GPU-Accelerated Locally Injective Shape Deformation

RENJIE CHEN, Max Planck Institute for Informatics, Germany
OFIR WEBER, Bar-Ilan University, Israel

Fig. 1. Locally injective isometric deformation of Rex, a doubly-connected planar domain (left). Note the hole between the legs. Our harmonic subspace has 392 DOF. Our solver converges after 6 iterations to the result depicted, which is computed in $0.072s$ when implemented on the GPU, and $0.375s$ on the CPU. We compare against an unmodified Newton solver which halts prematurely after 9 iterations at a high energy configuration due to a non-positive definite Hessian. The Newton-Eigen solver performs a full eigen-factorization Hessian modification at each iteration. It converges after 32 iterations and takes $2.467s$ which is $\times 34$ times slower than our GPU solver. The L-BFGS solver performes 1, 818 iterations and is not competitive in terms of runtime.

We present a highly efficient planar meshless shape deformation algorithm. Our method is based on an unconstrained minimization of isometric energies, and is guaranteed to produce C^∞ locally injective maps by operating within a reduced dimensional subspace of harmonic maps. We extend the harmonic subspace of [Chen and Weber 2015] to support multiply-connected domains, and further provide a generalization of the bounded distortion theorem that appeared in that paper. Our harmonic map, as well as the gradient and the Hessian of our isometric energies possess closed-form expressions. A key result is a simple-and-fast *analytic* modification of the Hessian of the energy such that it is positive definite, which is crucial for the successful operation of a Newton solver. The method is straightforward to implement and is specifically designed to harness the processing power of modern graphics hardware. Our modified Newton iterations are shown to be extremely effective, leading to fast convergence after a handful of iterations, while each iteration is fast due to a combination of a number of factors, such as the smoothness and the low dimensionality of the subspace, the closed-form expressions for the differentials, and the avoidance of expensive strategies to ensure positive definiteness. The entire pipeline is carried out on the GPU, leading to deformations that are significantly faster to compute than the state-of-the-art.

CCS Concepts: • **Computing methodologies** → *Computer graphics*; *Animation*; *Image manipulation*; *Shape analysis*;

Additional Key Words and Phrases: injective maps, harmonic maps, shape deformation, Newton method, GPU

This research was partially funded by the Israel Science Foundation (grants No. 1869/15 and 2102/15) and by the Max Planck Center for Visual Computing and Communication.

ACM Reference Format:
Renjie Chen and Ofir Weber. 2017. GPU-Accelerated Locally Injective Shape Deformation. *ACM Trans. Graph.* 36, 6, Article 214 (November 2017), 20 pages. https://doi.org/10.1145/3130800.3130843

1 INTRODUCTION

Shape deformation is a classical problem in computer graphics and animation that comes in a variety of forms and settings. Essential to all variants is the requirement to compute a map between different spaces with some desirable properties. Common choices include maps between curved manifolds embedded in $\mathbb{R}^3$, volumetric maps between solid shapes in $\mathbb{R}^3$, and mixed dimensional maps where, for example, a curved 2-manifold embedded in $\mathbb{R}^3$ is mapped to $\mathbb{R}^2$ (a parameterization). In this work we deal with the problem of *planar* maps where both the source and target domains are subsets of $\mathbb{R}^2$. Due to its relative simplicity, this setting is the most fundamental one that serves as a bridge for all other developments and generalizations, and as such it attracts a large amount of research efforts since the early days of graphics. Depending on the precise user requirements, even this simplified setting is challenging to deal with.

Some requirements such as smoothness are easily achievable while other, such as local injectivity or strict bound on the induced metric distortion are hard (sometimes impossible) to guarantee. In this work we consider an interactive deformation system that is driven by user specified positional constraints. That is, the user selects a small set of points in the source domain which is visualized as a textured image, and interactively drags them around to new positions. The goal is to be able to compute a map that adheres to these positional constraints yet at the same time is visually plausible. Plausibility is usually quantified by smoothness and the amount of

metric distortion, which can be either minimized on the average, or bounded above by some user-defined amount. Moreover, the map is expected to be locally injective, preventing degeneracies of the image and local overlaps while allowing global ones. These requirements pose a great computational challenge since they usually lead to a nonconvex optimization problem with many degrees of freedom for which finding an optimal solution (or even just a feasible one) is often intractable. Recent advances in the computation of locally injective and/or bounded distortion maps allow artists to produce high quality results with greater than ever success rates but the performance of current methods is still significantly worse compared to simple linear methods [Weber et al. 2009].

We cast the deformation problem as a nonlinear nonconvex unconstrained minimization problem that is performed in a reduced deformable subspace of harmonic maps, and allow the user to pick among a variety of smooth energies that measure isometric distortion. Harmonic functions have many appealing properties and are widely used in computer graphics applications. In particular, holomorphic and harmonic maps are utilized extensively in the context of planar shape deformation and interpolation [Chen and Weber 2015; Chien et al. 2016a; Hefetz et al. 2017; Levi and Weber 2016; Weber 2010, 2017; Weber et al. 2009, 2011; Weber and Gotsman 2010]. However, this entire line of work was restricted to simply-connected domains (topological disks). Our first contribution in this work is broadening the scope of previous research on harmonic maps from simply-connected to multiply-connected domains (disks with finite number of holes) such as those appearing in Figures 1, 3, 5. In particular, in Section 4 we extend the harmonic subspace of the Bounded Distortion Harmonic Mappings (BDHM) method [Chen and Weber 2015] to multiply-connected domains, as well as generalize the bounded distortion theorem of BDHM (Section 4.1). Furthermore, in Section 8 we provide a much tighter analysis of the Lipschitz constants provided in BDHM, leading to more accurate validation of the map and faster convergence.

We construct a custom-made Newton solver with unique features to efficiently address the deformation problem. The quality of the produced maps is superior and comparable to the other methods that operate within the harmonic subspace. However, our method is orders of magnitude faster than existing techniques. Our optimization is based on a second-order Newton approach which ensures that the convergence rate is high. In general, a single Newton iteration can be slow due to various reasons such as the need to approximate the Hessian numerically using finite differences (when analytical expressions are unavailable), as well as the need to modify the Hessian such that it becomes positive definite (which is necessary to ensure convergence to a local minimum).

Our novel solver overcomes these issues due to several unique properties, such as having closed-form expressions for the map, and its differentials, and an analytic modification scheme for the Hessian to ensure its positive definiteness. Combining these benefits leads to a Newton iteration that is considerably faster than a standard Newton iteration. Together with the small number of iterations that are typically required for convergence, the total running time of our solver is significantly shorter compared to state-of-the-art solvers. Moreover, every iteration of our solver is embarrassingly parallelized and perfectly suites implementation on massively parallel graphics processing units due to the structure and regularity of the problem. This allows for unprecedented performance.

2 PREVIOUS WORK

Linear blend skinning [Kavan et al. 2007] and methods that are based on barycentric coordinates [Hormann and Floater 2006; Lipman et al. 2007; Weber et al. 2009] use precomputed basis functions [Jacobson et al. 2011] and formulate the deformation as a linear combination of these basis functions with some coefficients that are automatically deduced from the user input. For instance a control polygon/mesh or rotations that are extracted from a skeletal structure. Despite their inability to produce maps with good distortion control, these methods are still highly popular due to their speed and in particular their suitability for GPU implementation.

More geometric motivated linear approaches merely require solving a single linear system with variable right hand sides where the left hand side matrix is fixed and can be factored in preprocessing [Igarashi et al. 2005; Lipman et al. 2005; Weber et al. 2009, 2007; Zayer et al. 2005]. Under extreme deformations, artifacts arise which lead to the introduction of more sophisticated nonlinear approaches.

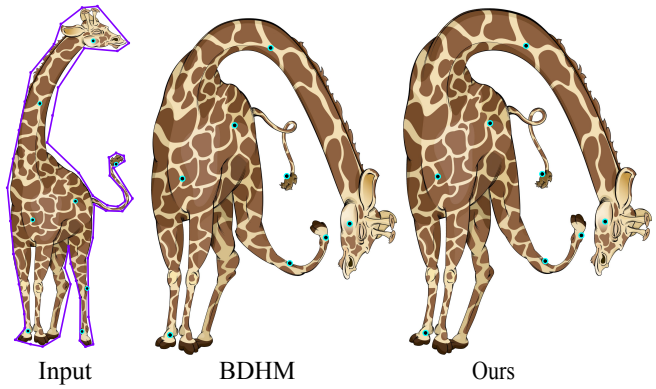

Fig. 2. Comparison with BDHM using the same harmonic subspace. We set user distortion bounds for BDHM: $(k, \sigma_1, \sigma_2) = (0.8, 5, 0.3)$. The resulted map is bounded distortion with bounds $(0.50, 4.3, 0.31)$. Without explicit distortion bounds, our method minimizes the symmetric Dirichlet energy producing a map with bounds $(0.32, 2.25, 0.43)$. Our GPU solver converges after 8 iterations with total runtime of $0.0297s$. In comparison, BDHM requires 28 iterations of second order cone programming [ApS 2017], with total runtime $3.71s$ which is $\times 125$ slower.

Recently, a successful line of approaches have been designed to specifically address quality concerns with a priority on producing injective maps [Bright et al. 2017; Fu et al. 2015; Schüller et al. 2013; Smith and Schaefer 2015] with bounded amount of distortion [Aigerman and Lipman 2013; Aigerman et al. 2014; Chen and Weber 2015; Chien et al. 2016b; Kovalsky et al. 2015; Lipman 2012; Poranne and Lipman 2014]. These are mostly based on interior point methods and are relatively efficient, though they cannot perform deformation with many degrees of freedom in real time rates and none of these is directly suitable for a GPU implementation. Our method outperforms these by a large margin. For example, in Figure 2, we show

a speedup of ×125 when comparing our GPU Newton solver with the Mosek interior point solver that was used in [Chen and Weber 2015]. To the best of our knowledge, the only bounded distortion deformation method that utilizes the GPU is [Hefetz et al. 2017] which projects maps to the bounded distortion space by using alternating tangential projections (ATP). It is extremely fast and provably converges to a bounded isometric-and-conformal distortion map with any user-specified bounds, albeit ATP does not handle positional constraints, nor minimize user-defined energies.

The Newton's algorithm (see [Nocedal and Wright 2006] for a comprehensive review) is an established method for the minimization of smooth nonlinear objective functions, and is widely known for its good convergence properties due to its second order nature. Each Newton iteration is very effective in reducing the energy, and typically a small amount of iterations is needed to reach low energy. For the Newton iteration to produce a descent direction, the Hessian of the energy must be positive definite and since the energies of interest are nonconvex in general, several strategies for modifying the Hessian to become positive definite have been developed (see [Nocedal and Wright 2006, Chapter 3.4] for a concise overview).

The most straightforward approach is to compute the eigen–factorization (also known as spectral decomposition) and to replace all the negative eigenvalues with sufficiently small positive values. The obtained modified Hessian is the closest (in Frobenius norm) positive definite matrix to the unmodified one. We compare our results with this method and show our superiority. Another common approach [Fu and Liu 2016] is to shift the spectrum by trying iteratively to add a positive constant to the diagonal of the Hessian until the Cholesky factorization succeeds (which is an indication for positive definiteness). In practice though, such modifications are expensive to compute.

A more affordable approach is to decompose the Hessian as a sum of individual per-element matrices and to perform the modification on each element separately. The approximated Hessian is then obtained by a summation of the modified per-element matrices. In [Teran et al. 2005], each per-element tensor is encoded as a 9×9 block diagonal matrix and in [Fu and Liu 2016], 4×4 or 9×9 matrices are constructed and modified numerically. This approach is in general more efficient than a full Hessian modification but is still time consuming due to the large number of required matrix factorizations. Our approach is conceptually similar, however, due to the infinite support of the harmonic basis functions, our per-element Hessians are dense $4n\times4n$ matrices (n is typically of order of hundreds). Performing the factorization numerically is practically impossible. A key contribution (Section 7) in our paper is that each such $4n\times4n$ factorization can be computed analytically. Furthermore, we show that the number of elements (samples) needed in practice to achieve good convergence rates can be relatively small, ensuring that the assembly of the individual Hessian contributions does not dominate the solver performance. The computation of the Hessian modification (as well as all the other parts of our algorithm) is done in parallel and is perfectly suitable for implementation on graphics hardware (Appendix G) leading to an algorithm that is orders of magnitude faster than state-of-the-art methods.

If the objective function is a nonlinear sum of squares, the Gauss-Newton (GN) or the Levenberg-Marquardt (LM) algorithms can be used. These use first order derivatives only, sidestepping the requirement to compute-and-modify the Hessian matrix. [Vaxman et al. 2015] uses LM to compute conformal deformations in 2-and-3 dimensions. Another class of first order methods which can be applied to more general type of objectives are Quasi-Newton algorithms. BFGS for example, incrementally approximate the Hessian based on the gradients and the updates from previous iterations. L-BFGS further avoids inverting the Hessian by directly approximating its inverse from k recent iterations [Liu et al. 2017]. A single iteration of L-BFGS is then more efficient than that of a second order method, but the number of iterations is typically much larger. [Smith and Schaefer 2015] used L-BFGS to optimize a particular choice of isometric energy. We compare our solver for this particular energy against L-BFGS, showing a dramatic reduction in the number of iterations and increase in speed.

The Accelerated Quadratic Proxy (AQP) method is another notable first order method [Kovalsky et al. 2016] that can optimize isometric-like energies. A main observation made by Kovalsky et al. is that the Hessian of the particular deformation energy can be approximated by the mesh Laplacian matrix which is the Hessian of the quadratic Dirichlet energy. In contrast to a Newton iteration which requires a full linear solve at each iteration, AQP merely requires a backward and forward substitutions against the factor of the *fixed* Laplacian matrix, leading to extremely fast iterations. In order to reduce the number of iterations, which is typically much larger than that of Newton's, an acceleration technique was used. Albeit, this advocates the use of hard positional constraints. Since finding a locally injective map satisfying hard positional constraints is a notoriously hard problem, AQP only produces orientation preserving maps which is a weaker requirement. To obtain an initialization, AQP first computes an arbitrary map satisfying the positional constraints, and then projects it using the method of [Kovalsky et al. 2015] to the space of orientation preserving maps. While [Kovalsky et al. 2015] is a relatively efficient and robust method, it does not guarantee convergence and utilizing it as initialization extends the overall computation time. The method of [Martin et al. 2013] takes a signal processing viewpoint for mesh optimization where powers of the Laplacian matrix are utilized.

Recently, Rabinovich et al. [2017] proposed Scalable Locally Injective Mappings (SLIM) for optimizing rotation-invariant mesh-based energies via a modification of the local-global ARAP algorithm [Liu et al. 2008]. Unlike AQP which uses a fixed Hessian approximation, SLIM reweights a proxy energy iteratively in a way that aligns its gradient with that of the true energy. While AQP and SLIM are more general and are not restricted to planar maps, we demonstrate that our solver is significantly faster, with much shorter per-iteration time and dramatically smaller number of iterations. In contrast to all the above mentioned mesh-based methods, which are based on sparse linear algebra, our method operates within a reduced dimensional subspace which is formulated in dense linear algebra terms. While the sparse direct linear solvers utilized in these mesh-based algorithms exploit multi-core CPU parallelism, existing implementations on graphics hardware (e.g. SuiteSparse, cuSolverSP) are still immature.

