Volume xx (200y), Number z, pp. 1–20

Adaptive Block Coordinate Descent for Distortion Optimization

Alexander Naitsat[1], Yufeng Zhu[2] and Yehoshua Y. Zeevi[1]

[1]Viterbi Faculty of Electrical Engineering, Technion - Israel Institute of Technology
[2]Facebook Reality Labs

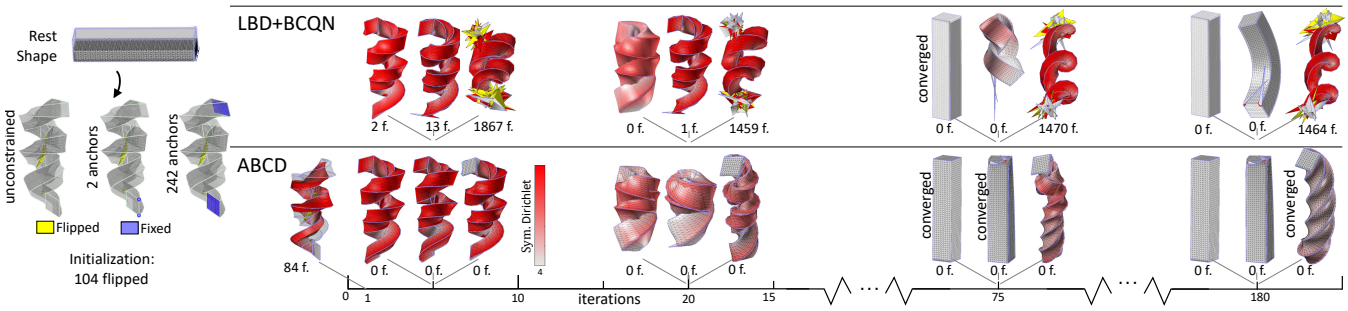


Figure 1: *Comparison of the alternating combination of LBD and BCQN (top) with ABCD(BCQN) (bottom) in 3D. The same initialization is used for the four shown trials: unconstrained problem, two fixed anchors, two fixed endpoints (the second problem of Fig. 18). We report the number of inverted elements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

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; (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, positively-oriented 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 modeling, 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; (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-

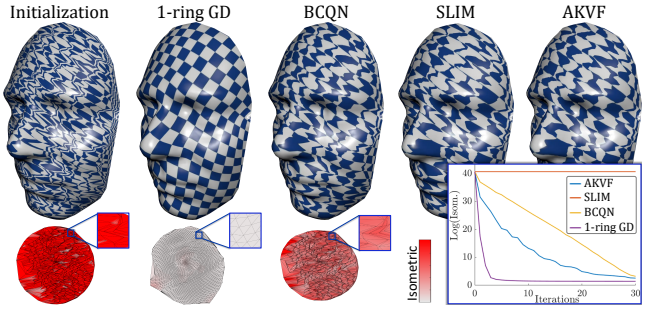


Figure 2: *Free boundary parametrization with a noisy initialization (intentionally distorted Tutte map without foldovers). 1-Ring GD denotes the gradient descent in blocks of single vertices; each block receives a single GD iteration per cycle depicted in the plot.*

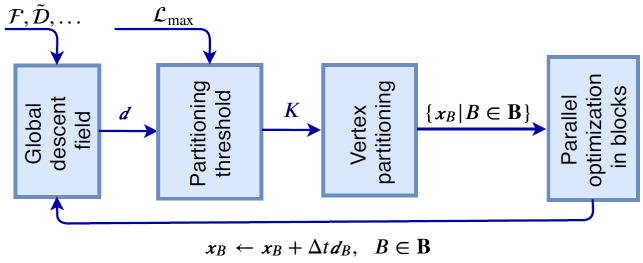


Figure 3: *ABCD main stages: 1) Computing descent field of alternatively selected distortions $\mathcal{D}_j = \mathcal{F}, \tilde{\mathcal{D}}$, defined by (12) and (10); 2) Estimating partitioning threshold K; 3) Vertex partitioning into blocks $\mathbf{B}$ by Algorithm 1; 4) Parallel optimization of (2) in coordinate blocks $B \in \mathbf{B}$.*

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.
3. An **adaptively blended local-global strategy** providing robustness to highly non-smooth problems, 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 $f : M \subset \mathbb{R}^n \to \mathbb{R}^m$ 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 $\boldsymbol{x} \in \mathbb{R}^{|\mathbf{V}|m}$ be the column stack of vertex positions under f. We express a distortion energy of the mapping $f[\boldsymbol{x}]$ as a weighted sum over simplices

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

where $w(c)$ are simplex weights, usually equal to 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 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 $2 \le m \le n \le 3$:

$$\underset{\boldsymbol{x}}{\text{argmin}} \quad E\left(f[\boldsymbol{x}]\right) \tag{2}$$

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

$$A\boldsymbol{x} = \boldsymbol{z}, \tag{4}$$

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 iteratively target coordinates $\boldsymbol{x}$ via

$$\boldsymbol{x}(t+\Delta t) = \boldsymbol{x}(t) + \Delta t \boldsymbol{d}\left(\nabla_{\boldsymbol{x}} E, \nabla^2_{\boldsymbol{x}} E\right), \tag{5}$$

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

$$\nabla_{\boldsymbol{x}} E = \frac{\partial E\left(\boldsymbol{x}(t)\right)}{\partial \boldsymbol{x}}, \ \nabla^2_{\boldsymbol{x}} E = \frac{\partial^2 E\left(\boldsymbol{x}(t)\right)}{\partial \boldsymbol{x}^2}. \tag{6}$$

In first order methods, $\boldsymbol{d} = \boldsymbol{d}(\nabla E)$ and it is computed, in general, by preconditioning the gradient with a sparse *proxy matrix*, i.e., in unconstrained problems $\boldsymbol{d}$ is a solution of a sparse linear system

$$H\boldsymbol{d} = -\nabla_{\boldsymbol{x}} E. \tag{7}$$

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

$$E_H(\boldsymbol{x}) = E(\boldsymbol{x}^i) + \left(\boldsymbol{x} - \boldsymbol{x}^i\right)^\top \nabla_{\boldsymbol{x}} E + \left(\boldsymbol{x} - \boldsymbol{x}^i\right)^\top H \left(\boldsymbol{x} - \boldsymbol{x}^i\right) / 2. \tag{8}$$

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 Map-

pings (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 preconditioner (AKVF).

Quasi-Newton methods [ZBK18; LBK17; SS15] optimize a quadratic proxy constructed from gradients of both the current and previous iterations, i.e., $\boldsymbol{d} = \boldsymbol{d}\left(\nabla_{\boldsymbol{x}} E^i, \nabla_{\boldsymbol{x}} E^{i-1}, \dots\right)$. 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 $\boldsymbol{d}$ is the function of both ∇E and $\nabla^2 E$. Since, the problem (2) is highly non-convex, these solvers compute positive semidefinite approximations of $\nabla^2 E$, most commonly, using the Hessian diagonal $H = \operatorname{diag}\left(\nabla^2 E(\boldsymbol{x})\right)$ in Jacobi gradient descent [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 semidefinite 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 $x(t)$ in (5) by an affine combination of the results from the current and previous iterations $\boldsymbol{x}(t) = (1+\theta)\boldsymbol{x}^i - \theta \boldsymbol{x}^{i-1}$. 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 $f[\boldsymbol{x}^i]$ 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 [SC18], 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, $|\mathbf{C}|$ 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 semidefinite programming (SDP).

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 $\mathcal{D}_{\text{iso}}$ [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 multi-objective optimization performs well on middle resolution meshes, Autocut supports only soft positional constraints in 2D and does not extend to volumetric domains.

We propose a novel algorithm, Adaptive Block Coordinate Descent (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 uni-

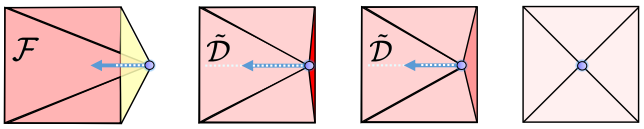


Figure 4: *A typical ABCD work-flow: initialization with flips (in yellow), fixer, optimizer and final convergence. Dashed lines along descent directions depict length of line search intervals (23).*

fied 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 2-3 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. Redesigning 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:

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

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

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 $\det(df_c) \le 0$. 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., $s_c = \mathrm{sign}\big(\det df_c^{i-1}\big)$; then we modify the original measure $\mathcal{D}$ as follows:

$$\widetilde{\mathcal{D}}(f_c) = \begin{cases} \mathcal{D}(\sigma_1,...,\sigma_m) & \sigma_1 > 0,\, s_c > 0\,, \\ \infty & \sigma_1 \le 0,\, s_c > 0\,, \\ \min\mathcal{D} & s_c \le 0\,, \end{cases} \tag{10}$$

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, $\widetilde{\mathcal{D}}$ equals the original measure plus the barrier term, 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 collapse 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

$$\det(df_c) = \prod\sigma_j < 0\,. \tag{11}$$

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:

$$\mathcal{D}_{\mathrm{flip}}(\boldsymbol{\sigma},\Lambda) = \begin{cases} \Lambda - \prod\sigma_i & \prod\sigma_i \le 0\,, \\ 0 & \text{else}\,, \end{cases} \tag{12}$$

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

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

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

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

$$j = 1,\dots,m : \frac{\partial\mathcal{D}_{\mathrm{collapse}}}{\partial\sigma_j} = \begin{cases} -1 & |\sigma_j| \le \varepsilon\,, \\ 0 & \text{else}\,. \end{cases} \tag{14}$$

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

$$\mathcal{F}(\boldsymbol{\sigma},\Lambda) = \begin{cases} \mathcal{D}_{\mathrm{flip}}(\boldsymbol{\sigma},\Lambda) & \sigma_1 < 0,\, \sigma_m \ge \varepsilon\,, \\ \mathcal{D}_{\mathrm{collapse}}(\boldsymbol{\sigma},\Lambda) & \text{else}\,, \end{cases} \tag{15}$$

where we set a higher priority for penalizing flips, since minimizing (12) is a more complex geometric problem than unfolding col-

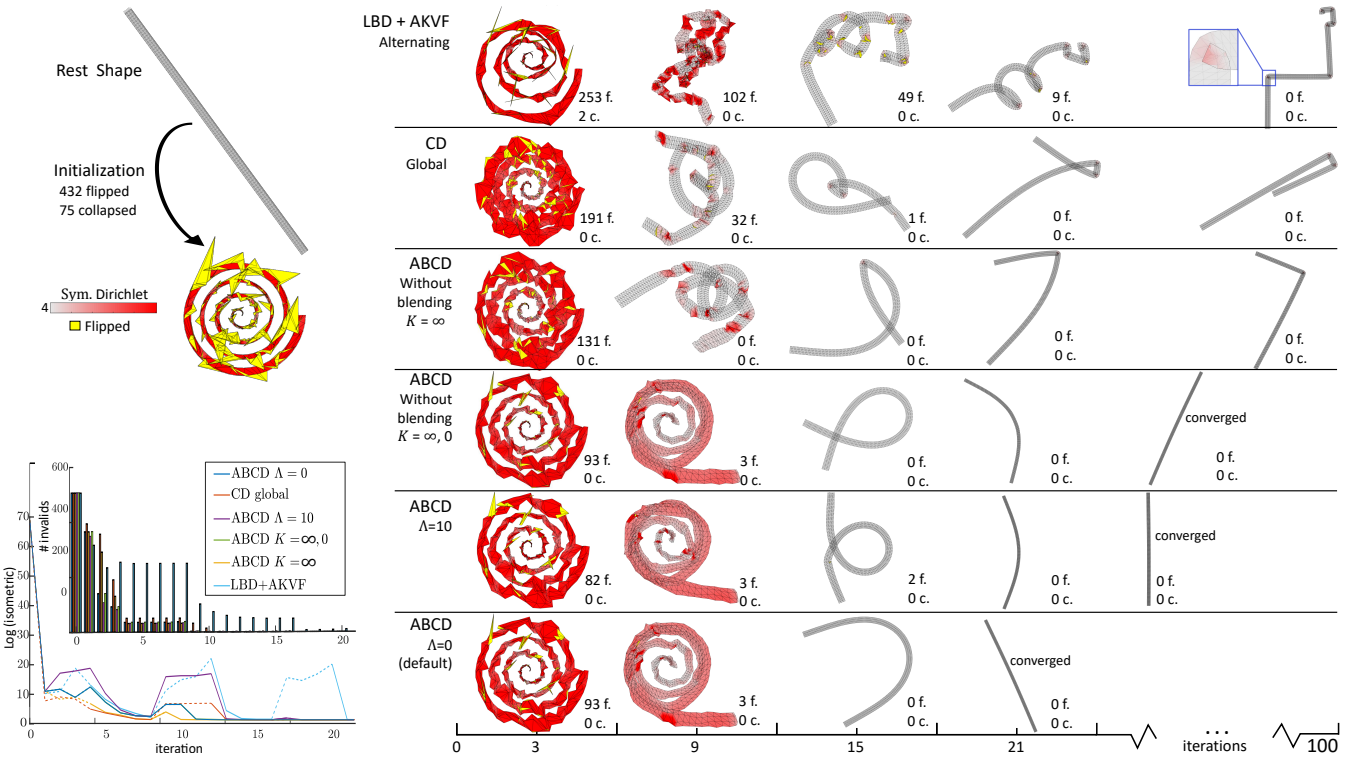


Figure 5: *Comparing ABCD(AKVF) (bottom) with other methods and with simplified variants of ABCD(AKVF).* ***Top to bottom****: alternating optimization of LBD and AKVF; global ABCD version (CD); ABCD without blending (BCD) that includes constant $K = \infty$ and constantly alternating thresholds $K =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

lapsed 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 (Fig. 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 $E_{\mathcal{F}} = \sum w(c)\mathcal{F}(\boldsymbol{\sigma}, \Lambda)$ 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 Fig. 5 and Fig. 24 in the supplemental.

