

Adaptive Block Coordinate Descent for Distortion Optimization

Alexander Naitsat,¹ Yufeng Zhu² and Yehoshua Y. Zeevi¹

¹Viterbi Faculty of Electrical Engineering, Technion - Israel Institute of Technology, Israel
anaitsat@campus.technion.ac.il, zeevi@ee.technion.ac.il

²Facebook Reality Labs, PA, USA
mike323zyf@gmail.com

Abstract

We present a new algorithm for optimizing geometric energies and computing positively oriented simplicial mappings. Our major improvements over the state-of-the-art are: (i) introduction of new energies for repairing inverted and collapsed simplices; (ii) adaptive partitioning of vertices into coordinate blocks with the blended local-global strategy for more efficient optimization and (iii) introduction of the displacement norm for improving convergence criteria and for controlling block partitioning. Together these improvements form the basis for the Adaptive Block Coordinate Descent (ABCD) algorithm aimed at robust geometric optimization. ABCD achieves state-of-the-art results in distortion minimization, even under hard positional constraints and highly distorted invalid initializations that contain thousands of collapsed and inverted elements. Starting with an invalid non-injective initial map, ABCD behaves as a modified block coordinate descent up to the point where the current mapping is cleared of invalid simplices. Then, the algorithm converges rapidly into the chosen iterative solver. Our method is very general, fast-converging and easily parallelizable. We show over a wide range of 2D and 3D problems that our algorithm is more robust than existing techniques for locally injective mapping.

Keywords: geometric optimization, inversion-free mapping, deformation, mesh parametrization, tetrahedral mesh

1. Introduction

Computing injective mappings with low distortions on triangulated domains is a fundamental problem in computer graphics, geometrical modelling and physical simulations. This problem often results in non-convex and non-linear optimization of geometric energies defined in a finite element manner on triangular and tetrahedral meshes.

The existing solutions to the above problem typically fall into two major categories: (1) *map fixers*: algorithms focused on the injectivity of maps; and (2) *core-solvers*: iterative descent algorithms focused on minimizing rotation-invariant energies. Map fixers are aimed at repairing simplices that have folded over (*foldovers*) and changed their orientation. The goal of map fixers is to repair all foldovers of a non-injective map and restraining its geometric distortions into a finite range. Core solvers start from a one-to-one initialization and ensure that optimization results remain one-to-one, at least locally. Efficient designs of map fixers and core-solvers have been extensively studied by the computer graphics community. But due to their inherent differences the treatment remained separate,

one would either provide a framework to solve the foldover problem or the energy minimization problem. Nevertheless, both aspects of the problem play crucial role in finding a good solution to a given geometric optimization. In this paper, we propose the Adaptive Block Coordinate Descent (ABCD) algorithm that successfully tackles both problems within the same routine. Our algorithm achieves state-of-the-art results in distortion minimization even with highly distorted invalid initializations that contain thousands of degenerate and inverted elements. Consequently, compared with core-solvers, our algorithm is much more robust. At the same time, compared with recent map fixer methods, our algorithm achieves superior results, making some previously intractable problems applicable. The core contributions of our algorithm are:

1. Introduction of new distortion measures designed for repairing inverted and degenerate simplices, as well as modification of classical energies to allow an efficient integration of these energies into the proposed framework.
2. An adaptive partitioning of vertices into blocks of varying sizes according to the geometric configuration of the problem.

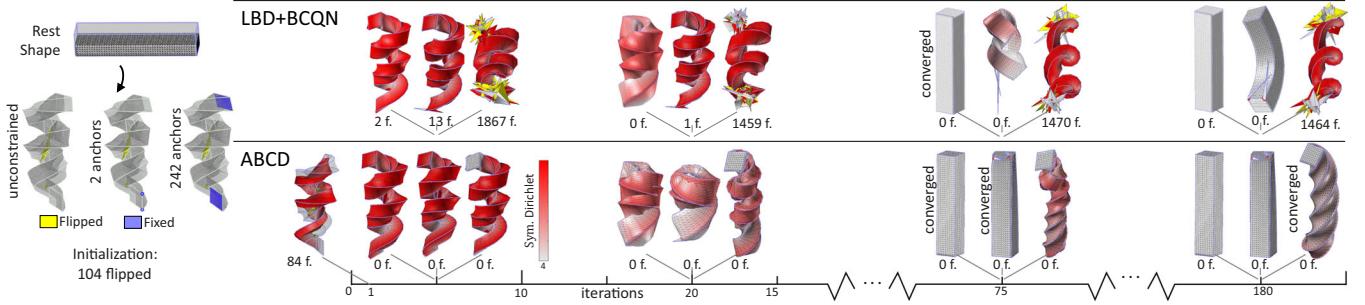


Figure 1: Comparison of the alternating combination of LBD and BCQN (top) with ABCD(BCQN) (bottom) in 3D. The same initialization is used for the three shown trials: unconstrained problem, two fixed anchors, two fixed endpoints (the second problem of Figure 18). We report the number of inverted elements below each snapshot.

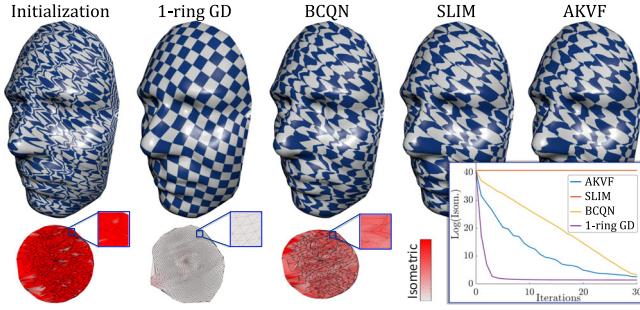


Figure 2: Free boundary parametrization with a noisy initialization (intentionally distorted Tutte map without flips). The 1-ring GD denotes the gradient descent in blocks of single vertices; each block receives a single GD iteration per cycle depicted in the plot.

3. An adaptively blended local-global strategy providing robustness to highly distorted elements, while allowing employment of state-of-the-art solvers for inducing global deformations.
4. A cured alternating optimization strategy designed to best exploit contributions 1–3 and provide enough flexibility for our algorithm to be combined with other popular methods. This stage includes number of novel sub-steps, such as an enhanced line search filtering.
5. Introduction of displacement norm for improving convergence criteria and for controlling the local-global blending.

2. Problem Formulation and Overview

We compute inversion-free simplicial maps that induce low distortions. Let $f : M \subset \mathbb{R}^n \rightarrow \mathbb{R}^m$ be a piecewise affine mapping defined on a mesh M with a m -dimensional simplex set \mathbf{C} and a vertex set \mathbf{V} , and let $\mathbf{x} \in \mathbb{R}^{|\mathbf{V}|^m}$ be the column stack of vertex positions under f . We express a distortion energy of the mapping $f[\mathbf{x}]$ as a weighted sum over simplices

$$E(f[\mathbf{x}]) = \sum_{c \in \mathbf{C}} w(c) \mathcal{D}(f_c[\mathbf{x}]), \quad (1)$$

where $w(c)$ are simplex weights, usually equal to $\text{volume}(c)$, f_c is f 's component on c and \mathcal{D} is a given distortion measure invariant to rotations and translations of source and target vertices. Denote

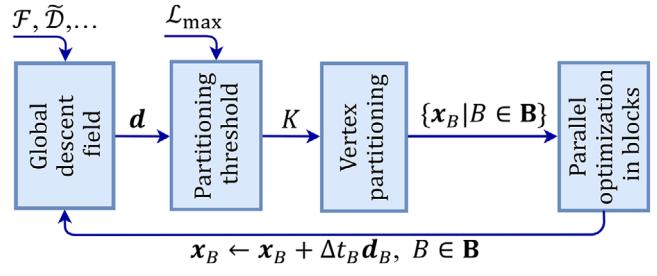


Figure 3: ABCD main stages: (a) Computing descent field of alternatively selected distortions $\mathcal{D}_j = \mathcal{F}, \tilde{\mathcal{D}}$, defined by (15) and (10); (b) estimating partitioning threshold K ; (c) vertex partitioning into blocks \mathbf{B} by Algorithm 1; (d) parallel optimization of (2) in coordinate blocks $B \in \mathbf{B}$.

by df_c the Jacobian of f_c modulo a rigid transformation of c from \mathbb{R}^n to \mathbb{R}^m . Then, we address the following distortion minimization problem for $2 \leq m \leq n \leq 3$:

$$\underset{\mathbf{x}}{\operatorname{argmin}} E(f[\mathbf{x}]), \quad (2)$$

$$\text{such that: } \det(df_c) > 0, \quad \forall c \in \mathbf{C}, \quad (3)$$

$$\mathbf{A}\mathbf{x} = \mathbf{z}, \quad (4)$$

where (3) guarantees that f is free of inversions and (4) are the given positional constraints.

Numerous first- and second-order techniques for solving (2) and its unconstrained version have been developed. A typical geometric solver updates target coordinates \mathbf{x} via

$$\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + \Delta t \mathbf{d}(\nabla_{\mathbf{x}} E, \nabla_{\mathbf{x}}^2 E), \quad (5)$$

where \mathbf{d} is the descent direction $\langle \mathbf{d}, \nabla_{\mathbf{x}} E \rangle < 0$, expressed as a function of the gradient and the Hessian

$$\nabla_{\mathbf{x}} E = \frac{\partial E(\mathbf{x}(t))}{\partial \mathbf{x}}, \quad \nabla_{\mathbf{x}}^2 E = \frac{\partial^2 E(\mathbf{x}(t))}{\partial \mathbf{x}^2}. \quad (6)$$

In first-order methods, $\mathbf{d} = \mathbf{d}(\nabla E)$ and it is computed, in general, by pre-conditioning the gradient with a sparse proxy matrix,

i.e. in unconstrained problems \mathbf{d} is a solution of a sparse linear system

$$H\mathbf{d} = -\nabla_{\mathbf{x}} E. \quad (7)$$

If \mathbf{x}^i are vertex coordinates at iteration i , then updating \mathbf{x} via (7) is equivalent to minimizing the following quadratic proxy:

$$E_H(\mathbf{x}) = E(\mathbf{x}^i) + (\mathbf{x} - \mathbf{x}^i)^T \nabla_{\mathbf{x}} E + (\mathbf{x} - \mathbf{x}^i)^T H(\mathbf{x} - \mathbf{x}^i) / 2. \quad (8)$$

For example, $H = I$ in the gradient descent (GD) [NSZ18, FLG15]. In Sobolev gradient descent (SGD), and in the related accelerated quadratic proxy (AQP) [KGL16], H is chosen to be the rest mesh Laplacian. The method of scalable locally injective mappings (SLIM) [RPPSH17] extends the local-global parametrization [LX*08] to general distortions by introducing a weighted proxy in the global step, which is equivalent to approximating H by reweighed Laplacian. The geometric approach of [CBSS17] introduces the Killing operator of discrete vector fields as an isometry-aware pre-conditioner (AKVF).

Quasi-Newton methods [ZBK18, LBK17, SS15] optimize a quadratic proxy constructed from gradients of both the current and previous iterations, i.e. $\mathbf{d} = \mathbf{d}(\nabla_{\mathbf{x}} E^i, \nabla_{\mathbf{x}} E^{i-1}, \dots)$. In particular, Zhu *et al.* [ZBK18] had proposed blended cured quasi-Newton (BCQN) strategy of a gradual blending between AQP [KGL16] and L-BFGS [SS15]. This approach benefits both from the rapid progress of AQP at the first iterations, and having the super-linear convergence of L-BFGS in the vicinity of the optimal point.

Similarly, second-order methods are based on ‘Newton’ update step in (5), where \mathbf{d} is the function of both ∇E and $\nabla^2 E$. Since the problem (2) is highly non-convex, these solvers compute positive semi-definite approximations of $\nabla^2 E$, most commonly, using the Hessian diagonal $H = \text{diag}(\nabla^2 E(\mathbf{x}))$ in Jacobi GD [WY16]; by projecting $m(m+1) \times m(m+1)$ blocks of $\nabla^2 E$ into PSD cone for each simplex in projected Newton (PN) methods [LKK*18, LBK16, TSIF05]. Specifically, in 2D, positive semi-definite H can be obtained via complex problem formulation [GSC18, CW17], or by using the composite majorization (CM) [SPSH*17] technique and the related closed-form expression of 2×2 Jacobian singular values.

Besides the aforementioned core-solvers, there are techniques intended for accelerating existing optimization algorithms. For instance, the update step in AQP is equipped with a Nesterov-like acceleration [Nes83] that approximates $\mathbf{x}(t)$ in (5) by an affine combination of the results from the current and previous iterations, $\mathbf{x}(t) = (1 + \theta)\mathbf{x}^i - \theta\mathbf{x}^{i-1}$. Progressive parametrization (PP) [LYNF18] accelerates core-solvers by decomposing an initial map into intermediate mappings with bounded singular values. Anderson acceleration [PDZ*18] adopts numerical analysis techniques for geometric optimization by treating solvers as fixed-point iterations. The method is intended for alternating local-global algorithms, such as [BDS*12, BML*14], where in the local step, vertex coordinates are projected into varying sets of positional constraints.