3 BACKGROUND

For completeness we include a short introduction to some basic concepts in optimization, complex analysis, and planar harmonic maps. For further reading we refer to the comprehensive book by Nocedal and Wright [2006] for optimization related topics, the book by Ahlfors for complex analysis related topics [1979], and the concise book by Duren regarding harmonic planar maps [2004].

3.1 Newton's Method

Given a scalar function $E(\boldsymbol{x})$ of a vector variable $\boldsymbol{x} \in \mathbb{R}^n$, Newton's method searches for a stationary point of $E(\boldsymbol{x})$ by iteratively approximating it using its second order Taylor expansion:

$$E(\boldsymbol{x}) = E(\boldsymbol{x_k} + \Delta\boldsymbol{x}) \approx E(\boldsymbol{x}_k) + \Delta\boldsymbol{x}^\mathsf{T}\nabla E(\boldsymbol{x}_k) + \frac{1}{2}\Delta\boldsymbol{x}^\mathsf{T}\nabla^2 E(\boldsymbol{x}_k)\Delta\boldsymbol{x},$$

where $\Delta\boldsymbol{x} = \boldsymbol{x} - \boldsymbol{x}_k \in \mathbb{R}^n$ is the search direction, and $\nabla E \in \mathbb{R}^n$ and $\nabla^2 E \in \mathbb{R}^{n\times n}$ denotes the gradient and the Hessian of E respectively. This second order polynomial in $\Delta\boldsymbol{x}$ has vanishing derivative when the following linear system is satisfied:

$$\nabla^2 E(\boldsymbol{x}_k)\,\Delta\boldsymbol{x} = -\nabla E(\boldsymbol{x}_k). \tag{1}$$

The direction $\Delta\boldsymbol{x}$ decreases the energy only if the second order polynomial is convex, or equivalently, the Hessian matrix $\nabla^2 E$ is positive definite. Hence, when needed, the Hessian is modified. The solution to the linear system produces the update for the current iteration $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta\boldsymbol{x}$. To ensure convergence (to a local minimum) from an arbitrary starting point, a line search along the search direction $\Delta\boldsymbol{x}$ should be used to find a step size t, that satisfies the Wolfe conditions [Nocedal and Wright 2006, Chapter 3.1].

3.2 Complex-valued Functions as Planar Maps

A real-valued function $u(x, y) : \Omega \to \mathbb{R}$, is harmonic if it satisfies the Laplace equation $u_{xx} + u_{yy} = 0$. A planar harmonic map is a map $f : \Omega \to \mathbb{R}^2$ where $f = [u(x, y), v(x, y)]$, and the component functions u and v are harmonic. When we identify $\mathbb{R}^2$ with the complex plane $\mathbb{C}$ using complex notation $z = x + \mathrm{i}y$, we can express f as a complex-valued function $f(z) = u(z) + \mathrm{i}v(z)$. Thus a complex-valued harmonic function $f : \Omega \subset \mathbb{C} \to \mathbb{C}$ can be interpreted as a harmonic planar *map*. A complex-valued function $f = u + \mathrm{i}v$ is said to be *holomorphic* if it satisfies the Cauchy-Riemann equations. $f(z)$ is called anti-holomorphic if $\overline{f(z)}$ is holomorphic.

The Wirtinger derivatives of a complex-valued function are defined as follows: $f_z := \frac{1}{2}(f_x - \mathrm{i}f_y)$ and $f_{\bar{z}} := \frac{1}{2}(f_x + \mathrm{i}f_y)$. Using the Wirtinger derivatives, the Cauchy-Riemann equations are expressed concisely as $f_{\bar{z}} \equiv 0$. The Laplacian $f_{xx} + f_{yy}$ is concisely expressed as $4f_{\bar{z}z}$, hence the condition for harmonicity (of $\mathrm{Re}f$ and $\mathrm{Im}f$) is simply $f_{\bar{z}z} \equiv 0$. It is then easy to see that the real and imaginary components of a holomorphic function are harmonic. The same is true for anti-holomorphic functions. The converse is not true since $f_{\bar{z}z} \equiv 0$ does not imply $f_{\bar{z}} \equiv 0$ nor $f_z \equiv 0$.

The Wirtinger derivatives are useful for defining many of the local geometric distortion quantities of a C^1 planar map. We have that the local change in area is: $\det(J_f) = |f_z|^2 - |f_{\bar{z}}|^2$, where J_f is the 2×2 Jacobian matrix of the map. A C^1 map f is locally injective and orientation preserving at a neighborhood of a point z if $\det(J_f(z)) > 0$, or alternatively if $|f_z(z)| > |f_{\bar{z}}(z)|$.

Isometric and conformal distortion measures are typically formulated in terms of the singular values of J_f, $0 \le \sigma_2 \le \sigma_1$:

$$\sigma_1(z) = |f_z| + |f_{\bar{z}}|, \quad \sigma_2(z) = \Big|\,|f_z| - |f_{\bar{z}}|\,\Big|. \tag{2}$$

Note that for orientation preserving locally injective maps, the expression on the right simplifies to $\sigma_2(z) = |f_z| - |f_{\bar{z}}| > 0$. The little dilatation $k(z) = |f_{\bar{z}}(z)|\,/|f_z(z)|$ is a measure of conformal distortion. We discuss isometric distortion measures in Section 6.

4 HARMONIC MAPS ON MULTIPLY-CONNECTED DOMAINS

As mentioned in the Introduction, our work generalizes existing techniques that are designed for simply-connected domains to multiply-connected domains. We base our construction on the following theorem which we prove in Appendix A.

Theorem 4.1 (Harmonic Decomposition). *Let Ω be a multiply-connected planar domain with N holes $K_1, \ldots, K_N$, and choose N arbitrary points ρ_i in K_i. Then, any harmonic map $f : \Omega \to \mathbb{C}$ can be represented as:*

$$f(z) = \tilde{\Phi}(z) + \overline{\tilde{\Psi}(z)} + \sum_{i=1}^{N} \omega_i \ln|z - \rho_i|, \tag{3}$$

where $\tilde{\Phi}, \tilde{\Psi} : \Omega \to \mathbb{C}$ are holomorphic functions, and $\omega_1, \ldots, \omega_N$ are some complex coefficients.

The harmonic decomposition (3) is unique up to an additive complex constant that can be chosen by setting e.g. $\tilde{\Psi}(z_0) = 0$ for an arbitrary point $z_0 \in \Omega$. Theorem 4.1 extends a similar result for simply-connected domains for which the summation term in (3) is missing (c.f. [Duren 2004] Section 1.2). While $\tilde{\Phi}(z) + \overline{\tilde{\Psi}(z)}$ is still harmonic on multiply-connected domains, the additional summation term is essential to represent certain harmonic maps and without it, the representation is incomplete.

4.1 Locally Injective Harmonic Maps

The main theoretical result in [Chen and Weber 2015] is a boundary-value characterization of the injectivity and the conformal-and-isometric distortion of planar harmonic maps on simply-connected domains. We generalize this result to the multiply-connected case (see Appendix B for the proof):

Theorem 4.2 (Bounded Distortion). *A planar harmonic map $f : \Omega \to \mathbb{C}$ on a multiply-connected domain with exterior boundary curve γ_0 oriented counterclockwise and interior boundary curves $\gamma_1 \cdots \gamma_N$ oriented clockwise, is locally injective with an upper bound $\boldsymbol{k} \in [0, 1)$ on the conformal distortion, a lower bound $\boldsymbol{\sigma_2} > 0$ on the small singular value of the Jacobian, and an upper bound $\boldsymbol{\sigma_1} < \infty$ on the large singular value at every point z in Ω if and only if:*

$$\oint_{\gamma_0} \frac{f_z'(w)}{f_z(w)}dw + \sum_{i=1}^{N}\oint_{\gamma_i} \frac{f_z'(w)}{f_z(w)}dw = 0, \tag{4a}$$

$$0 \le k(w) \le \boldsymbol{k} \qquad \forall w \in \partial\Omega, \tag{4b}$$

$$\sigma_1(w) \le \boldsymbol{\sigma_1} \qquad \forall w \in \partial\Omega, \tag{4c}$$

$$\boldsymbol{\sigma_2} \le \sigma_2(w) \qquad \forall w \in \partial\Omega. \tag{4d}$$

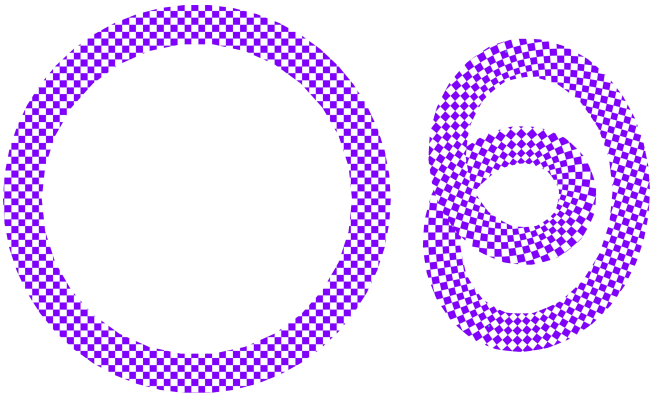

Fig. 3. A knot. Left: doubly-connected domain Ω. Right: an image of a map $f : \Omega \to \mathbb{C}$. While f is locally injective, f and the identity are not regular homotopic. Intuitively, one cannot "morph" the annulus on the left to the one on the right continuously without breaking local injectivity. The conditions of Theorem 4.2 on f hold. The integral (Equation (4a)) over the exterior boundary is $2\pi\mathrm{i}$ while the integral over the interior one is $-2\pi\mathrm{i}$.

Intuitively, Theorem 4.2 states that in order to induce global distortion bounds, it is sufficient to bound the distortion of a harmonic map on the *boundary* curves γ_i as long as (4a) holds. The boundary integral condition (4a) is equivalent to the condition that f_z is non-vanishing throughout the domain. Namely that $f_z(z) \neq 0$ (Appendix B). In contrast to the simply-connected domain, the integral over the exterior boundary γ_0 may be nonzero. This gives rise to maps which are not regular homotopic to the identity map but can still be locally injective (see Figure 3).

As opposed to [Chen and Weber 2015], our deformation algorithm does not *explicitly* impose user-defined bounds $\boldsymbol{k}, \boldsymbol{\sigma_1}, \boldsymbol{\sigma_2}$ on the distortion. Instead, we favor an overall lower *average* distortion. By using energies that become infinite when the map degenerates at a boundary point, a finite bound on distortion is naturally formed on the boundary. Theorem 4.2 then assures that this bound is also global. At each iteration, our algorithm certifies the map as locally injective by verifying that (4a) holds, as well as the following:

$$|f_z(w)| > |f_{\bar{z}}(w)| \qquad \forall w \in \partial\Omega. \tag{5}$$

In Section 8 we explain how to enforce (4a) and (5) in practice.

5 DISCRETIZATION

We discretize Equation (3) to obtain a finite dimensional space of harmonic maps. The holomorphic functions $\tilde{\Phi}$ and $\tilde{\Psi}$ are discretized by using the Cauchy complex barycentric coordinates [Weber et al. 2009] which are derived by approximating the boundary of the domain using a polygonal shape. Let $\hat{\mathrm{P}} = \{z_1, z_2, ..., z_m\}, \quad z_j \in \mathbb{C}$ be the vertices of a multiply-connected planar polygon (the cage). The vertices of the cage are split into sets such that the first set represents the exterior polygon oriented counterclockwise while the following sets represent the interior polygons oriented clockwise (see Figure 4).

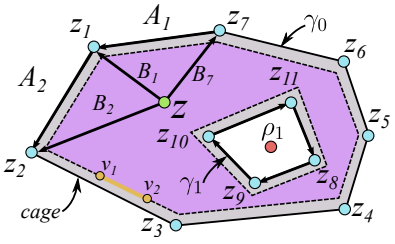

Fig. 4. Notations. The purple area is a doubly-connected domain Ω bounded by a polygon P with 11 vertices. The solid black polygon is an outward offset cage $\hat{\mathrm{P}} = \gamma_0 \cup \gamma_1$ with vertices z_j (cyan) composed of exterior polygon γ_0 oriented counterclockwise, and interior clockwise polygon γ_1. The point z (green) is in Ω or on $\partial\Omega$. $A_j = z_j - z_{j-1}$, $B_j(z) = z_j - z$, and $[v_i, v_{i+1}]$ is a small segment (orange) on P. The point ρ_1 (red) is in γ_1.

The holomorphic functions $\tilde{\Phi}$ and $\tilde{\Psi}$ are represented as:

$$\tilde{\Phi}(z) = \sum_{j=1}^{m} \tilde{C}_j(z)\varphi_j, \qquad \tilde{\Psi}(z) = \sum_{j=1}^{m} \tilde{C}_j(z)\psi_j, \tag{6}$$

where $\tilde{C}_j(z)$ is the j^{th} holomorphic Cauchy barycentric coordinate associated with vertex z_j and φ_j, ψ_j are complex coefficients. $\tilde{C}_j(z)$ possess a rather simple closed-form expression (see Appendix D). Note that while Weber et al. [2009] assumed that $\hat{\mathrm{P}}$ is simply-connected, their derivation can be extended to the case of multiple boundary components due to the fact that Cauchy's integral formula still holds. This can be done by integrating the Cauchy kernel over *all* the boundary components. Practically, representing holomorphic functions on such a domain simply boils down to using additional basis functions that correspond to vertices that lie on the interior boundaries. We note, that the complex barycentric map induced by the Cauchy coordinates on multiply-connected domains has a non-empty null space of (complex) dimension $2N$ which corresponds to a similarity transformation of the vertices of the interior boundary polygons. To remove this ambiguity, we fix the first two complex coefficients of each hole to zero.

We proceed by substituting the holomorphic functions $\tilde{\Phi}$ and $\tilde{\Psi}$ in the harmonic decomposition (3) with the expressions from (6). The additional term $\sum_{i=1}^{N} \omega_i \ln|z - \rho_i|$ in (3) is substituted with the expression:

$$\sum_{j=m+1}^{m+N} (\varphi_j + \overline{\psi_j}) \ln\left|z - \rho_{j-m}\right|, \tag{7}$$

where φ_j, ψ_j are $2N$ additional complex coefficients. Note that splitting ω_i into two variables creates some unnecessary redundancy, nevertheless it greatly simplifies the exposition as well as the implementation. Later on, we will remove this redundancy by enforcing a constraint $\varphi_j = \overline{\psi_j}$, $j = m+1, \ldots, m+N$ for each hole. Finally, rearranging terms and denoting $n = m + N$ leads to:

$$f(z) = \sum_{j=1}^{n} C_j(z)\varphi_j + \overline{\sum_{j=1}^{n} C_j(z)\psi_j}, \tag{8}$$

$$C_j(z) = \begin{cases} \tilde{C}_j(z) & j = 1, \ldots, m \\ \ln\left|z - \rho_{j-m}\right| & j = m+1, \ldots, n. \end{cases}$$

Equation (8) is our closed-form expression for a finite dimensional harmonic subspace with $2n$ complex variables $\{\varphi_j, \psi_j\}_{j=1}^n$. The dimension of the (complex) null space is $4N + N + 1$, where $4N$ is due to the 2 complex DOF per hole for each of the two Cauchy barycentric maps (as explained above), N is due to the redundancy introduced by Equation (7) and the term 1 is due to the constant DOF of the harmonic decomposition (3).