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 Fig. 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 $\boldsymbol{x}$ 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 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

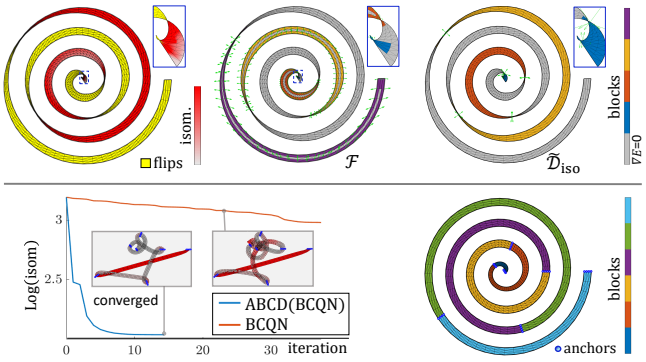


Figure 6: ***Top from left to right****: Unconstrained swirl planar deformation, initialized with foldovers, block partitioning in ABCD(GD) for $\mathcal{F}$ and $\widetilde{\mathcal{D}}_{iso}$ energies, respectively.* ***Bottom****: We compare BCQN and ABCD(BCQN) optimizations (left) of the constrained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

Algorithm 1 Vertex partitioning into blocks

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

Output: Vertex blocks $\mathbf{B}$.

1: $\mathbf{E} \leftarrow \emptyset$.
2: **for** $v \in \mathbf{V}$ **do**
3: **for** $u \in$ Neighbours(v) **do**
4: Add $\mathbf{E} \leftarrow \mathbf{E} \cup \{\{u,v\}\}$ **if** $\mathcal{L}_{uv}(\boldsymbol{d}, \boldsymbol{x}, \boldsymbol{y}) \le K$ 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 $(\mathbf{V}, \mathbf{E})$.

consider both the locality and the globality with respect to the selection of $\boldsymbol{x}$'s coordinate blocks in (5).

To highlight the importance of the raised question, consider a toy model, depicted in Fig. 2, where a simple 1-ring block gradient descent [HG00; NSZ18] easily outperforms much more advanced global solvers. In general, the global strategy works well when descent direction $\boldsymbol{d}$ 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 neighboring 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 $\boldsymbol{d}$ are rescaled to increase the line search range, the global update of $\boldsymbol{x}$ along $\boldsymbol{d}$ 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 their gradients are zero outside invalid element regions.

If $\boldsymbol{x}_{B_1}, \ldots, \boldsymbol{x}_{B_p}$ are blocks attained in the above scenario, then optimizing $\boldsymbol{x}_{B_i}$ has no effect on another block $\boldsymbol{x}_{B_j}$ as long as the inner boundary between the blocks is locked (see Fig. 6). Therefore, in such cases, it is more computationally efficient to employ BCD inherent parallelism for processing as many blocks as possible, 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 Fig. 6, and surface parametrization with standard harmonic initializations. Apparently, as illustrated by Fig. 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 on *adaptive block partitioning* strategy that solves problem (2) separately over blocks of target vertices $\{\boldsymbol{x}_B | B \in \mathbf{B}^i\}$, where the subscript in $\boldsymbol{x}$ denotes column stack of coordinates of vertices in B and superscript in $\mathbf{B}^i$ denotes that the block partitioning is recomputed for each iteration i. Similarly, we denote by $\boldsymbol{d}_v$, $\boldsymbol{x}_v$ and $\boldsymbol{y}_v$ 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* $\mathbf{V}_0 = \{v \in \mathbf{V} |\ \boldsymbol{d}_v = \mathbf{0}\}$ 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 $\boldsymbol{d}(\nabla E, \nabla^2 E)$ is the descent direction and denote by $u \sim v$ that u and v are neighboring 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:

$$\forall u, v \in B : \|\boldsymbol{d}_u - \boldsymbol{d}_v\|_2 \le K \|\boldsymbol{x}_u - \boldsymbol{x}_v\|_2 , \tag{16}$$

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 $\{u,v\}$, $u \nsim v$, can be updated independently of the rest of the mesh. Motivated by this fact, we construct a *connectivity graph* $(\mathbf{V}, \mathbf{E})$ in Algorithm 1 by iterating over vertices $u \sim v$ and measuring the following: quantity

$$\mathcal{L}_{uv} = \|\boldsymbol{d}_v - \boldsymbol{d}_u\|_2 \, \|\boldsymbol{y}_v - \boldsymbol{y}_u\|_2 \, \|\boldsymbol{x}_v - \boldsymbol{x}_u\|_2^{-1} . \tag{17}$$

The term $\|\boldsymbol{y}_v - \boldsymbol{y}_u\|$ is chosen in (17) so that it normalizes distances in the target domain. As specified in Algorithm 1, $\{u,v\}$ is an edge

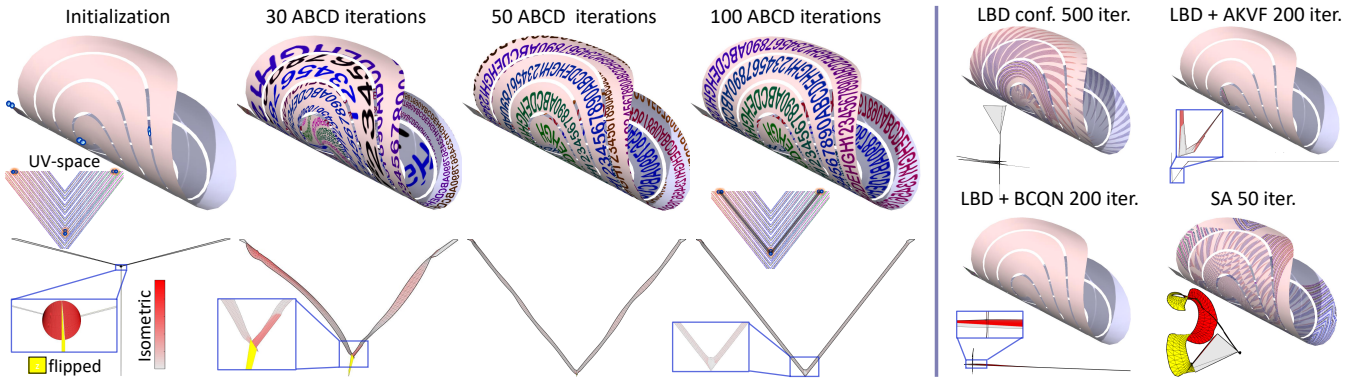


Figure 7: *Constrained parametrization of a space fitting curve. The source and target anchor positions are shown in the top-left corner. Free vertices are initialized by Tutte embedding. ABCD(AKVF) results (left) are compared with other methods (right): standalone conformal LBD, alternating combinations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

of $\mathbf{E}$ if $u \sim v$, $\mathcal{L}_{uv} \leq K$ and $\mathbf{B}^i$ are the 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 method of controlling partitioning is to set $K = \infty$. In this case, blocks $\mathbf{B}^i$ are the connected components of the proposed equivalence relation '$\sim$'. Optimizing separately free blocks $B_1,\ldots,B_p$, obtained in this partitioning scheme, is more effective than using a global solver because the BCD method can compute different step sizes Δt_{B_l} per block in (5). 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 $\boldsymbol{d}$ to be a *K-Lipschitz (continuous)* function of $\boldsymbol{x}$ 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 smoothness of $\boldsymbol{d}$ 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 Fig. 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 toward 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

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

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:

$$\mathrm{Disp}^i = \left\| \boldsymbol{x}^i - \boldsymbol{x}^{i-1} - \mathrm{Proj}_{\mathrm{Ker}\left(\boldsymbol{x}^{i-1}\right)} \left(\boldsymbol{x}^i - \boldsymbol{x}^{i-1}\right) \right\|_{\mathrm{Fro}}, \tag{19}$$

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

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

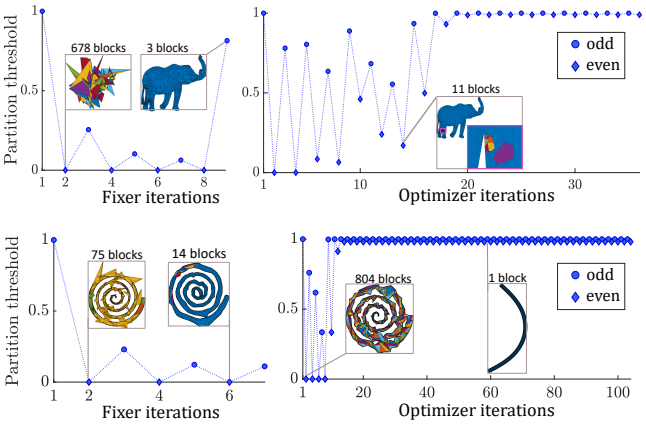


Figure 8: *Normalized partitioning thresholds ($K/\mathcal{L}_{\max}$) computed by Algorithm 2 for distortions $\mathcal{F}$ (left) and $\widetilde{\mathcal{D}}_{iso}$ (right). Snapshots are colored according to vertex partitioning.*

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

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 measure in a fixer phase (e.g., $\mathcal{F}$) vs 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

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

- 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 guaranties to avoid similar difficulties, we found that the alternating scheme works best. Figs. 7, 17 and 18, show that this option is much more robust than the other. 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 partition strategy avoids vertices being interlocked by allowing a separate processing of regions with 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 neighboring 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 Fig. 3. In addition to these stages, our procedure includes a number of sub-steps described below. Some

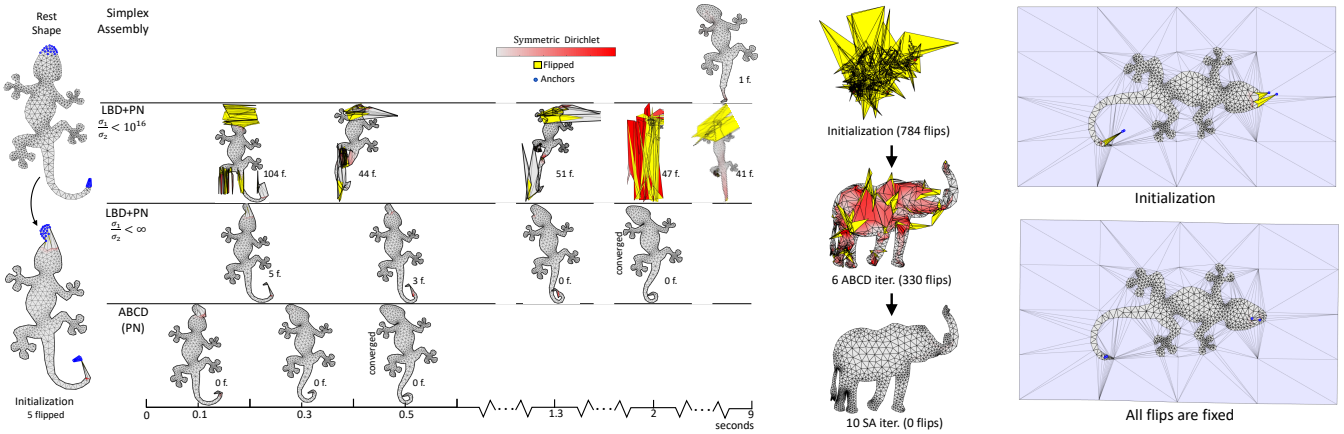


Figure 9: ***Left****: Comparing ABCD(PN) with LBD+PN and SA for a planar problem, initialized with flips. We used the following parameters for LBD: finite unbounded condition number* $\sigma_1/\sigma_2 < \infty$ *and* $\sigma_1/\sigma_2 < 10^{16}$. ***Middle****: Running SA with first six ABCD iterations to solve the failure case of SA from Fig. 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 $\widetilde{\mathcal{D}}$, defined according to (10). In our experiments, we minimize energies $\mathcal{F}$ and $\widetilde{\mathcal{D}}$ 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 1 and in Fig. 24 in supplemental. 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. So 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 preconditioning methods 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., $\boldsymbol{x}_B^{i+1} = \boldsymbol{x}_B^i + t\boldsymbol{d}_B$. We use the Armijo backtracking search to estimate a sufficiently good minimizer

$$\underset{t\in[0,T_{\max}]}{\operatorname{argmin}} E(\boldsymbol{x}_B^i + t\boldsymbol{d}_B)\,. \tag{21}$$

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

$$T_{\max} = (1-\delta^+)t_{\max}; \tag{22}$$

$$t_{\max} = \min_{c\in\mathbf{C}^+(B)} \{\infty, t \geq 0 \,|\, \det(df_c[t]) = 0\}\,, \tag{23}$$

where $\mathbf{C}^+(B)$ denotes the set of positively oriented simplices that share B's vertices, $df_c[t]$ is the Jacobian of c induced by mapping B to $\boldsymbol{x}_B^i + t\boldsymbol{d}_B$ and $\delta^+ > 0$ is a small number that we choose to avoid numerical errors on nearly collapsed elements.