Despite the abundance of techniques, the vast majority of the existing solvers are iterative algorithms that require an inversion-free initialization of f in (2). Moreover, in most methods, $f[\mathbf{x}^i]$ needs to be kept free of foldovers for each iteration $i \geq 0$. In view of the above limitation, different strategies have been proposed

to keep f satisfying (3) during the optimization. These strategies include: designing distortions with flip barriers [SS15, FLG15], inversion-aware line search [SS15], the recently proposed barrier-aware line search filtering [ZBK18] and employment of scaffold meshes [LKK*18, JSP17].

For unconstrained 2D distortion optimization, such as parametrization of disc-topology surfaces with free boundaries, the required positively oriented initialization f^0 is readily available via the Tutte’s embedding [Tut63] and its variants [Flo03]. However, computing feasible f^0 may still be challenging in number of geometry processing applications, including 2D and 3D shape deformations with positional constraints, shape matching and volumetric parametrizations. Moreover, Tutte’s mapping to unconvex regions cannot guarantee (3), while mapping complex shapes to a disc and other convex domains usually yields huge isometric distortions. Thus, producing f^0 free of foldovers often costs many more additional iterations for a typical parametrization algorithm. It therefore seems that mapping to domains that match the structure of a source mesh should be a much better starting point if the algorithm can fix occasional foldovers. Although there is a number of recent studies on embedding triangle mesh onto non-convex planar domains, our algorithm has unique properties: it operates both in 2D and 3D; it employs hard positional constraints, and therefore, unlike boundary first flattening (BFF) [SC18], it interpolates the exactly prescribed boundary shape; ABCD updates only vertex coordinates and, in contrast to the recently introduced progressive embedding (PE) [SJZP19], our algorithm does not involve mesh re-triangulation.

Among the recent studies that explicitly address unfeasible initializations are: bounding distortion mappings (BD and LBD) [AL13, KABL15] and simplex assembly (SA) [FL16]. BD solves the quadratic problem of constructing an optimal projection of f on the bounded distortion space, whereas LBD enhances this strategy by linearization of the BD space constraints and pre-factorization of the obtained KKT matrix. SA [FL16] optimizes significantly larger problem than (2), since its objective variables are linear and translation components of f_c represented, per simplex, by $n \times n + m$ unknowns instead of commonly employed m coordinates per vertex ($n \geq m$ and, in general, $|C|$ is much larger than $|\mathbf{V}|$ according to Euler formula).

The method of [KABL14] can start with trivial maps and converge to prescribed positional constraints, but it suffers from slow performance of semi-definite programming. Weber *et al.* [WMZ12] introduced the least square Beltrami (LSB) energy for computing extremal quasi-conformal mappings. Optimization of LSB can be initialized with foldovers, since the energy is finite on inverted triangles. However, this method is limited to 2D and it does not extend to other popular energies, such as symmetric Dirichlet energy D_{iso} [SS15] or AMIPS [FLG15].

The Autocuts algorithm [PTH*17], designed for a user-assisted global parametrization, can handle non-injective initializations because it treats the input mesh as a triangle soup. Nevertheless, the algorithm requires a consistent initial orientation of deformed triangles. Moreover, this method operates on disassembled simplices which, similarly to SA, significantly increases number of objective variables. Although the underlining homotopy approach for

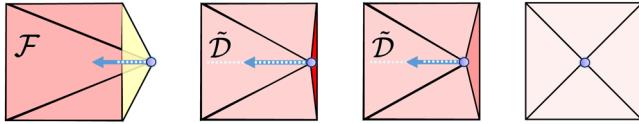


Figure 4: A typical ABCD workflow: initialization with flips (in yellow), fixer, optimizer and final convergence. Dashed lines along descent directions depict length of line search intervals (22).

multi-objective optimization performs well on middle-resolution meshes, Autocuts supports only soft positional constraints in 2D and does not extend to volumetric domains.

We propose a novel algorithm, ABCD, aimed at robust distortion optimization. By robustness, we mean that our algorithm is capable of solving challenging problems that previously available methods struggle with. These problems include cases of highly distorted, non-locally-injective, noisy initializations and mapping with complex positional constraints. Our algorithm combines the best of core-solvers (CM, PN, BCQN, AKVF, SLIM) and map fixers (BD, LBD, SA) into unified Algorithm 3. We show that, over a wide range of test cases, ABCD achieves superior results than mere cascade or alternating combinations of core-solvers and map fixers. Moreover, since our algorithm is very general, it can be further improved by employing novel acceleration techniques, such as PP [LYNF18].

For standard unconstrained problems, our algorithm behaves exactly as an integrated core-solver (BCQN, AKVF, PN, etc.), except for a very short period, typically two to three first iterations, for which computational cost is negligible. In the presence of highly distorted invalid initializations, ABCD passes through two phases: the first one of cleaning foldovers, where distortions are optimized in coordinate blocks of varying sizes, and the second stage of optimizing a positively oriented map, during which ABCD typically behaves as a global solver. Our algorithm exhibits such flexibility because, at each cycle, we adaptively modify coordinate blocks and switch between different energies. Unlike the common notion of optimization theory, where ‘adaptiveness’ is referring to a proper selection of block coordinate frames, we consider, instead, an adaptive strategy of the block partitioning and an adaptive blending of local-global solvers.

3. Distortion Measures

Distortion measures considered in (2) are rotation invariants which, according to [RPPSH17, NSZ18], can be expressed as functions of signed singular values $\sigma_1, \dots, \sigma_m$ of the Jacobian df_c :

$$\mathcal{D}(f_c) = \mathcal{D}(\sigma_1, \dots, \sigma_m). \quad (9)$$

We assume w.l.o.g. that only σ_1 can be negative in the signed SVD of the Jacobian, $df_c = U \text{diag}(\sigma_1, \dots, \sigma_m) V^\top$. Thus, on a non-degenerate simplex c , $\text{sign}(\sigma_1) = \text{sign}(\det(df_c))$.

The most common energies, considered in geometric optimization, are measures of isometric (length-wise) [CPSS10, SS15, FLG15, NSZ18] and conformal (angle-wise) [HG00, LPRM02, DMA02, FLG15] distortions. Usually, due to existence of a *barrier term*, these measures become infinite when $\det(df_c) \leq 0$. While this

prohibits appearance of new foldovers, it also prevents optimization from fixing the existing ones. Therefore, we aim to modify existing energies in such a way that both tasks of fixing invalid simplices (inverted or collapsed) and preventing generation of new ones are possible within the same procedure.

Denote by s_c the orientation of simplex c at the iteration $(i - 1)$, i.e. $s_c = \text{sign}(\det df_c^{i-1})$; then we modify the original measure \mathcal{D} as follows:

$$\tilde{\mathcal{D}}(f_c) = \begin{cases} \mathcal{D}(\sigma_1, \dots, \sigma_m) & \sigma_1 > 0, s_c > 0, \\ \infty & \sigma_1 \leq 0, s_c > 0, \\ \min \mathcal{D} & s_c \leq 0, \end{cases} \quad (10)$$

where $\min \mathcal{D}$ is the absolute minimum of distortion \mathcal{D} . Namely, at each iteration, we filter out the contribution of simplices that were previously inverted, while on valid simplices, $\tilde{\mathcal{D}}$ equals the original measure plus the barrier term in order to prevent new foldovers. This approach enables an efficient alternating optimization of classical rotation-invariant energies and of measures designed for fixing invalid simplices (*map fixer measures*) without badly interfering with each other so that our algorithm alternates between the two phases: minimization of map fixer measures and optimization of distortions, constructed by (10). We refer to these two phases as the *fixer* and *optimizer* phases, respectively. Since orientation of a zero-volume simplex is not well defined, we consider inverted simplices (flips) and collapsed simplices as the two separate classes of illegally deformed elements. Next, we describe how map fixer measures are devised to correct each of the two illegal deformations:

Inverted simplices: Assuming the positive orientation of the source elements, a simplex c is inverted under f if

$$\det(df_c) = \prod \sigma_j < 0. \quad (11)$$

Rather than using binary measures, or infinite barrier terms that can easily block optimization progress, we penalize both the number of foldovers and their target volumes (areas). Since foldover volumes are negative and proportional to $\det df_c$, we consider the following *flip penalty* measure:

$$\mathcal{D}_{\text{flip}}(\sigma, \Lambda) = \begin{cases} \Lambda - \prod \sigma_j & \prod \sigma_j \leq 0, \\ 0 & \text{else,} \end{cases} \quad (12)$$

where $\Lambda \geq 0$ is the uniform penalty cost of an inverted simplex.

Collapsed simplices: Similarly to $\mathcal{D}_{\text{flip}}$, the *collapsed simplex penalty* is the cost function of nearly zero-volume target elements

$$\mathcal{D}_{\text{collapse}}(\sigma, \Lambda) = \begin{cases} \Lambda & \exists j : |\sigma_j| < \epsilon, \\ 0 & \text{else,} \end{cases} \quad (13)$$

where ϵ is the singularity threshold. Unlike (12), $\mathcal{D}_{\text{collapse}}$ contains no smooth terms, and thus, we provide a separate definition of its pseudo-gradient by devising how σ should be modified to unfold a degenerate simplex. Specifically, if $\dim(\text{Ker}(df_c)) = k$, then $\sigma_{m-k+1}, \dots, \sigma_m = 0$, and thus $\partial \mathcal{D}_{\text{collapse}} / \partial \sigma_j$ should be non-zero for $m - k + 1 \leq j \leq m$. Consequently, we set gradient entries as follows:

$$j = 1, \dots, m : \frac{\partial \mathcal{D}_{\text{collapse}}}{\partial \sigma_j} = \begin{cases} -1 & |\sigma_j| \leq \epsilon, \\ 0 & \text{else.} \end{cases} \quad (14)$$

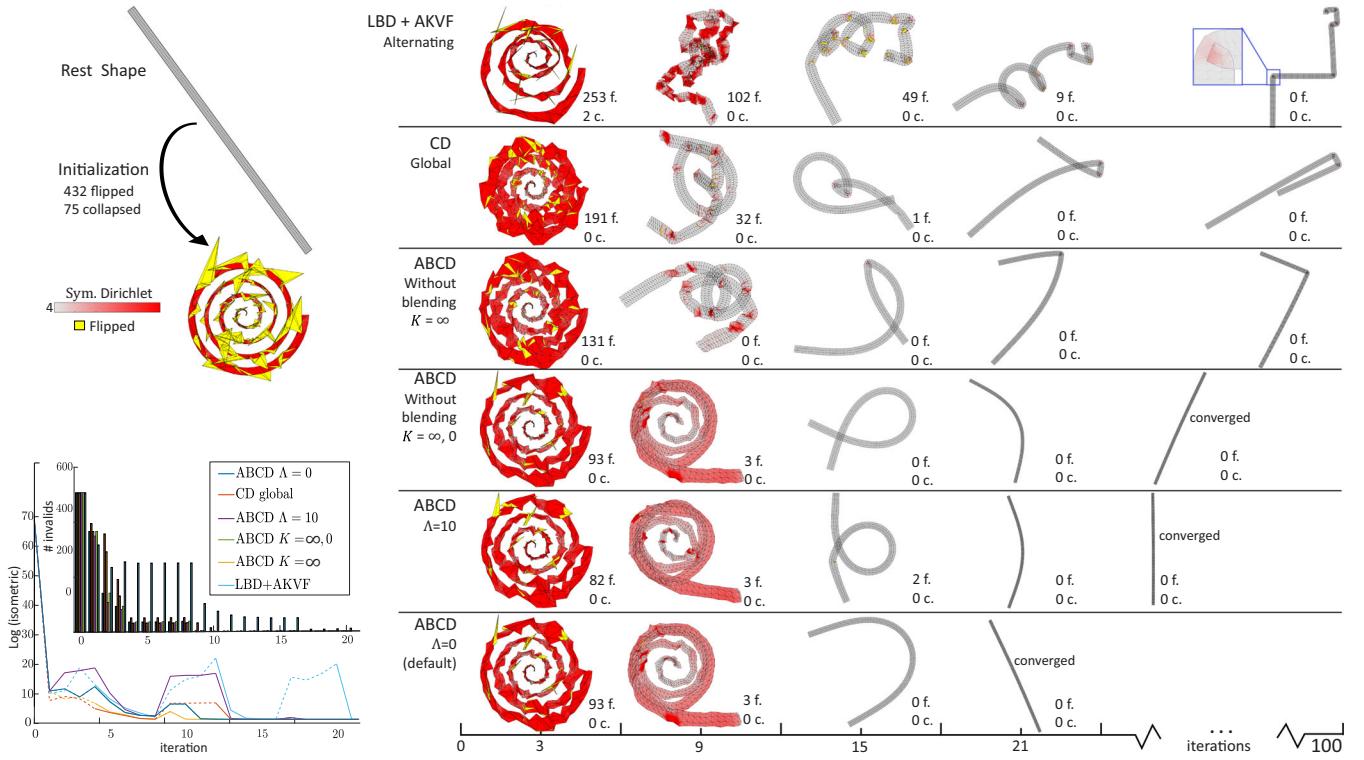


Figure 5: Comparing ABCD(AKVF) (bottom) with other methods and with simplified variants of ABCD(AKVF). Top to bottom: alternating optimization of LBD and AKVF; global ABCD version (CD); ABCD without blending that includes constant $K = \infty$ and constantly alternating thresholds $K = \infty, 0$; ABCD with $\mathcal{F}(\sigma, 10)$ and $\mathcal{F}(\sigma, 0)$. Each snapshot is annotated with the number of flipped and collapsed triangles, denoted by ‘f.’ and ‘c.’, respectively.