We proceed by constructing simple formulas for the Wirtinger derivatives of our harmonic map. Luckily, just like the map itself, it can be expressed in closed-form. By linearity of the Wirtinger operators, the fact that the derivative of the Cauchy coordinates with respect to $\overline{z}$ is 0, and assuming that $\varphi_j = \overline{\psi_j}, \forall j = m+1, \ldots, n$ we have:

$$f_z(z) = \sum_{j=1}^{n} D_j(z)\varphi_j, \qquad \overline{f_{\bar{z}}}(z) = \sum_{j=1}^{n} D_j(z)\psi_j, \tag{9}$$

$$D_j(z) = \begin{cases} \tilde{D}_j(z) & j = 1, \ldots, m \\ \frac{1}{z-\rho_{j-m}} & j = m+1, \ldots, n. \end{cases}$$

The derivatives of the Cauchy barycentric coordinates $\tilde{D}_j(z)$ (Appendix D) are holomorphic and so is $\frac{1}{z-\rho_{j-m}}$, hence it is clear that f_z and $\overline{f_{\bar{z}}}$ are both holomorphic (though not necessarily integrable).

6 ISOMETRIC ENERGIES

The main approach in interactive shape deformation is to design and minimize a distortion energy that aggregates a differential quantity that strives to keep the map as-isometric-as-possible. Such differential quantities are minimized when the map is locally isometric. However, since a perfect isometry cannot be obtained in general in the presence of positional constraints, the particular definition of proximity to isometry greatly affects the end result. It is a common practice to measure distortion at a point z as a function $\mathrm{E}(z) = \mathrm{E}(\sigma_1(z), \sigma_2(z))$ of the two singular values of the Jacobian matrix $J_f(z)$. Such a function is invariant to rigid motions of both the source and the target domains [Rabinovich et al. 2017]. $\mathrm{E}(z)$ should be minimized (at a single point z) if $\sigma_1(z) = \sigma_2(z) = 1$. Energies that minimize *conformal* distortion are also popular. Our harmonic subspace contains many pure (with zero distortion) conformal maps, hence, if desired, we can simply restrict the subspace by eliminating the ψ_j variables and use any of our isometric energies to regularize the result (Figure 5).

A popular choice in graphics is the as-rigid-as-possible (ARAP) energy [Igarashi et al. 2005; Liu et al. 2008; Sorkine and Alexa 2007] which defines local proximity to isometry via $\|J - R\|_F^2$, where $\|\cdot\|_F$ denotes the Frobenius norm, and R denotes the closest rotation to J. It can be expressed as $\mathrm{E}_{\mathrm{ARAP}} = (\sigma_1 - 1)^2 + (\sigma_2 - 1)^2$. The main limitation of ARAP is that it favors shrinkage over expansion, and in particular it stays finite even if the map is degenerated ($\sigma_2(z) = 0$). Hence, ARAP tends to attract the map to a configuration which is not locally injective. Our solver and derivation are applicable to $\mathrm{E}_{\mathrm{ARAP}}$, albeit ARAP is arguably a poor choice of energy if local injectivity is sought after.

Appropriate choices of isometric energy for designing locally injective maps become infinite when the map collapses locally, which serves as a natural barrier term. Since our optimization depends on high order derivatives, we focus on energies which are smooth in our variables φ_j, ψ_j. To this end, we express the distortion measure at z as a smooth function of $|f_z|^2$ and $|f_{\bar{z}}|^2$. Since $|f_z|^2$ and $|f_{\bar{z}}|^2$ are quadratic functions in φ_j, ψ_j, the composition is also smooth. The symmetric Dirichlet isometric energy $\mathrm{E}_{\mathrm{iso}} = \frac{1}{2}|J|_F^2 + \frac{1}{2}|J^{-1}|_F^2$ has been successfully used in [Kovalsky et al. 2016; Rabinovich et al. 2017; Schreiner et al. 2004; Smith and Schaefer 2015] for mesh parameterization. It can be expressed in terms of the singular values:

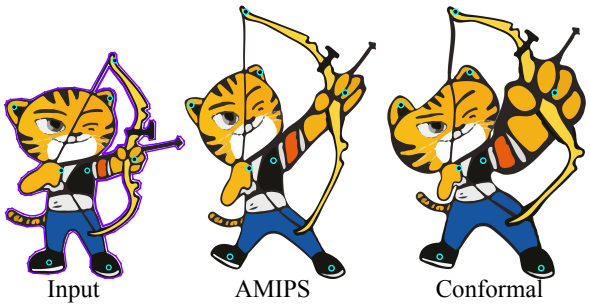

Fig. 5. Harmonic deformations of a triply-connected domain. The domain on the left has $N = 2$ holes in it. The result in the middle minimizes the Advanced MIPS energy, while the one on the right is a pure conformal map with least isometric distortion.

$$\mathrm{E}_{\mathrm{iso}}(\sigma_1, \sigma_2) = \frac{1}{2}\left(\sigma_1^2 + \sigma_1^{-2} + \sigma_2^2 + \sigma_2^{-2}\right) \tag{10a}$$

$$= \frac{\sigma_1^2 + \sigma_2^2}{2}\left(1 + \frac{1}{\sigma_1^2\sigma_2^2}\right). \tag{10b}$$

Recall that for an orientation preserving locally injective planar map f, we have $\sigma_1 = |f_z| + |f_{\bar{z}}|$ and $\sigma_2 = |f_z| - |f_{\bar{z}}|$. We get:

$$\frac{\sigma_1^2 + \sigma_2^2}{2} = |f_z|^2 + |f_{\bar{z}}|^2, \qquad \sigma_1\sigma_2 = |f_z|^2 - |f_{\bar{z}}|^2, \tag{11}$$

where the left term is the Dirichlet energy and the right term is $\det(J)$. Substituting into Equation (10b) gives:

$$\mathrm{E}_{\mathrm{iso}}(|f_z|^2, |f_{\bar{z}}|^2) = \left(|f_z|^2 + |f_{\bar{z}}|^2\right)\left(1 + \frac{1}{\left(|f_z|^2 - |f_{\bar{z}}|^2\right)^2}\right),$$

which is clearly smooth in $|f_z|^2$ and $|f_{\bar{z}}|^2$ since $|f_z|^2 \neq |f_{\bar{z}}|^2$.

In Table 1, we list several possible choices for such smooth isometric energies. These include the exponential symmetric Dirichlet isometric energy: $\mathrm{E}_{\mathrm{exp}} = \exp(s \cdot \mathrm{E}_{\mathrm{iso}})$ [Rabinovich et al. 2017]. The motivation to use $\mathrm{E}_{\mathrm{exp}}$ is to penalize more drastically high values of the distortion measure. As demonstrated in Figure 6, this allows the user to trade-off low average versus low maximal distortion. Finally, we include the Advanced MIPS energy [Fu et al. 2015] which provides user-controlled balance between area preservation and angle preservation.

We define the isometric distortion of a planar harmonic map f as a boundary integral over the pointwise distortion quantity $\mathrm{E}(w)$:

$$\mathrm{E}^f = \oint_{\partial\Omega} \mathrm{E}(w) ds. \tag{12}$$

Table 1. Smooth isometric energies. First column: isometric measure in terms of the singular values. Second column: isometric measure in terms of $|f_z|^2$, $|f_{\bar{z}}|^2$ which are denoted x, y for clarity. It is straightforward to see that these expressions are smooth in x, y wherever $x \neq y$, which is always the case for our locally injective maps. Third column: the parameters that are needed for instantiating the gradient (17) and the Hessian (18) expressions. Forth column: The four eigenvalues of $\boldsymbol{K}$ (not necessarily sorted). Potentially negative ones are highlighted.

	$\mathrm{E}(\sigma_1,\sigma_2)$	$\mathrm{E}(x,y)=\mathrm{E}(\lvert f_z\rvert^2,\lvert f_{\bar{z}}\rvert^2)$	$(\alpha_1,\alpha_2,\beta_1,\beta_2,\beta_3)^\mathsf{T}$	Eigenvalues
$\mathrm{E}_{\mathrm{iso}}$	$\frac{1}{2}\left(\sigma_1^2+\sigma_2^2+\sigma_1^{-2}+\sigma_2^{-2}\right)$	$\mathrm{E}_{\mathrm{iso}}=(x+y)\left(1+\frac{1}{(x-y)^2}\right)$	$\begin{pmatrix}\alpha_1^{\mathrm{iso}}=1-(x+3y)(x-y)^{-3}\\ \alpha_2^{\mathrm{iso}}=1+(3x+y)(x-y)^{-3}\\ \beta_1^{\mathrm{iso}}=2(x+5y)(x-y)^{-4}\\ \beta_2^{\mathrm{iso}}=2(5x+y)(x-y)^{-4}\\ \beta_3^{\mathrm{iso}}=-6(x+y)(x-y)^{-4}\end{pmatrix}$	$\underline{\boldsymbol{2\alpha_1}}, 2\alpha_2, \lambda_3, \lambda_4$
$\mathrm{E}_{\mathrm{exp}}$	$\exp\left(\frac{s}{2}\left(\sigma_1^2+\sigma_2^2+\sigma_1^{-2}+\sigma_2^{-2}\right)\right)$	$\exp\left(s\,(x+y)\left(1+\frac{1}{(x-y)^2}\right)\right)$	$s\,\mathrm{E}_{\mathrm{exp}}\begin{pmatrix}\alpha_1^{\mathrm{iso}}\\ \alpha_2^{\mathrm{iso}}\\ \beta_1^{\mathrm{iso}}+s(\alpha_1^{\mathrm{iso}})^2\\ \beta_2^{\mathrm{iso}}+s(\alpha_2^{\mathrm{iso}})^2\\ \beta_3^{\mathrm{iso}}+s\alpha_1^{\mathrm{iso}}\alpha_2^{\mathrm{iso}}\end{pmatrix}$	$\underline{\boldsymbol{2\alpha_1}}, 2\alpha_2, \lambda_3, \lambda_4$
$\mathrm{E}_{\mathrm{AMIPS}}$	$\exp\left(s\left(\frac{\sigma_1}{\sigma_2}+\frac{\sigma_2}{\sigma_1}\right)+\left(\sigma_1\sigma_2+\sigma_1^{-1}\sigma_2^{-1}\right)\right)$	$\exp\left(\frac{2s(x+y)+1}{x-y}+x-y\right)$	$\begin{pmatrix}\left(1-(4sy+1)(x-y)^{-2}\right)\mathrm{E}_{\mathrm{AMIPS}}\\ \left(-1+(4sx+1)(x-y)^{-2}\right)\mathrm{E}_{\mathrm{AMIPS}}\\ \alpha_1^2\mathrm{E}_{\mathrm{AMIPS}}^{-1}+2(4sy+1)(x-y)^{-3}\mathrm{E}_{\mathrm{AMIPS}}\\ \alpha_2^2\mathrm{E}_{\mathrm{AMIPS}}^{-1}+2(4sx+1)(x-y)^{-3}\mathrm{E}_{\mathrm{AMIPS}}\\ \alpha_1\alpha_2\mathrm{E}_{\mathrm{AMIPS}}^{-1}-(4sx+4sy+2)(x-y)^{-3}\mathrm{E}_{\mathrm{AMIPS}}\end{pmatrix}$	$\underline{\boldsymbol{2\alpha_1}}, \underline{\boldsymbol{2\alpha_2}}, \underline{\boldsymbol{\lambda_3}}, \underline{\boldsymbol{\lambda_4}}$

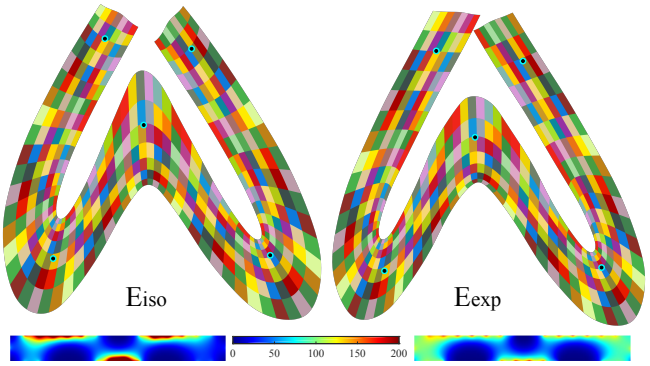

Fig. 6. Comparison of $\mathrm{E}_{\mathrm{iso}}$ (left) and $\mathrm{E}_{\mathrm{exp}}$ (right). A straight bar shape is deformed into an extreme pose. Exponentiating the $\mathrm{E}_{\mathrm{iso}}$ distortion measure reduces the maximal distortion (red color) but increases the average distortion. Both color maps show $\mathrm{E}_{\mathrm{iso}}$.

We use the superscript f throughout the paper to denote the boundary integrated distortion and omit it to denote the pointwise quantity. We also experimented with an area integral instead of the boundary one, but did not notice any benefit. Theorem 4.2 ensures that for any bound on $\mathrm{E}(w)$ along the boundary, a global upper bound on $\sigma_1(z)$, and a global lower bound on $\sigma_2(z)$ exist. Hence, a global bound on $\mathrm{E}(z)$ is naturally formed.

The gradient and the Hessian of the overall isometric energy are:

$$\nabla \mathrm{E}^f = \oint_{\partial\Omega} \nabla \mathrm{E}(w) ds \tag{13}$$

$$\nabla^2 \mathrm{E}^f = \oint_{\partial\Omega} \nabla^2 \mathrm{E}(w) ds. \tag{14}$$

The gradient and the Hessian of our energy measures have relatively simple closed-form expressions. A complete derivation is given in Appendix E. Let $D = (D_1, D_2, \ldots, D_n) \in \mathbb{C}^{1\times n}$ be a complex row vector, where D_j is defined as in (9). We use bold symbols to denote *real* vectors and matrices. Define the real matrix $\boldsymbol{D}$ (note the bold symbol) as:

$$\boldsymbol{D} = \begin{bmatrix} \mathrm{Re}\,(D) & -\mathrm{Im}\,(D) \\ \mathrm{Im}\,(D) & \mathrm{Re}\,(D) \end{bmatrix} \in \mathbb{R}^{2\times 2n}. \tag{15}$$

We express the complex Wirtinger derivatives as 2×1 real vectors:

$$\boldsymbol{f_z} = \begin{bmatrix} \mathrm{Re}\,(f_z) \\ \mathrm{Im}\,(f_z) \end{bmatrix}, \quad \overline{\boldsymbol{f_{\bar{z}}}} = \begin{bmatrix} \mathrm{Re}\left(\overline{f_{\bar{z}}}\right) \\ \mathrm{Im}\left(\overline{f_{\bar{z}}}\right) \end{bmatrix} \in \mathbb{R}^{2\times 1}. \tag{16}$$

The gradient of E with respect to the $4n$ real variables is:

$$\nabla \mathrm{E}(z) = 2\begin{bmatrix} \alpha_1 \boldsymbol{D}^\mathsf{T} \boldsymbol{f_z} \\ \alpha_2 \boldsymbol{D}^\mathsf{T} \overline{\boldsymbol{f_{\bar{z}}}} \end{bmatrix} \in \mathbb{R}^{4n\times 1}, \tag{17}$$

where α_1, α_2 are real parameters which depend on the particular choice of energy. Table 1 provides the parameters needed to instantiate some particular energies (out of many possible). The expression for the $4n\times 4n$ Hessian of $\mathrm{E}(z)$ at a single point is:

$$\nabla^2 \mathrm{E}(z) = \underbrace{\begin{bmatrix} \boldsymbol{D}^\mathsf{T} & 0 \\ 0 & \boldsymbol{D}^\mathsf{T} \end{bmatrix}}_{4n\times 4} \underbrace{\boldsymbol{K}}_{4\times 4} \underbrace{\begin{bmatrix} \boldsymbol{D} & 0 \\ 0 & \boldsymbol{D} \end{bmatrix}}_{4\times 4n} \in \mathbb{R}^{4n\times 4n}, \tag{18}$$

where $\boldsymbol{K}$ is the a 4×4 real matrix:

$$\boldsymbol{K} = \begin{bmatrix} 2\alpha_1 I + 4\beta_1 \boldsymbol{f_z}\boldsymbol{f_z}^\mathsf{T} & 4\beta_3 \boldsymbol{f_z}\overline{\boldsymbol{f_{\bar{z}}}}^\mathsf{T} \\ 4\beta_3 \overline{\boldsymbol{f_{\bar{z}}}}\boldsymbol{f_z}^\mathsf{T} & 2\alpha_2 I + 4\beta_2 \overline{\boldsymbol{f_{\bar{z}}}}\,\overline{\boldsymbol{f_{\bar{z}}}}^\mathsf{T} \end{bmatrix} \in \mathbb{R}^{4\times 4}. \tag{19}$$

The parameters $\beta_1, \beta_2, \beta_3$ are also energy dependent (see Table 1).

