Autocuts:
Simultaneous Distortion and Cut Optimization for UV Mapping

ROI PORANNE, ETH Zurich
MARCO TARINI, CNR-ISTI and Univ. Insubria
SANDRO HUBER, ETH Zurich
DANIELE PANOZZO, New York University
OLGA SORKINE-HORNUNG, ETH Zurich

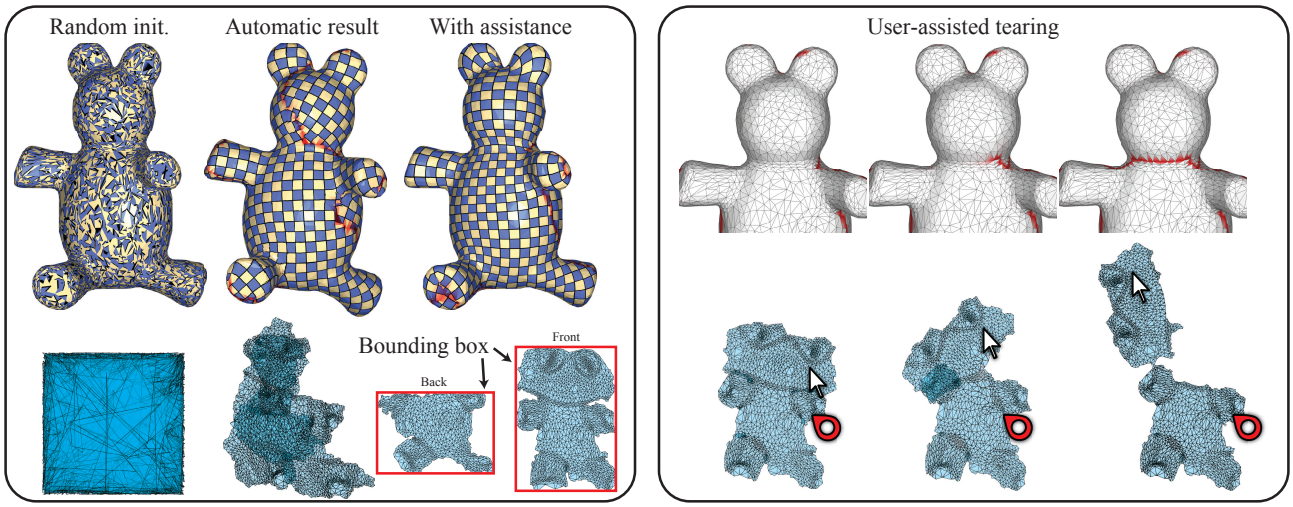


Fig. 1. Our method produces high quality UV maps, balancing the number of seams and the distortion of the map. The method runs at interactive rates and can provide artists with complete control over the UV map if desired. We offer interactive tools such as bounding boxes for packing islands (left), painting of regions that attract or discourage seam creation, and semi-automatic removal of overlapping regions. Each tool provides interactive feedback, drastically simplifying the design of complex UV maps.

We propose a UV mapping algorithm that jointly optimizes for cuts and distortion, sidestepping heuristics for placing the cuts. The energy we minimize is a state-of-the-art geometric distortion measure, generalized to take seams into account. Our algorithm is designed to support an interactive workflow: it optimizes UV maps on the fly, while the user can interactively move vertices, cut mesh parts, join seams, separate overlapping regions, and control the placement of the parameterization patches in the UV space. Our UV maps are of high quality in terms of both geometric distortion and cut placement, and compare favorably to those designed with traditional modeling tools. The UV maps can be created in a fraction of the time as existing methods, since our algorithm drastically alleviates the trial-and-error, iterative procedures that plague traditional UV mapping approaches.

This work is partially supported by the European Research Council under Grant No.: StG-2012-306877 (ERC Starting Grant iModel), NSF CAREER award (1652515), PRIN project "DSURF" (2015B8TRFM), and a gift from Adobe Research.

0730-0301/2017/11-ART215 $15.00
https://doi.org/10.1145/3130800.3130845

CCS Concepts: • **Computing methodologies** → **Computer graphics**; **Shape modeling**;

Additional Key Words and Phrases: UV mapping, parameterization, cuts, distortion minimization

ACM Reference format:
Roi Poranne, Marco Tarini, Sandro Huber, Daniele Panozzo, and Olga Sorkine-Hornung. 2017. Autocuts: Simultaneous Distortion and Cut Optimization for UV Mapping. *ACM Trans. Graph.* 36, 6, Article 215 (November 2017), 11 pages.
https://doi.org/10.1145/3130800.3130845

1 INTRODUCTION

UV maps are ubiquitously used in computer graphics to map regularly sampled 2D data, such as colors, normals, or displacements, onto surfaces embedded in 3D. The design of UV maps has received extensive attention in the research community in the last three decades. It is traditionally divided into two steps: the computation of optimal *cuts*, also called seams, and the minimization of the distortion of the resulting *patches* as they are mapped onto the plane. The two sub-problems are very different in nature: the former is discrete and combinatorial, since the cuts are selected from a discrete set

of mesh edges, and the latter is continuous, since the distortion depends on the UV coordinates of the mesh vertices. The cuts are typically computed first and heavily affect the quality of the resulting parameterization; if the quality of the UV map is not sufficiently good for the intended application, the user must alter the cuts and re-run the parameterization computation, possibly repeating this trial-and-error loop several times.

We present the first approach to jointly optimize cuts and geometric distortion. Our algorithm minimizes state-of-the-art distortion measures at interactive rates while identifying an adequate set of mesh edges to cut. While previous methods optimize for either cuts or distortion alone, our formulation automatically introduces cuts and minimizes distortion within the same energy optimization process. The key idea of our algorithm is to model the UV map by parameterizing each triangle individually, using an attraction energy to encourage the parameterization to be continuous over matching edge pairs. A sparsity inducing norm over the attraction energy results in only a few edge pairs not merging, and these are the seams of the final UV map. Intuitively, this corresponds to a relaxation of the classical binary notion of seams: instead of tagging each edge as either a seam or a regular edge, we allow it to be something in-between during the optimization. As the algorithm converges, the edges come closer and closer to either becoming seams or regular edges, until they are close enough to be "snapped" to one of the two binary choices.

While our algorithm can already be useful in automatic mode, its advantage is supporting the interactive workflow that is preferred by digital artists: our optimization runs in the background, while the user can interactively move vertices, cut mesh parts, join seams, remove self-overlapping regions and pack the resulting patches in the UV space (see Fig. 1 and the accompanying video). All such constraints can be either hard, overriding the energy-minimizing decisions favored by our algorithm, or soft, steering the optimization in the desired direction.

We expect our algorithm to impact the design of UV maps, since it eliminates the frustrating trial-and-error pipeline that is currently commonplace. Furthermore, it offers an interactive interface that enables artists to manually specify the semantically relevant portions of the map, while automatically handling the remaining optimization in the background. We provide a reference and open-source implementation of our method (see Section 4) to ensure replicability of our results and encourage integration into existing tools.

2 RELATED WORK

Surface parameterization, or UV map construction, is widely covered in the literature [Hormann et al. 2007; Sheffer et al. 2006]. The subproblems of computing the cuts and minimizing distortion have been extensively studied – however, mostly separately.

2.1 Minimizing distortion

Most applications require the parameterization mapping to be as isometric as possible, i.e., preserve angles and areas as much as possible. A large number of distortion measures have been proposed to quantify the discrepancy between a given mapping and an isometry. A chosen distortion measure is typically optimized by the UV map over the space of per-vertex UV assignments.

Conformal distortion measures, which only penalize angular distortion, can be expressed as quadratic functions of the UV variables [Desbrun et al. 2002; Lévy et al. 2002], leading to a sparse least-squares problem, which is very efficient to solve. Unfortunately, distortion measures that also consider area (e.g. [Hormann and Greiner 2000; Liu et al. 2008; Sander et al. 2001]) are nonlinear and non-convex, and hence challenging to minimize. This has elicited the search for good algorithmic and numeric solutions tailored to this specific purpose.

Among the most successful approaches, local/global optimization [Liu et al. 2008] alternates between computing a perfectly isometric but discontinuous mapping per face, and stitching it by global optimization to create a continuous but no longer isometric mapping. Our approach borrows the fruitful idea of assigning separate per-corner variables to each vertex, one for each incident triangle, effectively detaching each triangle in UV space.

Recent works formulate sophisticated and highly non-convex objective functions that enable finding nearly isometric and inversion-free parameterizations by continuous optimization, which works by minimizing an iteratively updated convex proxy [Kovalsky et al. 2016; Rabinovich et al. 2017; Smith and Schaefer 2015]. We propose to take another leap forward and include cut optimization together with a state-of-the-art distortion measure, in the same objective function. This is far from trivial because cut optimization and distortion minimization have drastically different behaviors: the latter tends to spread the error equally over the surface, while the former must concentrate the error in a discrete set (the cut vertices) and leave it at zero everywhere else, due to the binary nature of cutting. Hence, intuitively, an objective function that accommodates both cuts and low distortion must be extremely non-convex.