For processing all invalid elements with a single measure, we combine (12) and (13) into the *invalid simplex penalty*:

$$\mathcal{F}(\sigma, \Lambda) = \begin{cases} \mathcal{D}_{\text{flip}}(\sigma, \Lambda) & \sigma_1 < 0, \sigma_m \geq \epsilon, \\ \mathcal{D}_{\text{collapse}}(\sigma, \Lambda) & \text{else,} \end{cases} \quad (15)$$

where we set a higher priority for penalizing flips, since minimizing (12) is a more complex geometric problem than unfolding collapsed elements. Measure (15) is designed to behave well in GD. In particular, GD minimization of \mathcal{F} either flips back foldovers, or shrinks them to minimize magnitudes of their volumes (Figure 4).

Note that despite \mathcal{F} having non-differentiable points and separate definitions of energy values and energy gradients, map fixer measures are fully compatible with the geometric optimization framework. Particularly, Algorithm 3 receives \mathcal{D} and $\nabla \mathcal{D}$ in separate inputs and it can handle non-differentiable measures as long as these two inputs are consistent.

At first glance, it seems reasonable to set $\Lambda \gg 0$ in (15), since we are interested in minimizing the total number of invalid elements, rather than merely decreasing their total volume. However, the smoothness of $E_{\mathcal{F}} = \sum w(c)\mathcal{F}(\sigma, \Lambda)$ decreases in Λ . If Λ is large enough, then $E_{\mathcal{F}}$ is approximately equal to the number of invalid simplices times Λ . This quantity has a combinatorial structure, and thus minimizing $E_{\mathcal{F}}$ with common geometric solvers is not efficient for large Λ . Therefore, on average, the most significant

progress is achieved by setting Λ to be a tiny positive number. Our experiments with different values of Λ are depicted in Figures 5 and Figure S24 in the Supporting Information.

By definition, $\nabla \mathcal{F}$ is non-zero only on vertices of invalid elements, often distributed in patterns of ‘islands’ surrounded by positively oriented simplices (see second row of Figure 17). Following this observation, we suggest to construct block partitioning in which vertices sharing only inverted simplices and the rest of vertices are put in separate blocks. As discussed in the next section, our adaptive coordinate descent strategy has a number of apparent advantages over purely global or local approaches.

4. Local Versus Global Optimization

While most of optimization techniques are focused on approximating (1) with proxy energies, or on designing gradient preconditioner in (5), we adopt a new viewpoint and address a different aspect of the problem: How to divide objective variables \mathbf{x} into blocks to attain more efficient and robust optimization?

The vast majority of existing geometric solvers are *global methods* that update entire set of target coordinates simultaneously. Among the few purely *local methods* are block coordinate descent (BCD) strategies with constant small blocks [HG00, FLG15, FL16, NSZ18]. Note that our terminology here is different from the

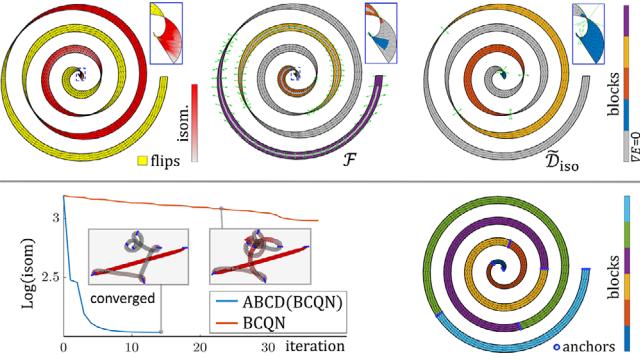


Figure 6: Top, from left to right: Unconstrained swirl planar deformation, initialized with foldovers, block partitioning in ABCD(GD) for F and \tilde{D}_{iso} energies, respectively. Bottom: We compare BCQN and ABCD(BCQN) optimizations (left) of the constrained swirl deformation, initialized without flips (right).

Algorithm 1. Vertex partitioning into blocks

Input:

- Mesh (\mathbf{V}, \mathbf{C}) with source and target coordinates \mathbf{y} and \mathbf{x} .
- Descent field \mathbf{d} and partitioning threshold K .

Output:

- 1: $\mathbf{E} \leftarrow \emptyset$.
- 2: **for** $v \in \mathbf{V}$ **do**
- 3: **for** $u \in \text{Neighbours}(v)$ **do**
- 4: Add $\mathbf{E} \leftarrow \mathbf{E} \cup \{u, v\}$ **if** $\mathcal{L}_{uv}(\mathbf{d}, \mathbf{x}, \mathbf{y}) \leq K$ and $u \sim v$.
- 5: **end for**
- 6: **end for**
- 7: $(\mathbf{V}, \mathbf{E}) \leftarrow$ Graph of nodes \mathbf{V} and edges \mathbf{E} .
- 8: $\mathbf{B} \leftarrow$ Connected components of (\mathbf{V}, \mathbf{E}) .

common notation of local-global solvers, such as [PDZ*18, RPPSH17, LX*08], where an update (5) is performed in the global stage, while auxiliary properties per simplex are computed in the local one. We consider both the locality and the globality with respect to the selection of \mathbf{x} 's coordinate blocks in (5).

To highlight the importance of the raised question, consider a toy model, depicted in Figure 2, where a simple 1-ring block GD [HG00, NSZ18] easily outperforms much more advanced global solvers. In general, the global strategy works well when descent direction \mathbf{d} is a smooth vector field. However, a noisy initialization produces chaotic gradients. Therefore, in this case, a global descent update of (5) can easily get stuck in the line search stage, since neighbouring vertices may be forced to move in completely random directions. In particular, repairing initializations with many flipped and collapsed simplices often introduces highly non-smooth gradients, and thus, cannot be effectively handled by a global approach alone. Although [ZBK18] proposed some line search enhancements, where components of the global descent field \mathbf{d} are rescaled to increase the line search range, the global update of \mathbf{x} along \mathbf{d} may still be subjected to an adverse coupling of vertices in highly distorted initializations. Moreover, a sufficiently high noise

level in vertex position often leads to poor preconditioning due to extremely ill-conditioned approximate Hessian. Hence, this issue cannot be solved solely by filtering line search directions or using other related approaches.

Another motivation for using an intelligent block partitioning is derived from observing standard optimization problems with Dirichlet positional constraints. This type of constraints often result in a natural block partition, where inner block boundaries pass through fixed vertices (*anchors*) with vanishing descent directions. Similar configurations occur in minimizing map fixer measures, since they have non-zero gradients only on vertices adjacent to invalid elements. Optimizing one block of vertices has no effect on vertices in another block if the boundary between the two blocks is locked (see Figure 6 bottom). Therefore, in such cases, it is more computationally efficient to employ BCD inherent parallelism for processing different blocks of vertices in parallel.

However, local methods are not effective in dealing with problems where the decrease in a distortion energy can be achieved only by modifying large groups of vertices, at once. These cases often include complex shape deformations, such as the one depicted in Figure 6, and surface parametrization with standard harmonic initializations. Apparently, as illustrated by Figure 5, distortion minimization can have mixed cases, where neither purely global, nor local approaches are suitable, and thus some blending of two opposite strategies should be adopted.

5. Adaptive Block Partitioning

In this section, we propose an *adaptive block partitioning* strategy that solves problem (2) separately over blocks of target vertices $\{\mathbf{x}_B | B \in \mathbf{B}^i\}$, where the subscript in \mathbf{x}_B denotes the column stack of coordinates of vertices in B and the superscript in \mathbf{B}^i denotes that the block partitioning is recomputed for each iteration i . Similarly, we denote by \mathbf{d}_v , \mathbf{x}_v and \mathbf{y}_v the search direction, target and source coordinates of a single vertex $v \in \mathbf{V}$, respectively.

Our partitioning scheme is designed to disconnect *static vertices* $\mathbf{V}_0 = \{v \in \mathbf{V} | \mathbf{d}_v = \mathbf{0}\}$ from the rest of the mesh and to group non-static (*free*) vertices $\mathbf{V} \setminus \mathbf{V}_0$ into edge-connected blocks with a sufficiently smooth descent field. More precisely, assume that $\mathbf{d}(\nabla E, \nabla^2 E)$ is the descent direction and denote by $u \sim v$ that u and v are neighbouring vertices such that both are in \mathbf{V}_0 or in $\mathbf{V} \setminus \mathbf{V}_0$. Then, we divide vertices in such a way that each block B is a connected set with respect to edges $\{u, v\} | u \sim v\}$ and it satisfies:

$$\forall u, v \in B : \|\mathbf{d}_u - \mathbf{d}_v\|_2 \leq K \|\mathbf{x}_u - \mathbf{x}_v\|_2, \quad (16)$$

where K is the adjustable threshold value that controls maximal deviation in descent direction. Since ‘geometric information’ cannot propagate through static vertices, a block surrounded by edges $\{u, v\}, u \sim v$, can be updated independently of the rest of the mesh. Motivated by this observation, we construct a *connectivity graph* (\mathbf{V}, \mathbf{E}) in Algorithm 1 by iterating over vertices $u \sim v$ and measuring the following quantity:

$$\mathcal{L}_{uv} = \|\mathbf{d}_v - \mathbf{d}_u\|_2 \|\mathbf{y}_v - \mathbf{y}_u\|_2 \|\mathbf{x}_v - \mathbf{x}_u\|_2^{-1}. \quad (17)$$

The term $\|\mathbf{y}_v - \mathbf{y}_u\|$ is chosen so that it normalizes distances in the target domain. As specified in Algorithm 1, $\{u, v\}$ is an edge

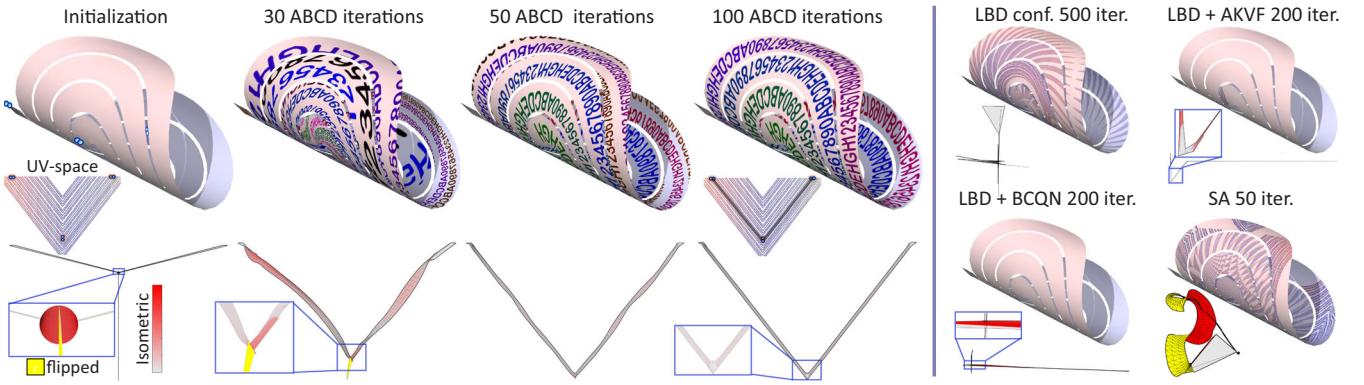


Figure 7: Constrained parametrization of a space fitting curve. The source and target anchor positions are shown in the top-left corner. Free vertices are initialized by Tutte embedding. ABCD(AKVF) results (left) are compared with other methods (right): stand-alone conformal LBD, alternating combinations of LBD with AKVF and BCQN, and SA. All the methods, except LBD, optimize isometric distortions.

of \mathbf{E} if $u \sim v$ and $\mathcal{L}_{uv} \leq K$, and blocks \mathbf{B}^i are connected components of (\mathbf{V}, \mathbf{E}) . Static blocks, obtained in Algorithm 1, include anchors and vertices that share only simplices with zero distortion energy.