6.1 Positional Constraints

The point-to-point (P2P) metaphor is an intuitive drag-and-drop user-interface that best fits interactive deformation tasks. The user of our system can add or remove (by clicking) positional constraints at any time during interaction. Dragging the P2P handles signals the application to invoke the optimization and to render the updated result. We incorporate the P2P constraints as soft constraints as was

advocated by [Chen and Weber 2015; Poranne and Lipman 2014; Rabinovich et al. 2017]. We define the P2P energy:

$$\mathrm{E}^f_{\mathrm{p2p}} = \frac{1}{2}\sum_{i=1}^{|\mathcal{P}|} |f(p_i) - q_i|^2,$$

where $p_i \in \mathcal{P} \subset \Omega$ is the position of the i^{th} handle in the source domain, and $q_i \in \mathbb{C}$ is its target desired position. The gradient and the Hessian of $\mathrm{E}^f_{\mathrm{p2p}}$ are derived in Appendix F. The full energy of our unconstrained minimization problem is:

$$\mathrm{E}^f_{\mathrm{Def}} = \mathrm{E}^f + \lambda \mathrm{E}^f_{\mathrm{p2p}}, \tag{20}$$

where λ is a user-defined weight that balances the two terms.

7 POSITIVE DEFINITE HESSIAN

The Hessian of $\mathrm{E}^f_{\mathrm{p2p}}$ is positive semi-definite (PSD). However, the Hessian of E^f is, in general, not PSD, and neither is the Hessian of $\mathrm{E}^f_{\mathrm{Def}}$ (Equation (20)). Our key observation here is that the closest (in Frobenius norm) PSD matrix to the Hessian $\nabla^2\mathrm{E}(z) \in \mathbb{R}^{4n\times 4n}$ at a *single* point z, can be expressed in closed-form. With that at hand we substitute Equation (14) with:

$$\nabla^2\mathrm{E}^{f+} = \oint_{\partial\Omega} \nabla^2\mathrm{E}^+(w)ds, \tag{21}$$

where $\nabla^2\mathrm{E}^+(w)$ is the closest PSD matrix to $\nabla^2\mathrm{E}(w)$. Since, the integral (or sum) of PSD matrices is PSD (due to the convex cone structure of the PSD set), it is clear that the modified Hessian of the isometric energy (21) is guaranteed to be PSD. In Appendix I, we further show that the dimension of the null space of (21) in the case of $\mathrm{E}_{\mathrm{iso}}$ and $\mathrm{E}_{\mathrm{exp}}$ is 2 or 3. Therefore with 2 or more positional constraints, $\nabla^2\mathrm{E}^{f+}$ is nonsingular. We refer to the variant of Newton's method that computes the closest PSD matrix to $\nabla^2\mathrm{E}^f$ as Newton-Eigen and stress that our modified Newton iterations are dramatically faster to compute. Moreover, throughout extensive experiments, we observe that our local modification leads to iterations which are more *effective*, where less iterations are required for convergence (Figures 1, 10, 11).

7.1 Hessian Modification

Our goal is to compute the eigen-factorization of the local Hessian matrix $\nabla^2\mathrm{E}$. On the one hand, doing it numerically is impractical, whereas on the other hand, analytic eigen-factorization of a $4n\times 4n$ matrix is challenging to derive. Our first observation is that each of the nontrivial eigenvectors (with nonzero eigenvalue) of $\nabla^2\mathrm{E}$ can be expressed as a product of $\boldsymbol{B} = \left[\begin{smallmatrix} \boldsymbol{D} & 0 \\ 0 & \boldsymbol{D} \end{smallmatrix}\right]$ and a corresponding eigenvector of $\boldsymbol{K}$ (Equation (19)). This is due to the fact that $\boldsymbol{B}$ has 4 rows that are orthogonal to each other (easy to verify by computing the inner product of each pair) and all the rows have the same norm. Furthermore, the eigenvalues of $\boldsymbol{K}$ and $\nabla^2\mathrm{E}$ are the same up to a positive scale. Hence, we have reduced our problem to that of analytically computing the eigenvalues of a 4×4 matrix. This is still challenging as these are the roots of a 4^{th} order polynomial. Denote the (unsorted) eigenvalues of $\boldsymbol{K}$ as $(\lambda_1, \lambda_2, \lambda_3, \lambda_4)$. Our second observation is that $\lambda_1 = 2\alpha_1$ and $\lambda_2 = 2\alpha_2$. To see why $2\alpha_1$ is an eigenvalue, subtract $2\alpha_1$ from the diagonal of $\boldsymbol{K}$. The first two rows of $\boldsymbol{K} - 2\alpha_1 I$ are:

$$\underbrace{\boldsymbol{f_z}}_{2\times 1} \underbrace{\left[4\beta_1 \boldsymbol{f_z}^{\mathsf{T}} \quad 4\beta_3 \overline{\boldsymbol{f_{\bar{z}}}}^{\mathsf{T}}\right]}_{1\times 4}, \tag{22}$$

where we can see that the 1×4 row vector on the right hand side of (22) is multiplied by two scalars (the elements of $\boldsymbol{f_z}$), hence, these two rows are linearly dependent, meaning that the matrix $\boldsymbol{K} - 2\alpha_1 I$ is singular and $2\alpha_1$ is a root of the characteristic polynomial. Showing that $2\alpha_2$ is an eigenvalue can be done similarly. Knowing that $2\alpha_1$ and $2\alpha_2$ are two eigenvalues of $\boldsymbol{K}$, we can compute the other two eigenvalues by directly solving the quartic characteristic equation. The derivation is long but straightforward, hence omitted. We get these expressions:

$$\lambda_{3,4} = s_1 \pm \sqrt{s_2^2 + 16\beta_3^2 |f_z|^2 |f_{\bar{z}}|^2}, \tag{23}$$

where $s_{1,2} = \alpha_1 + 2\beta_1 |f_z|^2 \pm \left(\alpha_2 + 2\beta_2 |f_{\bar{z}}|^2\right)$.
The corresponding four eigenvectors are:

$$\begin{aligned} &\left(\mathrm{Im}(f_z), \quad -\mathrm{Re}(f_z), \quad 0, \quad 0\right) \\ &\left(0, \quad 0, \quad \mathrm{Im}(\overline{f_{\bar{z}}}), \quad -\mathrm{Re}(\overline{f_{\bar{z}}})\right) \\ &\left(\mathrm{Re}(f_z), \quad \mathrm{Im}(f_z), \quad t_1\mathrm{Re}(f_{\bar{z}}), \quad t_1\mathrm{Im}(\overline{f_{\bar{z}}})\right) \\ &\left(\mathrm{Re}(f_z), \quad \mathrm{Im}(f_z), \quad t_2\mathrm{Re}(f_{\bar{z}}), \quad t_2\mathrm{Im}(\overline{f_{\bar{z}}})\right) \end{aligned} \tag{24}$$

where:

$$t_{1,2} = \frac{\lambda_{3,4} - 2\alpha_1 - 4\beta_1 |f_z|^2}{4\beta_3 |f_{\bar{z}}|^2}.$$

The signs of these eigenvalues depend on the particular choice of isometric energy, therefore in the most general case, the 4 eigenvalues are evaluated and negative ones are substituted with 0 to obtain the modified Hessian. For particular energy choices, it is possible to simplify matter even more. For the first two energies listed in Table 1, we have that only $\lambda_1 = 2\alpha_1$ can be (at times) negative. This allows us to directly express the modified matrix. For example, for $\mathrm{E}_{\mathrm{iso}}$, it turns out that λ_3, λ_4 have quite simple expressions:

$$\lambda_3 = 4(1 + 3(|f_z| + |f_{\bar{z}}|)^{-4}), \quad \lambda_4 = 4(1 + 3(|f_z| - |f_{\bar{z}}|)^{-4}).$$

The spectrum is then sorted as follows: $2\alpha_1 \le \lambda_3 \le 2\alpha_2 \le \lambda_4$. With this information at hand, the modification is done by checking for the sign of α_1, and if it is negative, we substitute $\boldsymbol{K}$ in (18) with:

$$\boldsymbol{K}^+ = \begin{bmatrix} (\frac{2\alpha_1}{|f_z|^2} + 4\beta_1)\boldsymbol{f_z}\boldsymbol{f_z}^{\mathsf{T}} & 4\beta_3 \boldsymbol{f_z}\overline{\boldsymbol{f_{\bar{z}}}}^{\mathsf{T}} \\ 4\beta_3 \overline{\boldsymbol{f_{\bar{z}}}}\boldsymbol{f_z}^{\mathsf{T}} & 2\alpha_2 I + 4\beta_2 \overline{\boldsymbol{f_{\bar{z}}}}\,\overline{\boldsymbol{f_{\bar{z}}}}^{\mathsf{T}} \end{bmatrix}. \tag{25}$$

8 LOCALLY INJECTIVE CERTIFICATION

To ensure that the map is locally injective at each iteration, we need to verify that Conditions (4a) and (5) hold (Section 4.1), and backtrack during line search otherwise. As (5) involves infinite number of inequalities: $|f_z(w)| > |f_{\bar{z}}(w)| \quad \forall w \in \partial\Omega$, we use a simpler sufficient condition based on a finite number of conditions following the approach of [Chen and Weber 2015] which utilizes the fact that the Wirtinger derivatives are Lipschitz continuous. Condition (5) can be enforced on the entire boundary by enforcing it (individually) on many small boundary segments. For each segment

$[v_i, v_{i+1}]$ (see Figure 4 for notations), we compute lower and upper bounds for $|f_z|$ and $|f_{\bar{z}}|$ respectively as follows:

$$|f_z|_{\min} := \frac{|f_z(v_i)| + |f_z(v_{i+1})|}{2} - \frac{L_{f_z} l}{2} \le \min_{w \in [v_i, v_{i+1}]} |f_z(w)|,$$

$$|f_{\bar{z}}|_{\max} := \frac{|f_{\bar{z}}(v_i)| + |f_{\bar{z}}(v_{i+1})|}{2} + \frac{L_{f_{\bar{z}}} l}{2} \ge \max_{w \in [v_i, v_{i+1}]} |f_{\bar{z}}(w)|.$$

L_{f_z} and $L_{f_{\bar{z}}}$ are the corresponding Lipschitz constants on the segment, and $l = |v_i - v_{i+1}|$. Condition (5) is then substituted with:

$$|f_z(v_i)| - |f_{\bar{z}}(v_i)| + |f_z(v_{i+1})| - |f_{\bar{z}}(v_{i+1})| \ge (L_{f_{\bar{z}}} + L_{f_z}) l, \quad (26)$$

or more concisely:

$$\sigma_2(v_i) + \sigma_2(v_{i+1}) \ge (L_{f_{\bar{z}}} + L_{f_z}) l. \quad (27)$$

It is crucial that the sufficient condition above is as tight as possible, as otherwise the Newton step size may become too small and will stop the Newton iterations prematurely. Note that the condition becomes tighter as the Lipschitz constants and/or the length of the segment l decrease. Since denser sampling requires more computations, it is advised to obtain as small as possible Lipschitz constants. In Appendix H, we derive the following Lipschitz constants that can be used on multiply-connected domains. More importantly, our newly-derived constants are significantly smaller than those of BDHM.

$$L_{f_z} = \frac{|f_z'(v_i)| + |f_z'(v_{i+1})|}{2} + \frac{l}{2}\left(\sum_{j=1}^{m} L_j |s_j - s_{j+1}| + \sum_{j=m+1}^{n} L_j^h |\varphi_j|\right)$$

$$L_{f_{\bar{z}}} = \frac{|f_{\bar{z}}'(v_i)| + |f_{\bar{z}}'(v_{i+1})|}{2} + \frac{l}{2}\left(\sum_{j=1}^{m} L_j |t_j - t_{j+1}| + \sum_{j=m+1}^{n} L_j^h |\psi_j|\right)$$

where $L_j = \frac{1}{2\pi d^2(z_j)}$, $L_j^h = \frac{2}{d^3(\rho_{j-m})}$, $d(z)$ is the distance from a point z to the segment, $s_j = \frac{\varphi_j - \varphi_{j-1}}{z_j - z_{j-1}}$, and $t_j = \frac{\psi_j - \psi_{j-1}}{z_j - z_{j-1}}$. In Figure 7 we compare the Lipschitz constants obtained with our formulas against that of BDHM and show that they are significantly smaller and lead to faster convergence and increased robustness.

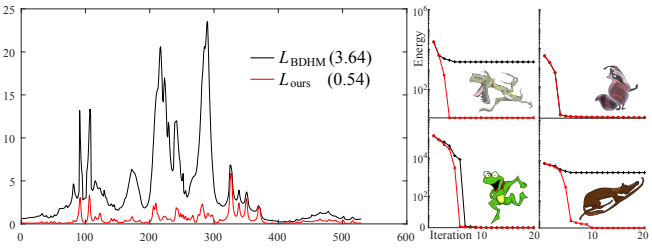

Fig. 7. Left: Comparison of the Lipschitz constants of f_z on 529 boundary segments. L_{BDHM} is the approach taken in [Chen and Weber 2015], while L_{ours} is our improved constants. Our constants are smaller on all the segments with an average (number in parentheses above) which is ×6.74 smaller. Right: Comparison of convergence behavior of our solver using BDHM constants (black) vs. ours (red). See how the iterations of the Raptor and Horse deformations stop immaturely when L_{BDHM} is used.

We now turn to Condition (4a) requiring that the boundary integral (or equivalently the number of zeros of f_z) is 0. Theorem 11 in [Chen and Weber 2015] provides a sufficient (but not necessary) condition that implies (4a), albeit their proof assumes that the domain is simply-connected. We show a much stronger result by deriving a new condition which is applicable to multiply-connected domains, and is both necessary and sufficient as long as (27) holds. The following theorem is proved in Appendix C.

Theorem 8.1. *Let $g(z)$ be an L-Lipschitz continuous holomorphic function in a neighborhood of a simple open curve with length l and two endpoints $v_i, v_{i+1} \in \mathbb{C}$ such that:*

$$|g(v_i)| + |g(v_{i+1})| > Ll, \quad (28)$$

then:

$$\int_{v_i}^{v_{i+1}} \frac{g'(z)}{g(z)} dz = \ln\left|\frac{g(v_{i+1})}{g(v_i)}\right| + \mathrm{i}\,\mathrm{Arg}\frac{g(v_{i+1})}{g(v_i)}, \quad (29)$$

where Arg *is the principle branch of the complex argument function.*

Theorem 8.1 is applicable to our f_z along any line segment $[v_i, v_{i+1}]$ since our algorithm always verifies that (27) holds (which implies (28)). Consequently we have the following corollary.

