Box Cutter: Atlas Refinement for Efficient Packing via Void Elimination

MAX LIMPER, Fraunhofer IGD / TU Darmstadt, Germany
NICHOLAS VINING, University of British Columbia, Canada
ALLA SHEFFER, University of British Columbia, Canada

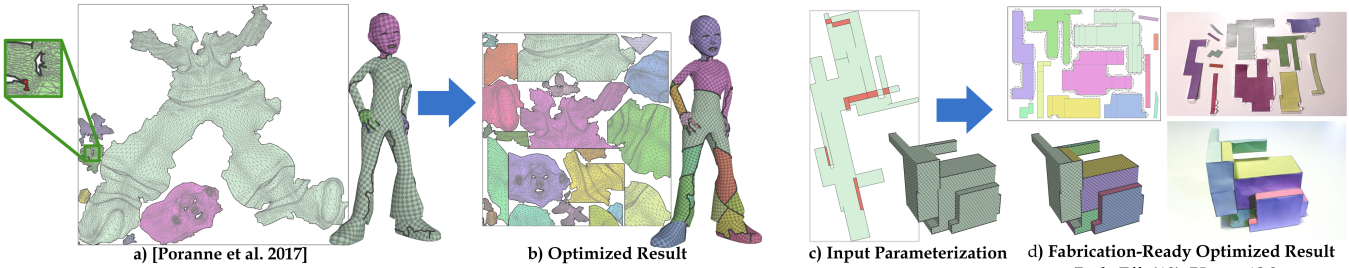


Fig. 1. Traditional texture atlas generation frameworks, such as [Poranne et al. 2017] (a), produce results which can have low packing efficiency and are not necessarily bijective (see inset). We produce an overlap-free atlas with the same parametric distortion and significantly higher packing efficiency by strategically segmenting and repacking the original charts directly in 2D space (b). We use the same framework to efficiently pack 2D patterns for 3D fabrication (c-e).

Packed atlases, consisting of 2D parameterized charts, are ubiquitously used to store surface signals such as texture or normals. Tight packing is similarly used to arrange and cut-out 2D panels for fabrication from sheet materials. *Packing efficiency*, or the ratio between the areas of the packed atlas and its bounding box, significantly impacts downstream applications.

We propose *Box Cutter*, a new method for optimizing packing efficiency suitable for both settings. Our algorithm improves packing efficiency without changing distortion by strategically cutting and repacking the atlas charts or panels. It preserves the local mapping between the 3D surface and the atlas charts and retains global mapping continuity across the newly formed cuts. We balance packing efficiency improvement against increase in chart boundary length and enable users to directly control the acceptable amount of boundary elongation. While the problem we address is NP-hard, we provide an effective practical solution by iteratively detecting large rectangular empty spaces, or *void boxes*, in the current atlas packing and eliminating them by first refining the atlas using strategically placed axis-aligned cuts and then repacking the refined charts. We repeat this process until no further improvement is possible, or until the desired balance between packing improvement and boundary elongation is achieved. Packed chart atlases are only useful for the applications we address if their charts are overlap-free; yet many popular parameterization methods, used as-is, produce atlases with global overlaps. Our pre-processing step eliminates all input overlaps while explicitly minimizing the boundary length of the resulting overlap-free charts. We demonstrate our combined strategy on a large range of input atlases produced by diverse parameterization methods, as well as on multiple sets of 2D fabrication panels. Our framework dramatically improves the output packing efficiency on all inputs; for instance with boundary length increase capped at 50% we improve packing efficiency by 68% on average.

CCS Concepts: • **Computing methodologies** → *Texturing*; *Mesh geometry models*; *Parametric curve and surface models*;

Additional Key Words and Phrases: Texture Atlas, 2D Patterns, Packing, Packing Efficiency

ACM Reference Format:
Max Limper, Nicholas Vining, and Alla Sheffer. 2018. Box Cutter: Atlas Refinement for Efficient Packing via Void Elimination. *ACM Trans. Graph.* 37, 4, Article 153 (August 2018), 13 pages. https://doi.org/10.1145/3197517.3201328

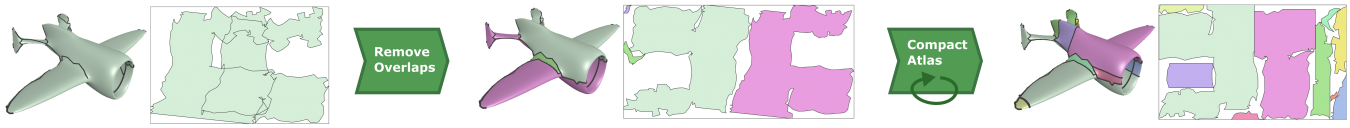
1 INTRODUCTION

2D parameterized chart atlases are ubiquitously used to store surface signals such as colors, textures, or normals [Hormann et al. 2007; Sheffer et al. 2007]. While many popular parameterization methods produce irregularly shaped atlas charts, the actual signals they encode are typically stored as 2D rectangular images defined over the bounding box of the packed atlas to allow for efficient access and GPU processing. Any space within this box that is not occupied by the atlas charts (Figure 1a) is wasted as it contains no actual signal. Wasting significant amounts of both memory and disk space is a concern for asset-heavy real-time applications such as games. Space waste considerations are also of concern for applications that use 2D panels for fabrication: when panels are cut out from flat material sheets, the material within the bounding box of the cut panels is often too fragmented to re-use for other needs and is consequently discarded [Koo et al. 2017] (Figure 1c). Our *Box Cutter* algorithm reduces material and memory waste by refining and repacking input atlases to explicitly maximize their *packing efficiency*, or the ratio between the sum of areas of the atlas charts and the area of their bounding box (Figure 1bd).

We achieve this goal by strategically cutting the existing charts to facilitate more efficient packing. In selecting the cuts, we seek to

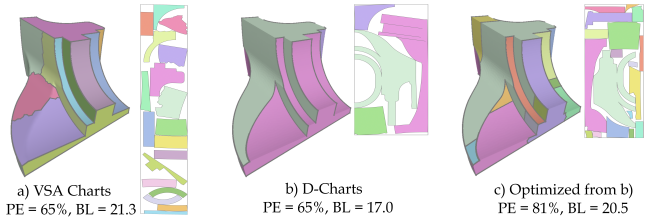
Authors' addresses: Max Limper, max.limper@siggraph.org, Fraunhofer IGD / TU Darmstadt, Fraunhoferstr. 5, Darmstadt, 64283, Germany; Nicholas Vining, nvining@cs.ubc.ca, University of British Columbia, 201-2366 Main Mall, Vancouver, British Columbia, V6T 1Z4, Canada; Alla Sheffer, sheffa@cs.ubc.ca, University of British Columbia, 201-2366 Main Mall, Vancouver, British Columbia, V6T 1Z4, Canada.

0730-0301/2018/8-ART153 $15.00
https://doi.org/10.1145/3197517.3201328



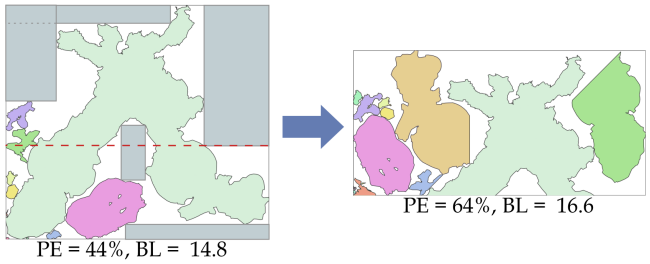
a) Input, PE = 69%, BL = 11.3 b) Bijective Atlas, PE = 68%, BL = 11.5 c) Final Atlas, PE = 82%, BL = 15.5

Fig. 2. Box Cutter overview: Overlaps in the input (a) are resolved and seams are closed (if feasible), leading to a bijective parameterization with short boundaries (b). The main stage of our algorithm then repeatedly cuts and repacks charts producing an atlas with high packing efficiency (PE) and controlled maximum boundary length (BL) (c).



a) VSA Charts PE = 65%, BL = 21.3 b) D-Charts PE = 65%, BL = 17.0 c) Optimized from b) PE = 81%, BL = 20.5

Fig. 3. Compactness versus packing efficiency: Packing compact, near convex, charts produced by VSA [Cohen-Steiner et al. 2004] (a) produces an atlas with the same packing efficiency as the much less compact ones produced by D-Charts [Julius et al. 2005] (b), which have much shorter boundaries. Our method refines the D-Charts atlas (b) to produce an atlas (c) with significantly higher packing efficiency, while constraining the boundary length to be less than the charts in (a).



PE = 44%, BL = 14.8 PE = 64%, BL = 16.6

Fig. 4. (left) Large void boxes in an existing atlas (grey) are strongly suggestive of desirable cut locations (dashed lines). (right) After executing the cuts and repacking the new charts, packing efficiency significantly improves.

cap or minimize boundary length elongation, since longer boundaries may lead to decreased rendering performance due to cache misses [Hakura and Gupta 1997], and may also cause texturing artifacts [Poranne et al. 2017]; resolving these artifacts requires duplicate storage of cross-seam signal content [González and Patow 2009]. Longer seams are similarly undesirable in fabrication settings, where they increase fabrication time and may affect the appearance of the output. Lastly, atlases used for either signal storage or fabrication must be overlap-free, yet the raw atlases produced by many parameterization methods contain overlaps (Figure 1c,Figure 2). Box Cutter eliminates these overlaps prior to packing using an effective boundary length minimizing strategy.

Previous approaches to atlas generation compute chart boundaries either prior to, or during, the parameterization stage (Section 2). Consequently, their choices impact not only packing efficiency but mapping distortion and continuity. Our method operates on previously computed atlases and does not change local chart geometry; therefore it does not affect distortion and maintains mapping continuity (up to a rigid transformation) across all newly generated seams. It is therefore well suited for use in conjunction with recent frameworks that cut the processed model during parameterization (e.g. [Myles and Zorin 2012; Poranne et al. 2017]). Users can first use one of these frameworks to provide the desired tradeoff between mapping distortion and boundary length, and then use our method to remove overlaps and optimize packing efficiency.

Computing charts that are amenable to efficient packing is known to be computationally hard [Chen et al. 2015]. A common approach in prior art has been to rely on geometric proxies, or properties of individual charts, as the means to predict packing efficiency. One commonly used proxy is chart compactness (convexity and roundness) [Sander et al. 2003; Zhou et al. 2004]. However, compactness is not a reliable proxy as both compact and highly non-compact charts with comparable boundary lengths can be packed with equal efficiency (Figure 3). Similarly, while pyramid-shaped geometries often allow for efficient packing [Chen et al. 2015], one can easily find examples where this is not the case: for instance, the packing efficiency of any individual triangle is just 50%.

Instead of looking for indirect proxy chart properties that make them amenable to efficient packing, we derive the location of our new cuts from actual packing solutions. We note that, given a packed atlas, the locations of the unused empty spaces - or *voids* - are strongly suggestive of potentially desirable cut locations (Figure 4). In particular, the *side lines* of rectangular axis-aligned voids, or *void boxes*, are strongly suggestive of efficiency improving cut locations. By extending these *side lines* we obtain a collection of refined charts that can often be rearranged to form a more efficiently packed atlas. We therefore use such void box elimination steps as the core operation of our optimization framework.