Our default approach to controlling vertex partitioning in Algorithm 1 is to set $K = \infty$. In this case, blocks \mathbf{B}^i are connected components of the graph $(\mathbf{V}, \{\{u, v\} \mid u \sim v\})$. If B_1, \dots, B_N are free blocks, obtained in this partitioning scheme, then optimizing separately these blocks is more effective than using a global solver because BCD methods update target coordinates in (5) with different step sizes $\Delta t_{B_1}, \dots, \Delta t_{B_N}$. This statement is proved in Appendix A. Note that if there are no inversions, then setting $K = \infty$, in most cases, results in a global optimization of unconstrained vertices. Next, we discuss another strategy in which parameter K is modified to control further partitioning of free vertices into smaller blocks.

5.1. Local-global blending

In continuous settings, condition (16) is equivalent to requiring \mathbf{d} to be a K -Lipschitz (*continuous*) function of \mathbf{x} in B . The block selection rule (16) is motivated by the well-known fact that optimization of K -Lipschitz functions converges fast for small K . However, a fast convergence of distortion energy in small blocks does not guarantee the same for the total energy (1), since it may take time for ‘information’ to propagate from one block to another. Therefore, parameter K should be well tuned to obtain an optimal trade-off between the number of blocks and the smoothness of \mathbf{d} , obtained in each block. A simple approach to control BCD is to alternate between number of constant thresholds parameters. For instance, one can alternate between global optimization and the finest partitioning, achieved by setting $K = 0$. However, this strategy is often wasteful, since, as illustrated by Figure 5, a significant part of the obtained iterations can be ineffective or even counterproductive. Moreover, as explained in Section 4, using purely global approach often results in slow progress at the beginning of the optimization, due to the presence of highly distorted elements.

We propose to estimate an optimal parameter K for vertex partitioning by means of the gradual blending between two opposite strategies: (1) partitioning into connected components of ‘ \sim ’, obtained if $K \geq \mathcal{L}_{\max} = \max_{u \sim v} \mathcal{L}_{uv}$; (2) fine partitioning into small blocks, obtained for a low value of K , which we denote by \mathcal{L}_{\min} . We refer to the proposed approach as *local-global blending* (LGB). Starting with $K^1 = \mathcal{L}_{\max}$ and $K^2 = \mathcal{L}_{\min}$ at the first and second iterations, we steadily update partitioning thresholds K^i for next iterations according to the observed progress towards achieving an optimal number of coordinate blocks.

Assume that \mathcal{P}^1 and \mathcal{P}^2 are estimates of the optimization progress achieved at the first two iterations, respectively. Then, $\mathcal{R}^{1/2} = \mathcal{P}^1/\mathcal{P}^2$ is the rate by which the first iteration outperforms the second one. Our blending procedure is designed to either increase or decrease the initial coefficients K^1, K^2 in such a way that their values are changed in proportion to $\mathcal{R}^{1/2}$. The same process is repeated for each successive pair of parameters (K^{2s-1}, K^{2s}) , $s \geq 2$. That is, we compute K^3 and K^4 according to Algorithm 2 at the third iteration, then we apply the algorithm again at the fifth iteration to compute (K^5, K^6) as a function of $\mathcal{R}^{3/4}, K^3$ and K^4 . This process is repeated until the blending termination criterion is reached (see Section 7.4).

It is difficult to predict how the non-linear optimization (2) evolves by observing energy values or energy derivatives alone. In fact, a slight modification of a single nearly collapsed simplex may consume more energy than a complex deformation of the entire shape. We therefore estimate optimization progress \mathcal{P}^i by observing both the decrease in the relative energy

$$\Delta E^i = \frac{E(\mathbf{x}^{i-1}) - E(\mathbf{x}^i)}{E(\mathbf{x}^{i-1})}, \quad (18)$$

and the magnitude of non-rigid motion by which target vertices are moved. We refer to the latter quantity as *displacement norm*, defined via Frobenius norm as follows:

$$\text{Disp}^i = \|\mathbf{x}^i - \mathbf{x}^{i-1} - \text{Proj}_{\text{Ker}(\mathbf{x}^{i-1})}(\mathbf{x}^i - \mathbf{x}^{i-1})\|_{\text{Fro}}, \quad (19)$$

where $\text{Ker}(\mathbf{x}^{i-1})$ denotes the linear space of rigid transformations of the target shape [CBSS17].

Figure 8 depicts optimization progress and plots partitioning thresholds, computed by Algorithm 2. Further details of the local-global blending are presented in Appendix B.

Algorithm 2. Local-global blending (LGB)

Input:

- Partitioning thresholds K^1, K^2 of the previous two iterations.
- Global-local performance ratio \mathcal{R} .
- \mathcal{L}_{\max} = maximum of coefficients in (17).

Output: Thresholds K^3, K^4 for the next two iterations.

```

1: if  $\mathcal{R} \geq 1$  then
2:    $(K^3, K^4) \leftarrow (\mathcal{L}_{\max}, K^1) - \frac{1}{\mathcal{R}}(\mathcal{L}_{\max} - K^1, K^1 - K^2)$ .
3: else
4:    $(K^3, K^4) \leftarrow (K^2 + \mathcal{R}(K^1 - K^2), \mathcal{R}K^2)$ .
5: end if

```

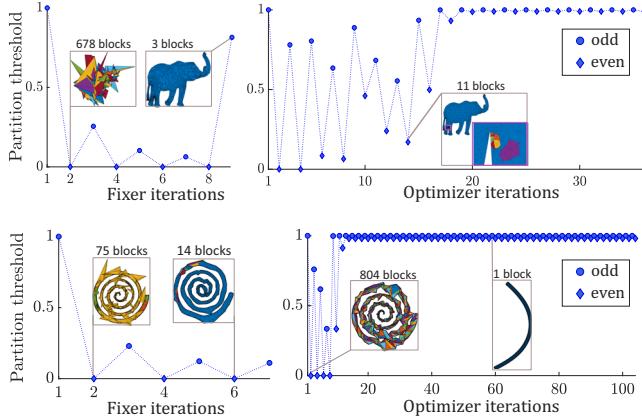


Figure 8: Normalized partitioning thresholds (K/\mathcal{L}_{\max}) computed by Algorithm 2 for distortions \mathcal{F} (left) and $\widehat{\mathcal{D}}_{iso}$ (right). Snapshots are coloured according to vertex partitioning.

6. Cured Alternating Optimization

In this section, we address the problem of optimizing multiple distortion measures. The main challenge in implementing ABCD stems from the fact that the algorithm copes with minimization of two potentially competing measures—distortion measures in a fixer phase (e.g. \mathcal{F}) versus distortion measures in optimization phase (e.g. isometric and conformal distortions)—reducing one might temporarily increase the other measure. Denote by E_1 and E_2 distortion energies of these two measures, obtained according to (1). One has to be cautious about how to combine E_1 and E_2 into a single optimization scheme. Roughly speaking, there are three options to be considered:

- processing measures in a *cascade*, i.e. optimizing the first energy till convergence before moving to the next one;
- combining all measures into a single objective. For example, optimizing a sum of energies

$$E_{\text{single}}(\mathbf{x}) = \alpha_1 E_1(\mathbf{x}) + \alpha_2 E_2(\mathbf{x}); \quad (20)$$

- processing measures in an *alternating* manner. That is, repeatedly cycle through E_1 and E_2 .

The cascade approach is not robust enough because, often, there are inversions that cannot be repaired by fixer alone without deforming valid simplices.

Often, minimizing a single objective (20) is not productive, since vertex positions can be locked easily in the regions of vanishing or inconsistent descent directions. Although alternating optimization of different energies has no guarantees to avoid similar difficulties, we found that the alternating scheme works best. Figures 7, 17 and 18 show that this option is much more robust than the others. In fact, in all these trials, we have not encountered any instances in which the optimization got stuck. Even for hard constrained problems with randomized initializations, optimization proceeds until convergence with no trouble. We posit that the reason for such a remarkable performance of the algorithm is the combination of distortion enhancements, introduced in Section 3, and our adaptive block partitioning method. In particular, our modified distortion measures significantly reduce any possible instabilities due to competing measures of the different optimization phases. In addition, our partitioning strategy avoids vertices being interlocked by allowing a separate processing of valid and invalid simplices.

It is important to mention that our algorithm is based on *inexact* block coordinate minimization of each individual measure E_j . That is, we optimize each block for a small number of successive iterations and repeat this process before the algorithm proceeds to the next measure in the queue. This is in contrast with the approach in which E_j , in its turn, is minimized until convergence or up to the point where the progress is lost. Advantages of the inexact approach were observed in number of recent studies [TRG16, FLG15]. When optimizer and map fixer distortions are alternatively optimized using the inexact approach, these measures support each other in a straightforward manner.

When the minimization of (20) is getting stuck, some recent papers propose to try to smoothen the objective more and more, until the optimization is able to resume again. This strategy may work well in Autocuts algorithm [PTH*17], since it is flexible enough to split problematic regions into disconnected sets. However, our tests show that the single objective approach is much less effective in seamless parametrization, and in parametrization with given cuts, as well as in more general 2D and 3D applications, where neighbouring simplices cannot be disassembled. The uniqueness of ABCD approach is in preserving target mesh topology during the entire optimization. In fact, dividing the vertex set into blocks in our algorithm can be considered as a ‘weak’ separation between simplices, which, unlike the standard simplex disassembling, does not duplicate even a single vertex.

7. ABCD Optimization Procedure

We are finally in position to present the full ABCD implementation, whose pseudo-code is given in Algorithm 3. This implementation integrates all the introduced strategies into four major steps, illustrated schematically in Figure 3. In addition to these stages, our procedure includes a number of sub-steps described below. Some

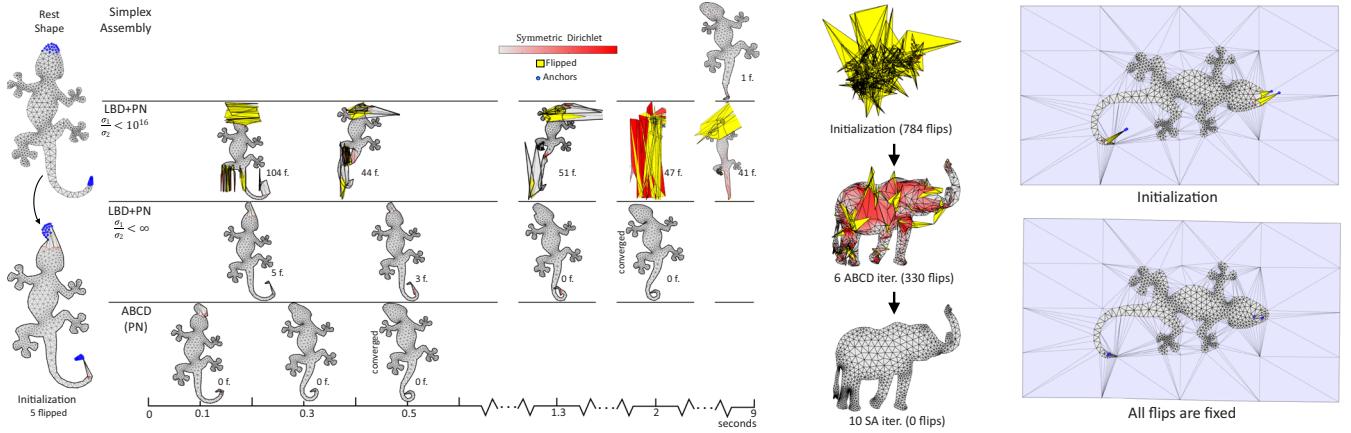


Figure 9: Left: Comparing ABCD(PN) with LBD+PN and SA for a planar problem, initialized with flips. We used the following parameters for LBD: finite unbounded condition number $\sigma_1/\sigma_2 < \infty$ and $\sigma_1/\sigma_2 < 10^{16}$. Middle: Running SA with first six ABCD iterations to solve the failure case of SA from Figure 17. Right: Using scaffold meshes to induce a globally injective map in ABCD(PN). Showing the two stages: anchors are moved to initialize the problem (top), all inversions are repaired (bottom).

of these sub-steps are unique for our approach, whereas others are well-known methods that we have adjusted to our needs.

7.1. Energy sequence specifications

To achieve its goal, the list of energies in ABCD should contain both an invalid simplex penalty (15) and an optimizer distortion $\tilde{\mathcal{D}}$, defined according to (10). In our experiments, we minimize energies \mathcal{F} and $\tilde{\mathcal{D}}$ and pick a small number of successive iterations n_j for each measure to exploit best the alternating optimization. We always begin with \mathcal{F} to provide a better starting point for the optimizer.

Not that the exact mix of the energies can be very general and there is also some flexibility in setting optimization parameters. We explore a more general configurations for ABCD in Table S1 and in Figure S24 in the Supporting Information. According to these results, our algorithm behaves consistently across different settings.