Corollary 8.2. *Under the assumption that (27) holds, f_z does not vanish inside the multiply-connected polygon* P *if and only if:*

$$\sum_{j=0}^{N} \sum_{i} \mathrm{Arg}\frac{f_z(v_{i+1}^j)}{f_z(v_i^j)} = 0, \quad (30)$$

where the first summation is over the polygonal loops, and the second one is over the segments in each loop.

Proof. It follows immediately from Cauchy's argument principle and (29) that:

$$2\pi\mathrm{i}\mathcal{N} = \oint_{\partial\Omega} \frac{f_z'(w)}{f_z(w)} dw = \sum_{j=0}^{N} \sum_{i} \left(\ln\left|\frac{f_z(v_{i+1}^j)}{f_z(v_i^j)}\right| + \mathrm{i}\mathrm{Arg}\frac{f_z(v_{i+1}^j)}{f_z(v_i^j)}\right),$$

where $\mathcal{N}$ is the number of zeros of f_z. The $\ln|\cdot|$ terms cancel since $\ln x/y = \ln x - \ln y$ and all the boundary polygons are closed. □

To conclude, we first verify that (27) holds on all the boundary segments and if so, we simply compute the sum in (30). If it is zero, the map is *guaranteed* to be locally injective everywhere.

9 IMPLEMENTATION

Algorithm 1 provides a pseudocode for our solver. The algorithm can be dramatically accelerated by exploiting parallelism. Most of the steps require only basic dense linear algebra operations and are straightforward to implement. Unlike sparse linear algebra operations, our method greatly benefits when these operations are performed on the GPU. In this section, we point out some of the practical aspects of the implementation. Furthermore, Appendix G contains elaborated description of our GPU implementation. In order to promote reproducibility and to encourage further development of GPU-based solvers, we provide a publicly available reference implementation (http://github.com/renjiec/GLID).

Algorithm 1 Modified Newton Solver

Input: $(\boldsymbol{\varphi}, \boldsymbol{\psi})$ ▷ initialization - locally injective map
$p_i \in \mathcal{P} \subset \Omega, \quad q_i \in \mathbb{C}$ ▷ P2P constraints
Output: $(\boldsymbol{\varphi}, \boldsymbol{\psi})$ ▷ new locally injective map

1: DoPreprocessing() ▷ Appendix G
2: **loop** ▷ Newton iterations
3: $\mathrm{E}^f_{\mathrm{Def}} \leftarrow \mathrm{E}^f + \lambda \mathrm{E}^f_{\mathrm{p2p}}$ ▷ evaluate energy
4: $\boldsymbol{g} \leftarrow \nabla \mathrm{E}^f + \lambda \nabla \mathrm{E}^f_{\mathrm{p2p}}$ ▷ evaluate gradients (G.7),(G.9)
5: $\boldsymbol{H} \leftarrow \nabla^2 \mathrm{E}^{f+} + \lambda \nabla^2 \mathrm{E}^f_{\mathrm{p2p}}$ ▷ modified Hessian (G.10)
6: $(\boldsymbol{H}, \boldsymbol{g}) \leftarrow$ EliminateVariables$(\boldsymbol{H}, \boldsymbol{g})$ ▷ Section 5
7: $\boldsymbol{H}\boldsymbol{d} = -\boldsymbol{g}, \quad \text{where } \boldsymbol{d} = \begin{bmatrix} \boldsymbol{d}_{\boldsymbol{\varphi}} \\ \boldsymbol{d}_{\boldsymbol{\psi}} \end{bmatrix}$ ▷ solve (1) using Cholesky
8: $(\boldsymbol{\varphi}, \boldsymbol{\psi}, t) \leftarrow$ LineSearch$(\mathrm{E}^f_{\mathrm{Def}}, \boldsymbol{d}, \boldsymbol{g}, \boldsymbol{\varphi}, \boldsymbol{\psi})$ ▷ backtracking
9: **if** $t \, \|\boldsymbol{d}\| \le \epsilon$ **then** ▷ solver converged
10: **return** $(\boldsymbol{\varphi}, \boldsymbol{\psi})$
11: **end if**
12: **end loop**

13: **procedure** LineSearch$(\mathrm{E}^f_{\mathrm{Def}}, \boldsymbol{d}, \boldsymbol{g}, \boldsymbol{\varphi}, \boldsymbol{\psi})$ ▷ Appendix G
14: $t \leftarrow 1$
15: **repeat**
16: $(\mathfrak{f}^t_z, \overline{\mathfrak{f}^t_z}) \leftarrow$ ComputeWirtinger() ▷ Equation (G.11)
17: **if** $\mathrm{E}^f_{\mathrm{Def}}(\mathfrak{f}^t_z, \overline{\mathfrak{f}^t_z}) < \mathrm{E}^f_{\mathrm{Def}} + c\, t(\boldsymbol{d}^{\mathsf{T}}\boldsymbol{g})$ **then** ▷ $c \in (0,1)$
18: **if** (27) and (30) hold **then** ▷ locally injective
19: **return** $(\boldsymbol{\varphi} + t\boldsymbol{d}_{\boldsymbol{\varphi}}, \boldsymbol{\psi} + t\boldsymbol{d}_{\boldsymbol{\psi}}, t)$
20: **end if**
21: **end if**
22: $t \leftarrow t/2$
23: **until** MaxNumSteps
24: **end procedure**

9.1 Isometric energy, gradient and Hessian evaluation

While the map and its differentials possess closed-form expressions, we are unable to solve the contour integrals that arise in Equations (12),(13),(14) analytically, and resort to numerical integration. Let $\mathcal{G} = \{w_1, w_2, \ldots, w_{|\mathcal{G}|}\} \subset \partial\Omega$ be a set of uniformly distributed samples. We apply the trapezoidal rule to (12) which due to the uniform sampling boils down to a simple sum:

$$\mathrm{E}^f \approx \frac{1}{|\mathcal{G}|} \sum_{i=1}^{|\mathcal{G}|} \mathrm{E}(w_i). \tag{31}$$

The gradient of E^f is approximated in the same fashion:

$$\nabla \mathrm{E}^f \approx \frac{1}{|\mathcal{G}|} \sum_{i=1}^{|\mathcal{G}|} \nabla \mathrm{E}(w_i), \tag{32}$$

however, it turns out that for the integral approximation of the Hessian, much coarser approximation can be used with negligible influence on the number of Newton iterations (see Figure 8). Hence, we use an additional set $\mathcal{H}$ of samples which is a subset of $\mathcal{G}$ but

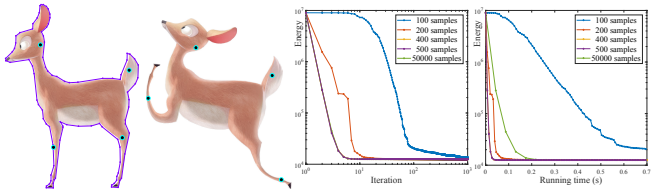

Fig. 8. Hessian approximation. We show the effect of changing the number of Hessian samples $|\mathcal{H}|$ on our algorithm when applied to the depicted deformation of the deer. We observed similar behavior for other models and deformations. With 200 samples or less, we see an increase in the number of iterations needed for convergence (left plot). However, with 400 samples, the convergence rate is identical to that of 50, 000 (note the three overlapping curves). On the right plot we see that with 50, 000 samples, the algorithm becomes 3 times slower due to the longer time it takes to assemble the Hessian, yet, 400 samples are sufficient, in which case the algorithm converges after $30ms$.

contains significantly less samples. The Hessian is approximated:

$$\nabla^2 \mathrm{E}^f \approx \frac{1}{|\mathcal{H}|} \sum_{i=1}^{|\mathcal{H}|} \nabla^2 \mathrm{E}^+(w_i), \tag{33}$$

where $\nabla^2 \mathrm{E}^+(w_i)$ is the *analytic* modification of the pointwise Hessian. We note on a theoretical level, that for any number $|\mathcal{H}|$ (as small as it may be), the Newton step produces a descent direction since the matrix $\nabla^2 \mathrm{E}^f$ in (33) is guaranteed to be PSD (strict positive definiteness can be ensured by adding ϵ to the diagonal). In the extreme case that $\mathcal{H}$ is an empty set, the algorithm degrades to gradient descent. We observed that in practice, choosing $|\mathcal{H}| = 0.1\,|\mathcal{G}|$ does not degrade the effectiveness of the Newton iterations and provides a good balance between the time it takes to perform the numerical integration and the other steps of the algorithm. We use this setting in all the results. Moreover, since in practice $|\mathcal{H}|$ is much larger than the dimension of our subspace, it is not necessary to add ϵ to the diagonal (Appendix I).

9.2 Line Search

To ensure convergence to a local minimum from an arbitrary starting point we use simple backtracking [Nocedal and Wright 2006, Algorithm 3.1] to find a step size that sufficiently decreases the energy (Algorithm 1, Line 17) and in addition ensures that the new map will be locally injective using the strategy of Section 8 (Algorithm 1, Line 18). This requires evaluations of the energy and the Wirtinger derivatives at different step sizes. Appendix G describes how to compute it efficiently.

10 RESULTS

We use the following default parameters for all our experiments unless stated otherwise. The number of samples is $|\mathcal{G}| = 10^4$, and $|\mathcal{H}| = 0.1\,|\mathcal{G}|$. The two energy terms E^f and $\mathrm{E}^f_{\mathrm{p2p}}$ are balanced using $\lambda = 10^5$. When comparing to mesh-based methods, a mesh with 10, 000 vertices is used. User prescribed P2P target positions q_i are visualized as cyan disks. The images of the P2P under the map $f(p_i)$ are visualized as smaller black dots. A black dot which is centered inside a cyan disk indicates P2P satisfaction. Our machine

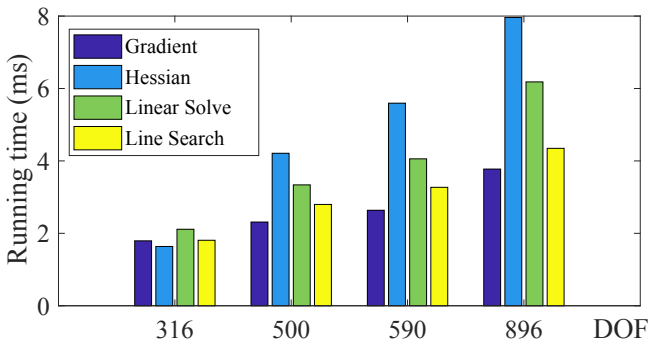

Fig. 9. Algorithm Balance. We show how the runtime of our GPU-based modified-Newton solver is spread across the 4 main stages that compose a single iteration. Each group of bins corresponds to a different model with variable amount of degrees of freedom (variables). For all models we use $|\mathcal{G}| = 10,000$, $|\mathcal{H}| = 1,000$.

runs Windows 10 with Intel Xeon (6 cores) CPU E5-2643 v4 3.40Ghz, 32GB, and has an NVIDIA TitanX graphics card.

In Figure 9 we demonstrate that our solver is well balanced with respect to the four different stages that compose a single Newton iteration: gradient evaluation, Hessian modification-and-assembly, Cholesky linear solve, and line search. Table 2 compares the runtime in milliseconds of a single iteration of various algorithms minimizing the symmetric Dirichlet energy. Among all methods, AQP iterations are the fastest (followed by our GPU iterations), albeit we observe that AQP requires significantly larger number of iterations compared to our GPU/CPU Newton solver. Moreover, the initialization required for AQP (see Section 2) further slows down the runtime (right most column). For comparisons with AQP we run 5 iterations of ARAP local/global [Liu et al. 2008] followed by the projection method of [Kovalsky et al. 2015] as was suggested in [Kovalsky et al. 2016].

Throughout our experiments, we consistently observe that the convergence rate of our algorithm is significantly better than that of the first-order methods (L-BFGS, SLIM, AQP) which we attribute to its second-order nature. Furthermore, the convergence rate is always better even when compared against the Newton-Eigen method which employs a full eigen-factorization [Nocedal and Wright 2006, Equation 3.40] of the Hessian at each iteration, followed by a truncation of negative eigenvalues to 1e−10. This is demonstrated in Figure 1 (6 vs. 32 iters), Figure 10 (15 vs. 35 iters), Figure 11 (8 vs. 11 iters) suggesting that our Hessian modification is not only faster to compute but also more effective in reducing the energy. Both the eigen-factorization and the solution to the linear system are computed using [cuSOLVER 2017].

The full potential of our deformation algorithm is leveraged when the algorithm runs on graphics hardware, yet our CPU parallel Matlab version (which is approximately 5 times slower on average) is still much faster than the alternatives we compare to. In Figure 10 we show a visual comparison of our algorithm against state-of-the-art methods. For each method we include small images of the maps obtained at intermediate stages of the algorithm to show that for the competing methods, many iterations are needed to get to a state where the progress is not visually apparent. Our method is between ×25 to ×58 faster than these competing methods.

Table 2. Comparison of runtime (ms) of a single iteration. SLIM and AQP use a mesh with 10, 000 vertices, while the other listed solvers operate within our harmonic subspace with the specified DOF.

	DOF	Ours/GPU	Ours/CPU	Newton Eigen	LBFGS	SLIM	AQP / Initialization	
deer	364	8.87	38.8	49.6	54.4	58.3	7.84	1239.7
archery	596	15.31	71.2	95.1	88.3	59.4	9.27	1266.3
giraffe	624	14.05	58.1	86.8	54.9	56.4	8.27	1182.9
rex	392	11.7	61.6	77.6	49.5	57.2	8.52	1141.0
racoon	320	7.25	34.4	39.9	68.1	53.7	9.01	904.6
raptor	416	10.18	45.2	56.5	82.3	55.1	8.64	883.4

We observe that qualitatively, the results obtained with the mesh-based methods tend to concentrate the distortion near the positional constraints. Refining the mesh does not seem to alleviate this issue completely while it increases the computation times even further. We experimentally find that a mesh resolution of 10^4 vertices provides a reasonable balance between quality and speed and use it to evaluate AQP and SLIM. Our algorithm on the other hand is rather indifferent to the mesh resolution and using a mesh with 10^5 vertices, leads to extremely smooth results with no degradation in speed.

Another comparison is given in Figure 11. Our algorithm fully converges in 0.081s (8 iterations) while AQP and SLIM take 2.51s and 9.36s respectively. For the comparisons, we initialize all methods with the identity map which is quite far from the final configuration that is determined by the P2P. In an actual interactive session, the solver utilizes the previous deformation for initialization. Convergence is obtained almost instantly, consequently our algorithm runs in realtime rates.

Figure 7 shows the improvement of our newly-derived Lipschitz constants compared to that of BDHM. These greatly contribute to the success of the algorithm as they ensure that the locally injective validation (line search) has minimal interference, allowing larger Newton steps to be taken.

The appendix contains an additional compilation of high resolution images of deformation results that are obtained with our algorithm. Some of these deformations are intentionally radical and are used to stress test the method and to demonstrate robustness. Our algorithm quickly converges on these extreme deformations and certifies the maps as locally injective. We also include an accompanying video with our paper. Watching it greatly assists in appreciating how fast the algorithm is.