Overview. We optimize packing efficiency using an iterative *cut-and-repack* process. At each iteration we pack our charts into an atlas, locate void boxes, and assess the impact of removing them using one of the side line cut options. We then select and execute the elimination step that best improves packing efficiency while avoiding extreme boundary elongation. To accurately and rapidly assess packing efficiency, we follow a common approach used by practitioners in the real-time computer graphics space [Dalmau 2003] and employ a rasterized working space. Raster space computation allows us to efficiently repack charts, detect void boxes, and compute different atlas properties.

To process raw atlases generated by popular parameterization techniques, before applying the cut-and-repack iterations, we eliminate any pre-existing global overlaps within the input charts (Section 4). We express the problem of finding the shortest necessary mesh cuts that separate the overlapping regions as a Minimum Cost

Lifted Multicut Problem (LMP) and efficiently solve it with existing algorithms. We then further reduce cut length by welding the resulting charts together across different sets of edges when local chart geometry allows for distortion-free and overlap-free welding.

Contribution. In summary, the contributions provided by this paper are as follows. We present a framework for optimizing packing efficiency of input 2D atlases that guarantees overlap-free atlases and avoids undesirably long chart boundaries. Our method is capable of processing input meshes with tens of thousands of triangles in just a few minutes. Our framework is based on two key novel technical components: an effective chart cutting heuristic based on void boxes, which facilitates efficient search for desirable atlas configurations, and a generator for initial overlap-free layouts that explicitly minimizes the resulting boundary length.

We validate our method by optimizing the packing efficiency of a large range of atlases produced by multiple popular parameterization methods (Section 5). Our framework eliminates all input overlaps and improves packing efficiency by 14% or more compared to both the initial and the overlap-free layouts, and a mean average of 54%, with an increase in boundary length of 30% or less. We compare our results to atlases generated using alternative approaches, and demonstrate our ability to control the tradeoff between packing efficiency and boundary length. We extend our algorithm to incorporate a range of user controls, and demonstrate the usability of our outputs for rendering and fabrication applications.

2 PREVIOUS WORK

Our work builds on prior research on mesh parameterization, segmentation for 2D parameterization and unfolding, cutting for fabrication, and atlas or polygon packing.

Mesh Parameterization. Most planar mesh parameterization methods compute 2D embeddings of 3D meshes with a given disc-topology boundary, while minimizing different distortion energies [Botsch et al. 2010; Hormann et al. 2007; Sheffer et al. 2007]. Fixed boundary methods [Floater 1997; Gu et al. 2002] allow for generation of strictly rectangular parameterization atlases with optimal packing efficiency; however these methods introduce more distortion than their free-boundary counterparts (such as [Lévy et al. 2002; Sander et al. 2001; Sheffer et al. 2005; Smith and Schaefer 2015]), and thus are less frequently used. None of these methods account for packing efficiency, and they offer no means to control it. Our method can process the outputs of any of these frameworks and improve packing efficiency of the resulting atlas while keeping the mapping distortion fixed. Our overlap resolution process is particularly suitable for processing outputs of global parameterization methods [Ben-Chen et al. 2008; Jin et al. 2004; Kharevych et al. 2006; Myles and Zorin 2012], which are typically prone to overlaps; our method leverages the global mapping continuity they provide to facilitate the boundary minimizing welding process (Section 4).

Our framework is not directly applicable for parameterizations that constrain chart coordinates to particular values, such as [Ray et al. 2010], but can be easily combined with methods such as [Liu et al. 2017] which edit the underlying 2D signal to generate seamless textures.

Mesh Cutting. Multiple methods cut 3D surface meshes prior to parameterization (e.g. [Gu et al. 2002; Sheffer and Hart 2002]), and try to balance anticipated mapping distortion and boundary length. Simultaneous cutting-and-unfolding methods [Poranne et al. 2017; Sorkine et al. 2002] balance distortion and cut length directly. None of these frameworks directly attempt to optimize for packing efficiency.

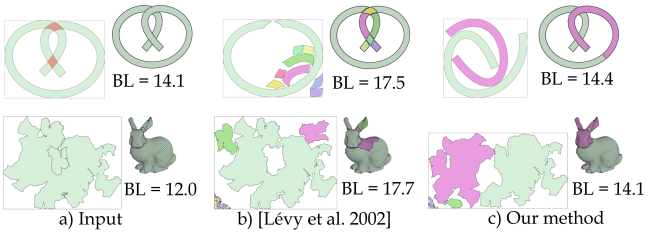


Fig. 5. Given an input parameterization with visible overlaps (left) the standard overlap removal approach of [Lévy et al. 2002] produces unnecessarily long boundaries (center); our results (right) have shorter boundaries and typically contain fewer charts (all examples packed using our method).

Several segmentation approaches [Julius et al. 2005; Lévy et al. 2002; Sander et al. 2002, 2003; Zhou et al. 2004] prioritize more compact (namely more convex, or round, charts), implicitly improving packing efficiency and reducing the likelihood of overlaps. As demonstrated by Figure 3 compactness does not accurately predict packing efficiency, and directly optimizing for it often leads to longer boundaries than necessary for achieving an acceptable packing outcome. Moreover, as pointed out by Poranne et al. [2017], cutting the surface prior to parameterization often introduces undesirable artifacts that can be avoided when the processes are performed in tandem.

Our framework allows users to separate packing efficiency from distortion considerations, and provides a direct trade-off between boundary length and packing efficiency. We use it to significantly improve the packing efficiency of atlases produced by a range of mesh cutting methods (Section 5).

Carr and Hart [2002] optimize packing efficiency by segmenting input meshes into charts that consist of two to six triangles each; these charts can then be efficiently packed into a given bounding box. Their technique combines low distortion with high packing efficiency, but results in an extremely high number of charts. This makes their approach unsuitable for applications such as modern rendering pipelines, both real-time and offline (cp. [Munkberg et al. 2016]), and for manufacturing where excessive seams cause both high manufacturing time and undesirable visual artifacts. Purnomo et al [2004] use tightly packed tiny quadrilateral charts to generate seamless textures. This framework suffers from similar drawbacks.

Overlap Avoidance and Resolution. Overlaps are a common feature of many free-boundary parameterization outputs [Hormann et al. 2007]. Recent methods allow for bijective atlas generation for inputs with fixed boundary connectivity [Jiang et al. 2017; Smith and Schaefer 2015]. Given a configuration in which unconstrained unfolding results in large overlaps (e.g. Figure 5), this approach can lead to major increases in mapping distortion. Methods that simultaneously cut and unfold the meshes [Poranne et al. 2017; Sorkine et al. 2002] can potentially prevent overlaps by adding new cuts on the fly whenever overlaps occur. However such cuts can drastically

increase the length of the resulting boundaries. Our framework allows users to employ boundary computation and mapping methods that do not explicitly prevent global overlaps, and to resolve these overlaps after the fact by introducing additional cuts where necessary. Consequently the resulting mapping distortion is not affected by overlap extent.

Our approach is consistent with the common industry practice of resolving overlaps once they occur. The standard approach to overlap removal, first proposed by Lévy et al. [2002], is to segment atlas charts which exhibit overlaps along projections of overlapping region boundaries. This approach may result in many small redundant charts and unnecessarily long boundaries (Figure 5b). By using a global cutting strategy designed to minimize boundary length, our method produces significantly shorter boundaries than this earlier method (Figure 5c).

Atlas Packing. Maximizing the packing efficiency for a given set of charts is NP-hard [Garey and Johnson 1979; Milenkovic 1999]; thus existing methods use heuristic strategies that balance packing efficiency against computation time [Lévy et al. 2002; Nöll and Stricker 2011; Sander et al. 2003]. While these methods keep chart geometry fixed, we introduce new cuts guided by the efficiency of actual packings, allowing for significant improvement in packing efficiency. We use a variation of the framework of Nöll et al [2011] for the computation of actual packings in our method (Section 3.3).

Cutting and Packing for Fabrication. The problem we address shares some similarity with packing of 3D shapes for fabrication or compact storage [Chen et al. 2015; Zhou et al. 2014]. Zhou et al. [2014] fold 3D objects into boxes by first discretizing each input using an optimized voxel grid designed to minimize wasted space. Such discretization unnecessarily reduces the degrees of freedom for our setting and would lead to longer than necessary boundaries. Koo et al. [Koo et al. 2017] minimize the wasted material used to fabricate furniture from flat pieces by subtly changing the geometry of these pieces. Our framework is designed to keep the local 2D geometry fixed, and to improve packing efficiency by using strategic cuts. The Dapper algorithm [Chen et al. 2015] cuts shapes into a small number of parts, such that the parts can fit into the build volume of a 3D fabrication device or can be packed into compact packages for transportation. The authors note that such cutting and packing problems are NP hard, and employ pyramidal decomposition as a heuristic solution strategy. On 2D data, our method achieves better packing efficiency while introducing shorter boundaries than their approach (Figure 17).

3 COMPACTING THE ATLAS

The core step of our algorithm takes an overlap free atlas as input, and improves its packing efficiency via a cut-and-repack strategy. We introduce *compacting cuts* that free empty spaces in the packing by cutting one or more charts located next to void boxes (Section 3.1), and then repack the resulting charts. We repeat the cutting and packing steps (Section 3.2) until no further improvement is possible without violating user constraints.

Users can constrain the amount of boundary elongation allowed, specify a minimal acceptable size for the resulting charts, bias cutting to avoid user-specified "no-cut" areas, put a time limit on the computation, or simply let the method run until no further improvement is possible. (e.g. Figures 6, 19). In the latter case, the algorithm will terminate when a fixed number of attempts does not improve packing efficiency by more than a given minimum improvement p_ϵ.

Our overall cutting and compacting strategy is agnostic to the choice of packing method used, and can operate in conjunction with any existing packing method (Section 2). However, since we repeatedly employ packing as an assessment tool to determine which cuts to employ, our runtimes are highly dependent on packing time. The packing framework we use (Section 3.3) is optimized for providing a suitable trade-off between packing efficiency and computation time, allowing us to use the packer as a black box evaluation tool between fifty to a hundred times throughout the computation, while keeping the overall computation time at around five minutes on average.

3.1 Cutting Strategy

The key observation behind our method is that inefficient packings most often result from the presence of large contiguous unused spaces, or *voids*, inside the packed chart atlas. Our goal, therefore, can be cast as identifying the best cuts that will help to remove such voids. We specifically focus on axis-aligned maximal *void boxes* - boxes whose sides are aligned with the sides of the atlas's bounding box, which contain no atlas triangles, and whose size cannot be further increased without intersecting an atlas chart (Figure 7). The axis-aligned lines that coincide with the sides of these boxes provide possible cut candidates, or *cut lines*, for packing improvement. For voids immediately next to bounding box corners, we have one cut line in each direction; for voids next to bounding box sides, we have one cut line in the direction of this side and two in the other; and for interior voids we have two lines in each direction. We observe that these lines bound a subset of charts, which we call the *supporting charts* (Figure 7b). If we cut the chart atlas along one of these pairs of lines and remove the supporting charts, then the packing efficiency of the remaining atlas can be trivially improved by collapsing the resulting empty space and moving the top line, and all portions of the atlas on top of it, so that the top and bottom lines coincide (Figure 7c). We then need to re-pack the removed supporting charts , which will in most cases lead to an improvement in packing efficiency, compared to the initial configuration (Figure 7a,d). While the size of the supporting charts depends on the choice of the cut lines, for large void boxes, at least one of the two cut line candidates typically produces relatively small supporting charts. Packing algorithms generally perform better given more and smaller charts, as they have more degrees of freedom in packing them. Thus while we have no guarantees that greedily guiding cuts by prioritizing the elimination of largest voids will always improve the packing outcome, this is very frequently the case. To avoid redundant cuts, rather than use the void box size as a proxy for packing efficiency improvement, we evaluate the impact of the cuts by directly computing the packing efficiency of the resulting atlas.