7.2. Core-solver specifications

Each energy E_j , can, in principle, be minimized by a different solver in Algorithm 3. Therefore, we can have S_1 and S_2 solvers, where each solver should meet the following two criteria: (i) it optimizes rotation invariant distortions; (ii) it is a line search based, i.e. it modifies vertex positions along the obtained descent direction. Although very general solvers can be used, it is important to keep in mind two things related to their usage:

- Often, at the beginning, iterations include one-ring vertex blocks in which expensive core-solvers lose their efficiency. After experimenting with various scenarios and estimating an average time required for solving (7) in first- and second-order methods, we suggest GD as a solver of choice for optimizing a single vertex block. Note that, in general, small blocks disappear soon after all elements attain positive orientation.
- We also recommend to use GD for minimizing \mathcal{F} . A simple GD actually works better for this measure than more advanced

solvers, both when optimizing over small and large blocks. The reason for this is that \mathcal{F} is specifically designed to fit GD, whereas state-of-the-art solvers, such as CM or SLIM, are designed to minimize isometric and conformal distortions. Moreover, in GD, vertices with vanishing gradients belong to static blocks, since they have a zero descent direction. As a result, ABCD fixer with GD updates only vertices that share invalid elements. Therefore, if only a small fraction of simplices are flipped, then fixer iterations run much faster with GD than with other solvers.

7.3. Enhanced line search filtering

The line search sub-step is aimed at modifying vertex coordinates for the next iteration along the given block descent field, i.e. $\mathbf{x}_B^{i+1} = \mathbf{x}_B^i + \Delta t_B^i \mathbf{d}_B^i$. We use the Armijo backtracking search to estimate a sufficiently good minimizer

$$\Delta t_B^i = \underset{t \in [0, T_{\max}]}{\operatorname{argmin}} E(\mathbf{x}_B^i + t \mathbf{d}_B^i). \quad (21)$$

We compute T_{\max} in (21) differently for map fixer measures and for optimizer distortions, defined by (10). For optimizer distortions, we employ the filtering of [SS15] over valid elements

$$T_{\max} = (1 - \delta^+) t_B^+, \quad (22)$$

$$t_B^+ = \min_{c \in C^+(B)} \{\infty, t > 0 \mid \det(df_c[t]) = 0\}, \quad (23)$$

where $C^+(B)$ denotes the set of positively oriented simplices that share vertices in B , $df_c[t]$ is the Jacobian of c , induced by mapping vertices in B to $\mathbf{x}_B^i + t \mathbf{d}_B^i$, and $\delta^+ > 0$ is a small number that we choose to avoid numerical errors on nearly collapsed elements.

Denote by $C^-(B)$ the set of inverted simplices adjacent to vertices in B . Then, for map fixer distortions, such as \mathcal{F} , the length of the search interval is set to $T_{\max} = (1 - \delta^-) t_B^-$, where t_B^- is the minimum of the right side of (23), computed over $C^-(B)$, and $\delta^- < 0$ is a negative number. Here, we use $\delta^- < 0$ instead of δ^+ because often

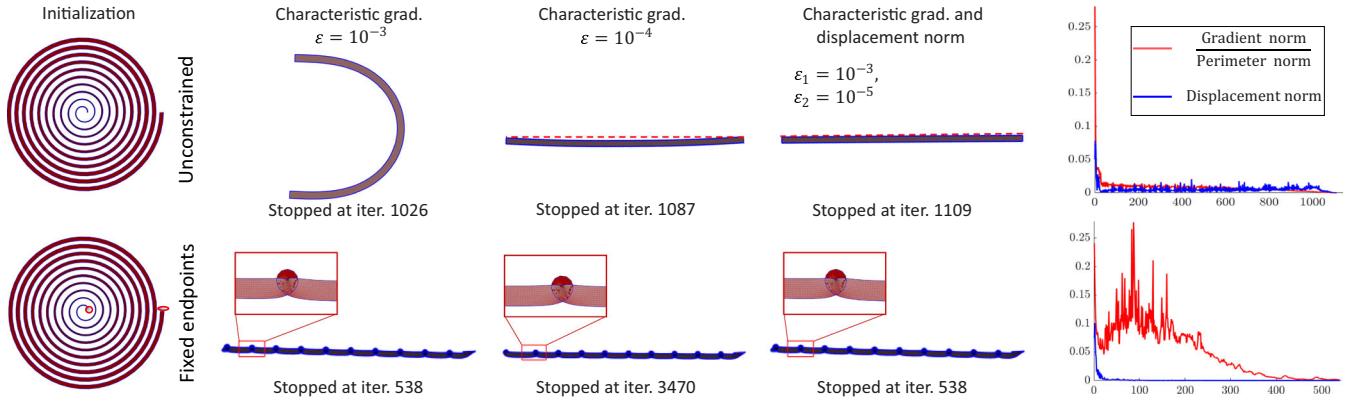


Figure 10: Characteristic gradient termination criteria [ZBK18] is compared against enhanced criteria (24). We test these criteria with BCQN solver and \mathcal{D}_{iso} distortion for unconstrained swirl deformation (top) and for the same problem with fixed endpoints (bottom).

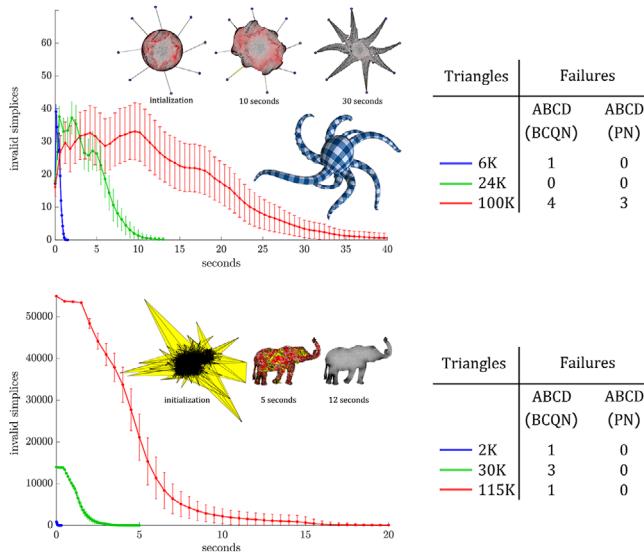


Figure 11: Testing ABCD on randomly generated deformations that include 600 trials of constrained parametrization with randomly placed anchors (top) and 600 trials of planar shape deformation with fixed boundary and randomized initialization of interiors (bottom). We tested 100 problems with Eigen library for three resolution levels: 2K, 30K and 115K triangles for the elephant model; 6K, 24K, 100K for the octopus. Left: Average number of flips in ABCD(PN) as a function of the runtime (vertical bars represent the standard deviation). Right: The number of failures encountered in a hundred trials of ABCD(BCQN) and ABCD(PN).

a minimizer of $E_{\mathcal{F}}(\mathbf{x}_B^i + t\mathbf{d}_B^i)$ lies outside the interval $[0, t_B^-]$. See Figure 4 for an illustration of the line search step.

7.4. Termination criteria

Unlike global solvers with a single objective, ABCD has several termination criteria—to stop processing of a current block, a current measure and the entire optimization, and the special criteria for

terminating or resetting the LGB. A successful operation of ABCD should encounter two phases: the first one of cleaning foldovers and the final one of $\tilde{\mathcal{D}}$ convergence. The first phase is completed when \mathcal{F} vanishes, and further processing is devoted solely to $\tilde{\mathcal{D}}$. Thus, the termination of an individual measure is claimed when its average value reaches the optimum. The final convergence criteria are met by either reaching the maximal iteration number (i_{max}), or by arriving at sufficiently accurate solution. To recognize such solution, we employ both the characteristic gradient (perimeter) norm [ZBK18] and the average of the proposed displacement norm (19). That is, we stop if the following two criteria are met:

$$\sum_j \|\nabla E_j\| < \varepsilon_1 \sum_j \text{Char}(E_j), \quad \frac{\text{Disp}^i(\mathbf{x})}{\Sigma w(c)} < \varepsilon_2, \quad (24)$$

where $\text{Char}(E_j)$ denotes the characteristic gradient norm of E_j . Although using $\text{Char}(E_i)$ is more robust than a direct gradient tolerance, we found that characteristic norm can still lead to incorrect termination claims if ε_1 is too high, or to redundant iterations if the threshold is too small. As exemplified by Figure 10, we can solve failure cases of the perimeter norm criterion by adding the displacement norm check, since it provides a simple, yet reliable, way for ensuring the absence of any further progress.

We use (24) for terminating optimization in blocks, too. However, we bound block iterations by a small number b_j , to follow our paradigm of inexact optimization. To minimize redundant operations in non-optimally selected blocks, we start with $b_j = 1$ and increment it at each cycle. Note that the impact of b_j disappears once block partitioning reaches a steady state. We stop updating parameters K by Algorithm 2 after the number of blocks in even iterations becomes equal to that of odd iterations. However, we always track the progress over last iterations and, if there is no decrease in a distortion energy or in the number of invalid elements, then the thresholds K are reset to their initial values.

7.5. Implementation

We have written the main body of Algorithm 3 in Matlab for attaining a highly customizable interface that supports multiple

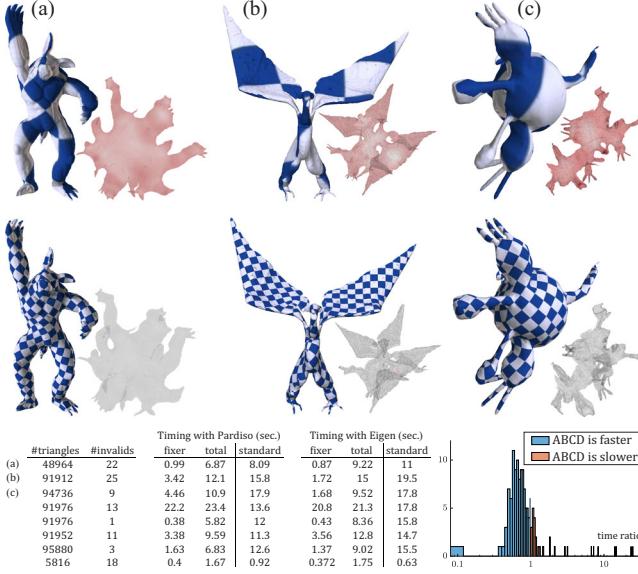


Figure 12: Using conformal flattening as a starting point for isometric parametrization. We depict few samples from our experiment in which BFF conformal maps contain flipped or collapsed triangles. Top: BFF maps. Middle: Isometric parametrization, obtained by ABCD(PN), initialized with BFF. Bottom: The table compares the speed of our method ('total') with the speed of the standard isometric parametrization with PN solver, initialized by Tutte ('standard'). The histogram depicts runtime ratio of our method and the standard one—when the ratio is less than 1, our method is faster.

core-solvers (GD, AKVF, BCQN, PN) and different variants of ABCD. Critical parts of the code were implemented in C++. We reimplemented AKVF and modified existing BCQN code to fully integrate these core-solvers with ABCD. The parallel version of our code supports PN solver and it is based on [CM83] graph colouring algorithm. We have tested Eigen and Pardiso libraries to solve (7) in ABCD(PN). We report runtimes for ABCD(PN) in Figures 9, 14, 12, 15 and 16 and in Figures S26, S21 and S24 in the Supporting Information.

8. Results

Since our algorithm unifies map fixers and core-solvers into a single framework, we compare our results with methods from both categories. To make a fair comparison, we examine different core-solvers for ABCD (BCQN, PN, AKVF, GD) and alternating combinations of state-of-the-art map fixers and core-solvers (e.g., LBD+BCQN, LBD+AKVF, LBD+PN, SA). To compare iterations of these methods with ABCD, we consider global solvers as BCD applied on a single block containing the entire vertex set. We rearrange alternating iterations of global solvers into the hierarchical structure of Algorithm 3, and then count overall number of b_j loops (line 9 of Algorithm 3).

We test meshes with different numbers of elements and summarize our results in Figures 5, 7, 11, 13 and 17 for 2D problems

Algorithm 3. Adaptive Block Coordinate Descent (ABCD)

Input:

- Source mesh (\mathbf{V} , \mathbf{C}) and vertex initialization \mathbf{x}^0 .
- Distortions \mathcal{D}_1 (fixer) and \mathcal{D}_2 (optimizer).
- Core solvers S_1 and S_2 with n_1 and n_2 successive iterations.