For map fixer distortions, such as $\mathcal{F}$ the minimum of (23) is computed over the set of inverted simplices $\mathbf{C}^-(B)$. Since the

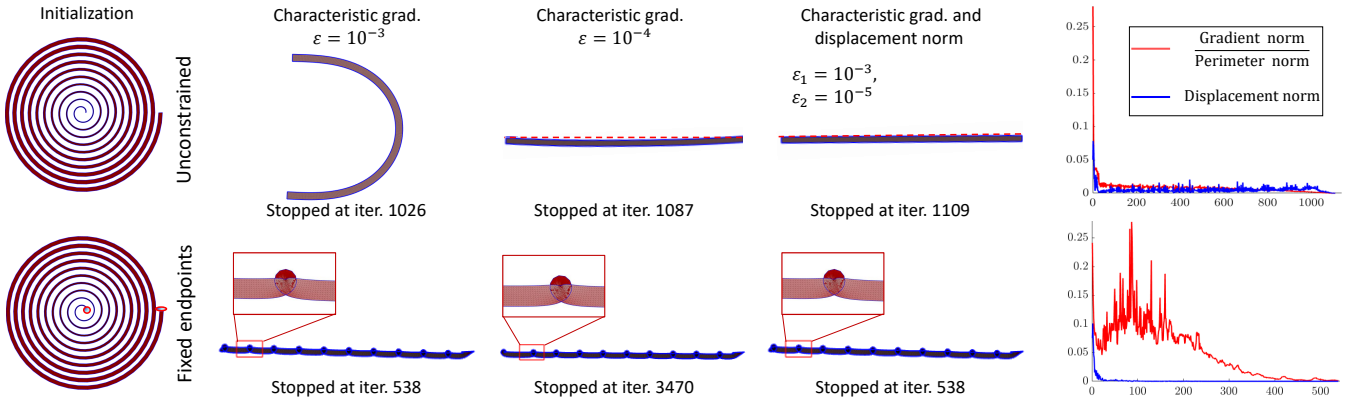


Figure 10: *Characteristic gradient termination criteria [ZBK18] is compared against enhanced criteria (24). We test these criteria with BCQN solver and $\mathcal{D}_{iso}$ distortion for unconstrained swirl deformation (top) and for the same problem with fixed endpoints (bottom).*

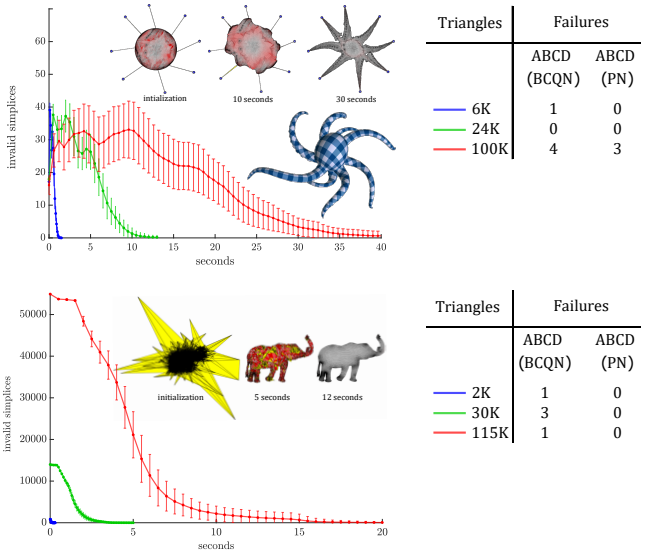


Figure 11: *Testing ABCD on randomly generated deformations that include 600 trials of constrained parametrization with randomly placed anchors (top) and 600 trials of planar shape deformation with fixed boundary and randomized initialization of interiors (bottom). We tested 100 problems with Eigen library 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).*

minimizer of $E_{\mathcal{F}}(\boldsymbol{x}_B^i + t\boldsymbol{d}_B)$ over $\mathbf{C}^-(B)$'s vertices lies beyond the interval $[0, t_{\max}]$, we use a negative number $\delta^- < 0$ in (22) instead of δ^+, i.e., $T_{\max} = (1-\delta^-)t_{\max}$. See the illustration in Fig. 4.

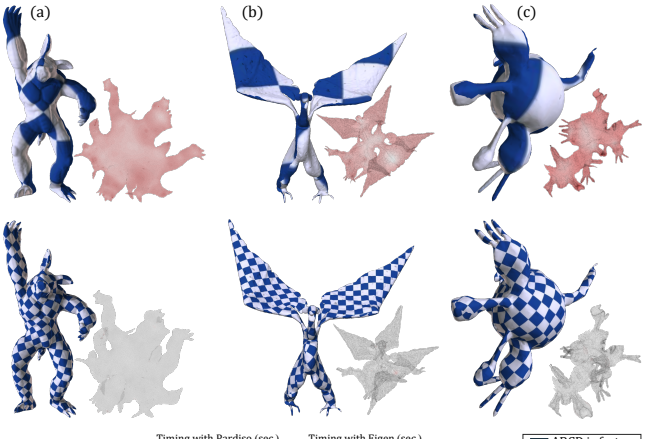
7.4. Termination criteria

Unlike global solvers with a single objective, ABCD has several termination criteria — to stop processing of a current block, a current measure, the entire optimization and the special criteria for terminating or reseting the LGB. A successful operation of ABCD should encounter two phases: the first one of cleaning foldovers, and the final one of $\widetilde{\mathcal{D}}$ convergence. The first phase is completed when $\mathcal{F}$ vanishes, and further processing is devoted solely to $\widetilde{\mathcal{D}}$. Thus, the termination of an individual measure is claimed when its average value reaches the optimum. The final convergence criteria is 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

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

where $\mathrm{Char}(E_j)$ denotes the characteristic gradient norm of E_j. Although using $\mathrm{Char}(E_i)$ 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 small, or to redundant iterations if the threshold is too high. As exemplified by Fig. 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 first with $b_j = 1$, steadily incrementing 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 Algortihm 2 after the number of blocks in local steps becomes equal to that of the global step. 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.



	#triangles	#invalids	Timing with Pardiso (sec.) fixer	total	standard	Timing with Eigen (sec.) fixer	total	standard
(a)	48964	22	0.99	6.87	8.09	0.87	9.22	11
(b)	91912	25	3.42	12.1	15.8	1.72	15	19.5
(c)	94736	9	4.46	10.9	17.9	1.68	9.52	17.8
	91976	13	22.2	23.4	13.6	20.8	21.3	17.8
	91976	1	0.38	5.82	12	0.43	8.36	15.8
	91952	11	3.38	9.59	11.3	3.56	12.8	14.7
	95880	3	1.63	6.83	12.6	1.37	9.02	15.5
	5816	18	0.4	1.67	0.92	0.372	1.75	0.63

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 one our method is faster.*

7.5. Implementation

We have written the main body of Algorithm 3 in Matlab for attaining a highly customizable interface that supports multiple 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 coloring algorithm. We have tested Eigen and Pardiso libraries to solve (7) in ABCD(PN). We report runtimes for ABCD(PN) in Figs. 9, 14, 12, 15, 16 and in Figs. 26, 21 and 24 in the supplemental.

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 bock 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).

Algorithm 3 Adaptive block coordinate descent (ABCD)

Input:

- Source mesh $(\mathbf{V}, \mathbf{C})$ and vertex initialization $\boldsymbol{x}^0$.
- 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: $E_j \leftarrow\!\shortmid \mathcal{D}_j$ energy of $f(\boldsymbol{x}^i)$ according to (1).
5: **if** average$(E_j) \le \min \mathcal{D}_j$ **continue**.
6: $\boldsymbol{d} \leftarrow\!\shortmid$ descent direction of E_j attained by S_j.
7: $\{\boldsymbol{x}^i_B\}_{B\in\mathbf{B}} \leftarrow\!\shortmid$ BlockPartitioning$(K, \boldsymbol{d})$ via Algorithm 1.
8: **for** $B \in \mathbf{B}$ in parallel **do**
9: $x^{i+1}_B \leftarrow\!\shortmid$ solve (2) in B by S_j with b_j max iterations.
10: **end for**
11: $i \leftarrow\!\shortmid i+1$.
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 $i = i_{\max}$.

We test meshes with different numbers of elements and summarize our results in Figs. 5, 17, 7, 11, 13 for 2D problems and in Figs. 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 $\widetilde{\mathcal{D}}_{\text{ARAP}}$ in Fig. 16 and $\widetilde{\mathcal{D}}_{\text{iso}}$ distortion in other ABCD trials, where $\mathcal{D}_{\text{iso}} = \sigma_1^2 + \sigma_1^{-2} + \cdots + \sigma_m^2 + \sigma_m^{-2}$ and $\mathcal{D}_{\text{ARAP}} = (\sigma_1 - 1)^2 + \cdots + (\sigma_m - 1)^2$.

First, 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 Fig. 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 Fig. 5 top-right). Although we can not 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 Fig. 9 (right).

Methods from Fig. 5 perform similarly over a wide range of unconstrained and constrained 2D and 3D problems. Furthermore, as shown in Fig. 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

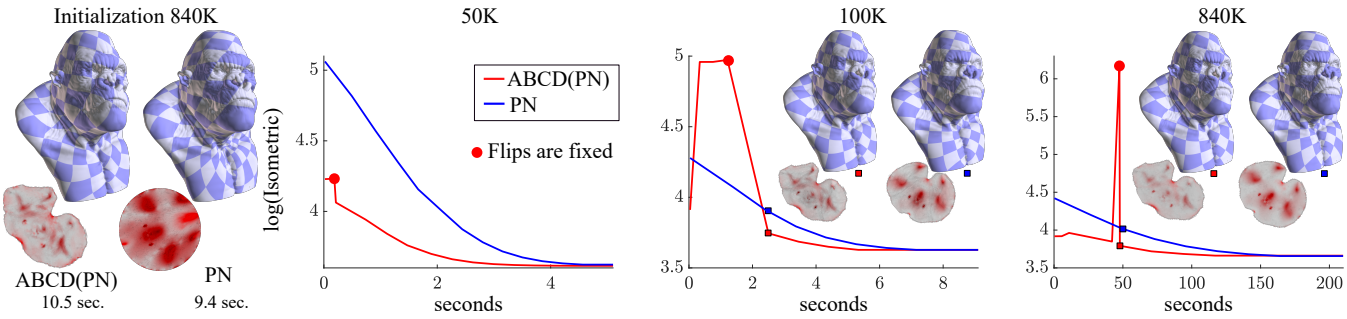


Figure 13: *We compare PN parametrization, initialized by Tutte embedding, and ABCD(PN) parametrization, initialized by mapping surfaces into UV-map of a decimated mesh. Specifically, we parametrize a mesh with 2K triangles and use the deformation embedding method [NZ19] for initializing the problem in a higher resolution (see Fig. 25 in supplemental). We run 12 iterations for each mesh resolution and depict $\widetilde{\mathcal{D}}_{iso}$ 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)	timing to fix (LGB)	iterations to fix	iterations to fix (LGB)	timing total (no LGB)	timing total (LGB)	iterations total	iterations total (LGB)
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 $\widetilde{\mathcal{D}}_{iso}$ (default) and $\widetilde{\mathcal{D}}_{ARAP}$ distortions. All experiments were timed on the four-core i7-8565U CPU.*

every trial. These tests show that ABCD is well-scalable and can recover from extremely distorted meshes of a high resolution. We compared ABCD(PN) for a planar shape deformation problem with other related methods. As demonstrated in Figs. 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 next examine failure cases of LBD and SA fixers in both 2D (Figs. 17, 9) and 3D (Figs. 18, 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 models (Figs. 18 and 9), respectively. Moreover, as illustrated in Fig. 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 the standalone 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 Figs. 7 and 17. To construct the starting point, we, first, compute the standard Tutte embedding, and then move anchors to their prescribed positions.

We experiment with alternative initialization schemes for surface parametrization. In Figs. 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 3 in supplemental). As demonstrated by Fig. 13, this initialization scheme can accelerate UV-map computations for meshes of a high resolution.

In Figs. 12 and 26 (supplemental), we compare the BFF conformal initialization scheme with the standard isometric parametrization. Specifically, we ran BFF conformal parametrization on 600 meshed 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, intro-

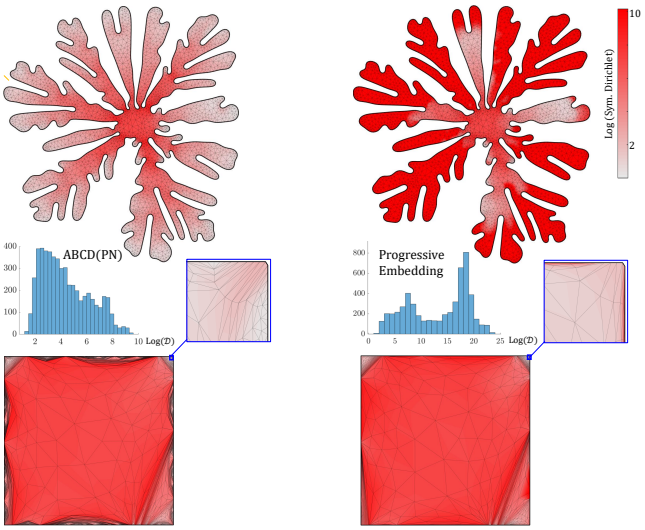


Figure 15: *Mapping of the Hele-Shaw polygon [SJZP19] (top) onto the square (bottom) using ABCD(PN) (left) and the PE method (right). The initial Tutte embedding produces 41 flipped and 845 degenerate triangles. It takes 25 seconds to fix all invalid triangles by ABCD(PN), while PE runs 60 seconds. Histograms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

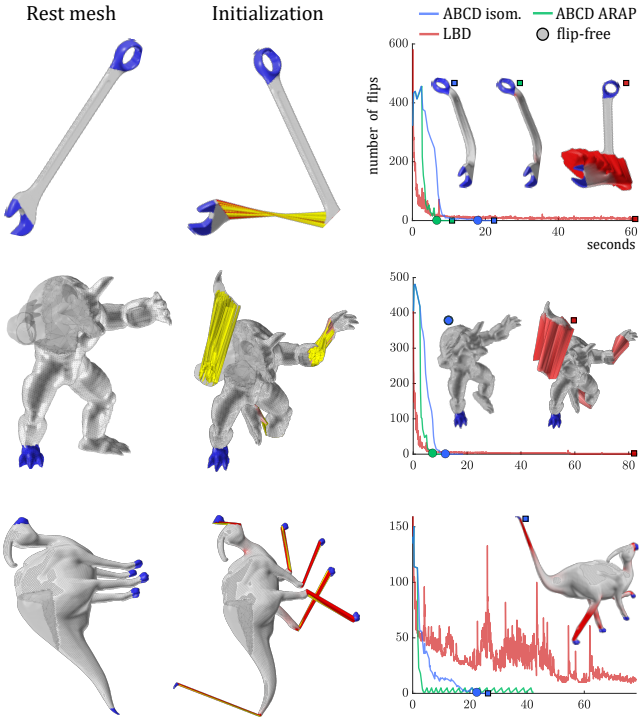


Figure 16: *ABCD(PN) with $\widetilde{\mathcal{D}}_{iso}$ and $\widetilde{\mathcal{D}}_{ARAP}$ distortions versus LBD on tetrahedral meshes. We bound condition number in LBD by $\sigma_1/\sigma_3 < 10^2$. **Left to right**: Rest mesh, initialization with anchor points, marked in blue, and numbers of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

duced in [SJZP19]. As demonstrated in Fig. 15, ABCD ran faster and converged to a mapping with lower isometric distortion.