The effectiveness of the global approach outlined above diminishes when no large voids exist inside the atlas. In such cases performing *global cuts*, which cross multiple charts inside an atlas, may lead to extensive boundary elongation with possibly only a small improvement in packing efficiency. At the same time, we observe that the overall packing quality of a chart atlas is often affected by the packing efficiency of individual charts, namely how efficiently

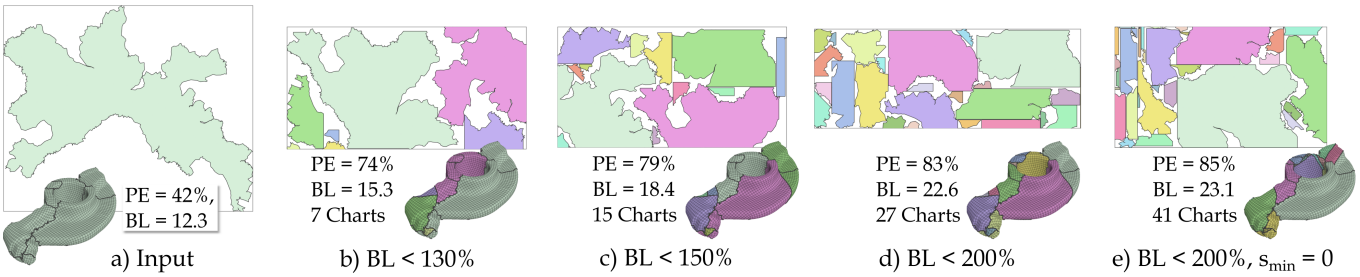


Fig. 6. Influence of different parameters when optimizing an input parameterization (a). Our algorithm can be configured to terminate at different boundary length budgets (b-d), as well as to prevent (b-d) or to allow (e) the creation of small pieces during cutting.

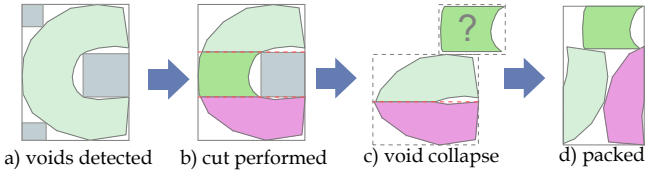


Fig. 7. Compacting cut: a) detected maximal void boxes; b) a pair of cut lines derived from one of the void boxes, and the associated supporting chart in green; c) conceptual collapsing of the void box after removal of the supporting chart; d) a more efficient packing achieved using these cuts.

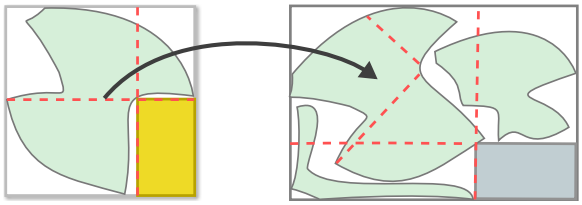


Fig. 8. Local and global cuts, as derived from local and global void boxes (shown in yellow and gray respectively). Left: an axis-aligned chart and a respective void box. Right: global atlas layout, including the chart from the left. Both boxes induce cuts that may help to improve the overall packing efficiency.

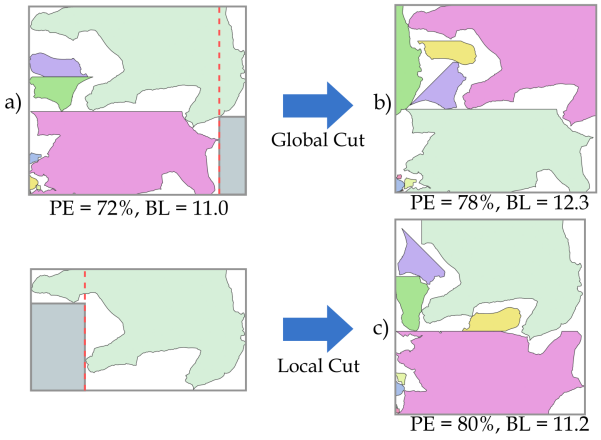


Fig. 9. Local and global cuts on the *horse* model. The atlas shown in (a) is already relatively compact. In this case, a global cut solution (b) is outperformed by a local cut (c).

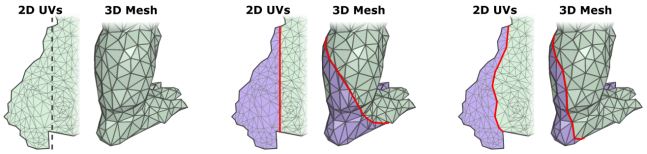
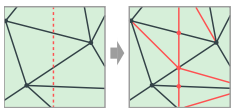


Fig. 10. Cutting charts (detail view on the *Armadillo* dataset). Left: Input piece and cut line. Center: Precise cutting, introducing new triangles. Right: Shortest-path cuts through existing mesh edges.

an individual chart is packed inside of its own *oriented* bounding box (Figure 8, Figure 9a). While an atlas packing method can often pack smaller charts within the void spaces surrounding charts with low packing efficiency, eliminating these voids directly can often dramatically improve the overall packing efficiency (Figure 8c). Therefore, we also use the same void box detection process, on a per-chart level, to evaluate a set of *local cut candidates*, where we apply cuts to individual charts and consider maximal void boxes that are oriented with respect to their parent chart.

3.2 Cut and Repack

The input to each cut-and-repack step is a packed atlas, with a given packing efficiency p and total boundary length b. Each cutting step generates one or two cut lines across the current charts and repacks the resulting refined charts. The step first detects both local and global cut candidates ($n = 4$ each) by locating the n largest void boxes in either the current atlas or in the bounding boxes of the current charts (Section 3.2.1). It then optimizes and ranks the candidate cuts, both local and global, selects the best one, performs the cut, and re-packs the chart atlas. We provide two possible methods for performing the cut. The first method precisely cuts through each affected chart by subdividing all triangles intersected by a cut line, triangulating any resulting quads and fixing any introduced T-Junctions (see inset figure).



Alternatively, our method is able to split charts following shortest paths along the existing mesh edges, for applications where the creation of new triangles is not desired. In this case, we first locate all triangle edges that intersect the cut line, resulting in one or multiple intersected segments for each chart. We then order all intersected edges along each cut line and find, for each segment, the closest vertex of its first and last intersected edge. The cut is then performed by connecting those edges via the shortest 2D path along the mesh edges between those closest vertices. Examples are shown

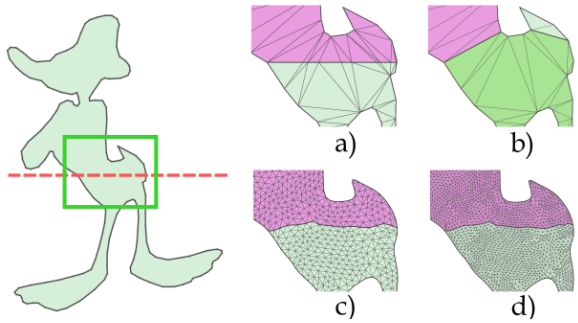
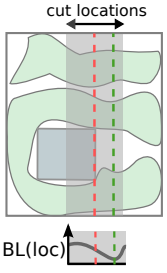


Fig. 11. Example of cutting through the *Duck* dataset). As an alternative to straight cuts through re-tessellation (a), our framework supports shortest-path cuts, which do not introduce new trianlges, but may lead to significantly different results depending on the mesh (b-d).

in Figure 10 and Figure 11. After cutting (using one the two possible methods), we repack and re-rasterize the new set of charts.

Locating Void Boxes. We extract candidate void boxes using an efficient raster space algorithm. Given a rasterized representation of either the entire atlas (global case), or of an individual chart (local case), we use a simple axis-aligned scanline algorithm to locate the largest void boxes. We store, for each pixel, the number of subsequent empty pixels in the horizontal direction inside a *skip buffer*, having the same size as the rasterized atlas. We then efficiently compute, for each empty pixel, the largest possible void box. This is done by iterating along a line over the vertical direction (up and down) until non-empty pixels or the chart boundary are reached, while tracking at the same time the largest possible horizontal extent using the skip buffer. We record all maximum empty boxes with size above a given threshold in a list. As this list may contain overlapping boxes, we filter them out by first sorting the list of boxes by size and then, for each entry, visiting all subsequent entries and deleting any box that has more than a predefined amount (in our case, 10%) of overlap. The result is a sorted list of void boxes with minimal or no overlap and the maximum extent possible; no box in the list can increase in size without intersecting a chart (Figure 8). We then select the n largest void boxes to induce axis-aligned vertical and horizontal cuts as cut candidates.

Optimizing Cut Locations. Our decision to operate on maximal voids is driven by packing efficiency; however, we also wish to account for boundary length when considering our choice of cuts. We note that minor axis-aligned shifts in the cut line locations can often significantly reduce the cut length, and help avoid formation of tiny charts (Figure 6,de). For each pair of candidate cut lines obtained, we consequently perform a local line search that computes a location that minimizes the resulting cut lengths. We define a range of evenly spaced offsets within 5% of the corresponding bounding box dimension, and explicitly evaluate the lengths of the resulting cuts for each. Since the computation is very fast (intersection with an axis aligned line) we use a dense set of 100 equally distributed offset samples. We then select the best location. Note that if the line does not cut any charts, then the operation will have no impact on



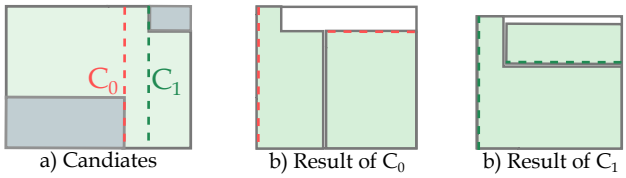


Fig. 12. Cut optimality as an estimate. Among two cut candidates (a), the one with shorter cut length and a larger void may still lead to a less efficient packing (b), compared to the cut that has lower score (c).

the packing. We thus always select the cut lines that cut across at least one chart.

Ranking Cuts. The impact of any candidate cut depends on our ability to efficiently re-incorporate the removed support charts into the atlas. Thus our estimate of candidate cut optimality is only an estimate - it may be that a less promising cut may outperform a more promising candidate (Figure 12).