- 1: Initialize block iterations (b_1, b_2) and thresholds (K_1, K_2).
 - 2: **repeat** {starting with $i = 0$ }
 - 3: **for** $j = 1, 2$ **do** { n_j times each}
 - 4: $E_j \leftarrow \mathcal{D}_j$ energy of $f[\mathbf{x}^i]$ according to (1).
 - 5: **if** average(E_j) $\leq \min \mathcal{D}_j$ **continue**.
 - 6: $\mathbf{d} \leftarrow$ descent direction of E_j attained by S_j .
 - 7: $\{\mathbf{x}_B^j\}_{B \in \mathbf{B}} \leftarrow$ BlockPartitioning(K, \mathbf{d}) via Algorithm 1.
 - 8: **for** $B \in \mathbf{B}$ in parallel **do**
 - 9: $\mathbf{x}_B^{j+1} \leftarrow$ solve (2) in B by S_j with b_j max iterations.
 - 10: **end for**
 - 11: $i \leftarrow i + 1$.
 - 12: Update b_j and K_j . [without blending use $K_j = \infty$]
 - 13: **end for**
 - 14: **until** (24) is true for each E_j or $i \geq i_{\max}$.
-

and in Figures 1, 18 and 16 for volumetric problems. We annotate our results as follows: ABCD(S) denotes that GD and S are the fixer and optimizer solvers, respectively; $S_1 + S_2$ refers to the alternating combination of S_1 and S_2 methods in which iterations are divided between S_1 and S_2 in the same way as in our algorithm. We use ARAP distortion $\tilde{\mathcal{D}}_{\text{ARAP}}$ in Figure 16 and $\tilde{\mathcal{D}}_{\text{ISO}}$ distortion in other ABCD trials, where $\mathcal{D}_{\text{ISO}} = \sigma_1^2 + \sigma_1^{-2} + \dots + \sigma_m^2 + \sigma_m^{-2}$ and $\mathcal{D}_{\text{ARAP}} = (\sigma_1 - 1)^2 + \dots + (\sigma_m - 1)^2$.

Firstly, we compare LGB strategy with the block partitioning scheme in which the partitioning threshold is constant, $K = \infty$. We conclude that ABCD with LGB is more robust, but it has a higher computation cost at the beginning of the optimization. The impact on the algorithm speed is not significant on meshes with a few dozens of thousands of simplices or less. However, in higher resolution, we recommend either to increase the value of \mathcal{L}_{\min} or to use ABCD without LGB.

Next, we compare ABCD with LBD + AKVF in Figure 5, to show that our strategies support each other, and, when combined together, attain the most significant progress. These and other related experiments with shape deformations and constrained parametrization indicate that ABCD is more likely to avoid poor local minima, where positively oriented maps are non-locally injective (see Figure 5 top-right). Although we cannot guarantee avoiding cases of poor local minima, our method is very general and it can be further modified for inducing globally injective mapping. This subject is briefly discussed in Appendix C and our experiment with scaffold meshes is depicted in Figure 9 (right).

Methods from Figure 5 perform similarly over a wide range of unconstrained and constrained problems in 2D and 3D. Furthermore, as shown in Figure 11, we have tested ABCD on 600 models with randomly generated initializations and encountered only few failures, while competing methods (LBD, SA) appear to fail constantly on every trial. These tests show that ABCD is well scalable and can recover from extremely distorted meshes of a high

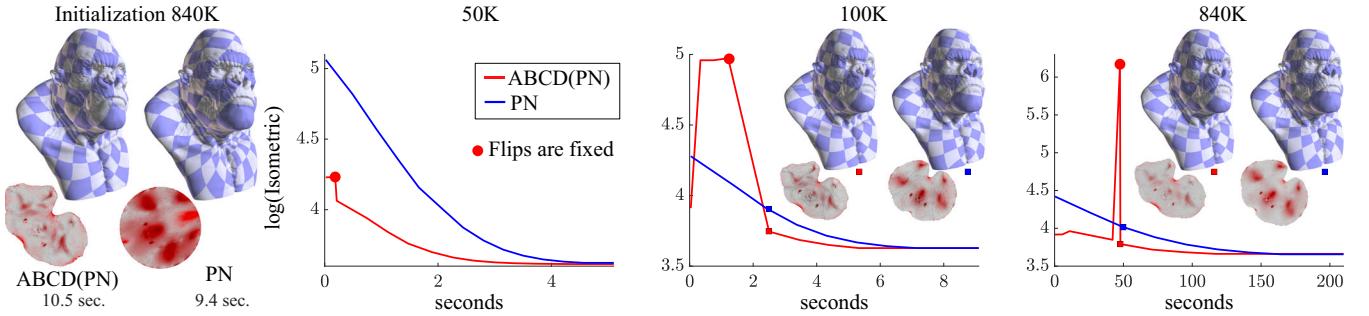


Figure 13: We compare PN parametrization, initialized by Tutte embedding, and ABCD(PN) parametrization, initialized by mapping surfaces into UV-map of a decimated mesh. Specifically, we parametrize a mesh with 2K triangles and use the deformation embedding method [NZ19] for initializing the problem in a higher resolution (see Figure S25 in Supporting Information). We run 12 iterations for each mesh resolution and depict \tilde{D}_{iso} as a function of the runtime. In these examples, we used Eigen library for linear algebra.

| Problem in 2D | #triangles | #invalid | #vertices / #anchors | timing to fix (no LGB) (LGB) | iterations to fix | timing total (no LGB) (LGB) | iterations total |
|---------------|------------|----------|----------------------|------------------------------|-------------------|-----------------------------|------------------|
| Gecko | 1238 | 5 | 804 / 45 | 0.04 0.121 | 3 7 | 0.17 0.25 | 13 22 |
| Elephant | 1796 | 412 | 1105 / 788 | 0.203 0.173 | 11 9 | 0.32 0.35 | 18 28 |
| Octopus UV | 5986 | 18 | 3924 / 8 | 0.469 0.313 | 13 9 | 0.91 0.89 | 20 27 |
| Grid 100x100 | 20000 | 4149 | 10201 / 0 | 1.105 1.571 | 9 7 | 2.38 3.11 | 15 20 |
| Octopus UV | 23944 | 18 | 13833 / 8 | 2.022 4.757 | 15 49 | 2.58 5.82 | 17 57 |
| Elephant | 28736 | 14038 | 15193 / 1648 | 2.178 5.819 | 12 38 | 4.33 7.10 | 21 47 |
| Octopus UV | 95776 | 19 | 51609 / 8 | 9.414 13.779 | 20 45 | 14.82 17.80 | 27 55 |
| Gorilla UV | 100000 | 7 | 50379 / 0 | 0.463 0.473 | 3 3 | 2.89 3.10 | 6 8 |
| Elephant | 114944 | 54900 | 59121 / 3296 | 7.403 7.023 | 12 14 | 14.06 13.08 | 20 28 |
| Gorilla UV | 839092 | 20 | 420408 / 0 | 5.736 5.828 | 3 3 | 34.85 37.41 | 7 10 |

| Problem in 3D | #tets | #invalid | #vertices / #anchors | timing to fix | iterations to fix | timing total | iterations total |
|-------------------------|-------|----------|----------------------|---------------|-------------------|--------------|------------------|
| Twisted bar | 12000 | 6961 | 2541 / 0 | 1.65 | 8 | 3.18 | 14 |
| Bar bending into spiral | 30000 | 104 | 6171 / 0 | 0.9 | 6 | 3.36 | 10 |
| Bar bending into spiral | 30000 | 104 | 6171 / 2 | 0.91 | 6 | 4.41 | 12 |
| Bar bending into spiral | 30000 | 104 | 6171 / 242 | 1.03 | 7 | 9.52 | 21 |
| Twisted wrench | 50122 | 120 | 14798 / 7942 | 2.16 | 9 | 5.52 | 13 |
| Bended wrench | 50122 | 322 | 14798 / 4792 | 18.2 | 49 | 23.7 | 53 |
| Bended wrench (ARAP) | 50122 | 322 | 14798 / 4792 | 7.04 | 20 | 11.8 | 25 |
| Armadillo | 56917 | 405 | 15791 / 662 | 12.4 | 24 | 20.1 | 29 |
| Armadillo (ARAP) | 56917 | 405 | 15791 / 662 | 7.27 | 18 | 14.6 | 23 |
| Dinosaur | 58191 | 141 | 16115 / 680 | 22.4 | 58 | 28.4 | 60 |

Figure 14: Reporting numbers of iterations and seconds until all invalid elements are fixed, and until the final convergence (24) of ABCD(PN) with $\varepsilon_1 = 10^{-3}$ and $\varepsilon_2 = 10^{-2}$. In 2D, we report both the results of ABCD without LGB and of ABCD with LGB for $\mathcal{L}_{min} = 0.1\mathcal{L}_{max}$. We used Pardiso linear solver to optimize \tilde{D}_{iso} (default) and \tilde{D}_{ARAP} distortions. All experiments were timed on the four-core i7-8565U CPU.

resolution. We compared ABCD(PN) for shape deformation problems with other related methods. As demonstrated in Figures 9 (left), 15 and 16, ABCD is both faster and more reliable than other techniques for computing positively oriented simplicial maps. Also, our experiments with different core-solvers show that ABCD(PN) is more robust than ABCD(S) for any tested first order solver S .

We examine failure cases of LBD and SA fixers in both 2D (Figures 17 and 9) and 3D (Figures 18 and 16), and find that these methods can, in general, handle only maps with relatively small fractions of inverted and collapsed simplices. These conclusions are exemplified for highly distorted initializations of the volumetric bar and planar model in Figures 18 and 17. Moreover, as illustrated in Figure 1, adding positional constraints reduces significantly LBD capabilities. Note that our experiments reveal that running LBD and state-of-the-art optimizers in the alternating manner is more effective than using stand-alone LBD or combining these methods into a single cascade.

We tested our algorithm for computing isometric parametrizations with fixed anchors and initial mappings to non-convex domains. Some of these scenarios are presented in Figures 7 and 17. To construct the starting point, we firstly compute the standard Tutte embedding, and then move anchors to their prescribed positions.

We experiment with alternative initialization schemes for surface parametrization. In Figures 13 and 17 (second row), we start with

Tutte mapping onto the non-convex region obtained by parameterizing a decimated mesh. That is, we initialize a problem by mapping a rest surface into the image of its UV-map, computed in a lower resolution (Section 12 in the Supporting Information). As demonstrated by Figure 13, this initialization scheme can accelerate UV-map computations for meshes of a high resolution.

In Figures 12 and S26 (Supporting Information), we compare the BFF conformal initialization scheme with the standard isometric parametrization. Specifically, we ran BFF conformal parametrization on 600 meshes from the dataset of [LYNF18] and collected 190 maps with invalid triangles. These maps were used to initialize ABCD(PN). In this experiment, we have only one mesh where ABCD(PN) failed to repair invalid elements in BFF, while Tutte embedding produced flipped or collapsed triangles in 20% of the trials. We tested the remaining meshes with valid Tutte maps and found that in 84% of the trials, our method ran faster than the standard parametrization, initialized by Tutte embedding.

As observed in our experiment, Tutte embedding has certain implementation-related limitations and, for challenging meshes, it can fail even if the target domain is convex. In particular, as demonstrated in [SJZP19], Tutte embedding may produce invalid triangles due to precision loss in the floating point arithmetic. We tested ABCD(PN) and PE on a challenging Tutte failure example, introduced in [SJZP19]. As demonstrated in Figure 15, ABCD ran faster and converged to a mapping with lower isometric distortion.

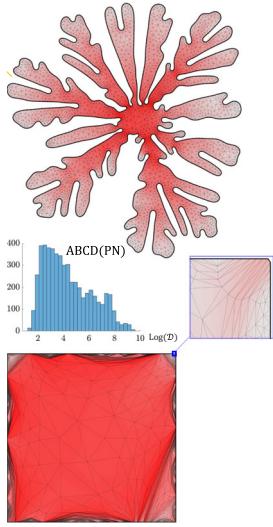


Figure 15: Mapping of the Hele-Shaw polygon [SJZP19] (top) onto the square (bottom) using ABCD(PN) (left) and the PE method (right). The initial Tutte embedding produces 41 flipped and 845 degenerate triangles. It takes 25 s to fix all invalid triangles by ABCD(PN), while PE runs 60 s. Histograms show final isometric distortions of triangles in a logarithmic scale.

According to our proposition in Appendix A, we presume that even for valid initializations, our vertex partitioning strategy can improve a core-solver performance, provided that positional constraints disconnect the mesh. See an example of such scenario at the bottom of Figure 6.

To summarize, we have demonstrated over a variety of geometric problems that ABCD exhibits superior performance compared to existing methods in terms of its speed, robustness to inverted and highly distorted simplices, and it can deal with more complex positional constraints. In particular, our algorithm can start with the proposed conformal initialization scheme to speed up isometric parametrization for challenging meshes.

9. Conclusion and Future Work

Until recently, a lack of locally-injective initialization was one of the major concerns for running geometric optimization. Our algorithm resolves this issue for a wide range of scenarios. In this, ABCD broadens the frontiers of tractable geometric processing problems and we believe that it can contribute a lot to modern design tools.