11 SUMMARY AND DISCUSSION

We construct a new finite dimensional subspace of harmonic maps for multiply-connected domains and extend the bounded distortion theorem of [Chen and Weber 2015] to this setting. The harmonic subspace is utilized to devise a novel deformation method with superior performance by casting the locally injective mapping problem as a boundary value problem, minimizing an isometric energy of choice. Within this low dimensional subspace, the induced map and its differentials are C^∞ functions and can be expressed in closed-form,

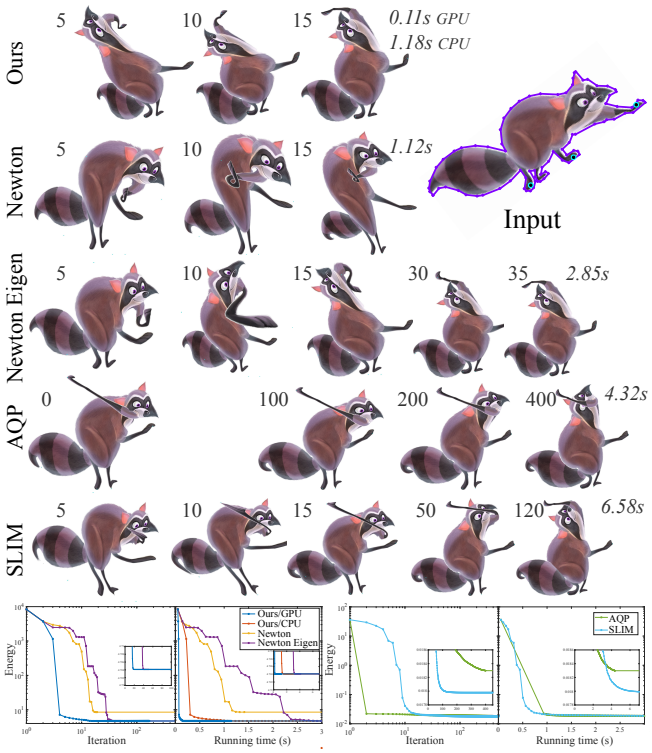

Fig. 10. Visual comparison of the progress of different solvers minimizing $E^f_{iso} + \lambda E^f_{p2p}$. The top three rows use the harmonic subspace, while AQP and SLIM are general mesh-based methods. The total number of iterations needed for each method as well as the total runtime is reported on the right of each row. The graphs depict the decrease in energy as a function of iteration and as a function of time, separating harmonic from mesh-based methods as they are scaled differently. The "jump" of $0.9s$ in AQP's graph (first iteration) is due to the initialization. Note that due to the large weight λ, the energy at the first few iterations is very large and E^f_{p2p} obscures the behavior of E^f_{iso}. The zoom-in graphs show how the energy decreases more clearly. Our method converges in 15 iterations. Newton's method halts after 15 iterations due to non-positive definite Hessian. Newton-Eigen performs 35 iterations and is ×25 slower than our GPU solver. AQP and SLIM require 400 and 120 iterations and are ×39 and ×58 slower respectively.

leading to simple, efficient and very accurate evaluation. Moreover, we provide a closed-form recipe for a positive definite modification of the Hessian of these isometric energies that leads to effective-and-efficient Newton iterations. Finally, our algorithm is specifically designed for efficient implementation on modern parallel graphics hardware architectures.

11.1 Limitations and Future Work

The main limitation of our method, which at the same time is also the reason for its superior quality and speed, is its dependence on the reduced dimensional subspace we operate within. Depending on the application, one might desire to produce maps which are not harmonic or not even smooth. Another limitation which is shared by all other methods is the fact that while our method is extremely robust and seems to converge to maps with very low distortion, global optimality is not guaranteed since the underlying energy is nonconvex. Furthermore, we cannot guarantee that the positional constraints will always be satisfied.

An interesting avenue for future research would be to continue using the harmonic subspace but in a mesh-based setting. While we argue that for the planar deformation setting, the C^∞ basis functions are the most suitable ones, using meshes will allow us to generalize the algorithm to curved manifolds which are the representation of choice for many geometry processing applications, such as surface parameterization, and remeshing. A first step in that direction would be to generalize Theorem 4.2 to smooth *curved* manifolds, and then to develop a discrete counterpart bounded distortion theorem [Bright et al. 2017].

ACKNOWLEDGMENTS

We thank Edward Chien for illuminating discussions on the integrability of the Wirtinger derivatives on multiply-connected domains and for writing the proof of Theorems 4.1, 4.2, to Markus Steinberger for helping with the GPU implementation, and to Eden Fedida Hefetz for proofreading.

REFERENCES

Ahlfors L.V. 1979. *Complex analysis: an introduction to the theory of analytic functions of one complex variable.* McGraw-Hill.
Aigerman Noam and Lipman Yaron. 2013. Injective and bounded distortion mappings in 3D. *ACM Transactions on Graphics* 32, 4 (2013), 106.
Aigerman Noam, Poranne Roi, and Lipman Yaron. 2014. Lifted bijections for low distortion surface mappings. *ACM Transactions on Graphics* 33, 4 (2014), 69.
ApS MOSEK. 2017. *The MOSEK optimization software.* http://www.mosek.com/
Axler Sheldon. 1986. Harmonic functions from a complex analysis viewpoint. *The American mathematical monthly* 93, 4 (1986), 246–258.
Bright Alon, Chien Edward, and Weber Ofir. 2017. Harmonic Global Parametrization with Rational Holonomy. *ACM Transactions on Graphics* 36, 4, Article 89 (2017).
Chen Renjie and Weber Ofir. 2015. Bounded distortion harmonic mappings in the plane. *ACM Transactions on Graphics* 34, 4, Article 73 (2015).
Chien Edward, Chen Renjie, and Weber Ofir. 2016a. Bounded Distortion Harmonic Shape Interpolation. *ACM Transactions on Graphics* 35, 4, Article 105 (2016).
Chien Edward, Levi Zohar, and Weber Ofir. 2016b. Bounded distortion parametrization in the space of metrics. *ACM Transactions on Graphics* 35, 6, Article 215 (2016).
cuBLAS. 2017. CUDA Basic Linear Algebra Subroutines. http://developer.nvidia.com/cublas. (2017). Accessed: 13-May-2017.
cuSOLVER. 2017. CUDA Linear Solvers Library. http://developer.nvidia.com/cusolver. (2017). Accessed: 13-May-2017.
Duren Peter. 2004. *Harmonic mappings in the plane.* Cambridge University Press.
Fu Xiao-Ming and Liu Yang. 2016. Computing inversion-free mappings by simplex assembly. *ACM Transactions on Graphics* 35, 6 (2016), 216.
Fu Xiao-Ming, Liu Yang, and Guo Baining. 2015. Computing locally injective mappings by advanced MIPS. *ACM Transactions on Graphics* 34, 4 (2015), 71.
Hefetz Eden Fedida, Chien Edward, and Weber Ofir. 2017. Fast Planar Harmonic Deformations with Alternating Tangential Projections. *Computer Graphics Forum* 36, 5 (2017), 175–188. Proceedings of Symposium on Geometry Processing 2017.
Hormann Kai and Floater Michael S. 2006. Mean value coordinates for arbitrary planar polygons. *ACM Transactions on Graphics* 25, 4 (2006), 1424–1441.
Igarashi Takeo, Moscovich Tomer, and Hughes John F. 2005. As-rigid-as-possible shape manipulation. *ACM Transactions on Graphics* 24, 3 (2005), 1134–1141.
Jacobson Alec, Baran Ilya, Popović Jovan, and Sorkine Olga. 2011. Bounded biharmonic weights for real-time deformation. *ACM Transactions on Graphics* 30, 4, Article 78 (2011).
Kavan Ladislav, Collins Steven, Žára Jiří, and O'Sullivan Carol. 2007. Skinning with dual quaternions. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games.* ACM, 39–46.
Kovalsky Shahar Z., Aigerman Noam, Basri Ronen, and Lipman Yaron. 2015. Large-scale bounded distortion mappings. *ACM Transactions on Graphics* 34, 6, Article 191 (2015).
Kovalsky Shahar Z., Galun Meirav, and Lipman Yaron. 2016. Accelerated Quadratic Proxy for Geometric Optimization. *ACM Transactions on Graphics* 35, 4, Article 134 (2016).

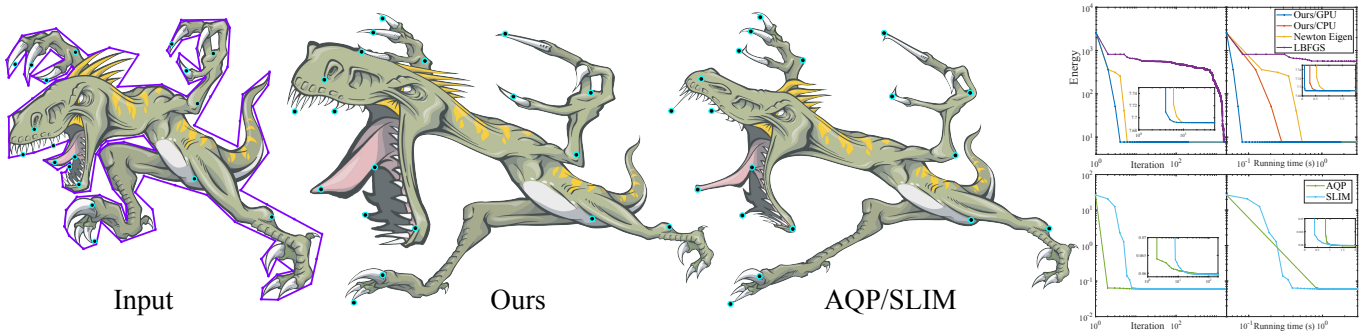

Fig. 11. Raptor. Our solver converges after 8 iterations ($0.081s$ on GPU, $0.31s$ on CPU) to the depicted map. The Newton-Eigen solver converges to the same result after 11 iterations which took $0.4s$ due to the longer iteration time. The first-order methods required many more iterations to converge. L-BFGS needs as many as 1, 612 iterations to converge to the same result. AQP and SLIM converge to the result depicted on the right after 188 and 170 iterations respectively. Note how the distortion is concentrated around the P2P in this result.

Levi Zohar and Weber Ofir. 2016. On the convexity and feasibility of the bounded distortion harmonic mapping problem. *ACM Transactions on Graphics* 35, 4, Article 106 (2016).
Lipman Yaron. 2012. Bounded distortion mapping spaces for triangular meshes. *ACM Transactions on Graphics* 31, 4 (2012), 108.
Lipman Yaron, Kopf Johannes, Cohen-Or Daniel, and Levin David. 2007. GPU-assisted positive mean value coordinates for mesh deformations. In *Symposium on geometry processing*, Vol. 257. 117–123.
Lipman Yaron, Sorkine Olga, Levin David, and Cohen-Or Daniel. 2005. Linear rotation-invariant coordinates for meshes. *ACM Transactions on Graphics* 24, 3 (2005), 479–487.
Liu Ligang, Zhang Lei, Xu Yin, Gotsman Craig, and Gortler Steven J. 2008. A local/global approach to mesh parameterization. *Computer Graphics Forum* 27, 5 (2008), 1495–1504.
Liu Tiantian, Bouaziz Sofien, and Kavan Ladislav. 2017. Quasi-Newton Methods for Real-Time Simulation of Hyperelastic Materials. *ACM Transactions on Graphics* 36, 3 (2017), 23.
Martin Tobias, Joshi Pushkar, Bergou Miklós, and Carr Nathan. 2013. Efficient Non-linear Optimization via Multi-scale Gradient Filtering. *Computer Graphics Forum* 32, 6 (2013), 89–100.
Nocedal Jorge and Wright Stephen. 2006. *Numerical Optimization.* Springer New York.
Poranne Roi and Lipman Yaron. 2014. Provably good planar mappings. *ACM Transactions on Graphics* 33, 4 (2014), 76.
Rabinovich Michael, Poranne Roi, Panozzo Daniele, and Sorkine-Hornung Olga. 2017. Scalable Locally Injective Mappings. *ACM Transactions on Graphics* 36, 2, Article 16 (2017).
Schreiner John, Asirvatham Arul, Praun Emil, and Hoppe Hugues. 2004. Inter-surface mapping. *ACM Transactions on Graphics* 23, 3 (2004), 870–877.
Schüller Christian, Kavan Ladislav, Panozzo Daniele, and Sorkine-Hornung Olga. 2013. Locally injective mappings. *Computer Graphics Forum* 32, 5 (2013), 125–135.
Smith Jason and Schaefer Scott. 2015. Bijective Parameterization with Free Boundaries. *ACM Transactions on Graphics* 34, 4, Article 70 (2015).
Sorkine Olga and Alexa Marc. 2007. As-rigid-as-possible Surface Modeling. In *Proceedings of the Symposium on Geometry Processing*. Eurographics, Switzerland, 109–116.
Teran Joseph, Sifakis Eftychios, Irving Geoffrey, and Fedkiw Ronald. 2005. Robust quasistatic finite elements and flesh simulation. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer Animation*. ACM, 181–190.
Vaxman Amir, Müller Christian, and Weber Ofir. 2015. Conformal mesh deformations with Möbius transformations. *ACM Transactions on Graphics* 34, 4, Article 55 (2015).
Weber Ofir. 2010. *Hybrid Methods for Interactive Shape Manipulation.* Ph.D. Dissertation. Technion - Israel Institute of Technology.
Weber Ofir. 2017. Planar Shape Deformation. In *Generalized Barycentric Coordinates in Computer Graphics and Computational Mechanics*, Kai Hormann and N. Sukumar (Eds.). CRC Press, Boca Raton, FL, Chapter 7, 28.
Weber Ofir, Ben-Chen Mirela, and Gotsman Craig. 2009. Complex Barycentric Coordinates with Applications to Planar Shape Deformation. *Computer Graphics Forum* 28, 2 (2009), 587–597.
Weber Ofir, Ben-Chen Mirela, Gotsman Craig, and Hormann Kai. 2011. A complex view of barycentric mappings. *Computer Graphics Forum* 30, 5 (2011), 1533–1542.
Weber Ofir and Gotsman Craig. 2010. Controllable conformal maps for shape deformation and interpolation. *ACM Transactions on Graphics* 29, 4, Article 78 (2010).
Weber Ofir, Sorkine Olga, Lipman Yaron, and Gotsman Craig. 2007. Context-Aware Skeletal Shape Deformation. *Computer Graphics Forum* 26, 3 (2007), 265–274.
Zayer Rhaleb, Rössl Christian, Karni Zachi, and Seidel Hans-Peter. 2005. Harmonic guidance for surface deformation. *Computer Graphics Forum* 24, 3 (2005), 601–609.

APPENDIX

A PROOF OF THEOREM 4.1

We first state the following theorem by Axler [1986]:

Theorem A.1 (Logarithmic Conjugation Theorem). *Let Ω be a multiply-connected planar domain with N holes $K_1, \ldots, K_N$, and choose N arbitrary points ρ_i in K_i. Then, any real-valued harmonic function $u : \Omega \to \mathbb{R}$ can be represented as:*

$$u(z) = \mathrm{Re}\,(g(z)) + \sum_{i=1}^{N} x_i \ln|z - \rho_i|\,, \tag{A.1}$$

where g is holomorphic in Ω, and $x_1, \ldots, x_N$ are real coefficients.