Thus instead of greedily applying the most promising cut, we assess all $2n$ possible candidate cuts by executing each of them, computing a new packing, and computing its packing efficiency p. Rather than selecting the cut that maximizes packing efficiency, we seek to balance efficiency against boundary elongation. We therefore compute the score of a given cut as $s = \frac{p}{b^\alpha}$, where b is the ratio of the boundary length to the length of the longest bounding box side of the initial atlas, and p is the packing efficiency measured in the rasterized space (Section 3.3). We use $\alpha = 0.2$ in all the examples in the paper. We choose the cut with the highest score s.

3.2.1 Local Bounding Boxes. To compute local candidate voids, we need to define a local atlas, or oriented bounding box, per-chart. In general, the tightest bounding box for a chart would provide the best packing efficiency for this individual chart; however, this is not our goal. Instead, we search for the oriented bounding box of the chart which produces the largest void boxes possible and thus has the most potential to improve our packing.

From this perspective, an orientation that maximally aligns the sides of the chart with the axis directions is a better alternative, since it is likely to align the sides of the maximal concavities on the chart with the major axes, and result in cuts that produce charts which are both convex and boxy and hence better suited for packing. We compute a suitable orientation by locating the orthogonal coordinate system whose edges are best aligned with the directions of the chart boundaries. Specifically we minimize the L_1 norm of the boundary edge vectors over all possible rotation angles θ:

$$E(\theta) = \sum |u_1(\theta)| + |v_1(\theta)| + \ldots + |u_n(\theta)| + |v_n(\theta)|.$$

Here $\{\langle u_1, v_1\rangle, \ldots, \langle u_n, v_n\rangle\}$ are the rotated boundary edge vectors. We obtain an approximate solution for the desired chart alignment via greedy search. When minimizing $E(\theta)$, we seek to optimize the alignment between the global boundary directions and the major axes. However, raw chart boundaries can contain high-frequency details. We ignore those details in the computation by pre-smoothing the raw edge direction vectors, using a simple averaging of the

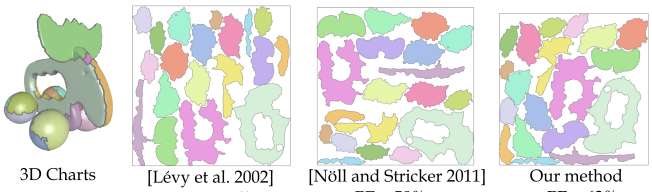
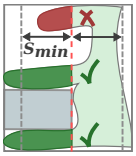


Fig. 13. Results of chart packing into a square-shaped atlas, using different packing algorithms. For [Nöll and Stricker 2011], the bounded (non-modulo) variant has been used.

immediate neighbors of each edge vector. θ itself is optimized via brute-force search.

3.2.2 User Constraints. We allow users to introduce a range of cutting constraints beyond boundary length restrictions.

Outward Offsetting. A range of applications require allocation of extra space around chart boundaries. The quality of rendered signals along boundaries can be improved by adding gutters [González and Patow 2009]; tailoring applications require seam allowances; and papercraft and other fabrication settings often benefit from flaps. In our setting, accounting for all of these extra space requirements is straightforward. Prior to each chart packing computation, we offset the charts outwards by the amount necessary for the target application, and then pack these extended charts as before. Our optimization naturally adapts to this change by taking the extra amount of space induced by the cuts into account during packing efficiency computation.

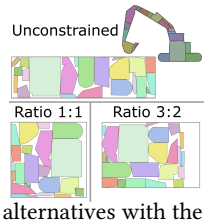


Preventing Small Charts. Small charts can negatively affect some of our target applications; for example small charts may require an inappropriately large amount of gutter space for rendering, compared to their area, and are hard to manipulate in fabrication settings. Unconstrained cuts, even after local cut optimization, may result in small charts being produced. We explicitly prevent the formation of small charts by adding a respective constraint to the cutting step and, likewise, to the cut evaluation during cut optimization. We check each new chart resulting from a cut to determine whether its extent, measured by orthogonal distance to the cut line, satisfies a minimum size threshold $s_{\min}$. If this is not the case, the respective segment of the cut line is ignored, leaving its particular region uncut. Figure 6e shows the effect of setting $s_{\min}$ to 0, allowing the formation of tiny charts. For all other examples in the paper we set $s_{\min}$ to 1% of the length of the largest side of the input atlas's bounding box.

Biasing Cut Locations. A variant of our algorithm allows the user to provide per-vertex weights, effectively specifying importance values for the different regions of the mesh. When computing scores for cut candidates, these importance values can be taken into account by multiplying the length of each resulting edge by its importance (which is computed as the average of the importance value of its two vertices). We then use this importance-biased boundary length b_{imp} instead of b to compute the score s for each possible solution.

3.3 Packing Computation

While our method can operate in conjunction with any packing method, we found the following method, which extends the framework of [Nöll and Stricker 2011], to provide the time versus packing efficiency trade off we need. As with prior work we perform packing in raster space. We follow the standard approach of sorting charts from large to small, and then greedily placing them inside the discrete working space. To assess placement we consider chart position and orientation (including mirroring).



In contrast to Nöll and others, who use horizon lines to track active area, we use the bounding boxes of the current charts. This allows placement of smaller charts inbetween the large ones. In selecting the optimal placement for each chart we by default choose the one that extends the current active area by the smallest number of pixels. Given multiple alternatives with the same minimal pixel count, we select the one that places the chart closest to the bottom left of the active area. This strategy pulls charts towards this single corner, keeping space free in other areas and facilitating our *pixel shifting* post-process. We support an optional strategy that computes an augmented active area which fits a given aspect ratio whenever a chart placement is evaluated. This allows us to produce packings that approximate a prescribed aspect ratio, which is useful in many practical scenarios (inset).

Hierarchical Optimization. In order to speed up the packing computation, we maintain a hierarchy of buffers, where the resolution of each coarser level is half of its previous one. The finest level of the working space contains the actual chart data, consisting of one chart ID per pixel. The remaining level pixels store the number of empty corresponding fine level ones. Storing this number along with each rasterized variant of a chart allows us to quickly reject placements in coarse pixels that do not contain enough free fine level pixels to accommodate a chart, and to directly compute the best possible placement within large empty regions. This feature reduces our run time by up to 60%. We explicitly store the boundary of each rasterized chart. This allows us to check the boundary pixels first when testing a possible placement at the finest level of the hierarchy. Thus we can resolve many cases earlier than it would be possible when checking chart pixels line by line.

Pixel-Shifting Post-Process. We prescribe an approximate resolution for the rasterized atlas, and use it to rasterize each chart. We rasterize charts conservatively: whenever a triangle partially overlaps a pixel, that pixel is set. The raster resolution needs to be kept moderate (128^2 or 256^2), since too many candidate evaluations would slow down the packing process. However, the choice of a resolution limits packing efficiency, as the unused continuous domain space between two charts that are densely packed next to each other in raster space may, in the extreme case, cover almost an entire pixel, which can be a notable distance at moderate resolutions. To eliminate such spaces, we optimize the packing in a post-process at higher resolution (1024^2). Our optimizer translates charts pixel by pixel towards a given *gravity* direction (e.g., to the bottom left corner), as long as no other chart is being intersected. This process is executed repeatedly for each chart until no chart can be moved any more. Despite the higher resolution of the rasterized atlas, this

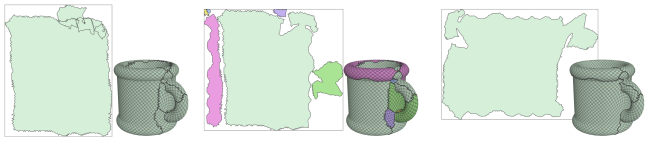


Fig. 14. Overlap removal. After removing overlaps from the input (left), resulting charts (center) can be welded together along a common 3D space seam segment if the parameterization is globally continuous across this segment, and if the combined chart (right) has no self-overlaps. Input has been globally parameterized using [Lipman 2012].

post-process consumes far less time than the actual packing, as we only need to evaluate translations of the boundary pixels of each chart in a given direction.

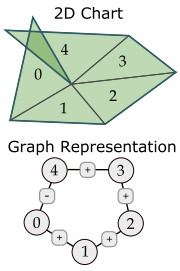
For typical scenarios, our packing method is able to efficiently pack arbitrarily shaped charts inside a common atlas within a second or less; an example packing is shown in Figure 13.

4 OVERLAP REMOVAL

We use an overlap resolution process that separates overlapping chart regions using minimal length cuts as a preprocess to our cut-and-pack algorithm. For globally continuous input atlases, we employ an additional optional step that shortens the boundary of the resulting atlas by attaching charts together, or welding them, across globally continuous boundaries.

4.1 Chart Splitting

We eliminate overlaps by casting overlap removal as a multi-cut problem. We find new charts by partitioning the graph into clusters; each generated cluster represents a new chart, and overlapping triangles are constrained to be assigned to different clusters. At the same time, we wish to minimize the length of the boundaries between these resulting clusters. We define a graph $G = (V, E)$ whose vertices v_i correspond to the triangles i of the input atlas. We construct graph edges as follows. First, we add an edge $e_{i,j}$ for every pair of triangles i and j that share a common edge within a given chart. We assign this edge the weight $l_{i,j}/l_{avg}$, where $l_{i,j}$ is the length of this edge and l_{avg} is the average edge length within the 2D atlas mesh. We then add an edge $e_{m,n}$ for every pair of overlapping triangles m and n, and assign them a weight of *-infinity*.



An example is shown on the right. Triangles 1 and 2 are neighbors inside the mesh; therefore they are connected through an edge inside the graph, which has a positive weight. Triangles 0 and 4 are not neighbors, but they are overlapping; therefore, the graph contains an additional edge with negative weight between those two triangles (nodes). In the presence of globally continuous chart boundaries, we add additional edges that bias the cut choice to prioritize cuts that facilitate subsequent boundary shortening via chart welding. For each chart that contains overlaps, we identify the globally continuous boundary segments, namely segments that are associated with the same 3D seam and that are a rigid transformation of one another in 2D space. For each pair of triangles i and j on opposite sides of such a boundary we create a graph edge $e_{i,j}$ with a weight $-l_{i,j}/l_{avg}$. These weights encourage cuts that separate such matching segments, indirectly increasing the likelihood of welding them in our subsequent boundary shortening step.

We then find the multi-cut that finds a set of cut edges that maximizes $\sum_{e_{ij}} W_{e_{ij}} Y_{e_{ij}}$; here $Y_{e_{ij}} = 1$ if the nodes i and j are in the same cluster and 0 otherwise. Our multicut approach has two advantages. First, by assigning infinite negative weights to edges, we ensure that any solution must assign overlapping nodes to different clusters, by definition; connecting overlapping triangles with edges with infinite negative weights therefore guarantees an output in which each cluster is free of overlaps. Second, the number of new clusters naturally emerges from this solution. While this problem is NP-complete in theory, excellent approximation algorithms exist. We use the extended Kernighan-Lin algorithm proposed by Keuper et al. [Kernighan and Lin 1970; Keuper et al. 2015], and obtain labels $Y_e \in \{0, 1\}$ for each graph edge e, indicating the edges along which we should cut the mesh. By performing these cuts, we obtain the desired set of overlap-free charts.