We suggest to further explore conformal flattening and Tutte mapping onto non-convex regions as an alternative starting point for accelerating parametrizations. We believe that these methods can be integrated with the recently proposed acceleration techniques [LYNF18, PDZ^{*}18] to attain even faster performance. Our algorithm is built upon a number of heuristics that work well empirically. However, we cannot prove that there is a guaranty to repair all invalid elements and reach the optimality. A supplementary mathematical analysis is required to better understand our empirical results and consider conditions for optimality.

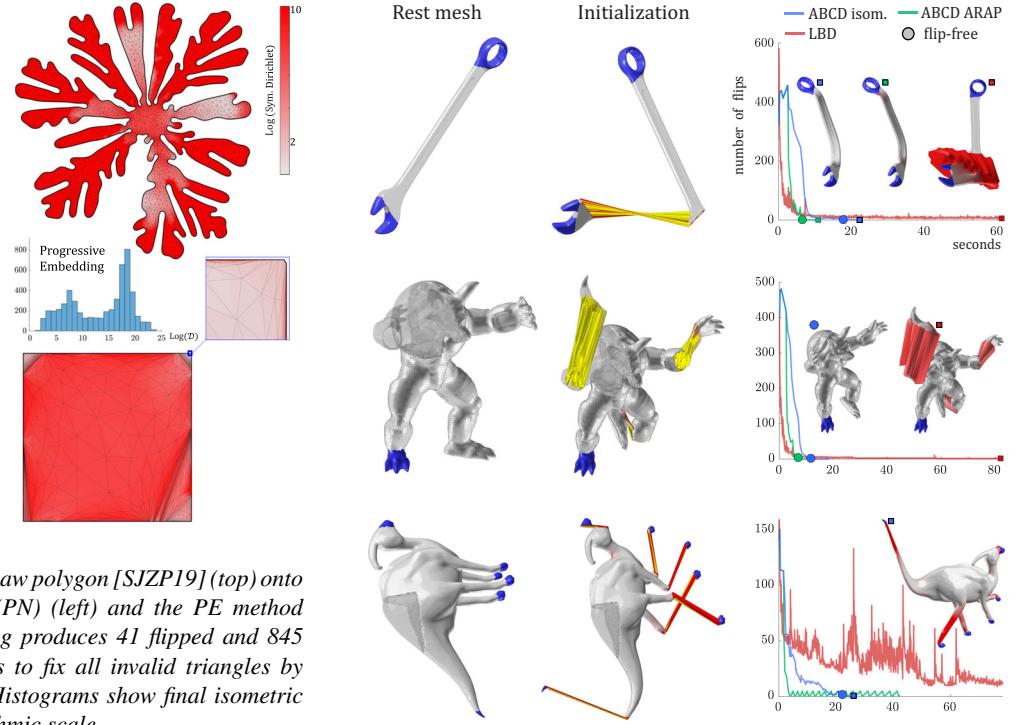


Figure 16: ABCD(PN) with \tilde{D}_{iso} and \tilde{D}_{ARAP} distortions versus LBD on tetrahedral meshes. We bound condition number in LBD by $\sigma_1/\sigma_3 < 10^2$. Left to right: Rest mesh, initialization with anchor points, marked in blue, and numbers of flipped tets per second.

Although a naive GD map fixer works well in our experiments, a more sophisticated approach can potentially yield even better results. For example, gradient preconditioning of many first-order solvers provides a better descent direction than a naive GD because preconditioning methods take into account the intrinsic metric of the deformation space. Unfortunately, at the moment it is not clear what is the right pre-conditioner for map fixer optimization steps. Considering a metric approach oriented at minimizing this class of geometric measures and finding the right pre-conditioner may be a promising direction for future research.

Acknowledgments

We thank Gregory Naitzat for proof-reading and giving valuable comments, and the anonymous reviewers for their constructive feedback. This research has been supported in part by the Ollendorff Minerva Center.

Appendix A: Proof of BCD Superiority

Proposition. Let S be a global core-solver, initialized by \mathbf{x}^0 . Denote by \mathbf{x} and $\bar{\mathbf{x}}$ results of running a single iterations of S and of $ABCD(S)$ with $K \geq \mathcal{L}_{\max}$, respectively. If the exact line search is used in (5) to minimize energy E , then $E(\bar{\mathbf{x}}) \leq E(\mathbf{x})$.

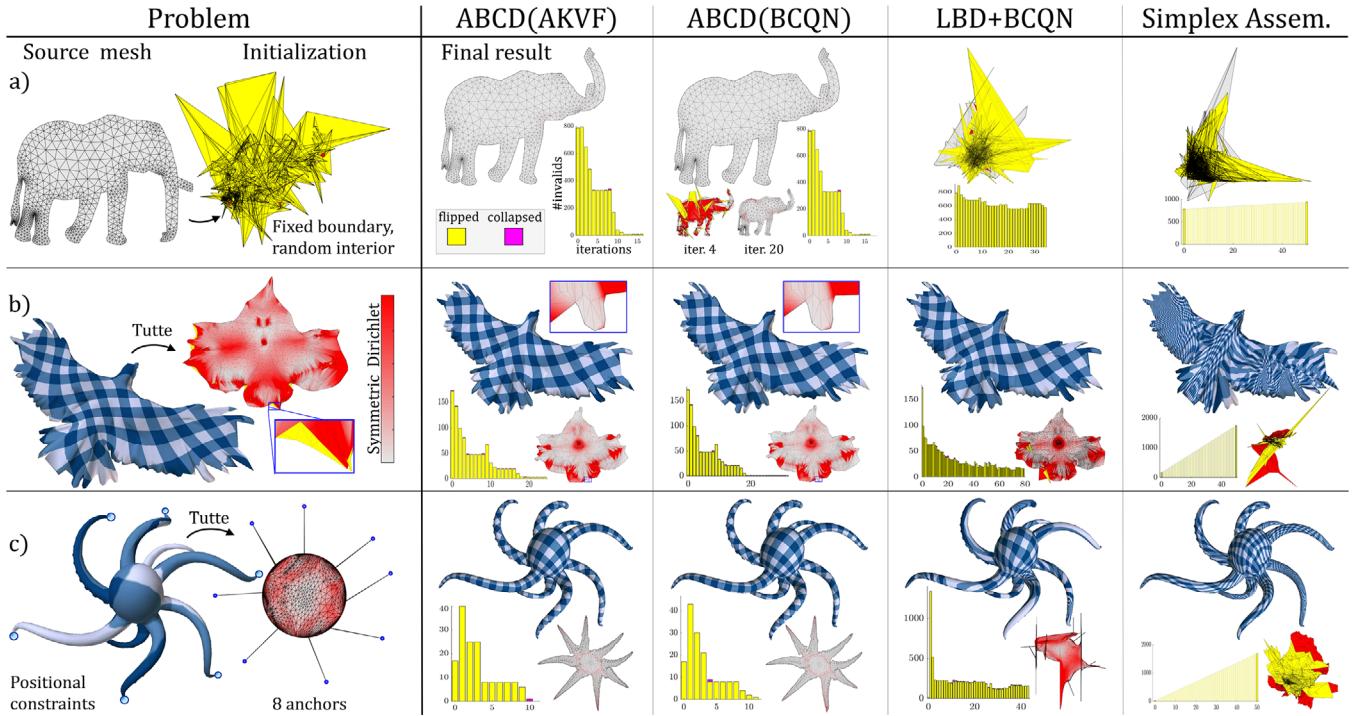


Figure 17: ABCD with first-order core solvers versus other related methods in 2D. Each element in the table shows the final result of the corresponding method and invalid triangle numbers per iteration. We plot Symmetric Dirichlet energy in logarithmic scale for each problem (bottom), where solid curves and dashed lines denote optimizer and fixer iterations, respectively (LBD is considered as a fixer). Note that SA results before the last iteration are our estimates, derived from the final output of SA.

Proof. Suppose w.l.o.g. that there are two free blocks B_1 and B_2 in ABCD and denote the remaining static vertices by $B_0 = \mathbf{V} \setminus (B_1 \cup B_2)$. Then, there are no mesh edges $\{u, v\}$ for $u \in B_1$ and $v \in B_2$ because, by our assumption on block partitioning, B_1 is disconnected from B_2 with respect to ‘~’ relation. As a result, we can represent E as a sum of energies, defined over disjoint simplices:

$$E = E_{B_0} + E_{B_1} + E_{B_2}; \quad E_{B_i} = \sum_{c \in \mathbf{C}(B_i)} w(c) \mathcal{D}(df_c),$$

where $\mathbf{C}(B_i)$, $i = 1, 2$, are simplices that share at least one vertex from B_i and $\mathbf{C}(B_0)$ are the rest of the elements. Denote by \mathbf{d} and $\bar{\mathbf{d}}$ the descent directions attained in S and $\text{ABCD}(S)$, respectively. Define $E[\mathbf{z}] = E(\mathbf{x}^0 + \mathbf{z})$, then the resulting energy in $\text{ABCD}(S)$ is

$$E(\bar{\mathbf{x}}) = \min_{\Delta t_1} E_{B_1} [\Delta t_1 \bar{\mathbf{d}}_{B_1}] + \min_{\Delta t_2} E_{B_2} [\Delta t_2 \bar{\mathbf{d}}_{B_2}] + \underbrace{E_{B_0}}_{\text{const.}}, \quad (\text{A.1})$$

while running a single iteration of S yields

$$E(\mathbf{x}) = \min_{\Delta t} (E_{B_1} [\Delta t \mathbf{d}_{B_1}] + E_{B_2} [\Delta t \mathbf{d}_{B_2}] + E_{B_0}). \quad (\text{A.2})$$

If $\bar{\mathbf{d}}_{B_i} = \mathbf{d}_{B_i}$ for $i = 1, 2$, then the proposition’s inequality (A.1) \leq (A.2) is proven because ‘min’ is sub-additive and E_{B_0} is constant by the definition (only free vertices are updated in S and $\text{ABCD}(S)$).

Note that H in (7) satisfies the following: (i) $H_{uv} \neq 0$ only if u and v share a common simplex c ; (ii) H_{uv} depends only on \mathbf{x}_p , where p are neighbours of c ’s vertices; (iii) $H_{uv} = 0$ for any $u \in B_1$ and $v \in B_2$, since there are no edges between B_1 and B_2 . Properties (i)–(iii) imply that, by reordering vertices, (7) can be rewritten as

$$\begin{pmatrix} H_{B_0} & * & * \\ * & H_{B_1} & 0 \\ * & 0 & H_{B_2} \end{pmatrix} \begin{pmatrix} 0 \\ \mathbf{d}_{B_1} \\ \mathbf{d}_{B_2} \end{pmatrix} = - \begin{pmatrix} \nabla E_{B_0} \\ \nabla E_{B_1} \\ \nabla E_{B_2} \end{pmatrix}, \quad (\text{A.3})$$

where each H_{B_i} is a block of H , computed for vertices in B_i . Then, we conclude by (A.3) that $\bar{\mathbf{d}}_{B_i} = \mathbf{d}_{B_i}$ for $i = 1, 2$. For N free blocks, B_1, \dots, B_N , we repeat the proof with $B_0 = \mathbf{V} \setminus (B_1 \cup B_2 \dots \cup B_N)$ and show that $\bar{\mathbf{d}}_{B_i} = \mathbf{d}_{B_i}$ for $i \geq 1$. \square

Appendix B: Blending Procedure

Denote odd (global) and even (local) iterations by $g(i) = 2i - 1$ and $l(i) = 2i$ for $i \geq 1$, respectively. We estimate the performance ratios of k global-local iterations by

$$\mathcal{R}^i = \frac{1}{\min\{k, i\}(1 + \mu)} \sum_{s=\lceil i-k+1 \rceil}^i \frac{\Delta E^{g(s)}}{\Delta E^{l(s)}} + \mu \frac{\text{Disp}^{g(s)}}{\text{Disp}^{l(s)}}, \quad (\text{B.1})$$

where $\lceil \bullet \rceil = \max\{\bullet, 1\}$, $\mu > 0$ is a constant blending parameter and the denominator value is chosen for obtaining $\mathcal{R}^i = 1$ for