According to our proposition in Appendix A, we presume that even for valid initializations, our block 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 Fig. 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, number of invalid simplices, distortion level 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.

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 such methods take into account the intrinsic metric of the deformation space. Unfortunately, at the moment it is not clear what is the right preconditioner for map fixer optimization steps. Considering a metric approach oriented at minimizing this class of geometric measures and finding the right preconditioner may be a promising direction for future research.

Appendix A. Proof of BCD Superiority

Proposition. Let S be a global core-solver, initialized by $\boldsymbol{x}^0$. Denote by $\boldsymbol{x}$ and $\bar{\boldsymbol{x}}$ 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 $E(\bar{\boldsymbol{x}}) \leq E(\boldsymbol{x})$.

Proof. Suppose w.l.o.g. that there are two free blocks B_1, B_2 in ABCD and denote the remaining static vertices by $B_0 = \mathbf{V} \setminus (B_1 \cup$

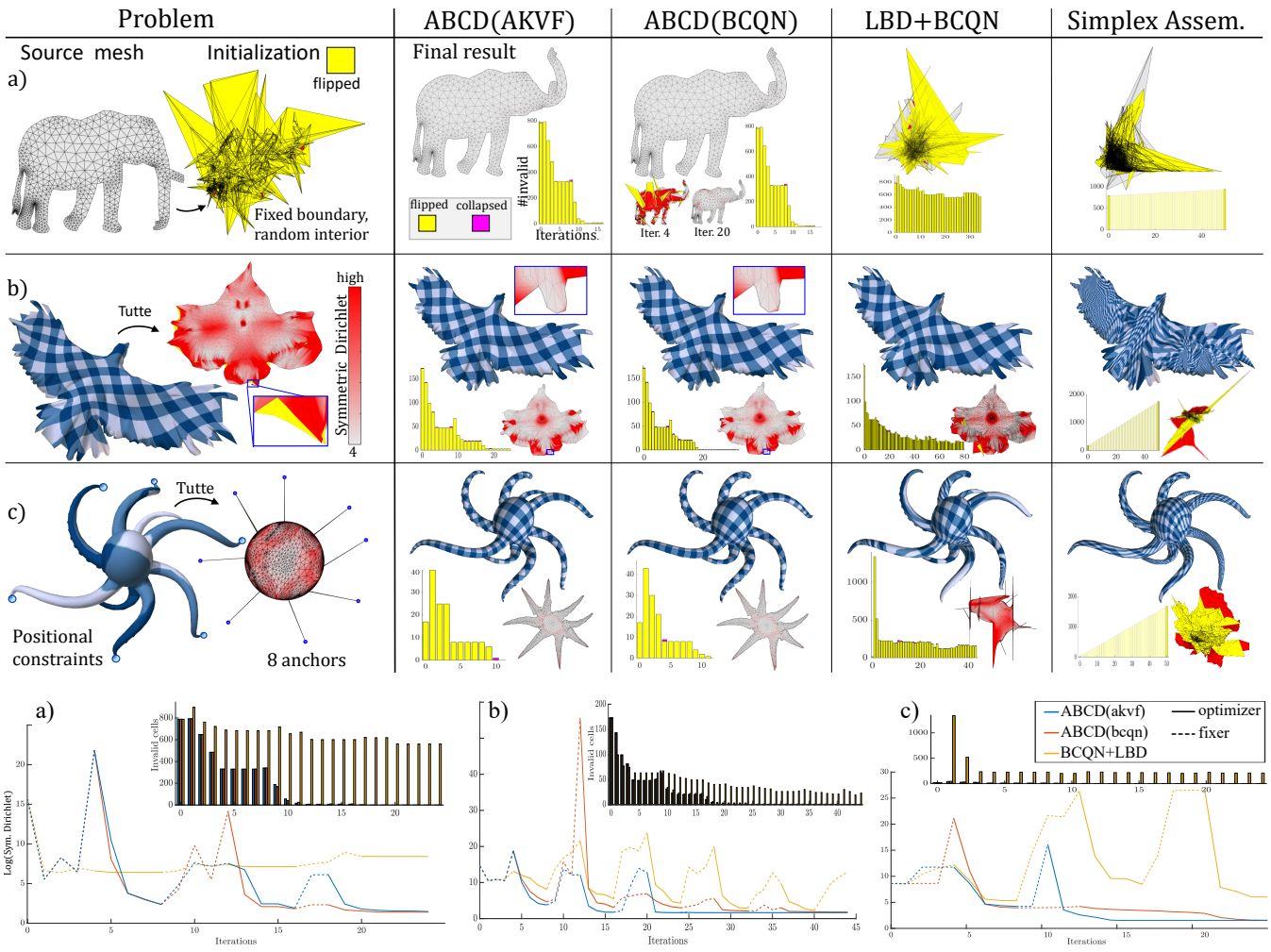


Figure 17: *ABCD with first order core-solvers versus other related methods in 2D. Each element in the table shows the final result of the corresponding method and invalid triangle numbers per iteration. We plot Symmetric Dirichlet energy in logarithmic scale for each problem (bottom), where solid curves 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.*

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 '$\sim$' relation. As a result, we can represent E as a sum of energies, defined over disjoint simplices:

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

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 $\boldsymbol{d}$ and $\bar{\boldsymbol{d}}$ the descent directions attained in S and ABCD(S), respectively. Define $E[\boldsymbol{z}] = E(\boldsymbol{x}^0 + \boldsymbol{z})$, then the resulting energy in ABCD(S) is

$$E(\bar{\boldsymbol{x}}) = \min_{\Delta t_1} E_{B_1}\left[\Delta t_1 \bar{\boldsymbol{d}}_{B_1}\right] + \min_{\Delta t_2} E_{B_2}\left[\Delta t_2 \bar{\boldsymbol{d}}_{B_2}\right] + \underbrace{E_{B_0}}_{\text{const.}}, \qquad (25)$$

while running a single iteration of S yields

$$E(\boldsymbol{x}) = \min_{\Delta t} \left(E_{B_1}\left[\Delta t \boldsymbol{d}_{B_1}\right] + E_{B_2}\left[\Delta t \boldsymbol{d}_{B_2}\right] + E_{B_0}\right) \qquad (26)$$

If $\bar{\boldsymbol{d}}_{B_i} = \boldsymbol{d}_{B_i}$ for $i = 1,2$, then the proposition's inequality (25)$\leq$(26) is proven because 'min' is sub-additive and E_{B_0} is constant by the definition (only free vertices are updated in S and 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 $\boldsymbol{x}_w$, where w are neighbors 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

$$\begin{pmatrix} H_{B_0} & * & * \\ * & H_{B_1} & 0 \\ * & 0 & H_{B_2} \end{pmatrix} \begin{pmatrix} 0 \\ \boldsymbol{d}_{B_1} \\ \boldsymbol{d}_{B_2} \end{pmatrix} = - \begin{pmatrix} \nabla E_{B_0} \\ \nabla E_{B_1} \\ \nabla E_{B_2} \end{pmatrix}, \qquad (27)$$

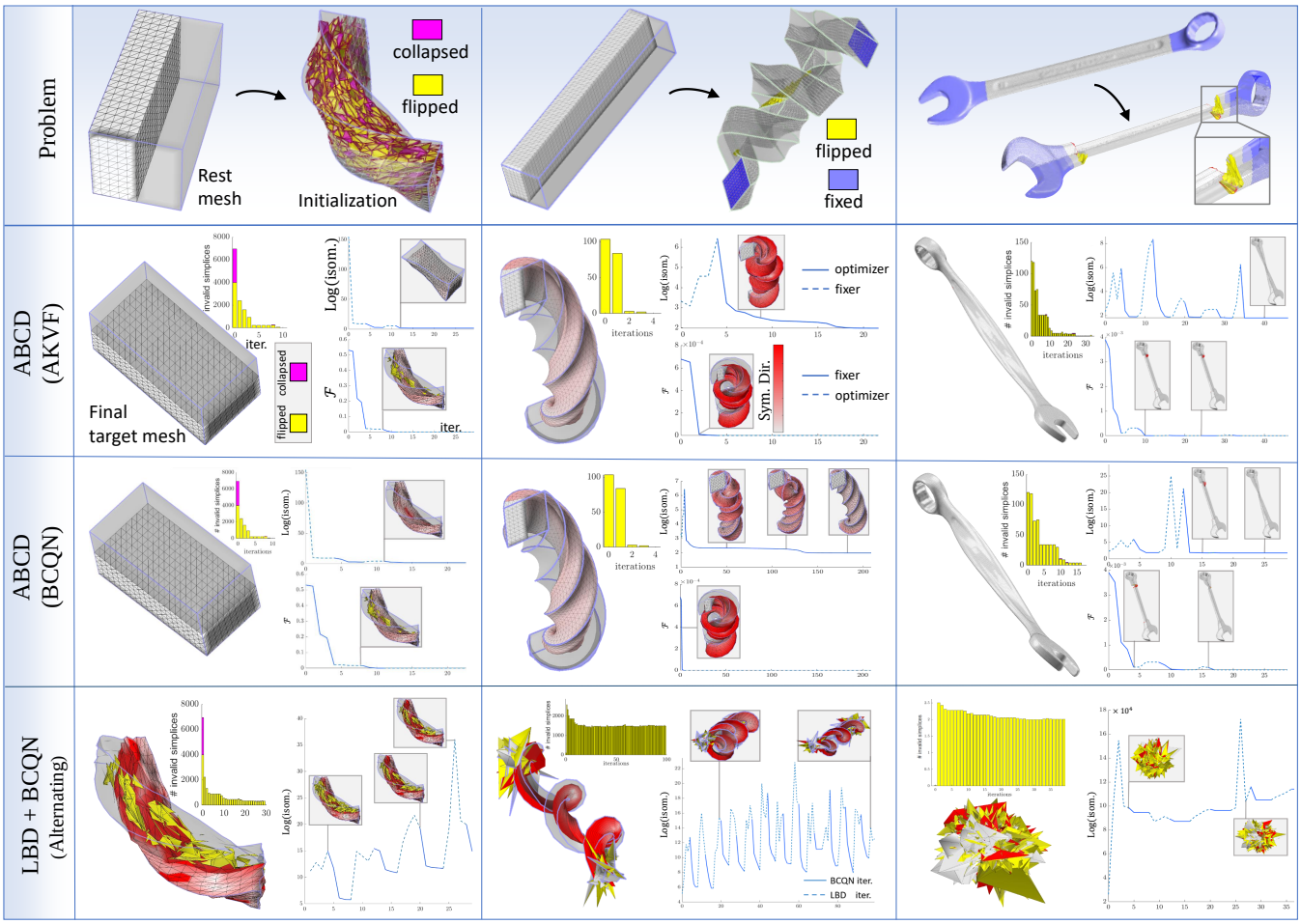


Figure 18: *ABCD versus alternating combination of LBD and BCQN on tet meshes. We show number of invalid tets and energy values per iteration. We plot energy E_j in solid lines for iterations in which E_j is minimized and in other iterations E_j is shown in dashed lines.*

where each H_{B_i} is a block of H, computed for vertices in B_i. Then, $\bar{\boldsymbol{d}}_{B_i} = \boldsymbol{d}_{B_i}$ for $i = 1, 2$ is concluded from (27). For N free blocks $B_1, \ldots, B_N$ we repeat the proof with $B_0 = \mathbf{V} \setminus (B_1 \cup B_2 \ldots \cup B_N)$ to show that $\bar{\boldsymbol{d}}_{B_i} = \boldsymbol{d}_{B_i}$ for $i \geq 1$.

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 estimates the performance ratios of k global-local iterations by

$$\mathcal{R}^i = \frac{1}{\min\{k,i\}(1+\mu)} \sum_{p=i}^{\lceil i-k+1 \rceil} \frac{\Delta E^{g(p)}}{\Delta E^{l(p)}} + \mu \frac{\mathrm{Disp}^{g(p)}}{\mathrm{Disp}^{l(p)}}, \qquad (28)$$

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 equally performed global-local iterations. We compute (28) and adjust partitioning thresholds for the next cycle via

$$\left(K^{2i+1}, K^{2i+2}\right) = \mathrm{Blend}\left(K^{2i-1}, K^{2i}, \mathcal{R}^i, \mathcal{L}_{\max}\right), \qquad (29)$$

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 $f : M \to \Omega$ 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 Fig. 9 (right) we attain a globally injective map by constructing a scaffold mesh [JSP17] and running Algorithm 3 with the line search step, modified according to [NSZ18], to prevent boundary self-intersections.

References

[AL13] AIGERMAN, NOAM and LIPMAN, YARON. "Injective and bounded distortion mappings in 3D". *ACM Trans. Graph. (TOG)* (2013) 3, 15.

[BDS*12] BOUAZIZ, SOFIEN, DEUSS, MARIO, SCHWARTZBURG, YULIY, et al. "Shape-Up: Shaping Discrete Geometry with Projections". *Computer Graphics Forum*. 2012, 1657–1667 3.

[BML*14] BOUAZIZ, SOFIEN, MARTIN, SEBASTIAN, LIU, TIANTIAN, et al. "Projective dynamics: fusing constraint projections for fast simulation". *ACM Transactions on Graphics (TOG)* 33.4 (2014), 154 3.

[CBSS17] CLAICI, S, BESSMELTSEV, M, SCHAEFER, S, and SOLOMON, J. "Isometry-Aware Preconditioning for Mesh Parameterization". *Computer Graphics Forum*. Vol. 36. Wiley Online Library. 2017, 37–47 3, 7.

[CM83] COLEMAN, THOMAS F and MORÉ, JORGE J. "Estimation of sparse Jacobian matrices and graph coloring blems". *SIAM journal on Numerical Analysis* 20.1 (1983), 187–209 11.

[CPSS10] CHAO, ISAAC, PINKALL, ULRICH, SANAN, PATRICK, and SCHRÖDER, PETER. "A simple geometric model for elastic deformations". *ACM transactions on graphics (TOG)* 29.4 (2010), 38 4.