Since our method operates on *triangles*, (in contrast to methods that operate solely on boundary edges), we are able to remove all possible types of overlaps, including local overlaps (triangle flips) that may occur in the interior without affecting the boundary, and overlaps that may occur without two points of boundary crossing. In our implementation, overlapping triangles are quickly detected by sorting 2D triangles into a regular grid and checking, for each triangle, all other triangles that are intersecting with its cells. Our weighting scheme can easy incorporate other criteria; for example, edges between adjacent triangles can be weighed to reflect visual importance, and to move cuts away from more visually important areas (Figure 19).

4.2 Chart Welding

Our overlap removal step minimally elongates the input boundaries. While in many cases this elongation is necessary, we often can further reduce the resulting boundary length by reattaching charts along *compatible boundaries*. We consider a pair of boundary segments on different charts to be compatible if: (1) these segments lie on opposite sides of a common 3D seam; (2) the parameterizaton is globally continuous across this segment; and (3) rigidly transforming one chart to match the other along this segment does not result in overlaps between the two charts (Figure 14). If all three criteria hold, we can reduce the boundary length in our chart atlas by welding the charts — transforming one chart to match the two boundaries, and then closing the seam by merging its vertices in 2D space.

We shorten the boundary of the overlap-free chart layout resulting from the previous stage by using a greedy strategy. We repeatedly choose the longest pair of compatible boundary segments and weld them together. While this process shortens the boundaries it can severely decrease packing efficiency. We thus only weld charts together if the packing efficiency of the resulting atlas remains within a fixed bound of the pre-welding one (here we use 10%). We repeat the welding steps until no more improvement is possible without violating this constraint.

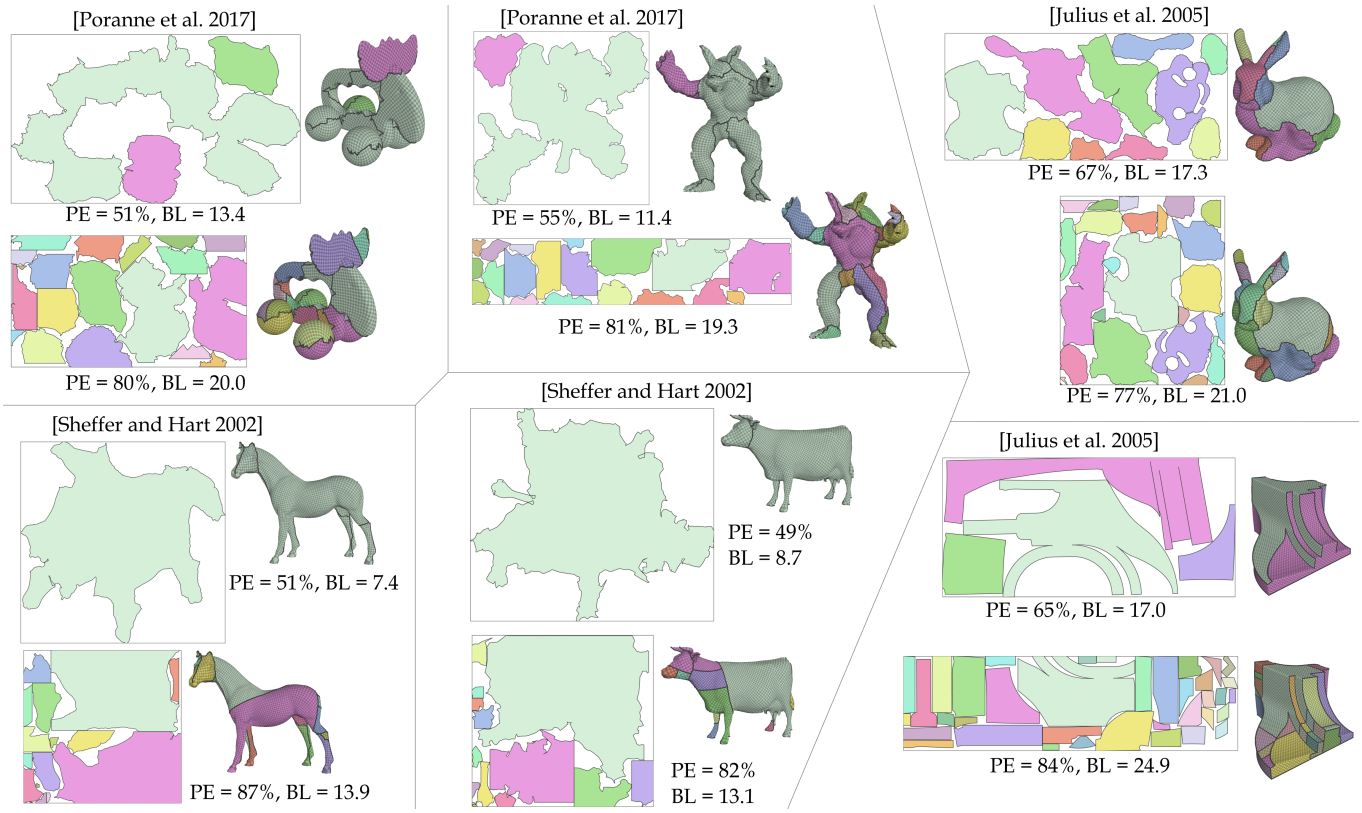


Fig. 15. Results of optimization on various kinds of input models, generated by different parameterization algorithms. The horse and cow models have been parameterized using ABF++ [Sheffer et al. 2005].

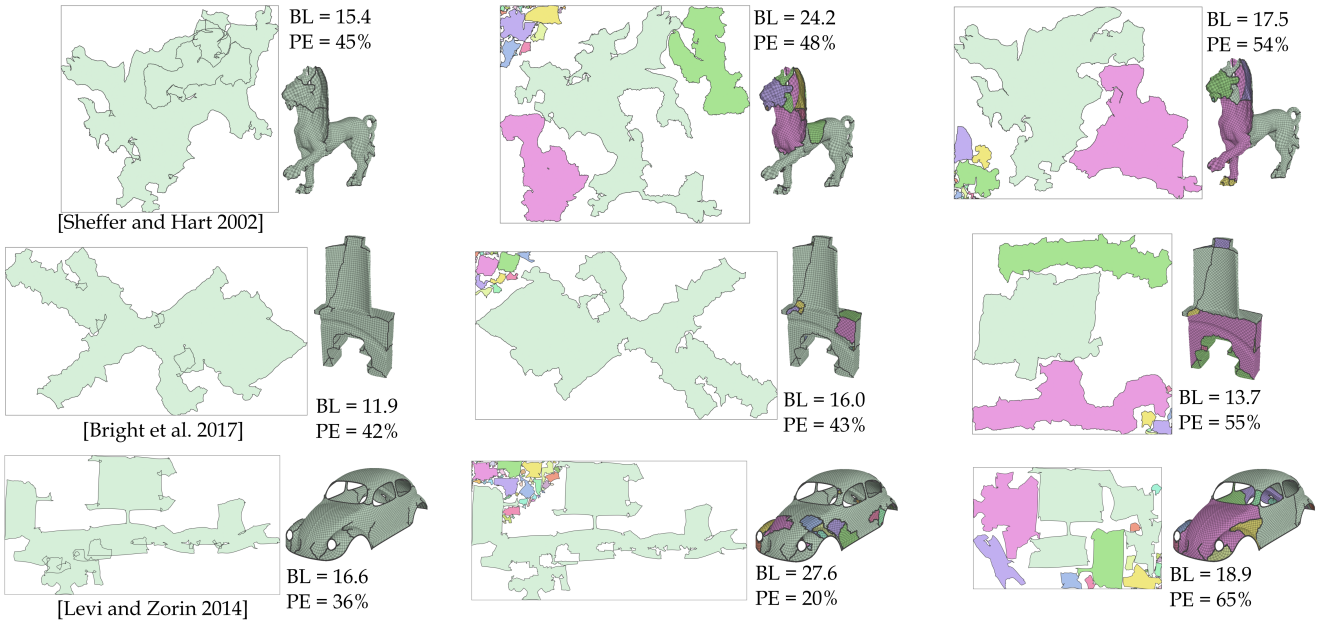


Fig. 16. Removing overlaps to obtain a bijective parameterization. From left to right: input, overlaps removed using the standard method of [Lévy et al. 2002], overlaps removed using our method. The feline model has been parameterized using ABF++ [Sheffer et al. 2005].

Table 1. Results of optimization of various input data, using straight cuts and re-tesselation(top) and shortest-path cuts through the mesh (bottom). The tables show, for different boundary length thresholds, the increase in packing efficiency compared to the overlap-free version, the output packing efficiency and boundary length.

Straight Cuts

Model [Method]	PE / BL	PE Improvement (PE / BL)		
	Bijective	BLen < 130%	BLen < 150%	BLen < 200%
armadillo [PTH+17]	55% / 11.3	**+33%** (73% / 14.4)	**+45%** (80% /15.9)	**+46%** (81% /19.3)
elk [PTH+17]	51% / 13.4	**+42%** (72% / 16.4)	**+58%** (80% /20.0)	**+58%** (80% /20.0)
girl [PTH+17]	44% / 14.8	**+75%** (77% / 18.7)	**+81%** (80% /21.5)	**+85%** (82% /22.8)
beethoven [SH02]	59% / 16.2	**+31%** (77% / 20.1)	**+36%** (80% /22.1)	**+36%** (80% /22.1)
bunny [SH02]	62% / 14.1	**+28%** (79% / 18.1)	**+30%** (81% /20.0)	**+31%** (81% /23.0)
cow [SH02]	49% / 9.1	**+57%** (77% / 11.4)	**+66%** (82% /13.1)	**+66%** (82% /13.1)
feline [SH02]	54% / 17.5	**+37%** (75% / 20.5)	**+41%** (77% /25.5)	**+41%** (77% /25.5)
gargoyle [SH02]	55% / 11.0	**+33%** (73% / 12.9)	**+53%** (83% /16.4)	**+56%** (85% /18.1)
horse [SH02]	51% / 7.8	**+34%** (69% / 9.7)	**+50%** (77% /11.1)	**+71%** (87% /13.9)
aircraft [Lip12]	68% / 12.2	**+23%** (84% / 15.7)	**+23%** (84% /15.7)	**+23%** (84% /15.7)
cup [Lip12]	69% / 6.9	**+16%** (80% / 8.4)	**+24%** (85% /9.6)	**+30%** (89% /11.3)
blade [BCW17]	55% / 13.7	**+43%** (78% / 17.6)	**+46%** (80% /18.9)	**+47%** (80% /20.9)
cow2 [BCW17]	64% / 12.6	**+17%** (74% / 15.2)	**+20%** (76% /16.9)	**+30%** (83% /24.2)
ramses [BCW17]	58% / 10.8	**+29%** (75% / 14.0)	**+32%** (77% /14.2)	**+38%** (80% /19.0)
camel [BCE+13]	49% / 21.4	**+50%** (74% / 26.5)	**+50%** (74% /26.5)	**+50%** (74% /26.5)
aircraft [MPZ14]	58% / 18.5	**+40%** (81% / 22.9)	**+50%** (87% /27.3)	**+51%** (88% /28.3)
santa [MPZ14]	61% / 27.1	**+25%** (77% / 32.0)	**+25%** (77% /32.0)	**+25%** (77% /32.0)
beetle [LZ14]	65% / 18.9	**+21%** (78% / 22.4)	**+21%** (78% /22.4)	**+21%** (78% /22.4)
bozbezbozzel [LZ14]	60% / 27.7	**+20%** (72% / 33.1)	**+20%** (72% /33.1)	**+20%** (72% /33.1)
bird [CZL+15]	30% / 9.4	**+131%** (70% / 11.5)	**+172%** (83% /13.7)	**+181%** (85% /18.4)
duck [CZL+15]	29% / 10.7	**+159%** (76% / 13.3)	**+160%** (76% /16.1)	**+169%** (79% /19.9)
excavator [CZL+15]	30% / 9.6	**+114%** (64% / 11.2)	**+167%** (80% /14.0)	**+181%** (84% /17.6)
jordan [CZL+15]	16% / 11.4	**+273%** (61% / 13.0)	**+370%** (76% /16.1)	**+388%** (79% /19.2)
tower [CZL+15]	38% / 10.6	**+40%** (54% / 12.5)	**+89%** (73% /13.9)	**+131%** (89% /19.3)
bunny [JKS05]	68% / 17.6	**+14%** (77% / 21.0)	**+14%** (77% /21.0)	**+14%** (77% /21.0)
fandisk [JKS05]	61% / 17.4	**+37%** (83% / 22.2)	**+39%** (84% /24.9)	**+39%** (84% /24.9)
rockerarm [JSP17]	42% / 12.4	**+75%** (74% / 15.3)	**+86%** (79% /18.4)	**+96%** (83% /22.6)
venus [JSP17]	59% / 5.4	**+25%** (73% / 6.4)	**+40%** (82% /7.8)	**+55%** (91% /10.7)
Min.	16% (5.4)	**+14%** (54% / 6.4)	**+14%** (72% / 7.8)	**+14%** (72% / 10.7)
Max.	69% (27.7)	**+273%** (84% / 33.1)	**+370%** (87% / 33.1)	**+388%** (91% / 33.1)
Average	52% (13.9)	**+54%** (74% / 17.0)	**+68%** (79% / 18.9)	**+74%** (82% / 20.1)
Median	55% (12.5)	**+36%** (75% / 15.5)	**+46%** (80% / 17.7)	**+48%** (82% / 20.5)