2.2 Optimizing cuts

The task of automatic computation of cuts is still an open problem despite extensive research on mesh parameterization. Many parameterization methods accept cuts as input and compute the mapping of a surface that has the topology of one or several disks. Earlier methods require the shape of the cut in the UV domain to be specified as well (e.g. [Floater 2003]). More recent free-boundary methods optimize the shape of the whole 2D patch, including the boundary (e.g. [Desbrun et al. 2002; Lévy et al. 2002; Liu et al. 2008; Sheffer et al. 2005]). Current work succeeds to avoid local and global self-intersections while leaving the boundary free and minimizing an isometric distortion measure [Smith and Schaefer 2015].

In several works, cuts are identified as part of the parametrization process. They can be roughly categorized into three different groups, detailed below.

Cuts for remeshing applications. A large portion of recent parameterization literature tends to focus on quadrilateral remeshing applications, while we are interested in texture mapping. In remeshing oriented parameterization, cut placement is largely inconsequential, as long as the two sides of every cut match in the UV space up to a rotation by an integer multiple of $\pi/2$ and an integer number of translation steps. A cut with this property is sometimes called "griddable", as it can be seamlessly traversed by a grid, and the parameterization is then termed "seamless". The problem of good cut

placement is therefore often disregarded in remeshing (see, e.g., [Bommes et al. 2009] and the survey [Bommes et al. 2013]). The placement of cone singularities, which can be understood as endpoints of griddable cuts, is still crucial, and can be the subject of a separate optimization [Ben-Chen et al. 2008; Myles and Zorin 2013]. Some works optimize the positions of the griddable cuts on the mesh (starting from the cones) for attributes like straightness or field alignment, and shortness [Campen and Kobbelt 2014; Tarini et al. 2011]. The interactive design of the singularity graph [Takayama et al. 2013; Tong et al. 2006] can also be seen as a special way of prescribing griddable cuts. Unfortunately, in the context of traditional texture mapping applications, griddable cuts are less useful and impose unnecessary constraints that are only necessary for the case of quadrilateral meshing. Although they can offer certain benefits for specific types of texture mapping [Ray et al. 2010], griddable cuts are not strictly required and hardly ever used, e.g., in the game industry. One key reason is that general cuts, but not griddable ones, serve the important purpose of trading Gaussian curvature inside patches for line curvature at their boundary, reducing distortion. This consideration is the basis for the algorithms described below.

Adding cuts while minimizing distortion. Many methods generate new cuts as a means to counter excessive local distortion. One way is to alternate between minimizing a distortion energy and greedily placing additional cuts every time the distortion exceeds a certain threshold, see e.g. [Gu et al. 2002; Sorkine et al. 2002]. An alternative is to first identify good candidate locations for cuts by analyzing the Gaussian curvature and visibility, see e.g. [Ben-Chen et al. 2008; Sheffer and Hart 2002]. All these methods attempt to balance distortion and cuts as part of the same iterative procedure; however, the cut placement relies on heuristics without directly being part of a global optimization of the map, unlike in our approach.

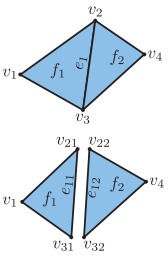
Generating cuts by mesh partitioning. Many parameterization methods developed in the course of over three decades partition the initial mesh into separate patches (i.e., separate connected components) with disk topology, thereby defining cuts as boundaries of the partitions. The partitioning can be guided by a variety of considerations, for example by clustering triangles according to features [Zhang et al. 2005], normals [Maillot et al. 1993] or developability criteria [Julius et al. 2005], seeding and expanding regions [Lévy et al. 2002], computing a centroidal Voronoi tessellation on the surface [Boier-Martin et al. 2004], simplifying the mesh and using the polygons of the base mesh to define charts [Khodakovsky et al. 2003], tracing cuts following a cross field [Campen and Kobbelt 2014], or abiding to a curved skeleton of the surface [Usai et al. 2015] (each item in this list is represented by one citation of many). While the choices behind these approaches are all inspired by valid, general considerations regarding their effect on the final parameterization quality, the *a priori* assumptions are indirect and cannot be expected to lead to optimal UV maps. Typically, the patches tend to be smaller and more numerous than desirable. An additional limitation is that by construction, cuts can only appear between elements belonging to different patches. This imposes an artificial constraint, which is often violated in many good cut layouts, such as the typical manually designed ones, where cuts are free to appear also between polygons belonging to the same patch.

Bypassing the need for cuts. A long lived trend in the context of research on texture mapping strives to bypass rather than solve the task of identifying good cuts [Tarini et al. 2017], either by providing a complete substitute for 2D parametrization and 2D texture mapping altogether [Benson and Davis 2002; Burley and Lacewell 2008; Christensen and Batali 2004; Lefebvre and Hoppe 2006; Tarini et al. 2004; Yuksel et al. 2010], or by diminishing some of the detrimental effects of cuts, thus making their positioning less crucial or even inconsequential [Lefebvre and Dachsbacher 2007; Purnomo et al. 2004; Ray et al. 2010; Tarini 2012, 2016]. All these methods provide new perspectives and tradeoffs. At the same time, the industry almost unanimously sides with the traditional format of UV and texture maps, probably due to a combination of benefits that they offer, which are currently not matched in full by any of the alternatives. Aside familiarity, these include, in terms of content creation: generality, full adaptability, fruitful analogy between texture and traditional 2D images (exploited in a variety of ways, like direct painting on 2D textures), and in terms of rendering: simplicity, time and resource efficiency, paramount GPU-friendliness, the ability to apply filters and pre-filtering (MIP-mapping). For all these reasons, we believe that the problem of defining good cuts in UV mappings remains highly relevant.

Existing tools for manual cut placement. In the graphics related industry, the task of producing UV mappings is invariably computer assisted rather than fully automatic; distortion minimization is successfully delegated to automatic algorithms, while the placement of cuts usually requires manual intervention by trained digital artists. Specialized interfaces and sophisticated interactive tools are provided by all modern modeling packages, yet the task remains time consuming (see the accompanying video). If the input mesh is irregular, the difficulty increases because the UV mapper cannot quickly select consecutive edges (so-called edge loops) to elect as cuts. Our method offers an improvement, as it removes the necessity of full manual cut selection and does not rely on regularity of the mesh connectivity.

3 UV OPTIMIZATION METHOD

One goal of UV mapping is to parameterize a mesh with minimal distortion and minimal length of cuts. One extreme is to do the minimal cuts necessary to induce disk topology, which results in highly distorted triangles useless for texture mapping applications, and the other extreme is placing cuts on all internal edges, such that each triangle is mapped without any distortion [Burley and Lacewell 2008]. In this work, we propose a method to interactively explore the solution space.



In our formulation, we treat the original mesh as a triangle *soup*, where each triangle is separated from the others. Every interior edge is duplicated and appears as two copies in the triangle soup. Each vertex is duplicated several times according to its degree, and the copies are called *corners*. We denote the vertices, edges and faces of the original mesh by v_i, e_j, f_k, and use the subscripts i, j, k to refer to vertex-, edge- and face-related quantities. For instance, A_k is the area of face f_k, and l_j is the

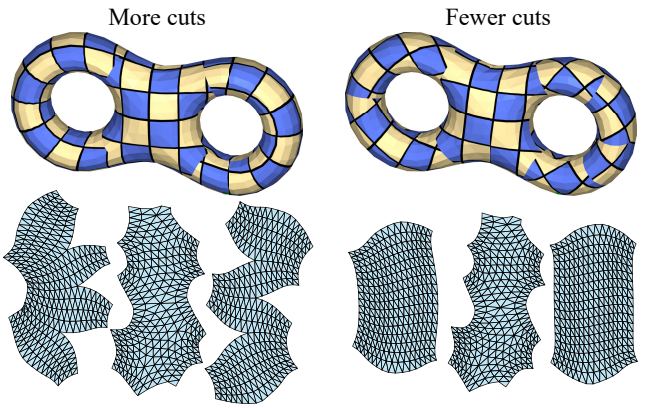


Fig. 2. Balancing between cuts and distortion. The example on the left shows a less distorted parameterization at the cost of more cuts, i.e. discontinuities. The example on the right is more distorted, but also more continuous.

length of edge e_j. We use mixed subscripts to identify edge copies and corners in the triangle soup, e.g., e_{jk} refers to the copy of edge e_j that is incident to face f_k, and v_{ik} refers to the corner vertex originating from v_i and residing in face f_k. The variables in our optimization are the positions of the corners in UV space, denoted by $\mathbf{X} = \{\mathbf{x}_{ik} = (U_{ik}, V_{ik})\}$. Finally, we use $f_k(\mathbf{X})$ to refer to all the variables related to f_k, meaning the three corner UVs $\mathbf{x}_{ik}$ of f_k (and similarly, $e_j(\mathbf{X}), e_{jk}(\mathbf{X})$, etc.).