[CW17] CHEN, RENJIE and WEBER, OFIR. "GPU-accelerated locally injective shape deformation". *ACM Transactions on Graphics (TOG)* 36.6 (2017), 214 3.

[DMA02] DESBRUN, MATHIEU, MEYER, MARK, and ALLIEZ, PIERRE. "Intrinsic parameterizations of surface meshes". *Computer graphics forum*. Vol. 21. 3. Wiley Online Library. 2002, 209–218 4.

[FL16] FU, XIAO-MING and LIU, YANG. "Computing inversion-free mappings by simplex assembly". *ACM Trans. Graph. (TOG)* 35.6 (2016), 216 3, 5.

[FLG15] FU, XIAO-MING, LIU, YANG, and GUO, BAINING. "Computing locally injective mappings by advanced MIPS". *ACM Transactions on Graphics (TOG)* 34.4 (2015), 71 2–5, 8.

[Flo03] FLOATER, MICHAEL. "One-to-one piecewise linear mappings over triangulations". *Mathematics of Computation* 72.242 (2003) 3.

[GSC18] GOLLA, BJÖRN, SEIDEL, HANS-PETER, and CHEN, RENJIE. "Piecewise linear mapping optimization based on the complex view". *Computer Graphics Forum*. Vol. 37. 7. Wiley Online Library. 2018 3.

[HG00] HORMANN, KAI and GREINER, GÜNTHER. *MIPS: An efficient global parametrization method*. Tech. rep. DTIC Document, 2000 4–6.

[JSP17] JIANG, ZHONGSHI, SCHAEFER, SCOTT, and PANOZZO, DANIELE. "Simplicial complex augmentation framework for bijective maps". *ACM Trans. Graph. (TOG)* 36.6 (2017), 186 3, 15.

[KABL14] KOVALSKY, SHAHAR Z, AIGERMAN, NOAM, BASRI, RONEN, and LIPMAN, YARON. "Controlling Singular Values with Semidefinite Programming". *ACM Trans. Graph. (TOG)* 33.4 (2014), 68 3.

[KABL15] KOVALSKY, SHAHAR Z, AIGERMAN, NOAM, BASRI, RONEN, and LIPMAN, YARON. "Large-scale bounded distortion mappings". *ACM Trans. Graph* 34.6 (2015), 191 3.

[KGL16] KOVALSKY, SHAHAR Z, GALUN, MEIRAV, and LIPMAN, YARON. "Accelerated quadratic proxy for geometric optimization". *ACM Trans. Graph. (TOG)* 35.4 (2016), 134 2, 3.

[LBK16] LIU, TIANTIAN, BOUAZIZ, SOFIEN, and KAVAN, LADISLAV. "Towards real-time simulation of hyperelastic materials". *arXiv preprint arXiv:1604.07378* (2016) 3.

[LBK17] LIU, TIANTIAN, BOUAZIZ, SOFIEN, and KAVAN, LADISLAV. "Quasi-newton methods for real-time simulation of hyperelastic materials". *ACM Trans. Graph. (TOG)* 36.4 (2017), 116a 3.

[LKK*18] LI, MINCHEN, KAUFMAN, DANNY M, KIM, VLADIMIR G, et al. "OptCuts: joint optimization of surface cuts and parameterization". *SIGGRAPH Asia 2018 Technical Papers*. ACM. 2018, 247 3.

[LPRM02] LÉVY, BRUNO, PETITJEAN, SYLVAIN, RAY, NICOLAS, and MAILLOT, JÉROME. "Least squares conformal maps for automatic texture atlas generation". *Acm transactions on graphics (tog)*. Vol. 21. 3. ACM. 2002, 362–371 4.

[LX*08] LIU, LIGANG, 0021, LEI ZHANG, XU, YIN, et al. "A Local/Global Approach to Mesh Parameterization". *Comput. Graph. Forum* 27.5 (2008), 1495–1504 3, 5.

[LYNF18] LIU, LIGANG, YE, CHUNYANG, NI, RUIQI, and FU, XIAO-MING. "Progressive parameterizations". *ACM Transactions on Graphics (TOG)* 37.4 (2018), 41 3, 4, 12, 13, 22.

[Nes83] NESTEROV, YURII E. "A method for solving the convex programming problem with convergence rate O (1/kˆ 2)". *Dokl. Akad. Nauk SSSR*. Vol. 269. 1983, 543–547 3.

[NSZ18] NAITSAT, ALEXANDER, SAUCAN, EMIL, and ZEEVI, YEHOSHUA Y. "Geometry-based distortion measures for space deformation". *Graphical Models* 100 (2018), 12–25 2, 4–6, 15.

[NZ19] NAITSAT, ALEXANDER and ZEEVI, YEHOSHUA Y. "Multi-resolution approach to computing locally injective maps on meshes". *ACM SIGGRAPH 2019 Posters*. ACM. 2019, 87 12, 18.

[PDZ*18] PENG, YUE, DENG, BAILIN, ZHANG, JUYONG, et al. "Anderson Acceleration for Geometry Optimization and Physics Simulation". *arXiv preprint arXiv:1805.05715* (2018) 3, 5, 13.

[PTH*17] PORANNE, ROI, TARINI, MARCO, HUBER, SANDRO, et al. "Autocuts: simultaneous distortion and cut optimization for UV mapping". *ACM Trans. Graph. (TOG)* 36.6 (2017), 215 3, 8.

[RPPSH17] RABINOVICH, MICHAEL, PORANNE, ROI, PANOZZO, DANIELE, and SORKINE-HORNUNG, OLGA. "Scalable locally injective mappings". *ACM Transactions on Graphics (TOG)* (2017) 3–5.

[SC18] SAWHNEY, ROHAN and CRANE, KEENAN. "Boundary first flattening". *ACM Transactions on Graphics (ToG)* 37.1 (2018), 5 3, 17, 19.

[SJZP19] SHEN, HANXIAO, JIANG, ZHONGSHI, ZORIN, DENIS, and PANOZZO, DANIELE. "Progressive embedding". *ACM Transactions on Graphics (TOG)* 38.4 (2019), 32 3, 12, 13.

[SPSH*17] SHTENGEL, ANNA, PORANNE, ROI, SORKINE-HORNUNG, OLGA, et al. "Geometric optimization via composite majorization". *ACM Trans. Graph* 36.4 (2017), 38 3.

[SS15] SMITH, JASON and SCHAEFER, SCOTT. "Bijective parameterization with free boundaries". *ACM Trans. Graph. (TOG)* 34.4 (2015), 70 3, 4, 9, 15.

[TRG16] TAPPENDEN, RACHAEL, RICHTÁRIK, PETER, and GONDZIO, JACEK. "Inexact coordinate descent: complexity and preconditioning". *Journal of Optimization Theory and Applications* 170.1 (2016) 8.

[TSIF05] TERAN, JOSEPH, SIFAKIS, EFTYCHIOS, IRVING, GEOFFREY, and FEDKIW, RONALD. "Robust quasistatic finite elements and flesh simulation". *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*. ACM. 2005, 181–190 3.

[Tut63] TUTTE, WILLIAM THOMAS. "How to draw a graph". *Proceedings of the London Mathematical Society* 3.1 (1963), 743–767 3.

[WMZ12] WEBER, OFIR, MYLES, ASHISH, and ZORIN, DENIS. "Computing Extremal Quasiconformal Maps". *Comput. Graph. Forum* 31.5 (2012), 1679–1689 3.

[WY16] WANG, HUAMIN and YANG, YIN. "Descent methods for elastic body simulation on the GPU". *ACM Trans. Graph. (TOG)* 35.6 (2016), 212 3.

[ZBK18] ZHU, YUFENG, BRIDSON, ROBERT, and KAUFMAN, DANNY M. "Blended cured quasi-newton for distortion optimization". *ACM Transactions on Graphics (TOG)* 37.4 (2018), 40 3, 6, 10, 20.

Adaptive Block Coordinate Descent for Distortion Optimization

Alexander Naitsat[1], Yufeng Zhu[2] and Yehoshua Y. Zeevi[1]

[1]Viterbi Faculty of Electrical Engineering, Technion - Israel Institute of Technology
[2]Facebook Reality Labs

Supplementary Document

In this document we provide some additional results and discuss ABCD limitations and certain technical details related to an optimal usage of our algorithm. We adopt abbreviations and notations introduced in the paper. In particular, $\mathcal{D}_j$ denotes a SVD-based distortion measures; its energy (1) is denoted by E_j and S_j refers to a core-solver chosen to optimizes E_j in ABCD.

1. Additional Results

We provide the following additional results:

- Fig. 26 presents results of conformal initialization scheme for all 190 meshes, collected from the Progressive Parametrization dataset and summarized in Fig. 12.
- In Fig. 24, we tested an impact of setting different hyper-parameters in ABCD algorithm. We examined both the fixer and optimizer hyper-parameters that control global and block iteration divisions, line search, LGB and the behavior of measure $\mathcal{F}$. Each trial, tested in Fig. 24, had a different set of hyper-parameters. These parameters are specified in Table 1.
- In Fig. 19, we experimented with as killing as possible constrained parametrization. In this figure, we test how ABCD behaves under positional constraints that enforce target simplices to be negatively oriented.
- In Fig. 21, we used ABCD(PN) to fix failures in Tutte embedding and parametrize space fitting curves with different initialization schemes. We compared the two initialization methods: Tutte embedding and the BFF mapping [SC18]. According to our results, the proposed BFF initialization scheme outperforms significantly the standard approach with Tutte embedding. Starting with BFF conformal embedding produces no flips in these experiments, whereas Tutte embedding produces a small number of foldovers due to the presence of extremely distorted triangles. At the same time, conformal initializations are very close to the final isometric parametrization.
- Fig. 23 compares ABCD(BCQN) and ABCD(PN) per iteration results obtained in our experiments with randomly placed anchors (Fig. 11 top) and randomized deformations (Fig. 11 bottom).

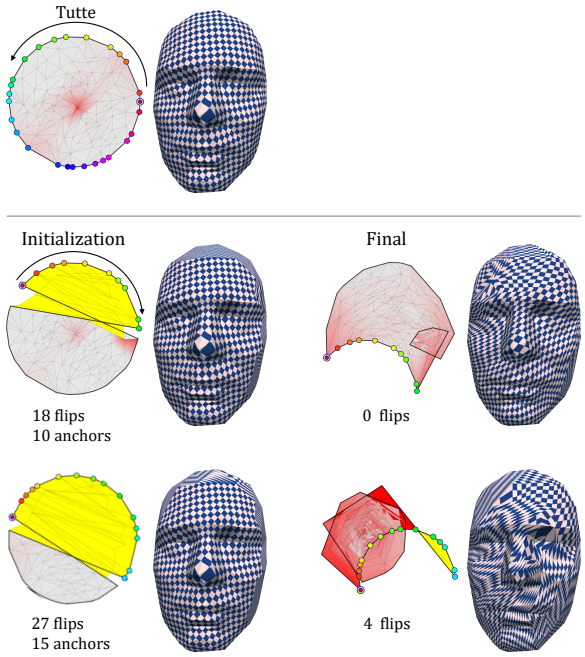


Figure 19: *Running a stress test of ABCD(PN) to identify positional constraints inconsistent with a positively oriented UV-map. To find failure cases in the algorithm, we deform Tutte mapping and fix positions of distorted vertices.* ***Top****: Positively-oriented Tutte mapping into a disc;* ***Middle****: We flip positions of 10 successive boundary anchors.* ***Bottom****: We flip positions of 15 successive boundary anchors.*

2. Failures and Challenging Cases

Because we use hard positional constraints, our algorithm cannot repair inversions if given positional constraints, $A\boldsymbol{x} = \boldsymbol{z}$, contradict the orientation requirement, $\forall c \in C : \det df_c > 0$.

Obviously, if $A \subset \mathbf{V}$ is a set of anchors, then the restriction of an initial map to A has to be an orientation-preserving map. In Fig. 19, we use this simple fact to test how many inconsistently placed anchors can be handled by our algorithm. We start with a positively

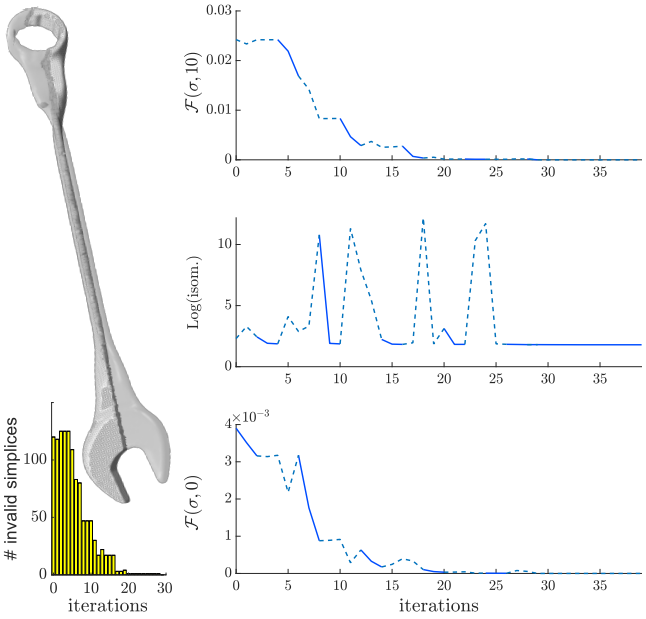


Figure 20: *Here we solve the third problem from Fig. 18 by minimizing three distortions: $\mathcal{F}(\boldsymbol{\sigma},0)$, $\widetilde{D}_{iso}$ and $\mathcal{F}(\boldsymbol{\sigma},10)$. These distortions are optimized according to Algorithm 4 with $n_j = 2$ and the results are visualized in the same way as in Fig. 18.*

oriented initialization and add gradually anchors to revert counter clockwise orientation of the boundary. Our algorithm succeeds to repair inversion for almost a half of inconsistently placed boundary vertices (Fig. 19 middle) and it fails when more than half of the boundary is reverted to the clockwise orientation. Note that, in the unconstrained version of the problem, ABCD(PN) repairs all the inversions without any difficulty.