Shortest-Path Cuts

Model [Method]	PE / BL	PE Improvement (PE / BL)		
	Bijective	BLen < 130%	BLen < 150%	BLen < 200%
armadillo [PTH+17]	55% (11.3)	**+33%** (73% / 14.6)	**+38%** (76% /16.9)	**+47%** (81% /20.2)
elk [PTH+17]	51% (13.4)	**+28%** (65% / 16.7)	**+46%** (74% /18.2)	**+56%** (79% /23.5)
girl [PTH+17]	44% (14.8)	**+66%** (73% / 17.7)	**+84%** (81% /22.2)	**+86%** (82% /25.5)
beethoven [SH02]	59% (16.2)	**+27%** (75% / 19.6)	**+32%** (78% /24.1)	**+36%** (80% /28.5)
bunny [SH02]	62% (14.1)	**+24%** (77% / 18.3)	**+27%** (79% /19.7)	**+34%** (83% /25.5)
cow [SH02]	49% (9.1)	**+48%** (73% / 10.7)	**+69%** (83% /13.7)	**+76%** (86% /15.9)
feline [SH02]	54% (17.5)	**+42%** (77% / 22.5)	**+48%** (81% /25.1)	**+52%** (83% /28.6)
gargoyle [SH02]	55% (11.0)	**+30%** (71% / 13.2)	**+40%** (76% /14.4)	**+51%** (82% /20.1)
horse [SH02]	51% (7.8)	**+30%** (67% / 8.5)	**+53%** (78% /10.7)	**+65%** (84% /15.1)
aircraft [Lip12]	68% (12.2)	**+19%** (81% / 14.7)	**+25%** (85% /17.7)	**+26%** (86% /20.4)
cup [Lip12]	69% (6.9)	**+14%** (78% / 8.6)	**+14%** (78% /8.6)	**+27%** (87% /13.8)
blade [BCW17]	55% (13.7)	**+44%** (78% / 17.6)	**+47%** (80% /20.0)	**+53%** (83% /24.7)
cow2 [BCW17]	64% (12.6)	**+19%** (76% / 16.3)	**+21%** (77% /17.4)	**+21%** (77% /17.4)
ramses [BCW17]	58% (10.8)	**+29%** (75% / 13.3)	**+42%** (83% /15.7)	**+42%** (83% /15.7)
camel [BCE+13]	53% (23.7)	**+47%** (77% / 30.8)	**+47%** (77% /30.8)	**+56%** (82% /41.8)
aircraft [MPZ14]	58% (18.5)	**+28%** (75% / 21.9)	**+40%** (81% /24.5)	**+45%** (85% /33.3)
santa [MPZ14]	57% (28.5)	**+32%** (76% / 36.9)	**+38%** (79% /40.2)	**+38%** (79% /40.2)
beetle [LZ14]	65% (18.8)	**+20%** (78% / 24.2)	**+20%** (78% /24.2)	**+30%** (84% /35.8)
bozbezbozzel [LZ14]	61% (27.7)	**+19%** (72% / 32.7)	**+23%** (74% /39.4)	**+36%** (83% /55.1)
bird [CZL+15]	30% (9.4)	**+134%** (71% / 11.6)	**+158%** (78% /13.7)	**+178%** (84% /14.5)
duck [CZL+15]	29% (10.7)	**+149%** (73% / 13.7)	**+168%** (79% /15.3)	**+177%** (81% /20.3)
excavator [CZL+15]	30% (9.6)	**+141%** (72% / 12.2)	**+162%** (78% /12.9)	**+162%** (78% /12.9)
jordan [CZL+15]	16% (11.4)	**+237%** (55% / 13.3)	**+379%** (78% /15.9)	**+396%** (81% /21.7)
tower [CZL+15]	38% (10.6)	**+40%** (54% / 12.5)	**+84%** (71% /14.8)	**+104%** (78% /19.4)
bunny [JKS05]	68% (17.6)	**+10%** (74% / 20.8)	**+15%** (77% /23.5)	**+15%** (78% /28.3)
fandisk [JKS05]	61% (17.4)	**+33%** (80% / 22.0)	**+36%** (82% /24.3)	**+36%** (82% /28.2)
rockerarm [JSP17]	42% (12.4)	**+57%** (67% / 14.9)	**+76%** (75% /17.4)	**+92%** (81% /23.3)
venus [JSP17]	59% (5.4)	**+21%** (71% / 7.0)	**+33%** (78% /7.5)	**+41%** (83% /10.5)
Min.	16% / (5.4)	**+10%** (54% /7.0)	**+14%** (71% /7.5)	**+15%** (77% /10.5)
Max.	69% / (28.5)	**+237%** (81% /36.9)	**+379%** (85% /40.2)	**+396%** (87% /55.1)
Average	52% / (14.0)	**+51%** (73% /17.4)	**+67%** (78% /19.6)	**+74%** (82% /24.3)
Median	55% / (12.5)	**+31%** (74% /15.6)	**+41%** (78% /17.5)	**+49%** (82% /22.5)

5 RESULTS

We tested our framework on a range of inputs, shown throughout the paper, and with additional inputs and results included in the supplementary material. We have tested our method on inputs produced via multiple combinations of cutting and parameterization methods: manually unwrapped inputs, seam generation [Sheffer and Hart 2002] followed by parameterization [Sheffer et al. 2005] for the *horse*, *cow* and *feline* models in Figures 15, 16; simultaneous cutting and parameterization [Poranne et al. 2017] for the *elk* and *armadillo* models Figure 15; different global parameterization methods [Bommes et al. 2013; Bright et al. 2017; Levi and Zorin 2014; Lipman 2012; Myles et al. 2014], chartification methods for *bunny* and *fandisk* models in Figures 15, parameterized using [Sheffer et al. 2005]; 2D data sets used for the *Dapper* paper [Chen et al. 2015], shown in Figure 17, and models parameterized using the bijective free-boundary method of Jiang et al. [2017].

Table 2. Boundary length increase through overlap removal (smaller is better), using [Lévy et al. 2002] and using our method. Models from the first section (cut using [Sheffer and Hart 2002]) do not have globally continuous parameterizations, hence no welding is possible. Models from the second section use globally continuous methods [Bommes et al. 2013; Bright et al. 2017; Levi and Zorin 2014; Lipman 2012; Myles et al. 2014].

Model [Method]	BLen Increase (Pack. Eff.)		
	[Lévy et al. 02]	Overl. Cut	Welded
beethoven [SH02]	**+76%** (62%)	**+13%** (59%)	**+13%** (59%)
bunny [SH02]	**+47%** (57%)	**+17%** (62%)	**+17%** (62%)
feline [SH02]	**+58%** (48%)	**+14%** (54%)	**+14%** (54%)
gargoyle [SH02]	**+30%** (54%)	**+15%** (55%)	**+15%** (55%)
aircraft [Lip12]	**+70%** (71%)	**+25%** (68%)	**+7%** (68%)
cup [Lip12]	**+37%** (81%)	**+42%** (71%)	**−13%** (69%)
blade [BCW17]	**+35%** (43%)	**+22%** (60%)	**+16%** (55%)
cow2 [BCW17]	**+70%** (56%)	**+23%** (65%)	**+16%** (64%)
ramses [BCW17]	**+35%** (58%)	**+18%** (56%)	**+18%** (58%)
camel [BCE+13]	**+92%** (61%)	**+26%** (57%)	**+8%** (49%)
aircraft [MPZ14]	**+57%** (64%)	**+12%** (66%)	**+1%** (58%)
santa [MPZ14]	**+58%** (60%)	**+15%** (66%)	**−1%** (61%)
beetle [LZ14]	**+66%** (40%)	**+24%** (66%)	**+14%** (65%)
bozbezbozzel [LZ14]	**+92%** (64%)	**+29%** (65%)	**+15%** (60%)
Min.	**+30%** (40%)	**+12%** (54%)	**−13%** (49%)
Max.	**+92%** (81%)	**+42%** (71%)	**+18%** (69%)
Average	**+59%** (59%)	**+21%** (62%)	**+10%** (60%)
Median	**+58%** (59%)	**+20%** (63%)	**+14%** (60%)

Our framework is agnostic to how the input was generated, and performs equally well on the different data sources (Figures 15, 20). A range of statistics for the models shown in the paper is summarized in Tables 1 and 2. Overall, our method improves output packing efficiency by an average of 54% when the increase in boundary length is capped at 30%, and an average of 74% when boundary length is allowed to double. It achieves the greatest improvement on the *jordan* model. For all data sets shown in table 1, the optimization process took 306 seconds on average with the boundary elongation constrained to be at most 30% (using an i7-3770 CPU at 3.4Ghz and 32GB of RAM). These are comparable to the runtimes of Chen et al. [Chen et al. 2015]. Since packing is an offline process typically done once per asset, we consider our runtimes acceptable for a large target user-base. In our tests, increasing triangle count by a factor of four via subdivision roughly doubled runtime. Our method achieves 25% to 46% PE improvement on models with 140K-200K triangles within a 30 minute time limit (see supplemental material). Time limits allow users to balance time spent against resulting PE. For example, when optimizing the *triceratops* data set, we get packing