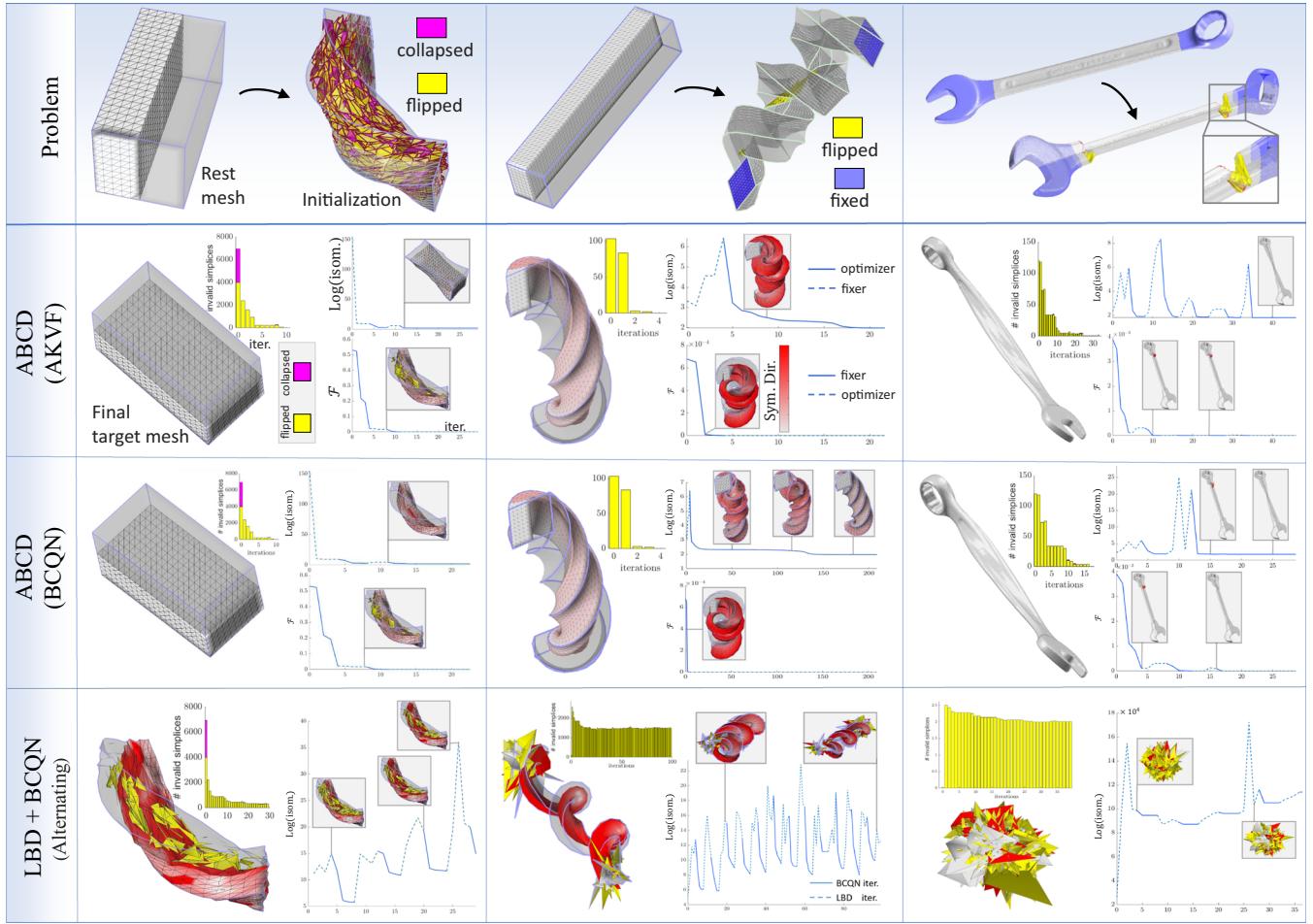


Figure 18: ABCD versus alternating combination of LBD and BCQN on tet meshes. We show number of invalid tets and energy values per iteration. We plot energy E_j in solid lines for iterations in which E_j is minimized and in other iterations E_j is shown in dashed lines.

equally performed global-local iterations. We compute (B.1) and adjust partitioning thresholds for the next cycle via

$$(K^{2i+1}, K^{2i+2}) = \text{Blend}(K^{2i-1}, K^{2i}, \mathcal{R}^i, \mathcal{L}_{\max}), \quad (\text{B.2})$$

where ‘Blend’ is the procedure described in Algorithm 2.

Appendix C: Map Injectivity

Although orientation-preserving maps are not always one-to-one, there is the following simple condition for guaranteeing the global bijectivity [AL13]: a positively oriented simplicial map $f : M \rightarrow \Omega$ is a global bijection if f maps bijectively ∂M onto $\partial\Omega$. As a result, ABCD produces a globally injective map upon its successful run if the optimized boundary is non-self-intersecting. We have experimented with techniques [JSP17, SS15] to avoid self-intersections between boundary edges. These methods meet the necessary conditions, listed in Section 7.2, and thus can be integrated with ABCD.

In a toy model example, demonstrated in Figure 9 (right), we attain a globally injective map by constructing a scaffold mesh [JSP17] and by running Algorithm 3 with the line search step modified to prevent boundary self-intersections [NSZ18].

References

- [AL13] AIGERMAN N., LIPMAN Y.: Injective and bounded distortion mappings in 3D. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–14.
- [BDS*12] BOUAZIZ S., DEUSS M., SCHWARTZBURG Y., WEISE T., PAULY M.: Shape-up: Shaping discrete geometry with projections. In *Computer Graphics Forum* (2012), pp. 1657–1667.
- [BML*14] BOUAZIZ S., MARTIN S., LIU T., KAVAN L., PAULY M.: Projective dynamics: Fusing constraint projections for fast simulation. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 154.

- [CBSS17] CLAICI S., BESSMELTSEV M., SCHAEFER S., SOLOMON J.: Isometry-aware preconditioning for mesh parameterization. In *Computer Graphics Forum* (2017), Vol. 36, Wiley Online Library, pp. 37–47.
- [CM83] COLEMAN T. F., MORÉ J. J.: Estimation of sparse jacobian matrices and graph coloring blems. *SIAM Journal on Numerical Analysis* 20, 1 (1983), 187–209.
- [CPSS10] CHAO I., PINKALL U., SANAN P., SCHRÖDER P.: A simple geometric model for elastic deformations. *ACM Transactions on Graphics (TOG)* 29, 4 (2010), 1–6.
- [CW17] CHEN R., WEBER O.: Gpu-accelerated locally injective shape deformation. *ACM Transactions on Graphics (TOG)* 36, 6 (2017), 1–13.
- [DMA02] DESBRUN M., MEYER M., ALLIEZ P.: Intrinsic parameterizations of surface meshes. In *Computer Graphics Forum* (2002), Vol. 21, Wiley Online Library, pp. 209–218.
- [FL16] FU X.-M., LIU Y.: Computing inversion-free mappings by simplex assembly. *ACM Transactions on Graphics (TOG)* 35, 6 (2016), 216.
- [FLG15] FU X.-M., LIU Y., GUO B.: Computing locally injective mappings by advanced mips. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 71.
- [Flo03] FLOATER M.: One-to-one piecewise linear mappings over triangulations. *Mathematics of Computation* 72, 242 (2003), 685–696.
- [GSC18] GOLLA B., SEIDEL H.-P., CHEN R.: Piecewise linear mapping optimization based on the complex view. In *Computer Graphics Forum* (2018), Vol. 37, Wiley Online Library.
- [HG00] HORMANN K., GREINER G.: Curve and Surface Design. *MIPS: An Efficient Global Parametrization Method*. International Conference on Curves and Surfaces (4th). Tech. Rep., DTIC Document, (2000). <https://apps.dtic.mil/dtic/tr/fulltext/u2/p012025.pdf>.
- [JSP17] JIANG Z., SCHAEFER S., PANZZO D.: Simplicial complex augmentation framework for bijective maps. *ACM Transactions on Graphics (TOG)* 36, 6 (2017), 1–9.
- [KABL14] KOVALSKY S. Z., AIGERMAN N., BASRI R., LIPMAN Y.: Controlling singular values with semidefinite programming. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 1–13.
- [KABL15] KOVALSKY S. Z., AIGERMAN N., BASRI R., LIPMAN Y.: Large-scale bounded distortion mappings. *ACM Transactions on Graphics* 34, 6 (2015), 1–10.
- [KGL16] KOVALSKY S. Z., GALUN M., LIPMAN Y.: Accelerated quadratic proxy for geometric optimization. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 1–11.
- [LBK16] LIU T., BOUAZIZ S., KAVAN L.: *Towards Real-Time Simulation of Hyperelastic Materials*. arXiv preprint arXiv:1604.07378 (2016). <https://www.cs.utah.edu/~ladislav/liu16towards/liu16towards.html>.
- [LBK17] LIU T., BOUAZIZ S., KAVAN L.: Quasi-newton methods for real-time simulation of hyperelastic materials. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 1.
- [LKK*18] LI M., KAUFMAN D. M., KIM V. G., SOLOMON J., SHEFFER A.: Optcuts: Joint optimization of surface cuts and parameterization. In *SIGGRAPH Asia 2018 Technical Papers* (2018), ACM, p. 247.
- [LPRM02] LÉVY B., PETITJEAN S., RAY N., MAILLOT J.: Least squares conformal maps for automatic texture atlas generation. *ACM Transactions on Graphics* 21, (2002), 362–371.
- [LX*08] LIU L., ZHANG L., XU Y., GOTSMAN C., GORTLER S. J.: A local/global approach to mesh parameterization. *Computer Graphics Forum* 27, 5 (2008), 1495–1504.
- [LYNF18] LIU L., YE C., NI R., FU X.-M.: Progressive parameterizations. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 41.
- [Nes83] NESTEROV Y. E.: A method for solving the convex programming problem with convergence rate $\mathcal{O}(1/k^2)$. In *Dokl. Akad. Nauk SSSR* (1983), Vol. 269, pp. 543–547.
- [NSZ18] NAITSAT A., SAUCAN E., ZEEVI Y. Y.: Geometry-based distortion measures for space deformation. *Graphical Models* 100 (2018), 12–25.
- [NZ19] NAITSAT A., ZEEVI Y. Y.: Multi-resolution approach to computing locally injective maps on meshes. In *ACM SIGGRAPH 2019 Posters* (2019), ACM, p. 87.
- [PDZ*18] PENG Y., DENG B., ZHANG J., GENG F., QIN W., LIU L.: Anderson acceleration for geometry optimization and physics simulation. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–14.
- [PTH*17] PORANNE R., TARINI M., HUBER S., PANZZO D., SORKINE-HORNUNG O.: Autocuts: Simultaneous distortion and cut optimization for UV mapping. *ACM Transactions on Graphics (TOG)* 36, 6 (2017), 1–11.
- [RPPSH17] RABINOVICH M., PORANNE R., PANZZO D., SORKINE-HORNUNG O.: Scalable locally injective mappings. *ACM Transactions on Graphics (TOG)* 36, (2017), 1.
- [SC18] SAWHNEY R., CRANE K.: Boundary first flattening. *ACM Transactions on Graphics (ToG)* 37, 1 (2018), 1–14.
- [SJZP19] SHEN H., JIANG Z., ZORIN D., PANZZO D.: Progressive embedding. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 1–13.

- [SPSH*17] SHTENGEL A., PORANNE R., SORKINE-HORNUNG O., KOVALSKY S. Z., LIPMAN Y.: Geometric optimization via composite majorization. *ACM Transactions on Graphics* 36, 4 (2017), 1–1.
- [SS15] SMITH J., SCHAEFER S.: Bijective parameterization with free boundaries. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 70.
- [TRG16] TAPPENDEN R., RICHTÁRIK P., GONDZIO J.: Inexact coordinate descent: Complexity and preconditioning. *Journal of Optimization Theory and Applications* 170, 1 (2016), 144–176.
- [TSIF05] TERAN J., SIFAKIS E., IRVING G., FEDKIW R.: Robust quasistatic finite elements and flesh simulation. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2005), ACM, pp. 181–190.
- [Tut63] TUTTE W. T.: How to draw a graph. *Proceedings of the London Mathematical Society* 3, 1 (1963), 743–767.
- [WMZ12] WEBER O., MYLES A., ZORIN D.: Computing extremal quasiconformal maps. *Computer Graphics Forum* 31, 5 (2012), 1679–1689.
- [WY16] WANG H., YANG Y.: Descent methods for elastic body simulation on the gpu. *ACM Transactions on Graphics (TOG)* 35, 6 (2016), 1–10.
- [ZBK18] ZHU Y., BRIDSON R., KAUFMAN D. M.: Blended cured quasi-newton for distortion optimization. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–14.

Supporting Information

Additional supporting information may be found online in the Supporting Information section at the end of the article.

Figure S19: Running a stress test of ABCD(PN) to identify positional constraints inconsistent with a positively oriented UV-map.

Figure S20: Here we solve the third problem from Figure 18 by minimizing three distortions.

Table S1: ABCD parameters are specified for each trial, tested in Figure S24.

Algorithm 4: ABCD with LGB for q measures.

Figure S21: Unconstrained parametrization of Hilbert curve of different resolutions.

Figure S22: Comparing ABCD(BCQN), ABCD(AKVF) and LBD+BCQN from Figure 18.

Figure S23: Showing an average number of invalid triangles per iteration, obtained in the experiments from Figure 11.

Figure S24: Testing the impact of setting different hyperparameters in ABCD(PN) algorithm.

Figure S25: Illustration of the deformation embedding method, described in Section 12.

Figure S26: All the results of the conformal initialization experiment (some were depicted by Figure 12).

Video S1