To resolve failures induced by inconsistent anchor positions, one should use soft positional constraints, instead of hard ones. That is, one should replace (4) with a cost function that penalizes anchor deviations from their prescribed positions. However, the subject on soft constraints is beyond the scope of this work because we have restricted objective functions in (2) to be SVD-based energies.

In some challenging cases, ABCD may struggle to repair inversions due to the presence of extreme differences in sizes of target elements. In particular, running BFF on some meshes from Fig. 26 produces conformal mapping with many "poles" — areas where target elements around inversions are shrunken to extremely small areas. It is difficult to set a proper singularity threshold ε in (13) that works well on these meshes. Setting ε too low may lead to numerical issues in the optimizer stage, setting it too hight may false-positively identify some proper triangles as invalid elements. Nevertheless, we succeeded to resolve these scenarios from Fig. 26, but it took significantly more time for our algorithm to converge. Not that most of these meshes cannot be processed by standard methods because, in these cases, Tutte map tends to produce similar poles with inverted triangles. See Section 8 for our discussion on Tutte embedding and precision lost issues.

Algorithm 4 ABCD with LGB for q measures

Input:

- Source mesh $(\mathbf{V}, \mathbf{C})$ and vertex initialization $\boldsymbol{x}^0$.
- Distortions $\mathbf{D} = \{\mathcal{D}_j\}_{j=1}^q$ and their gradients $\{\nabla\mathcal{D}_j\}_{j=1}^q$.
- Core solvers $\mathbf{S} = \{S_j\}_{j=1}^q$ with iteration numbers $\{n_j\}_{j=1}^q$.

$i \leftarrow 0$.
for $j = 1,\dots,q$ **do**
  $G_j \leftarrow 1$, $L_j \leftarrow 0$, $b_j \leftarrow 1$, state$(j) \leftarrow$ 'global'.
end for

repeat
  for $j = 1,\dots,q$ **do** $\{n_j$ times each$\}$
    $E_j \leftarrow \mathcal{D}_j$ energy of $f(\boldsymbol{x}^i)$ according to (1).
    if average$(E_j) \le \min \mathcal{D}_j$ **continue**.
    if state$(j) =$'local' **then**
      $K \leftarrow L_j$.
    else
      $K \leftarrow G_j$.
    end if
    $\boldsymbol{d} \leftarrow$ descent direction of E_j attained by S_j.
    $\{\boldsymbol{x}_B^i\}_{B\in\mathbf{B}} \leftarrow$ BlockPartition$(K, \boldsymbol{d})$ via Algorithm 1.
    for $B \in \mathbf{B}$ in parallel **do**
      $\boldsymbol{x}_B^{i+1} \leftarrow$ solution of (2) in B using S_j with b_j max iterations.
    end for
    if state$(j) =$'local' **then**
      $(G_j, L_j) \leftarrow$ Blend(G_j, L_j) via Algorithm 2.
      Update b_j.
    end if
    Switch state(j) between 'local' and 'global'.
  end for
  $i \leftarrow i+1$.
until (24) is true for each E_j or $i = i_{\max}$.

We found that if the parameter Λ in (13) is slightly increased, then it takes less ABCD iterations to process meshes with poles.

3. Initialization with Decimated UV-Map

We provide some explanations on the *embedding deformation* method [NZ19]. We used this method in Fig. 13 to construct a better starting point for surface parametrization.

Assume that Y is a decimated mesh of a mesh X, $\boldsymbol{y}^0$ is the Tutte embedding of Y and $\boldsymbol{y}$ is a parametrization of Y, computed by initializing isometric optimization with $\boldsymbol{y}^0$. The embedding deformation method is a simple approach aimed at mapping X onto the region contained in $\partial\boldsymbol{y}$ with only a small number of foldovers. Assume that $\boldsymbol{x}^0$ is Tutte embedding of X to a disc $(1-\varepsilon)D$. Then, the embedding deformation method includes the following steps:

1. Construct the simplical mapping $g : \boldsymbol{y}^0 \mapsto \boldsymbol{y}$.
2. For each vertex v of X, find simplex $s(v)$ of Y such that $\boldsymbol{x}_v^0$ is contained in the initial mapping of $s(v)$, i.e., $\boldsymbol{x}_v^0$ is located in the triangle with coordinates $\boldsymbol{y}_u^0$, $u \in s(v)$.
3. Map each vertex v of X to $\boldsymbol{x}_v = g_{s(v)}(\boldsymbol{x}_v^0)$.

The above steps are illustrated by Fig. 25. Note that we scale slightly $\boldsymbol{x}^0$ by a factor of $(1-\varepsilon)$ to avoid encountering numerical issues on the mesh boundary.

4. GD for Map Fixer Measures

To justify our definitions in (14), assume, first, that we have only a single simplex c and observe how updating $\boldsymbol{x}$ via GD for distortion $\mathcal{D}$ affects df_c. We can represent df_c as a linear function of singular values $\boldsymbol{\sigma}$, acting on its right singular vectors:

$$df_c[\boldsymbol{\sigma}] : \boldsymbol{v}_i \mapsto \sigma_i \boldsymbol{u}_i\,, \tag{30}$$

where $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$ are columns of U and V matrices that appear in the signed SVD of df_c, $df_c = U\mathrm{diag}(\sigma_1,...,\sigma_m)V^\top$. Since GD modifies df_c non-rigidly, its rotation components U and V are unchanged. Thus, the GD update step has a particularly simple form with respect to (30)

$$\boldsymbol{\sigma}(t+\Delta t) = \boldsymbol{\sigma}(t) - \Delta t \nabla_{\boldsymbol{\sigma}} \mathcal{D}\,. \tag{31}$$

Choose $\mathcal{D} = \mathcal{D}_{\text{collapse}}$, then $-\nabla\mathcal{D}_{\text{collapse}}$ is a decent direction of $\mathcal{D}_{\text{collapse}}$ and $\boldsymbol{\sigma}(t+\varepsilon)$ is its nearest minimum obtained in (31). If a number of simplices $c_1,\ldots,c_p$ share a common vertex v, then the contribution of each gradient $\nabla\mathcal{D}_{\text{collapsed}}(\boldsymbol{\sigma}(df_c))$ is summed up to form the energy gradient at v. Hence, repairing collapsed simplices with $\mathcal{D}_{\text{collapse}}$ reduces unnecessary deformation of valid elements, since (31) modifies df_c only on degenerate simplices along their collapsed axes. Similarly, only vertices of inverted elements are modified in GD optimization of $\mathcal{D}_{\text{flip}}$.

5. Algorithm Generalization

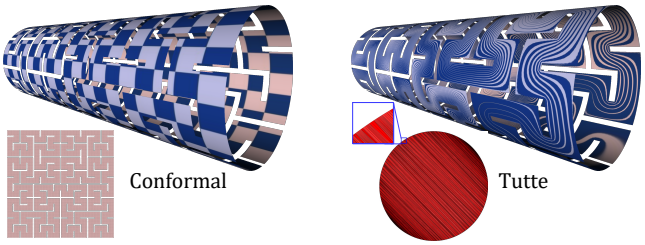
In general, our algorithm can optimize an arbitrary list of distortion measures. In particular, ABCD can function well with more than two measures, $\{\mathcal{D}_j\}_{j=1}^q$, if these measures do not contradict each other and there is at least one map fixer and at least one optimizer distortion (10). In this case, we can use Algorithm 4 that presents ABCD generalization for q measures. This algorithm uses the notion of the local and global iterations and other related notations, introduced in Appendix B.

Fig. 20 illustrates an example of running Algorithm 4 with three distortion measures.

6. Configurations for Local-Global Blending

If LGB is enabled in Algortihm 3, then we advise to set an even number of successive iterations n_j for each energy E_j to prevent the global-local cycle of one energy from being interrupted by iterations of another one. These interruptions adversely affect the local-global blending because the mutual impact of different measures can be mixed up in (28), leading to incorrect performance estimates.

We also suggest to start with the "global-local" order in LGB. That is, we suggest to set $K^1 = \mathcal{L}_{\max}$ and $K^2 = \mathcal{L}_{\min}$ in the first call to Algorithm 2. This order of partitioning thresholds leads to a better results because at the beginning global optimization tends to deform the entire shape of a target mesh, while local optimization are more concentrated on improving shapes of individual simplices.



#triangles	#invalids		initialization time		timing to fix		iterations to fix		timing total		iterations total	
	BFF	Tutte	BFF	Tutte	BFF	Tutte	BFF	Tutte	BFF	Tutte	BFF	Tutte
32228	0	1	0.498	0.094	0	0.926	0	8	0.805	1.915	11	23
51898	0	10	0.749	0.133	0	0.727	0	3	1.473	7.582	11	54
128912	0	32	1.983	0.384	0	25.31	0	41	3.233	35.87	11	84
515648	0	62	7.37	1.594	0	230.4	0	151	13.5	241.8	11	164

Figure 21: *Unconstrained parametrization of Hilbert curve of different resolutions. We use ABCD(PN) to fix inversions in Tutte maps and compare parametrization, initialized by Tutte, with the parametrization initialized by BFF [SC18] conformal mapping. We run these tests with Pardiso library on four-core i7 CPU and report runtimes in seconds.*

As a result, the progress of BCD step at the first iteration is often lost at the next step, if we start with $K^1 = \mathcal{L}_{\min}$, $K^2 = \mathcal{L}_{\max}$ instead.

7. Quasi-Newton Solvers

For an optimal integration with ABCD algorithm, the parameters for a quasi-newton solver need to be saved when it completes its iterations in the current cycle, and reloaded when we resume its iterations in the next cycle. In this way, one can achieve a better continuity in optimization process between different cycles, which experimentally yields faster convergence. For instance, if S_j is the L-BFGS solver, then the values of x_B^i and ∇E_B^i should be stored over a number of successive iterations. This data should be restored when S_j resumes its operation, since to write a secant equation one would like to consider the current gradient, and gradients from previous run of the preceding optimization cycle of S_j, rather than use gradients from iterations of a different measure.

8. Note on Enhanced Line Search

We added the following simple modification to the Armijo back tracking line search: we store coordinates $\boldsymbol{x}_B'(t)$ that induced the highest decrease in block energy (21) during the line search. If the Armijo-Goldstein condition is not satisfied within the maximal number of the line search iterations, then block coordinates for the next iteration are set to $\boldsymbol{x}_B'(t)$.

This enchantment is aimed at improving map fixer performance by making it less sensitive to line search parameters. In particular, in some configurations, map fixer can repair inversions but, at the same time, it may struggle to attain a sufficient decrease in $\mathcal{F}$ to satisfy the Armijo rule. In these scenarios, our method improves the fixer performance.

Note that core-solver S can modify a solution of the descent equation (7) to get more progress in the line search. We consider

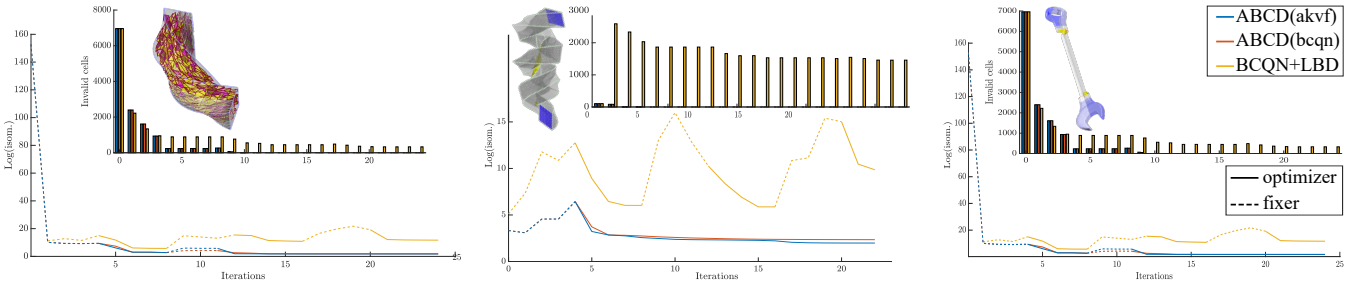


Figure 22: *Comparing ABCD(BCQN), ABCD(AKVF) and LBD+BCQN from Fig. 18. Here we plot number of inverted tetrahedrons and isometric energy per iteration in the same way as in Fig. 17.*

Table 1: *ABCD parameters are specified for each trial, tested in Fig. 24. We use the following notations: b'_j refers to the maximal number of block iterations; 'Search' refers to the line search filtering range, described in Section 7.3; $\mathcal{L}'_{\min} = \mathcal{L}_{\min}\ /\ \mathcal{L}_{\max}$ and $\mathcal{L}'_{\min}$ = '$\cdots$' denotes trials in which LGB is disabled.*

	Fixer parameters					Optimizer parameters			
Trial	Iterations n_j	Iterations b'_j	Search $1-\delta^-$	LGB $\mathcal{L}'_{\min}$	Measure Λ	Iterations n_j	Iterations b'_j	Search $1-\delta^+$	LGB $\mathcal{L}'_{\min}$
1	4	4	1.1	$\cdots$	0	4	1	0.5	$\cdots$
2	2	1	1.1	$\cdots$	0	1	1	0.5	$\cdots$
3	1	1	1.1	$\cdots$	0	1	1	0.5	$\cdots$
4	4	10	1.1	$\cdots$	0	4	1	0.5	$\cdots$
5	2	10	1.1	$\cdots$	0	2	1	0.5	$\cdots$
6	1	10	1.1	$\cdots$	0	1	1	0.5	$\cdots$
7	4	10	1.1	$\cdots$	0	4	4	0.5	$\cdots$
8	2	10	1.1	$\cdots$	0	1	4	0.5	$\cdots$
9	1	10	1.1	$\cdots$	0	1	4	0.5	$\cdots$
10	4	10	1.5	$\cdots$	0	4	4	0.5	$\cdots$
11	4	10	2.5	$\cdots$	0	4	4	0.5	$\cdots$
12	2	10	10	$\cdots$	0	2	4	0.5	$\cdots$
13	4	4	1.1	0.05	0	2	2	0.5	0.05
14	4	4	1.1	0.1	0	2	2	0.5	0.1
15	4	10	1.1	0.2	0	2	2	0.5	0.2
16	4	4	1.1	0.01	0	2	2	0.5	$\cdots$
17	4	4	1.1	0.1	0	2	2	0.5	$\cdots$
18	4	4	1.1	0.2	0	2	2	0.5	$\cdots$
19	4	4	1.1	$\cdots$	0.001	2	2	0.5	$\cdots$
20	4	4	1.1	$\cdots$	0.1	2	2	0.5	
21	4	4	1.1	$\cdots$	1	2	2	0.5	$\cdots$

these modifications as the part of the descent direction computation, employed in ABCD(S) algorithm. For example, the barrier-aware filter on search directions [ZBK18] is the part of BCQN solver and, therefore, it is also employed in ABCD(BCQN).