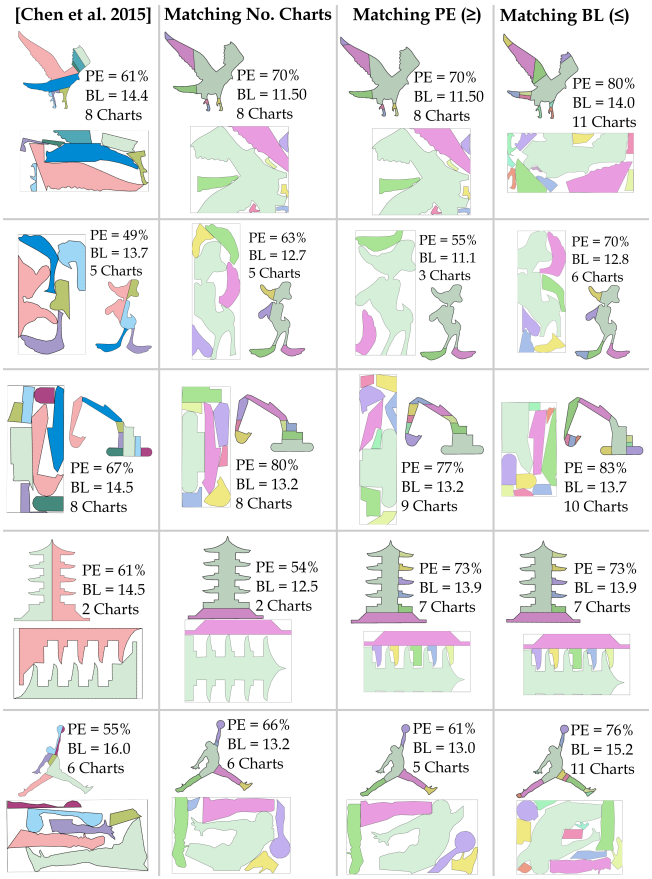


Fig. 17. Comparison results generated by Dapper [Chen et al. 2015](leftmost column) to those generated using method, using as termination criterion the respective Dapper result's number of pieces (second column), packing efficiency (third column), and boundary length (fourth column).

efficiency rates of 74%/83%/86% when limiting execution time to 15s/45s/180s respectively.

Our overlap removal method significantly improves on the state of the art as demonstrated by Figures 5 and 16 and summarized by Table 2. On average, while prior overlap removal methods elongated boundaries by roughly 60% our method resolves overlaps with only 10% elongation. Our method performs best on the outputs of globally continuous parameterization methods, where the initial atlases exhibit very large overlaps and also allow for subsequent welding by leveraging the extra degree of freedom provided by global continuity.

Since a common scenario consists in the use of a single UV atlas for an entire scene, we have also created a single arrangement of all the models shown in Figures 15, 16, which we optimized using Box Cutter - the result (created within 44 minutes) is shown in Figure 22.

Comparison to Prior Art. We have evaluated the performance of our method against the Dapper approach [Chen et al. 2015] which focuses on tradeoffs between packing efficiency and chart count. To allow for a comparison, we have used different termination criteria, running our algorithm until the number of generated charts, resulting boundary length or packing efficiency matches the respective

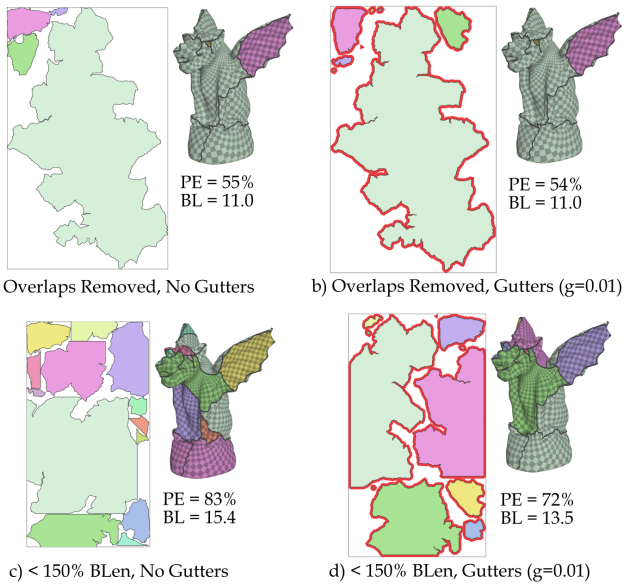


Fig. 18. Using gutter space with our algorithm. An overlap-free input (a-b) is compacted without gutter space (c) and with gutter space (d), using one percent of the normalized bbox side length.). Using gutter space around the chart boundaries decreases packing efficiency and leads to less, and shorter, cuts in the resulting compact atlas.

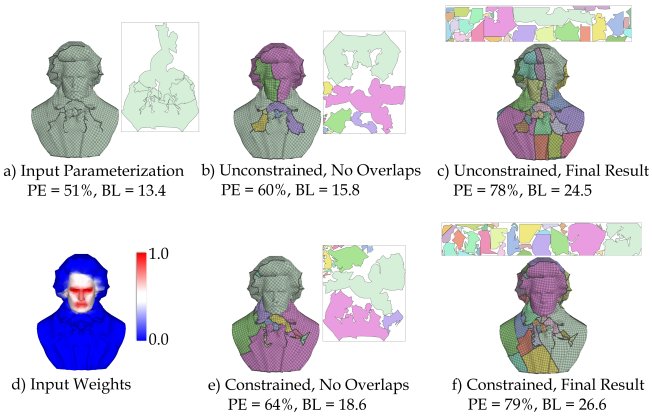


Fig. 19. Importance-weighted cutting. When optimizing an input parameterization (a) by removing overlaps (b) and performing compacting cuts (c), new cuts may cross important regions, such as the facial features in this case. Using importance weights (d), our algorithm is able to protect such regions from cutting during overlap removal (e) and compacting (f).

Dapper results. As can be seen in Fig. 17, our method outperforms their approach in terms of packing efficiency and boundary length for almost all cases. The only exception is the pagoda data set (rightmost column), where Box Cutter obtains lower packing efficiency results when constrained to produce only two charts.

Edge Cutting Modes. In some applications, it may not be desirable to cut triangles exactly with cutlines and retriangulate; in this case, we provide a method which cuts triangles along their edges, following the closest path to a cutline (Section 3.2). Using these paths instead of cutlines, we achieve a similar packing efficiency on

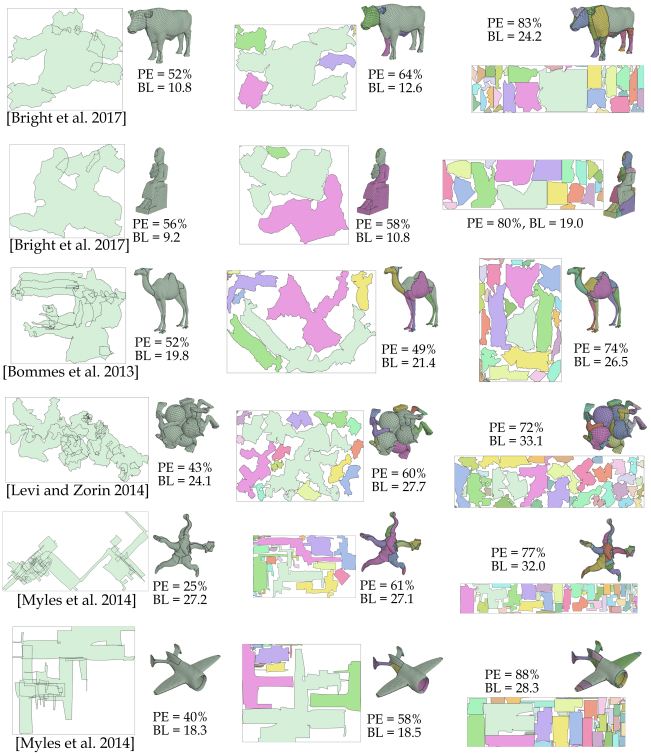


Fig. 20. Results for both stages of our algorithm on various kinds of input models, generated by different parameterization algorithms. From left to right: input parameterization, overlaps removed, compacted result.

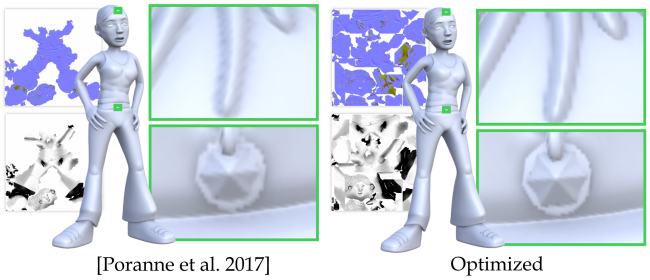


Fig. 21. Fixed-resolution textures (2048 × 2048px), used to store normals and occlusion, applied to the *girl* model (cp. Figure 1). Left: Maps generated using the unoptimized input of [Poranne et al. 2017]. Right: Maps generated using our optimized version. The higher packing efficiency of our parameterization leads to significantly sharper features on the rendered result.

average as the default framework, with only 6% larger increase in boundary length on average (compare BL in Table 1, top vs bottom).

User Control. Our framework can accommodate a range of user preferences. We support different termination criteria as shown in Figure 6. Our framework can penalize cuts in visually important areas, so that artists can redirect cuts away from key feature areas (Figure 19). We can also directly account for the allocation of extra space around the chart boundaries, which is necessary to support gutters for seamless signal storage (Figure 18 and for generating flaps for papercraft (Figure 1, 23).

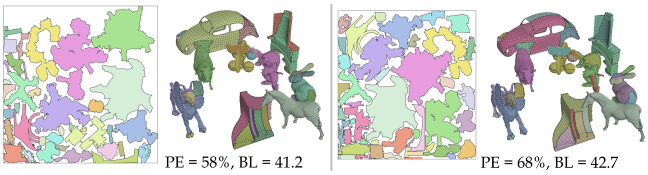


Fig. 22. Optimizing the common UV atlas of a collection of models.

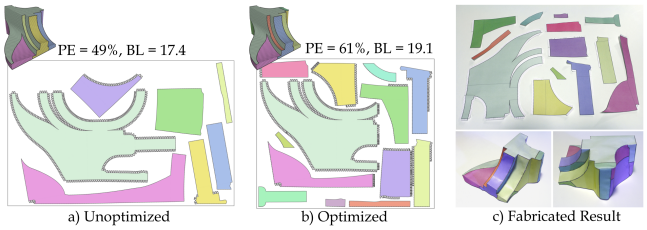


Fig. 23. Papercraft fabrication example. An unoptimized layout (a) is optimized (b) in order to efficiently exploit available material for fabrication (c).

Validation. To validate the impact of our cutlines derived from void boxes, we performed an experiment in which we replaced our candidate cutline set with random vertical and horizontal lines, while keeping the rest of the optimization framework - including ranking of cuts and local search - unchanged. Packing efficiency decreased across our set of all inputs by 15%, which is a significant drop.