A parameterization of a triangle soup is a mapping of each triangle to the plane. A cut occurs when two copies of an edge are separated and do not completely coincide. In the broadest sense, our approach consists in solving the following optimization problem:

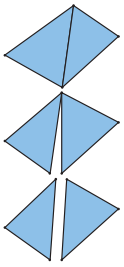
$$\min_{\mathbf{X}} \mathrm{E}(\mathbf{X}) = \min_{\mathbf{X}} \ (1-\lambda)\,\mathrm{D}(\mathbf{X}) + \lambda\,\mathrm{S}(\mathbf{X}), \tag{1}$$

where E is the total objective function, D is a distortion objective (such as preservation of angles or areas), S is a separation objective, and the parameter λ controls the balance between the two objectives (Fig. 2). The two objectives are defined as a weighted sum over the faces and edges, respectively:

$$\mathrm{D}(\mathbf{X}) = \sum_{\text{face } k} A_k\,\mathrm{E_d}(f_k(\mathbf{X}))\,,\ \mathrm{S}(\mathbf{X}) = \sum_{\text{edge } j} l_j\,\mathrm{E_s}\big(e_j(\mathbf{X})\big)\,, \tag{2}$$

where $\mathrm{E_d}$ is a triangle distortion measure and $\mathrm{E_s}$ is an edge separation measure. While distortion measures have been thoroughly discussed in previous work (Sec. 2), there is less relevant work related to separation measures. Identifying an appropriate such measure is one of the main challenges of this work. We discuss our choice next and briefly address distortion measures afterward.

The separation measure. Given two copies of an edge positioned in UV space, our goal is to find a meaningful measure of their separation. A somewhat similar problem has been discussed in the field of discontinuous Galerkin methods [Cockburn et al. 2000], where nonconforming (i.e., discontinuous) finite elements are used. Babuška and Zlámal [1973] proposed to measure the L_2 norm of the jump along the edge. A more modern approach uses the notion of *numerical flux*, which is usually defined as a combination of the integral of the average and the difference of a function along the edges [Arnold et al. 2001]. Recently, another separation measure appeared in [Fu and Liu 2016], utilized for computing bijective maps. This measure, called an *edge-assembly constraint*, is essentially the norm of the difference between the two edge vectors.



In our setting, we observe that an edge can have one of three distinct states when considering cuts: It can be uncut, half-cut (where one of the endpoint vertices is split but the other is not), and fully cut (see inset). The measures described above cannot distinguish between fully cut and half-cut states. Instead, we propose to consider each pair of corners at each of the edge endpoints *independently*. That is, the separation measure includes both distances between the two pairs of endpoints. More precisely, let f_{k_1} and f_{k_2} be two faces that share an edge e_j with two endpoints $v_i, v_{i'}$. We define a corner separation measure applied to both pairs $v_{ij}(\mathbf{X})$ and $v_{i'j}(\mathbf{X})$, i.e., the two pairs of corners at the endpoints of e_j:

$$\begin{aligned} \mathrm{E_s}(v_{ij}(\mathbf{X})) &= s\left(\left\|\mathbf{x}_{ik_1} - \mathbf{x}_{ik_2}\right\|\right), \\ \mathrm{E_s}(v_{i'j}(\mathbf{X})) &= s\left(\left\|\mathbf{x}_{i'k_1} - \mathbf{x}_{i'k_2}\right\|\right), \end{aligned} \tag{3}$$

where $s(\cdot)$ is a monotonic weight function defined later. We then rewrite the problem in Eq. (2) as follows:

$$\begin{aligned} &\text{minimize for } \mathbf{X} \quad \mathrm{E}(\mathbf{X}) = \\ &= (1-\lambda) \underbrace{\sum_{\text{face } k} A_k\,\mathrm{E_d}(f_k(\mathbf{X}))}_{\mathrm{D}(\mathbf{X})} + \lambda \underbrace{\sum_{\{i,k_1,k_2\}\sim j} l_j\, s\left(\left\|\mathbf{x}_{ik_1} - \mathbf{x}_{ik_2}\right\|\right)}_{\mathrm{S}(\mathbf{X})}, \end{aligned} \tag{4}$$

where the second sum is over all corners v_{ij} and faces f_{k_1}, f_{k_2} incident to edge e_j.

We define $s(\cdot)$ to achieve two goals: (i) we want to have as few cuts as possible, and (ii) if an edge is cut, there is no reason to encourage its two copies to remain spatially close to each other in the UV domain. The ideal function that satisfies these desiderata is:

$$s(t) = \begin{cases} 0, & t = 0 \\ 1, & \text{otherwise.} \end{cases}$$

For this choice for s, Eq. (4) is an L_0-regularized optimization problem, which is notoriously difficult to solve directly. In the following we discuss our approach for solving it using homotopy optimization.

The distortion measure. Since most texture mapping applications favor isometry, we use the *symmetric Dirichlet* energy [Smith and Schaefer 2015] for all results in this paper, defined by

$$\mathrm{E_d}(f_k(\mathbf{X})) = \|\mathbf{J}_k(f_k(\mathbf{X}))\|_F^2 + \left\|\mathbf{J}_k^{-1}(f_k(\mathbf{X}))\right\|_F^2, \tag{5}$$

where $\mathbf{J}_k(f_k(\mathbf{X}))$ is the Jacobian of the mapping on face f_k and $\|\cdot\|_F$ is the Frobenius norm. This measure is indeed minimal for rotations, and it has several additional favorable properties. First, it infinitely penalizes inverted elements, preventing them. Second, it is computationally efficient since it requires rather few elementary operations to evaluate. If required by specific applications, other energies, such as the conformal AMIPS [Fu et al. 2015], could be easily minimized by our algorithm with minimal modifications.

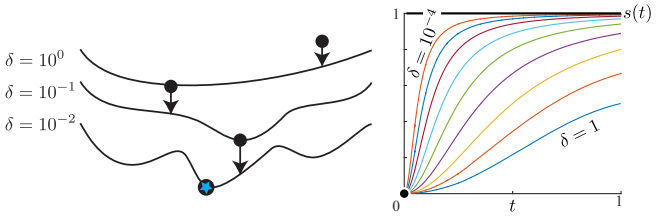


Fig. 3. Left: Homotopy optimization. The objective function E is first radically smoothed to obtain E_δ (top); it is then iteratively sharpened back, and a new minimum is found starting the optimization in the previous minimizer. As δ becomes smaller, E_δ approaches the original E and the found minima converge to a minimum of E (bottom). Right: Plots of our smooth approximations $s_\delta(t)$ (colored curves) and the discontinuous function $s(t)$ (in black) used in the separation measure. As δ approaches zero, $s_\delta(t)$ sharpens and approaches $s(t)$.

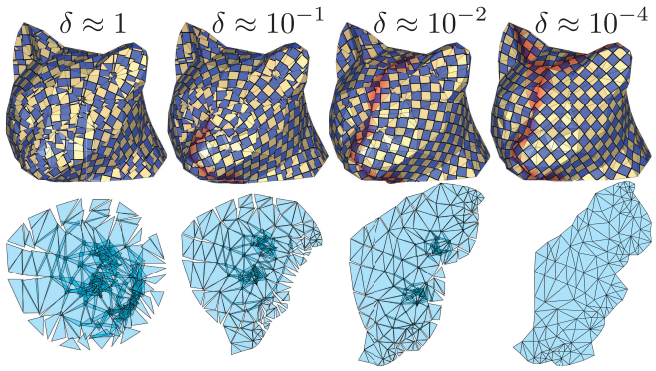


Fig. 4. The influence of the parameter δ on the generated UV map. Red hue on the 3D mesh marks edges with high separation energy.

3.1 Homotopy optimization

With the choices of separation and distortion energies above, Eq. (4) becomes non-smooth and non-convex. It is thus, difficult and computationally expensive to minimize directly. Nevertheless, practical algorithms such as iteratively reweighted least-squares [Daubechies et al. 2010], L_1 relaxation [Boyd and Vandenberghe 2004], and the homotopy optimization method perform well for such an energy. We use an approach similar to the latter, as explained below.

The homotopy optimization technique, also known as the continuation method [Allgower and Georg 2003], is a technique for solving highly non-convex problems. Similar to methods such as graduated optimization and deterministic annealing [Rose 1998], it eliminates many local minima by smoothing the objective function, then gradually sharpening it back [Mobahi and Fisher 2015]. In each iteration, the algorithm finds a local minimum, using the previous minimum as the starting point (see the sketch in Fig. 3, left). Specifically, given an objective $E(\mathbf{X})$, the method requires a family of functions $E_\delta(\mathbf{X})$, for a parameter $\delta > 0$ controlling the smoothness, such that $\lim_{\delta\to 0} E_\delta(\mathbf{X}) = E(\mathbf{X})$; δ is updated in each iteration according to a certain strategy (see later), until convergence. See Algorithm 1 below.