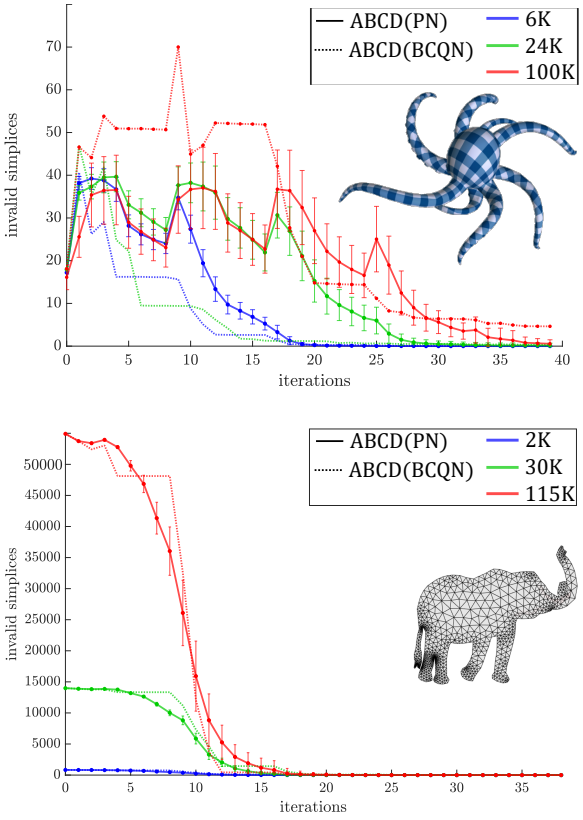


Figure 23: *Showing an average number of invalid triangles per iteration, obtained in the experiments from Fig. 11. Vertical bars show the standard deviation.*

9. Weighted Simplex Displacement Norm

We found that using the coordinates of simplex centers in (19), instead of vertices, yields better scalability, since $\Delta \boldsymbol{x}^i$ is more likely to accumulate numerical errors. We define a simplex-based displacement norm as follows:

$$\mathrm{Disp}_C^i = \frac{1}{\bar{e}} \left\| \mathrm{diag}(\boldsymbol{w}) \left(\Delta C^i - \mathrm{Proj}_{\mathrm{Ker}\left(C^{i-1}\right)} \left(\Delta C^i \right) \right) \right\|_{\mathrm{Fro}}, \qquad (32)$$

where $\boldsymbol{w}$ is the stack of weights $w(c)$ from (1), $C^i \in \mathbb{R}^{|\mathbf{C}| \times m}$ are simplex target centers in i^{th} iteration, $\Delta C^i = C^i - C^{i-1}$ and $\bar{e}$ is an approximate average length of edges in the target domain.

Problem	#triangles	#invalids	#anchors	iterations to fix			timing to fix			iterations total			timing total		
				trial 1	trial 2	trial 3	trial 1	trial 2	trial 3	trial 1	trial 2	trial 3	trial 1	trial 2	trial 3
Gecko	1238	5	45	9	11	13	0.106	0.162	0.193	37	39	38	0.193	0.28	0.313
Elephant	1796	784	788	11	17	15	0.138	0.263	0.23	38	43	39	0.237	0.41	0.355
Octopus	5986	17	8	12	38	59	0.293	0.933	1.67	39	66	65	0.6	1.38	1.777
Grid	20000	4149	0	9	12	13	0.612	0.874	1.021	36	43	43	1.759	2.67	2.717
Octopus	23944	17	8	28	336	78	1.769	22.034	6.522	55	336	89	3.041	22.11	7.254
Elephant	28736	14038	1648	43	52	23	3.253	4.994	2.654	69	76	54	4.692	6.77	4.867
Octopus	95776	19	8	41	160	132	11.401	34.335	34.596	74	170	139	18.491	36.44	36.129
Gorilla	100000	7	0	3	5	3	0.465	1.573	1.299	12	13	12	3.209	3.33	3.209
Elephant	114944	54900	3296	18	28	29	6.69	8.961	10.376	52	55	55	14.344	15.12	16.183
Gorilla	839092	20	0	3	4	3	4.98	13.701	13.616	16	16	15	34.459	37.57	35.113

Problem	#triangles	#invalids	#anchors	iterations to fix			timing to fix			iterations total			timing total		
				trial 13	trial 14	trial 15	trial 13	trial 14	trial 15	trial 13	trial 14	trial 15	trial 13	trial 14	trial 15
Gecko	1238	5	45	7	7	7	0.133	0.121	0.116	22	22	22	0.259	0.25	0.242
Elephant	1796	784	788	10	9	9	0.204	0.173	0.162	25	28	19	0.343	0.35	0.259
Octopus	5986	17	8	8	9	8	0.357	0.313	0.228	26	27	24	0.938	0.89	0.726
Grid	20000	4149	0	7	7	7	3.748	1.571	0.764	21	20	21	5.399	3.11	2.418
Octopus	23944	17	8	8	49	37	1.022	4.757	3.709	23	57	51	2.925	5.82	5.44
Elephant	28736	14038	1648	37	38	43	7.407	5.819	6.074	42	47	54	8.177	7.10	7.613
Octopus	95776	19	8	26	45	52	8.8	13.779	14.924	42	55	63	15.476	17.80	19.018
Gorilla	100000	7	0	4	3	3	1.276	0.473	0.478	9	8	8	3.286	3.10	3.063
Elephant	114944	54900	3296	14	14	19	10.326	7.023	8.489	28	28	32	16.365	13.08	14.091
Gorilla	839092	20	0	3	3	3	5.835	5.828	5.773	10	10	10	37.57	37.41	37.701

Problem	#triangles	#invalids	#anchors	iterations to fix			timing to fix			iterations total			timing total		
				trial 4	trial 5	trial 6	trial 4	trial 5	trial 6	trial 4	trial 5	trial 6	trial 4	trial 5	trial 6
Gecko	1238	5	45	9	6	7	0.145	0.109	0.121	37	35	41	0.279	0.25	0.289
Elephant	1796	784	788	11	10	11	0.173	0.16	0.175	38	38	37	0.313	0.32	0.373
Octopus	5986	17	8	10	18	24	0.306	0.531	0.671	38	42	54	0.762	0.92	1.168
Grid	20000	4149	0	9	7	7	0.726	0.672	0.61	36	40	39	2.296	2.54	2.435
Octopus	23944	17	8	17	15	23	1.429	1.513	2.417	51	43	61	3.544	3.25	4.787
Elephant	28736	14038	1648	78	31	15	7.241	3.523	1.977	104	59	50	9.099	5.55	4.494
Octopus	95776	19	8	22	61	27	6.231	17.361	8.191	60	87	65	14.784	22.90	16.62
Gorilla	100000	7	0	3	5	3	0.472	1.638	1.317	12	13	12	3.028	3.59	3.272
Elephant	114944	54900	3296	25	15	13	8.865	6.22	4.881	51	45	43	14.82	12.88	11.509
Gorilla	839092	20	0	3	3	3	5.339	13.839	13.861	16	15	15	35.328	35.42	35.43

Problem	#triangles	#invalids	#anchors	iterations to fix			timing to fix			iterations total			timing total		
				trial 16	trial 17	trial 18	trial 16	trial 17	trial 18	trial 16	trial 17	trial 18	trial 16	trial 17	trial 18
Gecko	1238	5	45	7	7	7	0.103	0.095	0.092	21	21	21	0.221	0.22	0.196
Elephant	1796	784	788	13	9	10	0.303	0.147	0.148	26	23	22	0.415	0.29	0.262
Octopus	5986	17	8	9	12	8	0.257	0.37	0.228	24	28	23	0.73	0.89	0.705
Grid	20000	4149	0	7	7	7	0.606	0.642	0.633	21	21	21	2.258	2.29	2.288
Octopus	23944	17	8	11	16	38	1.103	1.439	3.362	25	31	43	2.847	3.27	4.066
Elephant	28736	14038	1648	38	41	105	4.898	4.696	11.963	44	46	112	5.809	5.44	13.028
Octopus	95776	19	8	16	43	19	4.762	11.912	5.765	33	58	33	11.944	17.71	11.42
Gorilla	100000	7	0	2	3	3	0.341	0.49	0.509	7	8	8	3.062	3.19	3.268
Elephant	114944	54900	3296	14	14	14	7.258	6.962	6.932	28	28	28	13.232	12.90	12.961
Gorilla	839092	20	0	3	3	3	6.566	6.36	6.419	10	10	10	38.684	38.46	38.388

Problem	#triangles	#invalids	#anchors	iterations to fix			timing to fix			iterations total			timing total		
				trial 7	trial 8	trial 9	trial 7	trial 8	trial 9	trial 7	trial 8	trial 9	trial 7	trial 8	trial 9
Gecko	1238	5	45	8	5	5	0.131	0.076	0.086	13	13	12	0.206	0.18	0.191
Elephant	1796	784	788	9	9	9	0.175	0.152	0.164	16	14	19	0.292	0.24	0.318
Octopus	5986	17	8	12	13	13	0.43	0.435	0.538	19	21	17	0.862	0.93	0.804
Grid	20000	4149	0	8	7	7	0.987	0.678	0.763	13	15	14	2.075	2.27	2.176
Octopus	23944	17	8	11	10	19	1.184	1.028	2.792	20	18	24	3.078	2.75	4.013
Elephant	28736	14038	1648	31	18	10	5.486	2.947	1.946	33	23	16	6.109	4.28	3.478
Octopus	95776	19	8	19	36	26	9.054	15.656	14.793	24	43	30	12.976	20.89	18.14
Gorilla	100000	7	0	3	4	3	0.478	1.336	1.363	6	6	6	2.788	2.90	3.763
Elephant	114944	54900	3296	11	11	9	5.596	5.584	5.264	19	18	15	12.321	11.45	10.481
Gorilla	839092	20	0	3	3	3	5.433	12.58	14.531	7	6	6	34.776	35.36	36.443

Problem	#triangles	#invalids	#anchors	iterations to fix			timing to fix			iterations total			timing total		
				trial 19	trial 20	trial 21	trial 19	trial 20	trial 21	trial 19	trial 20	trial 21	trial 19	trial 20	trial 21
Gecko	1238	5	45	7	7	7	0.105	0.092	0.101	21	21	21	0.233	0.201	0.222
Elephant	1796	784	788	10	9	8	0.152	0.143	0.122	22	20	20	0.255	0.254	0.228
Octopus	5986	17	8	217	29	25	6.217	0.843	0.752	217	41	36	6.254	1.241	1.126
Grid	20000	4149	0	7	7	7	0.676	0.654	0.746	21	21	22	2.394	2.325	2.55
Octopus	23944	17	8	16	37	66	1.429	3.61	7.845	30	48	80	3.222	5.026	9.615
Elephant	28736	14038	1648	55	198	66	6.04	33.647	5.954	56	201	80	6.314	34.186	10.77
Octopus	95776	19	8	18	134	...	5.819	53.208	...	39	139	..	14.976	55.311	...
Gorilla	100000	7	0	3	3	3	0.522	0.51	0.58	8	8	8	3.231	3.213	3.349
Elephant	114944	54900	3296	13	16	61	6.876	7.924	24.325	27	31	64	12.872	14.136	25.948
Gorilla	839092	20	0	3	4	3	6.402	14.82	6.61	10	10	10	38.601	36.671	38.683

Problem	#triangles	#invalids	#anchors	iterations to fix			timing to fix			iterations total			timing total		
				trial 10	trial 11	trial 12	trial 10	trial 11	trial 12	trial 10	trial 11	trial 12	trial 10	trial 11	trial 12
Gecko	1238	5	45	4	3	2	0.067	0.04	0.039	13	13	11	0.177	0.17	0.145
Elephant	1796	784	788	11	11	10	0.2	0.203	0.18	16	18	16	0.291	0.32	0.271
Octopus	5986	17	8	14	13	13	0.615	0.469	0.535	18	20	18	0.883	0.91	0.879
Grid	20000	4149	0	9	9	8	1.083	1.105	1.112	15	15	13	2.356	2.38	2.203
Octopus	23944	17	8	24	15	18	3.447	2.022	2.513	24	17	22	3.648	2.58	3.461
Elephant	28736	14038	1648	12	12	13	2.204	2.178	3.017	20	21	20	4.193	4.33	4.782
Octopus	95776	19	8	25	20	25	13.346	9.414	13.26	30	27	28	17.251	14.82	15.973
Gorilla	100000	7	0	3	3	6	0.47	0.463	2.284	6	6	9	2.832	2.89	5.081
Elephant	114944	54900	3296	12	12	11	6.111	7.403	7.713	20	20	17	12.726	14.06	12.964
Gorilla	839092	20	0	3	3	5	5.684	5.736	22.346	7	7	10	34.824	34.85	64.176

