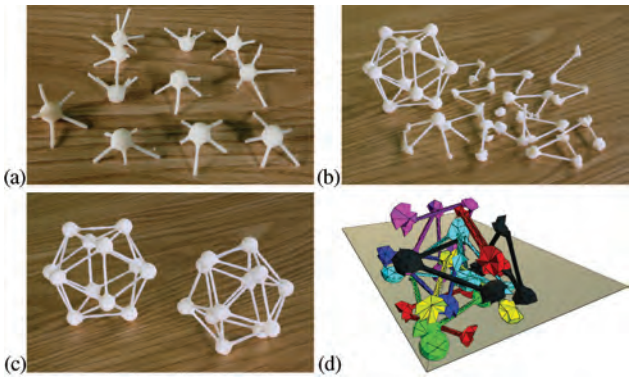

Figure 11: *The model was first segmented without considering the cross-sectional area resulting in a model that was impossible to assemble (a). The model was segmented again with the gluing area considered, optimized, printed (b) and assembled (c); (d) shows the printing tray with the packing configuration (almost 40% savings of material).*

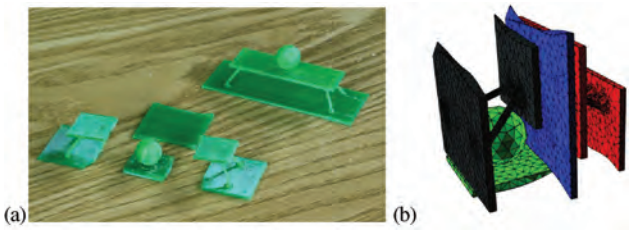

Figure 12: *Example of printing on an FDM printer. Packed configuration (b) and the printed object (a).*

the final model. The printing time for the complete model was almost 16 h and for the packed model it was 12 h, resulting in a 24% saving. The original model required 264 g of support material that was reduced to 160 g.

The table in Figure 12 was printed on an FDM printer (Makerbot Replicator). We observed that with FDM printers segmenting and packing does not always lead to material and time savings. A large number of segments is not desirable as those printers spend more time printing the object boundary than the inner space. Also supports on such printers behave differently as large overhang angles can be tolerated that can make estimation of the support costs inaccurate. In this case, printing of the packed group took only a few minutes less than the original object.

Multiple object packing shown in Figure 13 demonstrates that the packing stage can be separated from the rest of the pipeline and can be used to pack arbitrary groups of objects. A smaller cup was automatically packed inside the big cup and the two crystals were inserted inside the smaller cup that led to the minimum bounding volume. Separating and cleaning the cups after printing took less than 5 min.

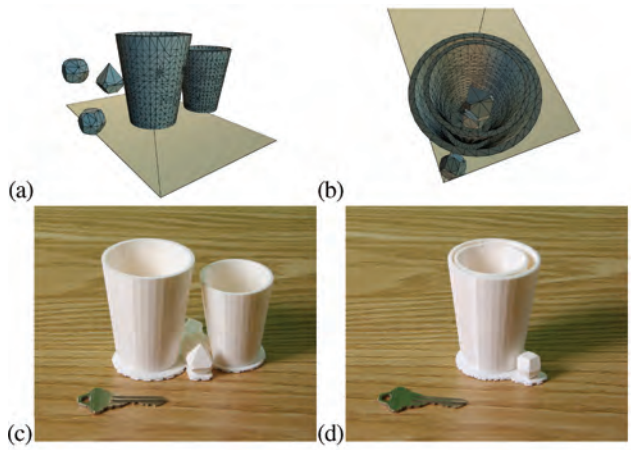

Figure 13: *A sample packing of groups of objects: (a) shows the initial, random placement of objects, (b) packed configuration and (c,d) printed results. Note that the two crystals have been placed inside the smaller cup and the smaller cup inside the larger cup.*

Results show that our system is not equally suited for all printer types. The best results were achieved with the Objet30 printer that uses a lot of support to print objects and printing time is influenced by the total object volume only. On the other hand, the worst results were achieved on FDM-based printers such as the Replicator. This can be explained by the printing mechanism of FDM printers where the printer spends much more time printing the object surface than the interior structure (which is a some kind of regular fill). With an increasing number of parts, a larger surface area is printed that results in slower printing.

The previous work of [WWY*13] also saves material and our method might not achieve such large savings (70%). However, there are important differences between both approaches. Our method optimizes both the support and the printing volume *without changing the object structure* (the only exception is conversion into the shell), whereas the work of [WWY*13] aims to reduce the primary material support with large transformations of the initial object into a skin-frame structure, without any optimization of the printing time and the support material.