Algorithm 1: Homotopy Optimization

Input:
$E_\delta(\mathbf{X})$, $\delta > 0$, such that $\lim_{\delta\to 0} E_\delta(\mathbf{X}) = E(\mathbf{X})$

$\delta \leftarrow \delta_0$
$\mathbf{X}^{(0)} \leftarrow \arg\min_{\mathbf{X}} E_\delta(\mathbf{X})$
$n \leftarrow 1$
while *criterion not reached* **do**
  Update δ according to strategy
  $\mathbf{X}^{(n)} \leftarrow \arg\min_{\mathbf{X}} E_\delta(\mathbf{X})$ using $\mathbf{X}^{(n-1)}$ as initialization
  $n \leftarrow n + 1$

Seamless L_0 penalty. Since the problematic non-convexity of Eq. (4) stems from the definition of $s(t)$, we need to find an appropriate family of functions $s_\delta(t)$ which smoothly approximate it. Li et al. [2012] introduce such a smooth approximation, termed the *seamless L_0 penalty*. Inspired by their proposal, we define the following approximation (Fig. 3, right):

$$s_\delta(t) = \frac{t^2}{t^2 + \delta}. \tag{6}$$

The argument t is squared so that $s_\delta(t)$ is differentiable at zero. It is easy to verify that $\lim_{\delta\to 0} s_\delta(t) = s(t)$; although the functions $s_\delta(t)$ are not globally convex, they are convex within a certain distance from the origin (defined by a closed-form formula) and are significantly easier to minimize than the discontinuous $s(t)$.

The parameter δ. The standard, fully automatic version of continuation method offers guarantees of convergence to a global minimum only under certain conditions on E_δ and on the strategy to update δ [Allgower and Georg 2003]; instead, we opt for an interactive version where δ is left as a parameter that users can manipulate to steer the optimization towards the goal they have in mind. The influence of δ on the resulting UV map is demonstrated in Fig. 4. In a basic workflow, the user starts with a relatively high δ; in this situation, all edges are cut, but corner pairs still stay close together. The result is in many cases very similar to single-patch parameterization (if the corners were to be snapped back together). As the user decreases δ, some of the corner pairs remain close, but others start to separate, generating a visible cut. Further decreasing δ lumps close corners together, making the map continuous along their edges, while the remaining corners stop influencing each other.

3.2 Numerical optimization

To solve Eq. (4), we use Newton's method with line search. In each iteration n, where the current iterate is $\mathbf{X}^{(n)}$, we solve for the Newton search direction $\mathbf{p}^{(n)}$:

$$\mathrm{H}_{\mathrm{E}}^{(n)}\,\mathbf{p}^{(n)} = -\mathrm{g}_{\mathrm{E}}^{(n)}, \quad \text{i.e.,} \tag{7}$$

$$\left((1-\lambda)\mathrm{H}_{\mathrm{D}}^{(n)} + \lambda \mathrm{H}_{\mathrm{S}}^{(n)}\right)\mathbf{p}^{(n)} = -(1-\lambda)\,\mathrm{g}_{\mathrm{D}}^{(n)} - \lambda\,\mathrm{g}_{\mathrm{S}}^{(n)}, \tag{8}$$

where $\mathrm{g}_{\mathrm{E}}^{(n)}, \mathrm{g}_{\mathrm{D}}^{(n)}, \mathrm{g}_{\mathrm{S}}^{(n)}$ and $\mathrm{H}_{\mathrm{E}}^{(n)}, \mathrm{H}_{\mathrm{D}}^{(n)}, \mathrm{H}_{\mathrm{D}}^{(n)}$ are the gradients and Hessians of the total energy E, distortion energy D and separation

energy S at $\mathbf{X}^{(n)}$, respectively. We then obtain the next iterate by

$$\mathbf{X}^{(n+1)} = \mathbf{X}^{(n)} + \alpha\, \mathbf{p}^{(n)}, \tag{9}$$

where step-size α is found with a line search; we use the strategy by Smith and Schaefer [2015] to ensure that no triangle gets flipped.

To find the gradients and Hessians of D($\mathbf{X}$), we iterate over the faces and accumulate the per-face gradients and modified Hessians using [Shtengel et al. 2017]. This ensures that the overall modified Hessian is positive definite while avoiding costly SVD computations of each face Hessian, as was done in [Fu and Liu 2016]. For the separation function S($\mathbf{X}$), we derive closed-form expressions for the gradient and positive modified Hessian of s_δ in Appendix A.

4 FRAMEWORK IMPLEMENTATION AND RESULTS

Implementation and performances. We ran our experiments on a 12-core Xeon clocked at 2.7 GHz, using the Pardiso solver [Kuzmin et al. 2013; Schenk et al. 2008, 2007] for the linear system solve in Eq. (8). For the mesh sizes we experimented with, performances scale roughly linearly, one iteration requiring 15 ms for 3K triangles, and about 100 ms for 20K triangles. A reference, open-source implementation is available at https://github.com/Roipo/Autocuts.

Parameters tuning. The optimization process runs indefinitely in the background, until the user is ready to save the result. We let the user interactively control the parameters δ and λ. Parameter λ is initialized at 0 and controlled freely in the full $[0, 1]$ range. δ is initialized at 1 and modified by halving or doubling it.

Initialization. Since we treat the input mesh as a triangle soup, initialization of the UV map can simply be done by placing the triangles individually in the plane with a rigid transformation: our distortion energy is rotation invariant, so the specific transformation used does not affect its minimum. Our method is extremely robust to different initializations, as demonstrated in Fig. 5.

Finalizing the UV map. Once the user is satisfied with the result, we generate a UV map that is continuous, except across the seams. To achieve this, we iterate over all pairs of corners and unify them at their average if their separation energy is lower than 0.5 (see for example Fig. 5 and 7). The UV map is also uniformly scaled and translated to fit into the canonical $[0, 1]^2$ texture space.

Unassisted cutting. Our method can achieve high quality results with no user input, or very little input in the form of λ and δ values (see Fig. 17 for examples of obtained UV maps). In contrast, professional artists must provide a lot of manual input in state-of-the-art software in order to cut a mesh for UV mapping. To the best of our knowledge, the most advanced commercial tool for UV mapping is ZBrush's *UV Master* [Pixologic 2017]. This tool allows the user to paint regions where cuts should appear or be avoided, and then provides a map; the user is then tasked with repainting the regions until the desired result is obtained. From our experiments with this tool, we suspect that the cuts are computed *a priori*, before the cut mesh is parameterized. We compare our method in unassisted mode against *UV Master* with substantial user input on the Bumpy Cube in Fig. 6: our method properly balances between cuts and distortions, and places the cuts along the "edges" of the cube, while the ZBrush result seems somewhat random. See Fig. 17 for more results.

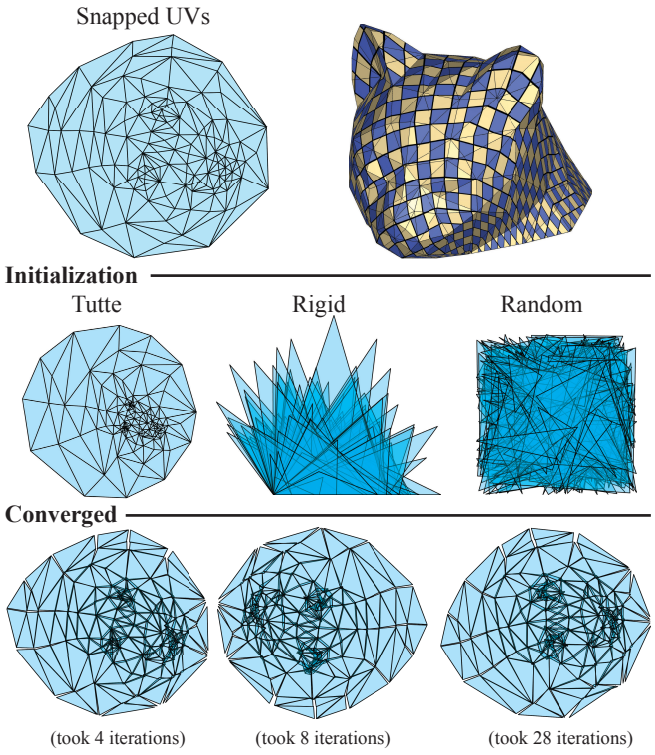


Fig. 5. Our method is extremely robust to bad initialization. The results obtained starting from Tutte (left), a rigid transformation of each triangle onto the UV plane (middle), and a completely random set of positions (right) are almost identical (up to a rotation and translation). For all results in the paper, we used the method in the middle, due to its efficiency and simplicity.