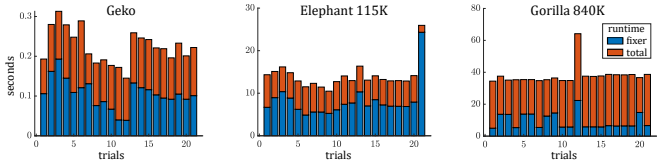


Figure 24: *Testing the impact of setting different hyper-parameters in ABCD(PN) algorithm. We ran 21 trials with different parameters, set according to Table 1. Each trial tests 10 different meshes. These results are laid out in the six tables; each table compares three successive trials. The runtimes (seconds) and iteration numbers are reported in the same ways as in Fig. 14. We denote by '...' the cases in which ABCD have failed to produce inversion-free maps. We select three problems and depict by the bar plot how much time took in each trial to process these problems. We used the relative energy termination criteria with* $\varepsilon = 10^{-3}$. *We ran all the tests with Pardiso solver on Intel i7-8565U CPU with 24GB RAM (4 phys. cores).*

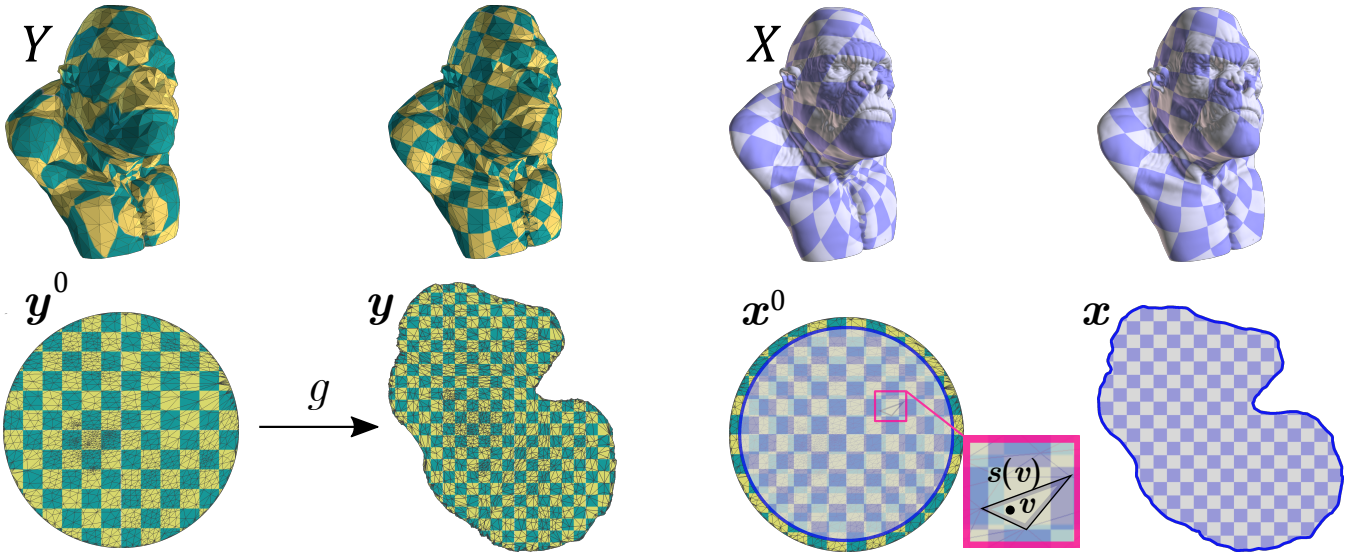


Figure 25: *Illustration of the deformation embedding method, descried in Section 3.* ***Left****: showing simplicial map g that deforms image of the Tutte embedding of a decimated mesh Y into the shape of its isometric parametrization.* ***Right****: map g is used to initialize mesh X of a higher resolution.*

Mesh	#Initial invalids	#Final invalids	Fixer runtime	Total runtime	Standard runtime
D1_00320	1	0	0.088	1.47	1.435
D1_00588	1	0	0.082	1.553	1.352
D1_00787	1	0	0.04	0.522	0.689
D1_00826	1	0	0.018	0.295	0.572
D1_00988	1	0	0.563	7.144	...
D1_01091	1	0	0.054	2.04	2.845
D1_01214	1	0	0.569	6.114	...
D1_01249	1	0	1.635	6.526	...
D1_01446	1	0	0.046	0.565	0.845
D1_01511	1	0	0.053	1.762	3.081
D1_01681	1	0	0.215	6.379	9.758
D1_01682	1	0	0.565	6.962	10.334
D1_02559	1	0	0.056	1.667	3.237
D1_02567	1	0	0.097	2.065	2.911
D1_02584	1	0	0.053	2.025	3.21
D1_02612	1	0	18.605	20.144	3.003
D1_02624	1	0	0.052	1.67	3.267
D1_02664	1	0	0.054	1.743	...
D1_02694	1	0	0.053	1.883	3.561
D1_02701	1	0	0.221	5.598	13.503
D1_03199	1	0	0.546	6.142	...
D1_03357	1	0	0.164	0.598	0.518
D1_03358	1	0	0.273	1.958	0.932
D1_03615	1	0	0.055	2.269	2.978
D1_03670	1	0	0.093	1.593	3.106
D1_03712	1	0	0.091	2.096	3.14
D1_03717	1	0	0.05	2.99	2.698
D1_03974	1	0	0.459	6.537	...
D1_04216	1	0	0.545	5.849	12.265
D1_04635	1	0	0.823	3.289	2.456
D1_04736	1	0	0.55	6.387	...
D1_04861	1	0	0.218	6.631	...
D1_04862	1	0	0.236	6.889	...
D1_04937	1	0	0.382	5.815	12.006
D1_05109	1	0	0.053	1.744	3.005
D1_05114	1	0	0.366	1.768	3.037
D1_01071	2	0	0.053	3.124	2.986
D1_01312	2	0	0.531	6.243	6.888
D1_01313	2	0	0.557	9.215	8.205
D1_01447	2	0	0.261	1.007	...
D1_01490	2	0	0.384	5.904	15.437
D1_01516	2	0	0.054	1.866	3.242
D1_01924	2	0	0.369	2.382	3.14
D1_02571	2	0	1.557	6.72	...
D1_02574	2	0	0.099	1.687	3.259
D1_02676	2	0	0.546	7.062	...
D1_02711	2	0	0.537	7.112	...
D1_02761	2	0	0.542	6.04	13.497
D1_03180	2	0	1.705	7.214	9.37
D1_03439	2	0	0.537	12.801	12.21
D1_03973	2	0	0.526	6.236	...
D1_04205	2	0	0.099	2.414	2.828
D1_04247	2	0	0.143	2.079	3.057
D1_04427	2	0	0.143	1.586	2.791
D1_04757	2	0	0.543	7.84	9.873
D1_04914	2	0	0.054	2.345	2.151
D1_05091	2	0	0.102	1.796	2.908
D1_05093	2	0	0.383	6.15	12.336
D1_05096	2	0	0.787	2.872	3.133
D1_05098	2	0	0.219	6.508	...
D1_05129	2	0	0.056	1.734	2.892
D1_05149	2	0	0.052	1.665	3.047
D1_00310	3	0	59.524	60.868	1.853
D1_00932	3	0	0.065	0.9	0.932
D1_01028	3	0	0.57	6.691	...
D1_01101	3	0	0.056	2.768	...
D1_01121	3	0	1.632	6.83	12.591
D1_01428	3	0	0.062	0.501	...
D1_01488	3	0	0.099	2.077	3.128
D1_01493	3	0	0.145	1.846	3.242
D1_01558	3	0	0.146	2.317	2.935
D1_01561	3	0	0.37	2.313	3.25
D1_01918	3	0	1.924	4.319	...
D1_01936	3	0	0.094	2.691	3.121
D1_02576	3	0	1.524	7.79	...
D1_02599	3	0	0.098	1.665	3.183
D1_02639	3	0	0.055	1.986	3.275
D1_02644	3	0	0.055	2.03	3.213
D1_03182	3	0	0.38	2.012	3.002
D1_03223	3	0	1.562	7.275	13.805
D1_04188	3	0	0.286	6.828	10.74
D1_04229	3	0	0.055	1.66	3.058
D1_04231	3	0	3.122	6.989	11.65
D1_04629	3	0	1.538	6.843	...
D1_05104	3	0	0.147	1.979	3.064
D1_05146	3	0	0.555	9.468	12.154
D1_05167	3	0	0.844	2.912	3.177
D1_00497	4	0	1.863	6.473	6.501
D1_00960	4	0	3.221	7.041	...
D1_00976	4	0	0.088	2.352	...
D1_01031	4	0	0.149	1.818	3.166
D1_01151	4	0	0.232	5.858	13.055
D1_01525	4	0	0.1	2.561	3.335
D1_01892	4	0	0.101	2.433	3.205
D1_01927	4	0	0.145	1.936	3.187
D1_02445	4	0	1.56	7.478	...
D1_02606	4	0	3.157	7.607	...
D1_03188	4	0	0.098	2.036	3.104
D1_03211	4	0	0.553	7.144	...
D1_03640	4	0	0.096	1.98	2.958
D1_04235	4	0	0.144	2.855	2.98
D1_04672	4	0	0.1	2.122	2.901
D1_04881	4	0	0.14	2.495	2.101
D1_04883	4	0	3.161	8.381	14.011
D1_05154	4	0	0.097	2.312	3.146
D1_00289	5	0	27.221	28.299	1.923
D1_00318	5	0	0.831	2.618	1.805
D1_00511	5	0	2.108	7.124	6.751
D1_00943	5	0	0.13	1.49	...
D1_01018	5	0	1.674	7.664	...
D1_01530	5	0	0.156	1.992	3.238
D1_01900	5	0	0.587	6.745	...
D1_01948	5	0	0.109	2.945	3.137
D1_02440	5	0	0.4	2.266	3.169
D1_02614	5	0	0.156	2.381	3.29
D1_03440	5	0	41.283	43.301	14.78
D1_04647	5	0	0.15	1.905	3.09
D1_04697	5	0	0.824	3.152	3.196
D1_04739	5	0	1.607	8.063	1.119
D1_00423	6	0	63.743	65.11	...
D1_00963	6	0	3.458	11.805	...
D1_01104	6	0	0.105	2.463	3.151
D1_01106	6	0	1.628	6.909	...
D1_01501	6	0	0.152	2.125	3.203
D1_01527	6	0	0.578	6.982	...
D1_01535	6	0	0.059	2.471	3.128
D1_01540	6	0	0.105	2.617	3.204
D1_02437	6	0	0.153	1.972	3.096
D1_04237	6	0	0.578	9.281	12.935
D1_04952	6	0	0.58	6.468	10.825
D1_04962	6	0	1.627	8.133	11.363
D1_00548	7	0	3.429	8.061	6.332
D1_01485	7	0	0.108	2.382	3.238
D1_01939	7	0	0.154	2.55	3.156
D1_02425	7	0	0.921	3.219	3.22
D1_04211	7	0	0.393	2.531	3.097
D1_04746	7	0	0.107	2.431	2.765
D1_04760	7	0	1.613	8.019	11.689
D1_01506	8	0	0.109	0.328	3.139
D1_01891	8	0	1.642	7.901	...
D1_01894	8	0	0.587	7.648	...
D1_01963	8	0	0.824	2.432	3.311
D1_02442	8	0	1.632	9.099	...
D1_03587	8	0	0.404	2.762	2.497
D1_04749	8	0	0.389	2.912	2.725
D1_04919	8	0	0.647	7.424	...
D1_04965	8	0	1.544	9.269	10.496
D1_00507	9	0	1.949	6.683	7.848
D1_00968	9	0	3.453	8.034	...
D1_01013	9	0	4.46	10.9	...
D1_01532	9	0	3.77	8.843	...
D1_02387	9	0	0.831	2.833	3.482
D1_02413	9	0	0.108	2.78	3.149
D1_02431	9	0	9.371	10.745	3.382
D1_04662	9	0	0.111	2.726	2.922
D1_00540	10	0	2.175	6.431	6.765
D1_01566	10	0	0.16	2.386	3.172
D1_01904	10	0	0.11	2.489	3.265
D1_01930	10	0	0.867	3.115	3.134
D1_01945	10	0	0.865	3.45	3.237
D1_04927	11	0	3.387	9.587	11.328
D1_01885	12	0	0.596	7.965	...
D1_02405	12	0	0.841	2.896	3.565
D1_02418	12	0	4.189	8.942	...
D1_02434	12	0	0.817	3.192	3.284
D1_02443	12	0	0.395	2.633	3.261
D1_01906	13	0	1.704	9.053	1.089
D1_01950	13	0	22.161	23.431	12.667
D1_04222	13	0	3.463	8.119	...
D1_00940	14	0	3.621	8.251	...
D1_02433	14	0	27.261	30.852	...
D1_00474	15	0	7.127	9.981	...
D1_01942	15	0	0.876	3.499	3.253
D1_00950	16	0	4.171	8.483	...
D1_00725	18	0	0.395	1.667	0.917
D1_02421	19	0	3.381	9.514	...
D1_02407	21	0	3.369	9.044	...
D1_00307	**22**	**8**	**...**	**...**	**...**
D1_00452	22	0	9.51	9.727	0.692
D1_00478	22	0	0.991	6.869	8.26
D1_02386	23	0	3.653	11.541	...
D1_00973	24	0	24.135	24.487	...
D1_02404	24	0	3.851	9.328	...
D1_02392	25	0	3.419	12.083	...
D1_03178	25	0	246.433	259.389	10.469
D1_03177	26	0	220.732	228.33	9.732
D1_02395	28	0	18.26	20.743	...
D1_02398	45	0	5.76	16.529	...
D1_01008	52	0	23.894	27.549	...
D1_02401	102	0	42.487	42.851	...

Figure 26: *All the results of the conformal initialization experiment (some were depicted by Fig. 12). Each table shows from left to right: a mesh file name from PP dataset [LYNF18]; an initial number of invalid (inverted and collapsed) triangles, obtained in BFF mapping; a number of invalid triangles, obtained after running ABCD(PN); time (seconds) until ABCD(PN) fixes all invalid triangles; a time until ABCD(PN) converges; a time that took to compute a standard isometric parametrization with PN solver, initialized by Tutte mapping into a disc. We denote by '...' the cases in which ABCD(PN) or Tutte embedding fail to produce inversion-free maps. We used the relative energy termination criteria with* $\varepsilon = 10^{-3}$. *These meshes were tested with Pardiso solver on Intel i7-6500U CPU with 16GB RAM (2 phys. cores).*