Proof. Since $\mathrm{Re}\,(f)$ and $\mathrm{Im}\,(f)$ are harmonic in Ω, Theorem A.1 states that there exist holomorphic functions g_1, g_2 and some real constants x_i, y_i such that:

$$\mathrm{Re}\,(f(z)) = \mathrm{Re}\,(g_1(z)) + \sum_{j=1}^{N} x_i \ln|z - \rho_i|\,,$$

$$\mathrm{Im}\,(f(z)) = \mathrm{Re}\,(g_2(z)) + \sum_{j=1}^{N} y_i \ln|z - \rho_i|\,.$$

Simple algebraic manipulation gives us:

$$f(z) = \mathrm{Re}\,(g_1(z)) + \mathrm{iRe}\,(g_2(z)) + \sum_{j=1}^{N}(x_i + \mathrm{i}y_i)\ln|z - \rho_i|$$

$$= \frac{g_1(z) + \mathrm{i}g_2(z)}{2} + \overline{\frac{g_1(z) - \mathrm{i}g_2(z)}{2}} + \sum_{j=1}^{N}(x_i + \mathrm{i}y_i)\ln|z - \rho_i|\,,$$

allowing us to set:

$$\tilde{\Phi} = (g_1 + \mathrm{i}g_2)/2, \quad \tilde{\Psi} = (g_1 - \mathrm{i}g_2)/2, \quad \omega_i = x_i + \mathrm{i}y_i. \qquad \square$$

Supplementary Material
GPU-Accelerated Locally Injective Shape Deformation

ADDITIONAL APPENDICES

B PROOF OF THEOREM 4.2

Proof. Recall the decomposition of Theorem 4.1:

$$f(z) = \tilde{\Phi}(z) + \overline{\tilde{\Psi}(z)} + \sum_{i=1}^{N} \omega_i \ln|z - \rho_i|\,.$$

Direct computation of the Wirtinger derivatives and some manipulation leads to:

$$f_z = \tilde{\Phi}' + \frac{1}{2}\sum_{i=1}^{N} \frac{\omega_i}{z-\rho_i}, \qquad f_{\bar{z}} = \overline{\tilde{\Psi}'} + \frac{1}{2}\sum_{i=1}^{N} \frac{\omega_i}{\overline{z-\rho_i}}, \tag{B.1}$$

where the $(\cdot)'$ notation is the standard complex derivative of holomorphic functions. It is clear from (B.1) that, similarly to the simply-connected case, the Wirtinger derivatives of a harmonic map on a multiply-connected domain are holomorphic and anti-holomorphic respectively. Nevertheless, as opposed to the simply-connected case, f_z and $f_{\bar{z}}$ do not posses an anti-derivative in general. Knowing that f_z is holomorphic, we can apply the argument principle:

$$\oint_{\partial\Omega} \frac{f_z'(w)}{f_z(w)}dw = \oint_{\gamma_0} \frac{f_z'(w)}{f_z(w)}dw + \sum_{j=1}^{N} \oint_{\gamma_i} \frac{f_z'(w)}{f_z(w)}dw = 2\pi \mathrm{i}\mathcal{N}, \tag{B.2}$$

where $\mathcal{N}$ denotes the number of zeros of f_z in Ω. Hence, Equation (4a) is equivalent to the condition $\mathcal{N} = 0$ meaning that $f_z \neq 0$. The rest of the proof is identical to the proof provided in [Chen and Weber 2015, Theorem 4] for simply-connected domains where the subharmonicity of $|f_z|$ and $|f_{\bar{z}}|$ is again leveraged. □

C PROOF OF PROPOSITION 8.1

We start by proving a lemma that will be used to prove the theorem.

Lemma C.1. *Assuming the conditions of Theorem 8.1 hold, the change in the argument of $g(z)$ between the two endpoints of the curve is bounded by π:*

$$|\arg g(v_{i+1}) - \arg g(v_i)| < \pi, \tag{C.1}$$

where arg *is a continuous branch of the argument function.*

Proof. Equation (28) ensures that g does not vanish along the curve, hence, a *continuous* branch of the argument function $\arg(g(z))$ exists. We define the constant $\tilde{t} \in (0,1)$ as follows:

$$\tilde{t} = \frac{|g(v_i)|}{|g(v_i)| + |g(v_{i+1})|}.$$

Since $(1-\tilde{t})$ and $\tilde{t}$ are positive, and $|g(v_i)| + |g(v_{i+1})| > Ll$, we have:

$$\tilde{t}\big(|g(v_i)| + |g(v_{i+1})|\big) > \tilde{t}\,L\,l \tag{C.2}$$

$$(1-\tilde{t})\big(|g(v_i)| + |g(v_{i+1})|\big) > (1-\tilde{t})\,Ll. \tag{C.3}$$

Substituting $\tilde{t}$ in (C.2) and (C.3) leads to:

$$|g(v_i)| > \tilde{t}Ll, \tag{C.4}$$

$$|g(v_{i+1})| > (1-\tilde{t})Ll. \tag{C.5}$$

Assume $t \in [0,\tilde{t}]$ and define $v(t)$ to be a point on the curve such that the arc length between the starting point v_i and $v(t)$ is $t \cdot l$. Since g is L-Lipschitz we have:

$$|g(v_i) - g(v(t))| \leq tLl. \tag{C.6}$$

Dividing both sides of (C.6) by $|g(v_i)|$:

$$\left|1 - \frac{g(v(t))}{g(v_i)}\right| \leq \frac{tLl}{|g(v_i)|} \leq \frac{\tilde{t}Ll}{|g(v_i)|} < \frac{|g(v_i)|}{|g(v_i)|} = 1, \tag{C.7}$$

where the last inequality is due to (C.4). The geometric interpretation of (C.7) is that the complex number $g(v(t))/g(v_i)$ lies strictly inside a unit disk centered at $1 + 0\mathrm{i}$. Denote the difference between the arguments of $g(v(t))$ and $g(v_i)$ as $\Delta\theta(t)$:

$$\Delta\theta(t) = \arg g(v(t)) - \arg g(v_i).$$

For any $t \in [0,\tilde{t}]$, $\Delta\theta(t)$ must belong to one of these ranges:

$$\ldots, \left[-\frac{5\pi}{2}, -\frac{3\pi}{2}\right], \left[-\frac{\pi}{2}, \frac{\pi}{2}\right], \left[\frac{3\pi}{2}, \frac{5\pi}{2}\right], \ldots$$

Furthermore, $\Delta\theta(t)$ is continuous in t, and since $\Delta\theta(0) = 0$ is in $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$, we must have that $-\frac{\pi}{2} < \Delta\theta(\tilde{t}) < \frac{\pi}{2}$.

On the other hand, we can assume $t \in [\tilde{t}, 1]$. Since g is Lipschitz:

$$|g(v_{i+1}) - g(v(t))| \leq (1-t)Ll.$$

Dividing by $|g(v_{i+1})|$:

$$\left|1 - \frac{g(v(t))}{g(v_{i+1})}\right| \leq \frac{(1-t)Ll}{|g(v_{i+1})|} \leq \frac{(1-\tilde{t})Ll}{|g(v_{i+1})|} < \frac{|g(v_{i+1})|}{|g(v_{i+1})|} = 1,$$

where the last inequality is due to (C.5). Using the same geometric reasoning as before, we have:

$$-\frac{\pi}{2} < \arg g(v_{i+1}) - \arg g(v(\tilde{t})) < \frac{\pi}{2}.$$

Finally, since the change in argument between v_i to $v(\tilde{t})$ is smaller than $\pi/2$, and so is the change between $v(\tilde{t})$ to v_{i+1}, the total change in the argument of g between the endpoints is bounded by π. □

Equipped with Lemma C.1 we now prove the theorem.

Proof. A continuous branch of complex log is defined as:

$$\log g(z) = \log g(v_i) + \int_{v_i}^{z} \frac{g'(w)}{g(w)}dw. \tag{C.8}$$

Substituting $z = v_{i+1}$, and rearranging we get:

$$\begin{aligned}\int_{v_i}^{v_{i+1}} \frac{g'(w)}{g(w)}dw &= \log g(v_{i+1}) - \log g(v_i)\\ &= \ln\left|\frac{g(v_{i+1})}{g(v_i)}\right| + \mathrm{i}\,(\arg g(v_{i+1}) - \arg g(v_i))\\ &= \ln\left|\frac{g(v_{i+1})}{g(v_i)}\right| + \mathrm{i}\mathrm{Arg}\frac{g(v_{i+1})}{g(v_i)},\end{aligned}$$

where Arg is the principal branch of arg, and the last equality is due to Lemma C.1. □

D EXPRESSIONS FOR CAUCHY COORDINATES

The expressions for the Cauchy coordinates and its first and second derivatives are taken from appendices B, C, D in [Weber 2010]. Using the notations from Figure 4, the j^{th} Cauchy coordinate is:

$$\tilde{C}_j(z) = \frac{1}{2\pi \mathrm{i}} \left(\frac{B_{j+1}(z)}{A_{j+1}} \mathrm{Log} \frac{B_{j+1}(z)}{B_j(z)} - \frac{B_{j-1}(z)}{A_j} \mathrm{Log} \frac{B_j(z)}{B_{j-1}(z)} \right). \quad \text{(D.1)}$$

The first complex derivative of the j^{th} Cauchy coordinate is:

$$\tilde{D}_j(z) = \tilde{C}_j'(z) = \frac{1}{2\pi \mathrm{i}} \left(\frac{1}{A_{j+1}} \mathrm{Log} \frac{B_j(z)}{B_{j+1}(z)} + \frac{1}{A_j} \mathrm{Log} \frac{B_j(z)}{B_{j-1}(z)} \right). \quad \text{(D.2)}$$

The second complex derivative of the j^{th} Cauchy coordinate is:

$$\tilde{D}_j'(z) = \tilde{C}_j''(z) = \frac{1}{2\pi \mathrm{i}} \left(\frac{z_{j+1} - z_{j-1}}{B_{j-1}(z) B_j(z) B_{j+1}(z)} \right) \quad \text{(D.3)}$$

$$= \frac{1}{2\pi \mathrm{i}} \left(\frac{1}{B_{j-1}(z) B_j(z)} - \frac{1}{B_j(z) B_{j+1}(z)} \right). \quad \text{(D.4)}$$

E FIRST AND SECOND ORDER DIFFERENTIALS OF THE SMOOTH ISOMETRIC ENERGIES

In the following, we will derive the gradient and the Hessian w.r.t the free variables for the energy $\mathrm{E}(|f_z|^2, |f_{\bar{z}}|^2)$ of the harmonic map at a given point $z \in \Omega$. For brevity, we drop the argument z in the derivation below. Throughout the paper we use bold symbols to denote **real** vectors and **real** matrices.

E.1 Gradient of the Isometric Energy

Let $\varphi = (\varphi_1, \varphi_2, \ldots, \varphi_n)^\mathsf{T} \in \mathbb{C}^{n \times 1}$ and $\psi = (\psi_1, \psi_2, \ldots, \psi_n)^\mathsf{T} \in \mathbb{C}^{n \times 1}$ be complex column vectors containing our variables. We convert the complex expressions into real as follows:

$$\boldsymbol{\varphi} = \begin{bmatrix} \mathrm{Re}(\varphi) \\ \mathrm{Im}(\varphi) \end{bmatrix} \in \mathbb{R}^{2n \times 1} \qquad \boldsymbol{\psi} = \begin{bmatrix} \mathrm{Re}(\psi) \\ \mathrm{Im}(\psi) \end{bmatrix} \in \mathbb{R}^{2n \times 1}. \quad \text{(E.1)}$$

By using the multivariable chain rule, and the fact that $f_{\bar{z}}$ is independent of $\boldsymbol{\varphi}$, and f_z is independent of $\boldsymbol{\psi}$, the gradient of E with respect to the $4n$ variables $\boldsymbol{\varphi}, \boldsymbol{\psi}$ is:

$$\nabla \mathrm{E} = \begin{bmatrix} \nabla_{\boldsymbol{\varphi}} \mathrm{E} \\ \nabla_{\boldsymbol{\psi}} \mathrm{E} \end{bmatrix} = \begin{bmatrix} \alpha_1 \nabla_{\boldsymbol{\varphi}} |f_z|^2 + \cancel{\alpha_2 \nabla_{\boldsymbol{\varphi}} |f_{\bar{z}}|^2} \\ \alpha_2 \nabla_{\boldsymbol{\psi}} |f_{\bar{z}}|^2 + \cancel{\alpha_1 \nabla_{\boldsymbol{\psi}} |f_z|^2} \end{bmatrix} \in \mathbb{R}^{4n \times 1} \quad \text{(E.2)}$$

where α_1, α_2 are real scalars:

$$\alpha_1 = \frac{d\mathrm{E}}{d|f_z|^2} \in \mathbb{R}, \qquad \alpha_2 = \frac{d\mathrm{E}}{d|f_{\bar{z}}|^2} \in \mathbb{R}. \quad \text{(E.3)}$$

Table 1 contains the expressions for α_1, α_2 for various isometric energies.

Let $D = (D_1, D_2, \ldots, D_n) \in \mathbb{C}^{1 \times n}$ be a complex row vector, where D_j is defined in (9). By defining the matrix $\boldsymbol{D}$ (note the bold symbol to denote a real matrix):

$$\boldsymbol{D} = \begin{bmatrix} \mathrm{Re}(D) & -\mathrm{Im}(D) \\ \mathrm{Im}(D) & \mathrm{Re}(D) \end{bmatrix} \in \mathbb{R}^{2 \times 2n}, \quad \text{(E.4)}$$

we can express the Wirtinger derivatives as real vectors using a matrix-vector multiplication:

$$\begin{aligned} \boldsymbol{f_z} &= \boldsymbol{D\varphi} = \begin{bmatrix} \mathrm{Re}(f_z) \\ \mathrm{Im}(f_z) \end{bmatrix} \in \mathbb{R}^{2 \times 1} \\ \overline{\boldsymbol{f_{\bar{z}}}} &= \boldsymbol{D\psi} = \begin{bmatrix} \mathrm{Re}\left(\overline{f_{\bar{z}}}\right) \\ \mathrm{Im}\left(\overline{f_{\bar{z}}}\right) \end{bmatrix} \in \mathbb{R}^{2 \times 1} \end{aligned} \quad \text{(E.5)}$$

We get:

$$\begin{aligned} \nabla_{\boldsymbol{\varphi}} |f_z|^2 = \nabla_{\boldsymbol{\varphi}} \|\boldsymbol{f_z}\|^2 = \nabla_{\boldsymbol{\varphi}} \|\boldsymbol{D\varphi}\|^2 = \nabla_{\boldsymbol{\varphi}} \left(\boldsymbol{\varphi}^\mathsf{T} (\boldsymbol{D}^\mathsf{T} \boldsymbol{D}) \boldsymbol{\varphi} \right) \\ = 2(\boldsymbol{D}^\mathsf{T} \boldsymbol{D}) \boldsymbol{\varphi} = 2\boldsymbol{D}^\mathsf{T} \boldsymbol{f_z}, \end{aligned} \quad \text{(E.6)}$$

where $|\cdot|$ denotes the modulus of a complex number while $\|\cdot\|$ denotes the Euclidean vector norm. Similarly we have:

$$\begin{aligned} \nabla_{\boldsymbol{\psi}} |f_{\bar{z}}|^2 = \nabla_{\boldsymbol{\psi}} \left\| \overline{\boldsymbol{f_{\bar{z}}}} \right\|^2 = \nabla_{\boldsymbol{\psi}} \|\boldsymbol{D\psi}\|^2 = \nabla_{\boldsymbol{\psi}} \left(\boldsymbol{\psi}^\mathsf{T} (\boldsymbol{D}^\mathsf{T} \boldsymbol{D}) \boldsymbol{\psi} \right) \\ = 2(\boldsymbol{D}^\mathsf{T} \boldsymbol{D}) \boldsymbol{\psi} = 2\boldsymbol{D}^\mathsf{T} \overline{\boldsymbol{f_{\bar{z}}}}. \end{aligned}$$