Assisted cutting. In many cases, the user wishes to create (or avoid) cuts in specific regions. We enable this by letting the user select the edges whose weight should be modified in the separation objective, either by clicking on them or painting. Encouraging cuts is simply implemented by reducing (resp. increasing) the weight of the relevant terms for the two pairs of corners in the optimization problem in Eq. (4). We observe that in many cases, it is sufficient to seed a cut along a single edge, and the rest of it propagates on its own during the optimization (see Fig. 7). Additionally, we allow the user to deform, tear, and attach the UV island simply by dragging (see, e.g., Fig. 1 and the attached video).

Bounding rectangles. In many situations, it is useful to require the UV shape to be bounded by a certain UV rectangle. This can improve texture space efficiency and aid packing. We enable interactive drawing and manipulation of bounding rectangles in UV space, and we add a *soft* constraint to the objective function, as a "box" energy B penalizing UV corner positions outside the relevant rectangle R. We add B to our objective in (4), with a certain weight β (in our system we used $\beta = 100$). Specifically, R is defined as the Cartesian product of two given intervals, $R = [\, \underline{u}\,,\, \overline{u}\,] \times [\, \underline{v}\,,\, \overline{v}\,]$, and B is the sum, over all affected UV positions $\mathbf{x} = (u, v)$, of

$$c(\, \underline{u}\,,\, u\,) + c(\, u\,,\, \overline{u}\,) + c(\, \underline{v}\,,\, v\,) + c(\, v\,,\, \overline{v}\,) \tag{10}$$

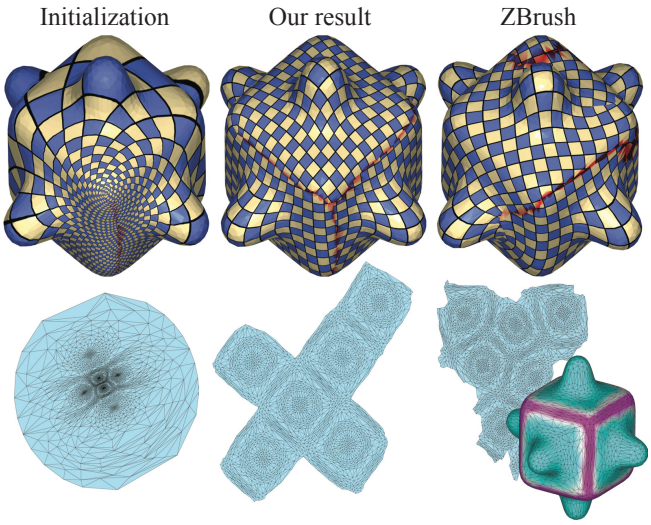


Fig. 6. Result of an automatic UV mapping of the Bumpy Cube using our method, without user interaction and with default parameter settings, which produces an intuitive cut along the edges of the cube. In contrast, the *assisted* UV map of ZBrush fails to find this cut even with extensive manual suggestions, which we provided by manually painting candidate regions (purple regions, bottom right).

where $c(a, b)$ is a C^1 function penalizing $a > b$:

$$c(a,b) = \begin{cases} 0, & a \le b \\ (a-b)^2, & a > b \end{cases} \tag{11}$$

See Fig. 8 and the attached video for examples of using bounding rectangles.

UV maps for quad meshes. Our approach also supports the design of UV maps for quadrilateral or polygonal meshes, which are common in the graphics community. Since the distortion measure is defined for triangles, we simply triangulate all polygons and prevent the introduced edges from separating by increasing the weights of their associated separation energy terms by a factor 100 (Fig. 12).

UV maps for irregular meshes. In some applications, such as modeling for video games, and in stark contrast with geometry processing, meshes with highly irregular faces and non standard topologies are acceptable. Since our approach treats the mesh as a triangle soup, we can handle such meshes without any special considerations. In Fig. 10 we show an example of an irregular mesh and the UVs we obtained using our system. Note that despite its appearance as a single manifold, the mesh is in fact comprised of many connected components, hence the numerous patches we obtain in the UV map.

Comparison with [Sorkine et al. 2002]. We empirically compare our method with [Sorkine et al. 2002]. This method also targets low distortion and small number of cuts, but does so in a greedy manner. Starting from an isometrically parameterized single triangle, the method appends more triangles to it to create a patch, unless doing so exceeds a distortion bound or creates self-intersections. When no additional triangle can be added, the current patch is finalized and a new one is seeded from another triangle. In Fig. 11 we show the result on the Octopus mesh, for a specified distortion bound of 3 and an area-to-perimeter ratio of 10000; the map is over-segmented, and contains numerous patches consisting of a single triangle or short triangle strips. In contrast, the result obtained with our interactive system features fewer and better-shaped patches.

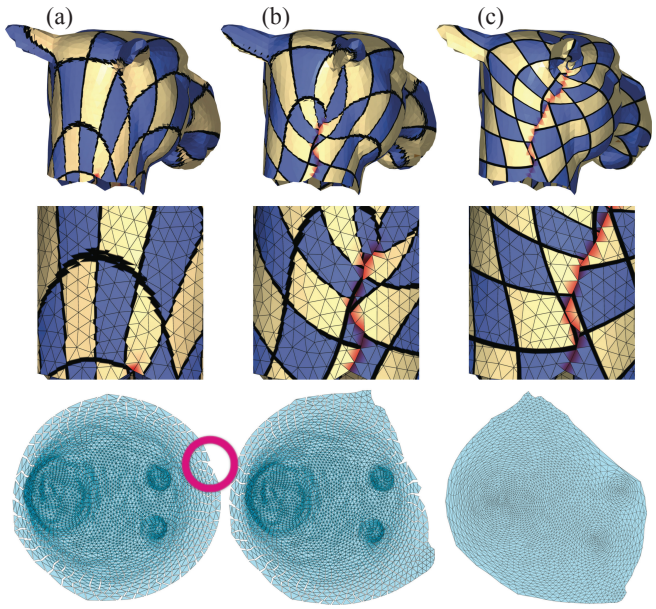


Fig. 7. In many cases, manually cutting one edge triggers the creation of a complete cut line. (a) User force-cuts a single edge in the highlighted region. (b) The cut automatically propagates through the mesh, considerably reducing the distortion. (c) Final result.

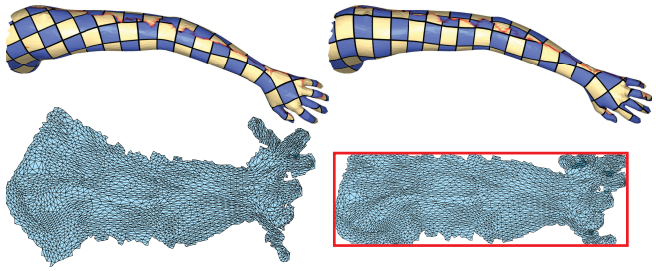


Fig. 8. Left: unconstrained parameterization; right: adding bounding rectangle constraints.

Removing self-overlaps. Texture mapping requires the map to be free from global-overlaps (as well as local ones, i.e. triangle flips, which are prevented by our energy functions). We offer no advancement toward the fulfillment of this requirement. Existing fully automatic techniques such as [Smith and Schaefer 2015] *prevent* global overlaps from appearing during the optimization, but this requires a valid, overlap-free map to start from, and the map must be kept overlap-free during all intermediate phases, which unnecessarily limits the solution space and hinders interactive UV map production. In contrast, in traditional interactive UV authoring suites, global overlaps are dealt with by the digital artist, for example by manually

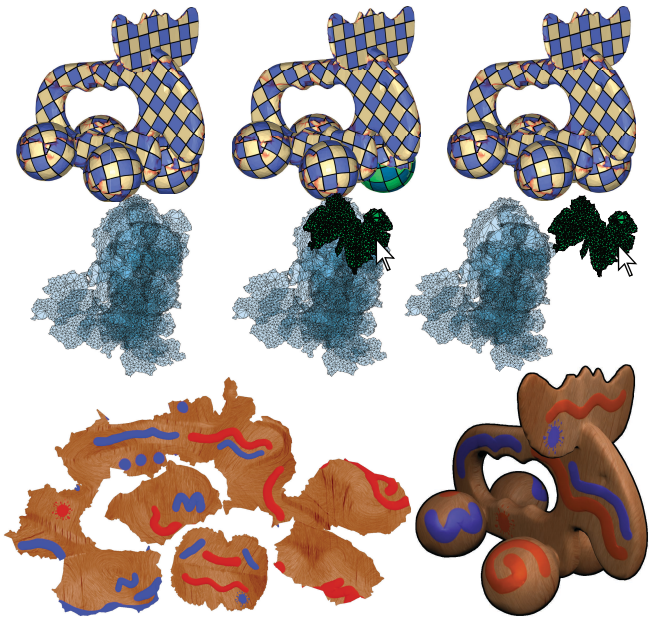


Fig. 9. From left to right: Using our tool, the user can select a maximal non-overlapping region and move it away, resolving overlaps. Bottom: the final non-overlapping UVs with a texture.

splitting self-overlapping charts. Our interactive framework can be employed, just as well, to this end. A difference is that, in our case, the task of identifying a proper location for the extra cuts can benefit from the assisted tools of our framework, which favors good cuts; for example, one can use a tear-up gesture for this purpose (see Fig. 1 and the accompanying video). Figures 11 and 14–16 show examples of overlap free UV maps constructed in this way. Many traditional suites also offer optional single-touch tools which the user can invoke at any moment to fix current global overlaps, using some automatic heuristic; the result is rarely immediately usable, and needs to be manually adjusted; still, this is usually faster than removing overlaps from scratch. Similar functionalities can be integrated in our framework too. In our prototype, we experimented with a greedy procedure based on [Sorkine et al. 2002] (see Fig. 11, right): starting from an arbitrary seed triangle, we visit the mesh in breadth-first order, avoiding triangles which would oversteps in UV any already visited triangle (as in [Sorkine et al. 2002], this test only affects the boundary of the visited region, making it efficient); when no new triangle can be added, we start with a new seed, and so on until all triangles are visited.