8. Conclusions

We have presented PackMerger, a framework that improves the efficiency of printing 3D objects by breaking them into smaller parts and packing them tightly. Our algorithm converts the input object into shells that are divided into a set of small segments. The segments are merged and packed together. The result of our algorithm is a small volume with tightly packed objects that saves the printing material and decreases the printing time. Comparison of our approach to the existing packing algorithms is problematic. Most 3D packing algorithms work only with restricted shapes or, like [CSR10], improve an initial configuration. Our algorithm starts with a random configuration and uses packing together with the part order optimization.

Our framework can be applied to printing a single object, printing multiple objects or to printing objects that do not fit the printing area and would need to be printed in multiple passes. We show that, depending on the object and printer type, the printing time can be reduced by up to 30% while simultaneously saving up to 60% of support material. The objects require manual gluing after they are printed. We have tested our framework on three different printers. Although some printers do not use support material, the speed of printing was higher for most of the examples in our paper.

The obvious limitation of our method is the need for manual assembly and that users might prefer to print the object as whole even though the cost will be higher. Possible assembly problems could occur if the method would be applied to CAD parts with intricate internal structure since our method is optimized for the thin shells extracted from solid objects. Also, if the model has a very complex geometry with thin prevalent features (tree, chain, cage, etc.), the segmentation and assembly would be problematic due to low cross-sectional areas of the connecting parts. Another one is its limited use for FDM printers that already print only the thin shell with a less dense structure inside the object, yet even for those printers we can use the method to save support material and to split the object in order to print it at a scale that does not fit the print volume.

There are many possible avenues for future work. We would like to test our method with alternative state-of-the-art packing algorithms to see if any additional efficiency can be gained. Our segmentation is not taking into account visibility of the seams, and for better visual quality we could make cuts along natural seams on the model or use other, more advanced approaches for high-quality segmentation reviewed in Section 1.1. Although we have not found that random merging of pieces is a limitation in our work, an even tighter coupling between merging and packing could be explored. Finally, we would like to automatically add connectors between pieces like in the work [LBRM12] along with automatically generated assembly instructions that can ease the model assembly process.

References

[AFS06] Attene M., Falcidieno B., Spagnuolo M.: Hierarchical mesh segmentation based on fitting primitives. *Vision Computing 22*, 3 (Mar. 2006), 181–193.

[BBJP12] Bächer M., Bickel B., James D. L., Pfister H.: Fabricating articulated characters from skinned meshes. *ACM Transactions on Graphics 31*, 4 (July 2012), 47:1–47:9.

[BBO*10] Bickel B., Bächer M., Otaduy M. A., Lee H. R., Pfister H., Gross M., Matusik W.: Design and fabrication of materials with desired deformation behavior. *ACM Transactions on Graphics 29*, 3 (2010), 63:1–63:10.

[CCA*12] Calì J., Calian D. A., Amati C., Kleinberger R., Steed A., Kautz J., Weyrich T.: 3D-printing of non-assembly, articulated models. *ACM Transactions on Graphics 31*, 6 (Nov. 2012), 130:1–130:8.

[CLD*13] Chen D., Levin D. I. W., Didyk P., Sitthi-Amorn P., Matusik W.: Spec2Fab: A reducer-tuner model for translating specifications to 3D prints. *ACM Transactions on Graphics 32*, 4 (2013), 135:1–135:10.

[CPT09] Crainic T. G., Perboli G., Tadei R.: Ts2pack: A two-level tabu search for the three-dimensional bin packing problem. *European Journal of Operational Research 195*, 3 (2009), 744–760.

[CSAD04] Cohen-Steiner D., Alliez P., Desbrun M.: Variational shape approximation. In *ACM SIGGRAPH 2004 Papers* (New York, NY, USA, 2004), SIGGRAPH '04, ACM, pp. 905–914.

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

[Dow93] Dowsland K. A.: Some experiments with simulated annealing techniques for packing problems. *European Journal of Operational Research 68*, 3 (1993), 389–399.

[GF08] Golovinskiy A., Funkhouser T.: Randomized cuts for 3D mesh analysis. *ACM Transactions on Graphics 27*, 5 (Dec. 2008), 145:1–145:12.

[HBA13] Hildebrand K., Bickel B., Alexa M.: Orthogonal slicing for additive manufacturing. *Computers & Graphics 37*, 6 (2013), 669–675.

[HDZ05] Hansen G., Douglass R., Zardecki A.: *Mesh Enhancement: Selected Elliptic Methods, Foundations and Applications*. World Scientific Publishing Company, London, 2005.