Applications. We have tested the two core applications of our methodology - signal storage and fabrication. Figure 21 shows ambient occlusion and normal maps captured from a high-resolution input and baked onto a low-resolution mesh using our atlases. Our more efficient packing allows for a more efficient use of texture space, and consequently a higher quality reproduction of the baked normals and occlusion data, resulting in sharper edges and creases in high-frequency areas. Figures 1 and 23 demonstrate the usability of our method for fabrication. We started from two original atlases containing overlaps, making them unsuitable for fabrication. After overlap removal the atlas packing efficiency was 31% and 49% respectively, which subsequently improved to 61% in both cases, following our optimization. Given a target model size of $11.3 \times 13 \times 7.2$cm, we used a 27.9×43.2cm sheet of paper to create the bunny model using our output atlas. To create this model using the original layout would have required a 56.7×36.9 sheet of paper.

6 CONCLUSION

We have introduced *Box Cutter*, a new method for improving chart packing efficiency suitable for both rendering and fabrication settings. Our method provides users with direct control of the tradeoff between chart boundary length and packing efficiency, and allows for the specification of key parameters such as visual importance, gutter sizes, and minimal acceptable chart size. Additionally, we propose a new algorithm for overlap removal which produce bijective outputs while minimizing output boundary length.

Limitations and Future Work. While our algorithm works in practice, we cannot offer any theoretical guarantees on its outputs. In our worst case model, the bunny, our packing efficiency improvement is

only 14%. In the future, an obvious extension of this approach would be to look at three-dimensional void boxes and cuts for packing 3D objects. We believe this will work in practice, but this has not been tested.

ACKNOWLEDGMENTS

This work was supported by NSERC and by the FITweltweit program of the German Academic Exchange Service (DAAD). We would also like to thank Julian-Alexander Neagu (https://www.artstation.com/julianalexanderneagu) for modeling the sculpted details and providing the baked textures of the *girl* model. Our thanks also go to Chrystiano Araújo for implementing the flaps for our 2D output and creating the papercraft models, as well as to Luciano S. Burla for creating the video. We are deeply grateful to Tobias Nöll for providing results for various packing algorithms. Finally, we would like to thank Tobias A. Franke for his helpful feedback and suggestions.

REFERENCES

Mirela Ben-Chen, Craig Gotsman, and Guy Bunin. 2008. Conformal Flattening by Curvature Prescription and Metric Scaling. *Computer Graphics Forum* 27, 2 (2008), 449–458.

David Bommes, Marcel Campen, Hans-Christian Ebke, Pierre Alliez, and Leif Kobbelt. 2013. Integer-grid Maps for Reliable Quad Meshing. *ACM Trans. Graph.* 32, 4, Article 98 (July 2013), 12 pages.

Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno LÃĪvy. 2010. *Polygon Mesh Processing*. AK Peters. 250 pages.

Alon Bright, Edward Chien, and Ofir Weber. 2017. Harmonic Global Parametrization with Rational Holonomy. *ACM Trans. Graph.* 36, 4, Article 89 (July 2017), 15 pages.

Nathan A. Carr and John C. Hart. 2002. Meshed Atlases for Real-time Procedural Solid Texturing. *ACM Trans. Graph.* 21, 2 (April 2002), 106–131.

Xuelin Chen, Hao Zhang, Jinjie Lin, Ruizhen Hu, Lin Lu, Qixing Huang, Bedrich Benes, Daniel Cohen-Or, and Baoquan Chen. 2015. Dapper: Decompose-and-pack for 3D Printing. *ACM Trans. Graph.* 34, 6, Article 213 (Oct. 2015), 12 pages.

David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. 2004. Variational Shape Approximation. In *ACM SIGGRAPH 2004 Papers (SIGGRAPH '04)*. ACM, New York, NY, USA, 905–914.

Daniel Sanchez-Crespo Dalmau. 2003. *Core Techniques and Algorithms in Game Programming*. New Riders Games.

Michael S. Floater. 1997. Parametrization and Smooth Approximation of Surface Triangulations. *Comput. Aided Geom. Des.* 14, 3 (April 1997), 231–250.

Michael R. Garey and David S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA.

Francisco González and Gustavo Patow. 2009. Continuity Mapping for Multi-chart Textures. *ACM Trans. Graph.* 28, 5 (2009), 109:1–109:8.

Xianfeng Gu, Steven J. Gortler, and Hugues Hoppe. 2002. Geometry Images. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '02)*. ACM, New York, NY, USA, 355–361.

Ziyad S. Hakura and Anoop Gupta. 1997. The Design and Analysis of a Cache Architecture for Texture Mapping. In *Proceedings of the 24th Annual International Symposium on Computer Architecture (ISCA '97)*. ACM, New York, NY, USA, 108–120.

K. Hormann, B. Lévy, and A. Sheffer. 2007. Mesh Parameterization: Theory and Practice. In *SIGGRAPH 2007 Course Notes*. ACM Press, San Diego, CA, vi+115.

Zhongshi Jiang, Scott Schaefer, and Daniele Panozzo. 2017. Simplicial Complex Augmentation Framework for Bijective Maps. *ACM Trans. Graph.* 36, 6 (2017), 186:1–186:9.

M. Jin, Y. Wang, S. T. Yau, and X. Gu. 2004. Optimal global conformal surface parameterization. In *IEEE Visualization 2004*. 267–274.

Dan Julius, Vladislav Kraevoy, and Alla Sheffer. 2005. D-Charts: Quasi-Developable Mesh Segmentation. *Computer Graphics Forum* 24, 3 (2005), 581–590.

B. W. Kernighan and S. Lin. 1970. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal* 49, 2 (Feb 1970), 291–307.

Margret Keuper, Evgeny Levinkov, Nicolas Bonneel, Guillaume Lavoue, Thomas Brox, and Bjorn Andres. 2015. Efficient Decomposition of Image and Mesh Graphs by Lifted Multicuts. In *The IEEE International Conference on Computer Vision (ICCV)*.

Liliya Kharevych, Boris Springborn, and Peter Schröder. 2006. Discrete Conformal Mappings via Circle Patterns. *ACM Trans. Graph.* 25, 2 (April 2006), 412–438.

B. Koo, J. Hergel, S. Lefebvre, and N. J. Mitra. 2017. Towards Zero-Waste Furniture Design. *IEEE Transactions on Visualization and Computer Graphics* 23, 12 (Dec 2017), 2627–2640.

Zohar Levi and Denis Zorin. 2014. Strict Minimizers for Geometric Optimization. *ACM Trans. Graph.* 33, 6, Article 185 (Nov. 2014), 14 pages.

Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérome Maillot. 2002. Least Squares Conformal Maps for Automatic Texture Atlas Generation. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '02)*. ACM, New York, NY, USA, 362–371.

Yaron Lipman. 2012. Bounded Distortion Mapping Spaces for Triangular Meshes. *ACM Trans. Graph.* 31, 4, Article 108 (July 2012), 13 pages.

Songrun Liu, Zachary Ferguson, Alec Jacobson, and Yotam Gingold. 2017. Seamless: Seam erasure and seam-aware decoupling of shape from mesh resolution. *ACM Trans. Graph.* 36, 6 (2017), 216:1–216:15.

Victor J. Milenkovic. 1999. Rotational polygon containment and minimum enclosure using only robust 2D constructions. *Computational Geometry* 13, 1 (1999), 3 – 19.

Jacob Munkberg, Jon Hasselgren, Petrik Clarberg, Magnus Andersson, and Tomas Akenine-Möller. 2016. Texture Space Caching and Reconstruction for Ray Tracing. *ACM Trans. Graph.* 35, 6, Article 249 (Nov. 2016), 13 pages.

Ashish Myles, Nico Pietroni, and Denis Zorin. 2014. Robust Field-aligned Global Parametrization. *ACM Trans. Graph.* 33, 4, Article 135 (July 2014), 14 pages.

Ashish Myles and Denis Zorin. 2012. Global parametrization by incremental flattening. *ACM Trans. Graph.* 31, 4, Article 109 (July 2012), 11 pages.

Tobias Nöll and Didier Stricker. 2011. Efficient Packing of Arbitrarily Shaped Charts for Automatic Texture Atlas Generation. In *Proceedings of the Twenty-second Eurographics Conference on Rendering (EGSR '11)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 1309–1317.

Roi Poranne, Marco Tarini, Sandro Huber, Daniele Panozzo, and Olga Sorkine-Hornung. 2017. Autocuts: Simultaneous Distortion and Cut Optimization for UV Mapping.

Budirijanto Purnomo, Jonathan D. Cohen, and Subodh Kumar. 2004. Seamless Texture Atlases. In *Proc. Symp. Geometry Processing*. 65–74.

Nicolas Ray, Vincent Nivoliers, Sylvain Lefebvre, and Bruno Lévy. 2010. Invisible Seams. In *Proc. Eurographics Conference on Rendering*. 1489–1496.

Pedro V. Sander, Steven J. Gortler, John Snyder, and Hugues Hoppe. 2002. Signal-specialized Parametrization. In *Proc. Eurographics Workshop on Rendering*. 87–98.

Pedro V. Sander, John Snyder, Steven J. Gortler, and Hugues Hoppe. 2001. Texture Mapping Progressive Meshes. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)*. ACM, New York, NY, USA, 409–416.

P. V. Sander, Z. J. Wood, S. J. Gortler, J. Snyder, and H. Hoppe. 2003. Multi-chart Geometry Images. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing (SGP '03)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 146–155.

Alla Sheffer and John C. Hart. 2002. Seamster: Inconspicuous Low-distortion Texture Seam Layout. In *Proceedings of the Conference on Visualization '02 (VIS '02)*. IEEE Computer Society, Washington, DC, USA, 291–298.

Alla Sheffer, Bruno Lévy, Maxim Mogilnitsky, and Alexander Bogomyakov. 2005. ABF++: Fast and Robust Angle Based Flattening. *ACM Transactions on Graphics* (Apr 2005).

Alla Sheffer, Emil Praun, and Kenneth Rose. 2007. Mesh Parameterization Methods and Their Applications. *Foundations and TrendsÂő in Computer Graphics and Vision* 2, 2 (2007), 105–171.

Jason Smith and Scott Schaefer. 2015. Bijective Parameterization with Free Boundaries. *ACM Trans. Graph.* 34, 4, Article 70 (July 2015), 9 pages.

Olga Sorkine, Daniel Cohen-Or, Rony Goldenthal, and Dani Lischinski. 2002. Bounded-distortion Piecewise Mesh Parameterization. In *Proceedings of the Conference on Visualization '02 (VIS '02)*. IEEE Computer Society, Washington, DC, USA, 355–362.

Kun Zhou, John Synder, Baining Guo, and Heung-Yeung Shum. 2004. Iso-charts: Stretch-driven Mesh Parameterization Using Spectral Analysis. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing (SGP '04)*. ACM, New York, NY, USA, 45–54.

Yahan Zhou, Shinjiro Sueda, Wojciech Matusik, and Ariel Shamir. 2014. Boxelization: Folding 3D Objects into Boxes. *ACM Trans. Graph.* 33, 4, Article 71 (July 2014), 8 pages.