Cut length vs. number of cuts. By default, we minimize the total *length* of the cuts, by weighting the separation measure by edge length in Eq. (1). This reflects application penalties associated with cuts, such as texel replication or bleeding artifacts, which can be considered proportional to the length. Other penalties, like the need for vertex duplication, are rather associated with the *number* of cut edges. We can choose to minimize this number instead of the total length by giving all edges a weight of 1. This potentially leads to different results (see Fig. 13).

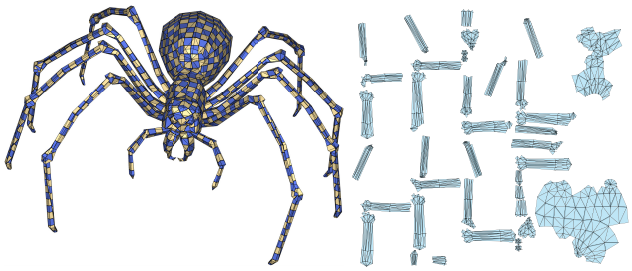


Fig. 10. Parameterization of an irregular, "video game style" mesh.

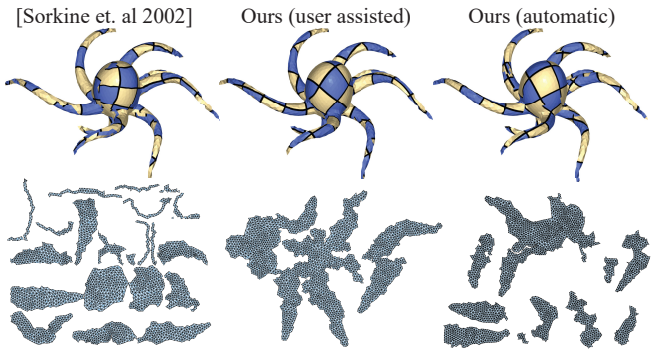


Fig. 11. Comparison with [Sorkine et al. 2002]. Our user-assisted optimization (middle column) results in fewer cuts and fewer patches. Right: automatic result obtained with our heuristic to add cuts and remove overlaps.

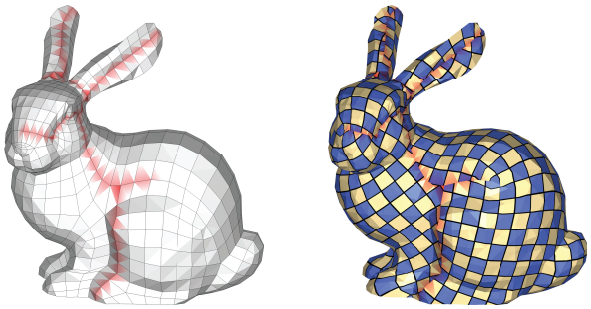


Fig. 12. UV-mapping of a quad mesh: internally, quad faces are triangulated, but cuts never split them.

5 LIMITATIONS AND CONCLUDING REMARKS

We proposed an algorithm to address a central challenge of parameterization design, that is, the identification of cuts and distortion minimizing unfolding within a unified minimization process of a single energy function. This is a stark, qualitative improvement over previous approaches, where the two problems are solved in cascade or in alternation. Our framework makes the entire interactive UV-map creation process faster and more intuitive. We conjecture that a commercial grade software suite based on our method would further improve it. We asked professional artists to experiment with our software, and received positive feedback. Initial experimentation showed that our approach can speed up the process of manual UV

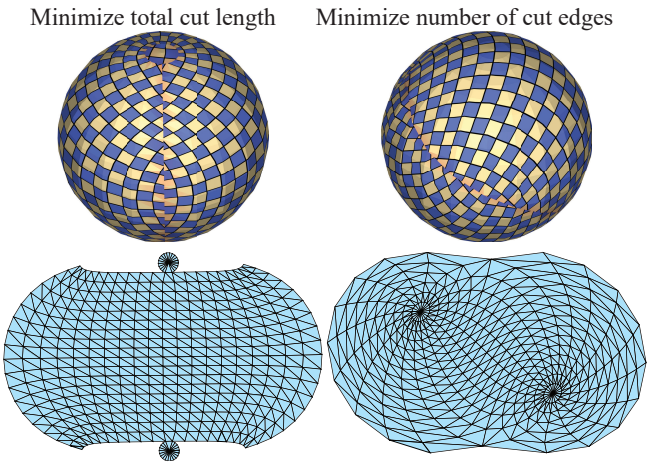


Fig. 13. By default, we minimize the total length of cuts (left), but we can opt to minimize their number instead (right).

Fig. 15. An example of texture from a high-poly Armadillo model (right) baked on a low-poly Armadillo model that has been UV mapped with our technique. Left: resulting textured low-res model.

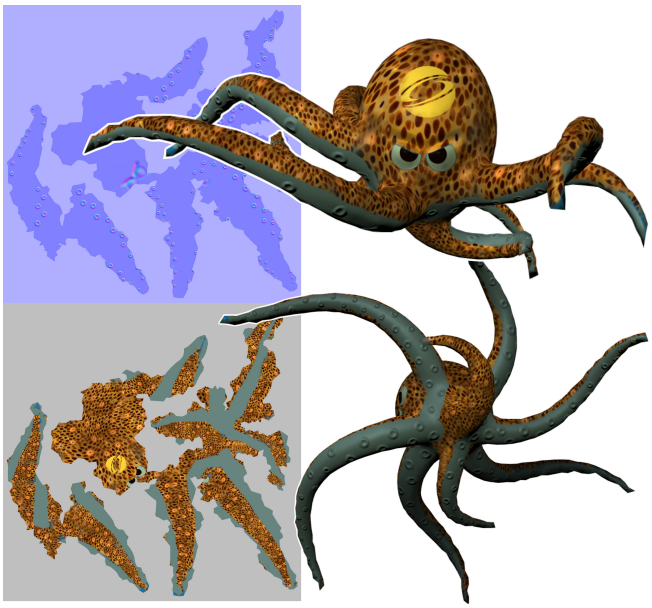


Fig. 14. Left: a texture set (top: tangent-space normal map, bottom: diffuse map), manually 3D painted over one model UV-mapped with our technique. Right: two views of the final model.

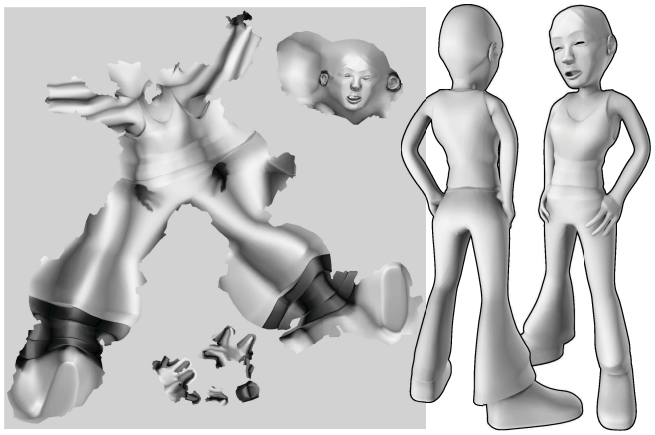


Fig. 16. A light-texture baked over a model that has been UV mapped with our technique. The illumination on the renderings on the right comes exclusively from the texture.

mapping considerably. While it does require a slight re-adjustment of the UV-mapping pipeline, the artists expressed interest in using our method for their future UV-mapping tasks, exploring its limitations and providing us with case studies for future development.