[HKSY25] Hiroyuki Y., Keishi S., Shigetoshi N., Yoji K.: The 3D-packing by meta data structure and packing heuristics. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sci. 83*, 4 (2000), 639–645.

[KHS10] Kalogerakis E., Hertzmann A., Singh K.: Learning 3D mesh segmentation and labeling. *ACM Transactions on Graphics 29*, 4 (July 2010), 102:1–102:12.

[KLT05] Katz S., Leifman G., Tal A.: Mesh segmentation using feature point and core extraction. *The Visual Computer 21*, 8-10 (2005), 649–658.

[KS98] Kreher D. L., Stinson D. R.: *Combinatorial Algorithms: Generation, Enumeration, and Search*. CRC Press, Florida, 1998.

[KT03] Katz S., Tal A.: Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Transactions on Graphics 22*, 3 (July 2003), 954–961.

[LBRM12] Luo L., Baran I., Rusinkiewicz S., Matusik W.: Chopper: Partitioning models into 3D-printable parts. *ACM Transactions on Graphics 31*, 6 (Nov. 2012), 129:1–129:9.

[LHMR08] Lai Y.-K., Hu S.-M., Martin R. R., Rosin P. L.: Fast mesh segmentation using random walks. In *ACM Symposium on Solid and Physical Modeling* (New York, NY, USA, 2008), pp. 183–191.

[LPRM02] LÉVY B., PETITJEAN S., RAY N., MAILLOT J.: Least squares conformal maps for automatic texture atlas generation. *ACM Transactions of Graphics 21*, 3 (July 2002), 362–371.

[mes13] MESHLAB: Retrieved from http://meshlab.sourceforge.net/, May 2013.

[MPV00] MARTELLO S., PISINGER D., VIGO D.: The three-dimensional bin packing problem. *Operations Research 48*, 2 (2000), 256–267.

[PF09] PAVLIDIS V. F., FRIEDMAN E. G.: *Three-Dimensional Integrated Circuit Design*. Morgan Kaufmann Publishers Inc., Massachusetts, 2009.

[PWLSH13] PRÉVOST R., WHITING E., LEFEBVRE S., Sorkine-Hornung O.: Make it stand: Balancing shapes for 3D fabrication. *ACM Transactions on Graphics 323*, 4 (2013), 81:1–81:10.

[Sha08] SHAMIR A.: A survey on mesh segmentation techniques. *Computer Graphics Forum 27*, 6 (2008), 1539–1556.

[Si10] SI H.: Constrained Delaunay tetrahedral mesh generation and refinement. *Finite Elements in Analysis and Design 46* (2010), 33–46.

[Si13] SI H.: Tetgen: A quality tetrahedral mesh generator. Retrieved from http://tetgen.berlios.de/, May 2013.

[SSCO08] SHAPIRA L., SHAMIR A., COHEN-OR D.: Consistent mesh partitioning and skeletonisation using the shape diameter function. *Vision Computing 24*, 4 (Mar. 2008), 249–259.

[SVB*12] STAVA O., VANEK J., BENES B., CARR N., MĚCH R.: Stress relief: Improving structural strength of 3D printable objects. *ACM Transactions on Graphics 31*, 4 (July 2012), 48:1–48:11.

[SWG*03] SANDER P. V., WOOD Z. J., GORTLER S. J., SNYDER J., HOPPE H.: Multi-chart geometry images. In *Symposium on Geometry Processing* (Aachen, Germany, 2003), pp. 146–155.

[TJ11] TELEA A., JALBA A.: Voxel-based assessment of printability of 3D shapes. In *International Conference on Mathematical Morphology and Its Applications to Image and Signal Processing* (Verbania-Intra, 2011), pp. 393–404.

[VWRKM13] VIDIMČE K., WANG S.-P., RAGAN-KELLEY J., MATUSIK W.: Openfab: A programmable pipeline for multi-material fabrication. *ACM Transactions on Graphics 32* (July 2013), 136:1–136:12.

[WC10] WANG H., CHEN Y.: A hybrid genetic algorithm for 3D bin packing problems. In *BIC-TA'10* IEEE, Changsha (2010), pp. 703–707.

[WWY*13] WANG W., WANG T. Y., YANG Z., LIU L., TONG X., TONG W., DENG J., CHEN F., LIU X.: Cost-effective printing of 3D objects with skin-frame structures. *ACM Transactions on Graphics 32*, 5 (2013), 177:1–177:10.

[ZPZ13] ZHOU Q., PANETTA J., ZORIN D.: Worst-case structural analysis. *ACM Transactions on Graphics 32*, 4 (July 2013), 137:1–137:12.

Supporting Information

Additional Supporting Information may be found in the online version of this article at the publisher's web site:

Video S1