Scalability with mesh resolution. Our system is sufficiently responsive only if the input mesh is within the range of approximately 20k triangles. Many models employed by the game industry are already in the acceptable range. Our method can still be used offline, but the results then might require manual retouches to be finalized. One future direction for exploration is to reduce the number of degrees of freedom in the problem. Currently, we treat all edges as candidates for cut placement; this is can be excessive, since the placement does not always have to be so accurate. Another direction for future work is adapting acceleration techniques [Kovalsky et al. 2016; Liu et al. 2017] to our formulation.

Insensitivity to global overlaps. While our framework unifies cut placement and distortion minimization, two other tasks of UV-mapping authoring pipelines are left out of it: the removal of global overlaps and the final packing of charts are simply delegated to the manual intervention by the designer (assisted by the proposed interactive tools); in this, our framework offers no direct advancement over current UV authoring practices. A challenging avenue for future work is to embed the two tasks into the energy to be minimized, similarly to what we now introduced for cut selection.

Fig. 17. Several results obtained using our system with no assistance (Hand, Elk, Octopus, Moai statue), painting the front facing side to have no cuts (Eight, Girl) or painting regions that should have cuts (High-Genus Tet). See the accompanying video for more examples.

ACKNOWLEDGMENTS

We would like to thank Mario Botsch, Francis Williams, and the anonymous reviewers for their helpful comments and suggestions.

REFERENCES

E. L. Allgower and Kurt Georg. 2003. *Introduction to Numerical Continuation Methods.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

Douglas N. Arnold, Franco Brezzi, Bernardo Cockburn, and L. Donatella Marini. 2001. Unified Analysis of Discontinuous Galerkin Methods for Elliptic Problems. *SIAM J. Numer. Anal.* 39, 5 (2001), 1749–1779. DOI: http://dx.doi.org/10.1137/S0036142901384162

Ivo Babuška and M. Zlámal. 1973. Nonconforming Elements in the Finite Element Method with Penalty. *SIAM J. Numer. Anal.* 10, 5 (1973), 863–875. DOI: http://dx.doi.org/10.1137/0710071 arXiv:http://dx.doi.org/10.1137/0710071

Mirela Ben-Chen, Craig Gotsman, and Guy Bunin. 2008. Conformal Flattening by Curvature Prescription and Metric Scaling. *Comput. Graph. Forum* 27, 2 (2008), 449–458. DOI: http://dx.doi.org/10.1111/j.1467-8659.2008.01142.x

D. Benson and J. Davis. 2002. Octree textures. *ACM Trans. Graph.* 21, 3 (2002), 785–790.

Ioana Boier-Martin, Holly Rushmeier, and Jingyi Jin. 2004. Parameterization of Triangle Meshes over Quadrilateral Domains. In *Proc. Symposium on Geometry Processing.* 193–203. DOI: http://dx.doi.org/10.1145/1057432.1057459

David Bommes, Bruno Lévy, Nico Pietroni, Enrico Puppo, Claudio Silva, Marco Tarini, and Denis Zorin. 2013. Quad-Mesh Generation and Processing: A Survey. *Computer Graphics Forum* 32, 6 (2013), 51–76.

David Bommes, Henrik Zimmer, and Leif Kobbelt. 2009. Mixed-integer Quadrangulation. *ACM Trans. Graph.* 28, 3, Article 77 (2009), 10 pages. DOI: http://dx.doi.org/10.1145/1531326.1531383

Stephen Boyd and Lieven Vandenberghe. 2004. *Convex Optimization.* Cambridge University Press, New York, NY, USA.

Brent Burley and Dylan Lacewell. 2008. Ptex: Per-Face Texture Mapping for Production Rendering. In *Proc. Eurographics Symp. on Rendering.* 1155–1164.

M. Campen and L. Kobbelt. 2014. Quad Layout Embedding via Aligned Parameterization. *Comput. Graph. Forum* 33, 8 (2014), 13. DOI: http://dx.doi.org/10.1111/cgf.12401

Per H. Christensen and Dana Batali. 2004. An Irradiance Atlas for Global Illumination in Complex Production Scenes. In *Proc. Eurographics Symp. on Rendering.* 133–141. DOI: http://dx.doi.org/10.2312/EGWR/EGSR04/133-141

B. Bernardo Cockburn, George Karniadakis, and Chi-Wang Shu (Eds.). 2000. *Discontinuous Galerkin methods : theory, computation, and applications.* Springer, Berlin, New York. http://opac.inria.fr/record=b1096869

Ingrid Daubechies, Ronald Devore, Massimo Fornasier, and C. Sinan Güntürk. 2010. Iteratively reweighted least squares minimization for sparse recovery. *Comm. Pure Appl. Math* 63, 1 (2010), 1–38.

Mathieu Desbrun, Mark Meyer, and Pierre Alliez. 2002. Intrinsic Parameterizations of Surface Meshes. *Comput. Graph. Forum* 21, 3 (2002), 209–218. DOI: http://dx.doi.org/10.1111/1467-8659.00580

M. S. Floater. 2003. Mean Value Coordinates. *Computer Aided Geometric Design* 20, 1 (2003), 19–27.

Xiao-Ming Fu and Yang Liu. 2016. Computing Inversion-free Mappings by Simplex Assembly. *ACM Trans. Graph.* 35, 6, Article 216 (2016), 12 pages. DOI: http://dx.doi.org/10.1145/2980179.2980231

Xiao-Ming Fu, Yang Liu, and Baining Guo. 2015. Computing Locally Injective Mappings by Advanced MIPS. *ACM Trans. Graph.* 34, 4, Article 71 (2015), 12 pages. DOI: http://dx.doi.org/10.1145/2766938

Xianfeng Gu, Steven J. Gortler, and Hugues Hoppe. 2002. Geometry Images. *ACM Trans. Graph.* 21, 3 (2002), 355–361. DOI: http://dx.doi.org/10.1145/566654.566589

K. Hormann and G. Greiner. 2000. MIPS: An Efficient Global Parametrization Method. In *Proc. Curve and Surface Design*. 153–162.

K. Hormann, B. Lévy, and A. Sheffer. 2007. Mesh parameterization: Theory and practice. In *ACM SIGGRAPH Course Notes*.

Dan Julius, Vladislav Kraevoy, and Alla Sheffer. 2005. D-Charts: Quasi-Developable Mesh Segmentation. *Comput. Graph. Forum* 24, 3 (2005).

Andrei Khodakovsky, Nathan Litke, and Peter Schröder. 2003. Globally Smooth Parameterizations with Low Distortion. *ACM Trans. Graph.* 22, 3 (2003), 350–357. DOI: http://dx.doi.org/10.1145/882262.882275

Shahar Z. Kovalsky, Meirav Galun, and Yaron Lipman. 2016. Accelerated quadratic proxy for geometric optimization. *ACM Trans. Graph.* 35, 4 (2016). DOI: http://dx.doi.org/10.1145/2897824.2925920

A. Kuzmin, M. Luisier, and O. Schenk. 2013. Fast Methods for Computing Selected Elements of the Green's Function in Massively Parallel Nanoelectronic Device Simulations. In *Proc. Euro-Par*. 533–544. DOI: http://dx.doi.org/10.1007/978-3-642-40047-6_54

Sylvain Lefebvre and Carsten Dachsbacher. 2007. Tiletrees. In *Proc. of the Symp. on Interact. 3D Graph. and Games*. ACM, 25–31.

S. Lefebvre and H. Hoppe. 2006. Perfect spatial hashing. *ACM Trans. Graph.* 25, 3 (2006), 579–588.

Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and JÃľrome Maillot. 2002. Least Squares Conformal Maps for Automatic Texture Atlas Generation. *ACM Trans. Graph.* 21, 3 (2002), 362–371. DOI: http://dx.doi.org/10.1145/566654.566590

Zilin Li, Sijian Wang, and Xihong Lin. 2012. Variable selection and estimation in generalized linear models with the seamless L_0 penalty. *Canadian Journal of Statistics* 40, 4 (2012), 745–769. DOI: http://dx.doi.org/10.1002/cjs.11165

Ligang Liu, Lei Zhang, Yin Xu, Craig Gotsman, and Steven J. Gortler. 2008. A Local/Global Approach to Mesh Parameterization. In *Proc. Symposium on Geometry Processing*. 1495–1504. http://dl.acm.org/citation.cfm?id=1731309.1731336

Tiantian Liu, Sofien Bouaziz, and Ladislav Kavan. 2017. Quasi-Newton Methods for Real-Time Simulation of Hyperelastic Materials. *ACM Transactions on Graphics (TOG)* 36, 3 (2017), 23.

Jérôme Maillot, Hussein Yahia, and Anne Verroust. 1993. Interactive Texture Mapping. In *Proc. ACM SIGGRAPH*. 27–34. DOI: http://dx.doi.org/10.1145/166117.166120

Hossein Mobahi and John W. Fisher. 2015. On the Link between Gaussian Homotopy Continuation and Convex Envelopes. In *Proc. Energy Minimization Methods in Computer Vision and Pattern Recognition*. 43–56. DOI: http://dx.doi.org/10.1007/978-3-319-14612-6_4

Ashish Myles and Denis Zorin. 2013. Controlled-distortion Constrained Global Parametrization. *ACM Trans. Graph.* 32, 4, Article 105 (2013), 14 pages. DOI: http://dx.doi.org/10.1145/2461912.2461970

Pixologic. 2017. ZBrush. http://pixologic.com/. (2017). Accessed: 2017-02-01.

Budirijanto Purnomo, Jonathan D Cohen, and Subodh Kumar. 2004. Seamless texture atlases. In *Proc. Symposium on Geometry Processing*. 65–74.

Michael Rabinovich, Roi Poranne, Daniele Panozzo, and Olga Sorkine-Hornung. 2017. Scalable Locally Injective Mappings. *ACM Trans. Graph.* 36, 2 (2017), 16:1–16:16.

Nicolas Ray, Vincent Nivoliers, Sylvain Lefebvre, and Bruno Lévy. 2010. Invisible Seams. In *Proc. Eurographics Symp. on Rendering*. DOI: http://dx.doi.org/10.1111/j.1467-8659.2010.01746.x

Kenneth Rose. 1998. Deterministic Annealing for Clustering, Compression, Classification, Regression, and Related Optimization Problems. In *Proceedings of the IEEE*. 2210–2239.

Pedro V. Sander, John Snyder, Steven J. Gortler, and Hugues Hoppe. 2001. Texture mapping progressive meshes. In *Proc. ACM SIGGRAPH*. 409–416. http://doi.acm.org/10.1145/383259.383307

Olaf Schenk, Matthias Bollhöfer, and Rudolf A. Römer. 2008. On Large-Scale Diagonalization Techniques for the Anderson Model of Localization. *SIAM Rev.* 50, 1 (2008), 91–112. DOI: http://dx.doi.org/10.1137/070707002

Olaf Schenk, Andreas Wächter, and Michael Hagemann. 2007. Matching-based preprocessing algorithms to the solution of saddle-point problems in large-scale nonconvex interior-point optimization. *Computational Optimization and Applications* 36, 2-3 (2007), 321–341. DOI: http://dx.doi.org/10.1007/s10589-006-9003-y

Alla Sheffer and John C. Hart. 2002. Seamster: Inconspicuous Low-distortion Texture Seam Layout. In *Proc. Visualization*. 8. http://dl.acm.org/citation.cfm?id=602099.602144

Alla Sheffer, Bruno Lévy, Maxim Mogilnitsky, and Alexander Bogomyakov. 2005. ABF++: Fast and Robust Angle Based Flattening. *ACM Trans. Graph.* 24, 2 (2005), 311–330. DOI: http://dx.doi.org/10.1145/1061347.1061354

A. Sheffer, E. Praun, and K. Rose. 2006. Mesh parameterization methods and their applications. *Foundations and Trends® in Computer Graphics and Vision* 2, 2 (2006), 105–171.

Anna Shtengel, Roi Poranne, Olga Sorkine-Hornung, Shahar Z. Kovalsky, and Yaron Lipman. 2017. Geometric Optimization via Composite Majorization. *ACM Trans. Graph.* 36, 4, Article 38 (July 2017), 11 pages. DOI: http://dx.doi.org/10.1145/3072959.3073618

Jason Smith and Scott Schaefer. 2015. Bijective Parameterization with Free Boundaries. *ACM Trans. Graph.* 34, 4, Article 70 (2015), 9 pages. DOI: http://dx.doi.org/10.1145/2766947

Olga Sorkine, Daniel Cohen-Or, Rony Goldenthal, and Dani Lischinski. 2002. Bounded-distortion piecewise mesh parameterization. In *Proc. Visualization*. 355–362.

Kenshi Takayama, Daniele Panozzo, Alexander Sorkine-Hornung, and Olga Sorkine-Hornung. 2013. Sketch-Based Generation and Editing of Quad Meshes. *ACM Trans. Graph.* 32, 4 (2013), 97:1–97:8. DOI: http://dx.doi.org/10.1145/2461912.2461955

M. Tarini. 2012. Cylindrical and toroidal parameterizations without vertex seams. *Journal of Graphics Tools* 16, 3 (2012), 144–150.

Marco Tarini. 2016. Volume-encoded UV-maps. *ACM Trans. Graph.* 35, 4, Article 107 (2016), 13 pages. DOI: http://dx.doi.org/10.1145/2897824.2925898

M. Tarini, K. Hormann, P. Cignoni, and C. Montani. 2004. PolyCube-Maps. *ACM Trans. Graph.* 23, 3 (2004), 853–860. DOI: http://dx.doi.org/10.1145/1015706.1015810

Marco Tarini, Enrico Puppo, Daniele Panozzo, Nico Pietroni, and Paolo Cignoni. 2011. Simple Quad Domains for Field Aligned Mesh Parametrization. *ACM Trans. Graph.* 30, 6, Article 142 (2011), 12 pages. DOI: http://dx.doi.org/10.1145/2070781.2024176

Marco Tarini, Cem Yuksel, and Sylvain Lefebvre. 2017. Rethinking Texture Mapping. In *ACM SIGGRAPH 2017 Courses (SIGGRAPH '17)*. ACM, New York, NY, USA, Article 11, 139 pages. DOI: http://dx.doi.org/10.1145/3084873.3084911

Y. Tong, P. Alliez, D. Cohen-Steiner, and M. Desbrun. 2006. Designing Quadrangulations with Discrete Harmonic Forms. In *Proc. Symp. Geom. Processing*. 201–210. http://dl.acm.org/citation.cfm?id=1281957.1281983

Francesco Usai, Marco Livesu, Enrico Puppo, Marco Tarini, and Riccardo Scateni. 2015. Extraction of the Quad Layout of a Triangle Mesh Guided by Its Curve Skeleton. *ACM Trans. Graph.* 35, 1, Article 6 (2015), 13 pages. DOI: http://dx.doi.org/10.1145/2809785

Cem Yuksel, John Keyser, and Donald H. House. 2010. Mesh colors. *ACM Trans. Grap.* 29, 2, Article 15 (2010), 11 pages. DOI: http://dx.doi.org/10.1145/1731047.1731053

Eugene Zhang, Konstantin Mischaikow, and Greg Turk. 2005. Feature-based Surface Parameterization and Texture Mapping. *ACM Trans. Graph.* 24, 1 (Jan. 2005), 1–27. DOI: http://dx.doi.org/10.1145/1037957.1037958

APPENDIX A SEPARATION GRADIENT AND HESSIAN

We write the expressions for the gradient and Hessian of the separation measure between two corners. Let the two corners be $\mathbf{x}_{ik_1}$ and $\mathbf{x}_{ik_2}$. Then the separation measure (Eq. (3)) between them is

$$\hat{s}\left(\left\|\mathbf{x}_{ik_1}-\mathbf{x}_{ik_2}\right\|^2\right), \text{ where } \hat{s}(t)=\frac{t}{t+\delta}.$$

The gradient and modified Hessian of s is as follows. Define the column vector $\mathbf{d}=2\begin{pmatrix}\mathbf{x}_{ik_1}-\mathbf{x}_{ik_2}\\ \mathbf{x}_{ik_2}-\mathbf{x}_{ik_1}\end{pmatrix}$. Then $\nabla\hat{s}\left(\left\|\mathbf{x}_{ik_1}-\mathbf{x}_{ik_2}\right\|^2\right)=\frac{\partial\hat{s}}{\partial t}\mathbf{d}$ and the Hessian H is

$$H=\nabla^2\hat{s}\left(\left\|\mathbf{x}_{ik_1}-\mathbf{x}_{ik_2}\right\|^2\right)=\frac{\partial^2\hat{s}}{\partial t^2}\mathbf{d}\mathbf{d}^{\top}+\frac{\partial\hat{s}}{\partial t}\begin{pmatrix}+1&0&-1&0\\0&+1&0&-1\\-1&0&+1&0\\0&-1&0&+1\end{pmatrix}$$

To ensure that the total Hessian is positive definite, a well known trick is to project H of each face onto the set of positive semidefinite matrices before summing them up. This requires to compute one SVD for each face, and can be time consuming; instead, after [Shtengel et al. 2017], we simply remove the non-convex part in the expression for $\nabla^2\hat{s}$. The second term in the expression is always positive semidefinite, while the first term depends on the second derivative of $\hat{s}$, which can be negative; therefore, its elimination leads to a modified positive semidefinite H:

$$\nabla^2\hat{s}\left(\left\|\mathbf{x}_{ik_1}-\mathbf{x}_{ik_2}\right\|^2\right)=\frac{\partial\hat{s}}{\partial t}\mathbf{E}.$$

By using this modified Hessian, we observe an increase in the frame rate, while the effect on convergence is marginal.