By substituting into Equation (E.2), the expression for the gradient of E with respect to the $4n$ real variables is:

$$\nabla \mathrm{E} = 2 \underbrace{\begin{bmatrix} \alpha_1 \boldsymbol{D}^\mathsf{T} \boldsymbol{f_z} \\ \alpha_2 \boldsymbol{D}^\mathsf{T} \overline{\boldsymbol{f_{\bar{z}}}} \end{bmatrix}}_{4n \times 1} = \underbrace{\begin{bmatrix} \boldsymbol{D}^\mathsf{T} & 0 \\ 0 & \boldsymbol{D}^\mathsf{T} \end{bmatrix}}_{4n \times 4} \underbrace{\begin{bmatrix} 2\alpha_1 \boldsymbol{f_z} \\ 2\alpha_2 \overline{\boldsymbol{f_{\bar{z}}}} \end{bmatrix}}_{4 \times 1}. \quad \text{(E.7)}$$

E.2 Hessian of the Isometric Energy

The Hessian of E can be expressed as a block matrix:

$$\nabla^2 \mathrm{E} = \begin{bmatrix} \nabla^2_{\boldsymbol{\varphi}^2} \mathrm{E} & \nabla^2_{\boldsymbol{\varphi}\boldsymbol{\psi}} \mathrm{E} \\ \nabla^2_{\boldsymbol{\psi}\boldsymbol{\varphi}} \mathrm{E} & \nabla^2_{\boldsymbol{\psi}^2} \mathrm{E} \end{bmatrix} \in \mathbb{R}^{4n \times 4n} \quad \text{(E.8)}$$

Let us first define the following real scalars:

$$\beta_1 = \frac{d\alpha_1}{d|f_z|^2}, \quad \beta_2 = \frac{d\alpha_2}{d|f_{\bar{z}}|^2}, \quad \beta_3 = \frac{d\alpha_1}{d|f_{\bar{z}}|^2} = \frac{d\alpha_2}{d|f_z|^2}. \quad \text{(E.9)}$$

Table 1 contains the expressions for $\beta_1, \beta_2, \beta_3$ for various isometric energies. The upper left block of the Hessian is:

$$\nabla^2_{\boldsymbol{\varphi}^2} \mathrm{E} = \nabla_{\boldsymbol{\varphi}} (\nabla_{\boldsymbol{\varphi}} \mathrm{E}) = \nabla_{\boldsymbol{\varphi}} (2\alpha_1 \boldsymbol{D}^\mathsf{T} \boldsymbol{f_z}) = \nabla_{\boldsymbol{\varphi}} (2\alpha_1 \boldsymbol{D}^\mathsf{T} \boldsymbol{D\varphi})$$

By using the product rule we get:

$$\begin{aligned} &= \nabla_{\boldsymbol{\varphi}} (\boldsymbol{D}^\mathsf{T} \boldsymbol{D\varphi})(2\alpha_1) + \nabla_{\boldsymbol{\varphi}} (2\alpha_1) \left(\boldsymbol{\varphi}^\mathsf{T} \boldsymbol{D}^\mathsf{T} \boldsymbol{D} \right) \\ &= 2\boldsymbol{D}^\mathsf{T} \boldsymbol{D} \alpha_1 + 2\nabla_{\boldsymbol{\varphi}} (\alpha_1) \left(\boldsymbol{\varphi}^\mathsf{T} \boldsymbol{D}^\mathsf{T} \boldsymbol{D} \right) \end{aligned}$$

Applying the multivariable chain rule to $\nabla_{\boldsymbol{\varphi}} (\alpha_1)$ above:

$$= 2\alpha_1 \boldsymbol{D}^\mathsf{T} \boldsymbol{D} + 2 \left(\frac{d\alpha_1}{d|f_z|^2} \nabla_{\boldsymbol{\varphi}} (|f_z|^2) + \cancel{\frac{d\alpha_1}{d|f_{\bar{z}}|^2} \nabla_{\boldsymbol{\varphi}} (|f_{\bar{z}}|^2)} \right) \boldsymbol{\varphi}^\mathsf{T} \boldsymbol{D}^\mathsf{T} \boldsymbol{D}$$

Using the definition of β_1 from (E.9), the definition of $\nabla_{\boldsymbol{\varphi}} (|f_z|^2)$ from (E.6), and the definition of $\boldsymbol{f_z}$ from (E.5) we get:

$$\begin{aligned} &= 2\alpha_1 \boldsymbol{D}^\mathsf{T} \boldsymbol{D} + 2 \left(\beta_1 2 \boldsymbol{D}^\mathsf{T} \boldsymbol{f_z} \right) \boldsymbol{f_z}^\mathsf{T} \boldsymbol{D} \\ &= \boldsymbol{D}^\mathsf{T} \left(2\alpha_1 I + 4\beta_1 \boldsymbol{f_z} \boldsymbol{f_z}^\mathsf{T} \right) \boldsymbol{D}, \end{aligned}$$

where I is the 2×2 identity matrix. The other three blocks of the Hessian can be derived similarly:

$$\nabla^2_{\boldsymbol{\psi}^2}\mathrm{E} = \boldsymbol{D}^\mathsf{T}\left(2\alpha_2 I + 4\beta_2\overline{\boldsymbol{f}_{\bar{z}}}\,\overline{\boldsymbol{f}_{\bar{z}}}^\mathsf{T}\right)\boldsymbol{D}$$

$$\nabla^2_{\boldsymbol{\varphi}\boldsymbol{\psi}}\mathrm{E} = \boldsymbol{D}^\mathsf{T}\left(4\beta_3\boldsymbol{f}_z\overline{\boldsymbol{f}_{\bar{z}}}^\mathsf{T}\right)\boldsymbol{D} = (\nabla^2_{\boldsymbol{\psi}\boldsymbol{\varphi}}\mathrm{E})^\mathsf{T}$$

Finally, the Hessian of E is:

$$\nabla^2\mathrm{E} = \tag{E.10}$$

$$\underbrace{\begin{bmatrix}\boldsymbol{D}^\mathsf{T} & 0\\ 0 & \boldsymbol{D}^\mathsf{T}\end{bmatrix}}_{4n\times 4}\underbrace{\begin{bmatrix}2\alpha_1 I + 4\beta_1\boldsymbol{f}_z\boldsymbol{f}_z^\mathsf{T} & 4\beta_3\boldsymbol{f}_z\overline{\boldsymbol{f}_{\bar{z}}}^\mathsf{T}\\ 4\beta_3\overline{\boldsymbol{f}_{\bar{z}}}\boldsymbol{f}_z^\mathsf{T} & 2\alpha_2 I + 4\beta_2\overline{\boldsymbol{f}_{\bar{z}}}\,\overline{\boldsymbol{f}_{\bar{z}}}^\mathsf{T}\end{bmatrix}}_{4\times 4}\underbrace{\begin{bmatrix}\boldsymbol{D} & 0\\ 0 & \boldsymbol{D}\end{bmatrix}}_{4\times 4n}$$

F GRADIENT AND HESSIAN OF THE P2P ENERGY

The P2P energy at a *single* handle can be expressed using complex numbers:

$$\mathrm{E}_{\mathrm{p2p}}(p_i) = \frac{1}{2}|f(p_i) - q_i|^2 = \frac{1}{2}|C(p_i)\varphi + \overline{C(p_i)\psi} - q_i|^2, \tag{F.1}$$

where $C(p_i) = (C_1(p_i), C_2(p_i), \ldots, C_n(p_i)) \in \mathbb{C}^{1\times n}$ is a complex row vector, where C_j is defined in (8), and $\varphi, \psi \in \mathbb{C}^{n\times 1}$. The complete energy is given by the sum:

$$\mathrm{E}^f_{\mathrm{p2p}} = \sum_{i=1}^{|\mathcal{P}|}\mathrm{E}_{\mathrm{p2p}}(p_i). \tag{F.2}$$

Equation (F.1) can be expressed using real numbers:

$$\mathrm{E}_{\mathrm{p2p}}(p_i) = \frac{1}{2}\left\|\boldsymbol{Q}(p_i)\begin{bmatrix}\boldsymbol{\varphi}\\ \boldsymbol{\psi}\end{bmatrix} - \begin{bmatrix}\mathrm{Re}(q_i)\\ \mathrm{Im}(q_i)\end{bmatrix}\right\|^2,$$

where $\boldsymbol{Q}(p_i)$ is a real matrix defined as:

$$\begin{bmatrix}\mathrm{Re}\,(C(p_i)) & -\mathrm{Im}\,(C(p_i)) & \mathrm{Re}\,(C(p_i)) & -\mathrm{Im}\,(C(p_i))\\ \mathrm{Im}\,(C(p_i)) & \mathrm{Re}\,(C(p_i)) & -\mathrm{Im}\,(C(p_i)) & -\mathrm{Re}\,(C(p_i))\end{bmatrix} \in \mathbb{R}^{2\times 4n}$$

$\mathrm{E}_{\mathrm{p2p}}(p_i)$ is a quadratic form with *constant* PSD Hessian matrix:

$$\nabla^2\mathrm{E}_{\mathrm{p2p}}(p_i) = \boldsymbol{Q}^\mathsf{T}(p_i)\boldsymbol{Q}(p_i) \quad \in \mathbb{R}^{4n\times 4n} \tag{F.3}$$

The gradient of the quadratic form is:

$$\nabla\mathrm{E}_{\mathrm{p2p}}(p_i) = \boldsymbol{Q}^\mathsf{T}(p_i)\boldsymbol{Q}(p_i)\begin{bmatrix}\boldsymbol{\varphi}\\ \boldsymbol{\psi}\end{bmatrix} - \boldsymbol{Q}^\mathsf{T}(p_i)\begin{bmatrix}\mathrm{Re}\,(q_i)\\ \mathrm{Im}\,(q_i)\end{bmatrix}. \tag{F.4}$$

In order to sum the gradients and the Hessians over all the P2P handles $\mathcal{P}$, we construct a matrix $\mathfrak{Q}$ by stacking $\boldsymbol{Q}(p_i)$:

$$\mathfrak{Q} = \begin{bmatrix}\boldsymbol{Q}(p_1)\\ \boldsymbol{Q}(p_2)\\ \vdots\\ \boldsymbol{Q}(p_{|\mathcal{P}|})\end{bmatrix} \in \mathbb{R}^{2|\mathcal{P}|\times 4n}. \tag{F.5}$$

Finally, the full Hessian and full gradient are given by:

$$\nabla^2\mathrm{E}^f_{\mathrm{p2p}} = \mathfrak{Q}^\mathsf{T}\mathfrak{Q}, \tag{F.6}$$

$$\nabla\mathrm{E}^f_{\mathrm{p2p}} = \mathfrak{Q}^\mathsf{T}\left(\mathfrak{Q}\begin{bmatrix}\boldsymbol{\varphi}\\ \boldsymbol{\psi}\end{bmatrix} - \mathfrak{q}\right), \tag{F.7}$$

where $\mathfrak{q}$ is a real column vector stacking the target positions:

$$\mathfrak{q} = \left[\mathrm{Re}\,(q_1), \mathrm{Im}\,(q_1), \ldots, \mathrm{Re}\,\left(q_{|\mathcal{P}|}\right), \mathrm{Im}\,\left(q_{|\mathcal{P}|}\right)\right]^\mathsf{T} \in \mathbb{R}^{2|\mathcal{P}|\times 1} \tag{F.8}$$

G GPU IMPLEMENTATION

We designed our GPU implementation in a way that utilizes standard libraries as much as possible. The implementation mostly involves only dense linear algebra operations, while the parts that require coding of a specialized kernel are relatively simple, embarrassingly parallel operations. We have utilized the cuBLAS [cuBLAS 2017], and cuSolverDN [cuSOLVER 2017] (BLAS and LAPACK implementations) dense linear algebra libraries that are freely distributed with NVIDIA's CUDA toolkit with double precision accuracy. This not only simplifies the implementation, but also ensures that the code will be optimized for different GPU architectures and will run smoothly on any NVIDIA device. Moreover, as graphics hardware constantly evolves and these libraries get optimized further, the speed of such implementation is expected to automatically improve over time.

Matrix and vector notations

The Wirtinger derivatives at a single point can be expressed concisely in *vector* form:

$$f_z(z) = D(z)\varphi, \qquad \overline{f_{\bar{z}}} = D(z)\psi, \tag{G.1}$$

where $D(z) = (D_1(z), D_2(z), \ldots, D_n(z)) \in \mathbb{C}^{1\times n}$ is a complex row vector, while $\varphi = (\varphi_1, \varphi_2, \ldots, \varphi_n)^\mathsf{T}$ and $\psi = (\psi_1, \psi_2, \ldots, \psi_n)^\mathsf{T}$ are complex column vectors.

We will need to evaluate the complex-valued function f and its differentials at a large collection of points inside the domain. It would be convenient to express these in matrix notations. For example, assume that we want to evaluate the map f on a set of points $\mathcal{V} \in \Omega$ for the sake of visualization.

We define the complex matrix $\mathfrak{C}(\mathcal{V}) \in \mathbb{C}^{|\mathcal{V}|\times n}$. The i^{th} row of $\mathfrak{C}$ corresponds to the point $v_i \in \mathcal{V}$ such that $\mathfrak{C}_{i,j} = C_j(v_i)$. Similarly, we use the complex matrix $\mathfrak{D}(\mathcal{V}) \in \mathbb{C}^{|\mathcal{V}|\times n}$ such that $\mathfrak{D}_{i,j} = D_j(v_i)$. Using these notations, we can evaluate the map f at the points of the set $\mathcal{V}$ by computing the matrix product:

$$\underbrace{\begin{bmatrix}\Phi & \Psi\end{bmatrix}}_{|\mathcal{V}|\times 2} = \mathfrak{C}(\mathcal{V})\underbrace{\begin{bmatrix}\varphi & \psi\end{bmatrix}}_{n\times 2}, \tag{G.2}$$

followed by the computation of the vector sum $\Phi + \overline{\Psi}$. Similarly, one can evaluate the Wirtinger derivatives on the set $\mathcal{V}$ with the product:

$$\underbrace{\begin{bmatrix}\mathfrak{f}_z & \overline{\mathfrak{f}_{\bar{z}}}\end{bmatrix}}_{|\mathcal{V}|\times 2} = \mathfrak{D}(\mathcal{V})\underbrace{\begin{bmatrix}\varphi & \psi\end{bmatrix}}_{n\times 2}. \tag{G.3}$$

Preprocessing

In preprocessing, we triangulate the polygon P with a dense triangulation whose vertices form the set $\mathcal{V}$. The triangulation is used for visualization purposes only. We simply use hardware texture mapping to render the final deformed image. The rendering can be performed efficiently on the GPU with extremely high mesh-and-texture resolutions without affecting the runtime of the solver. We compute and store on the device's memory the following matrices:

$$\begin{aligned}&\mathfrak{C}(\mathcal{V}) \in \mathbb{C}^{|\mathcal{V}|\times n} \qquad \mathfrak{C}(\mathcal{P}) \in \mathbb{C}^{|\mathcal{P}|\times n},\\ &\mathfrak{D}(\mathcal{G}) \in \mathbb{C}^{|\mathcal{G}|\times n} \qquad \mathfrak{D}(\mathcal{H}) \in \mathbb{C}^{|\mathcal{H}|\times n},\end{aligned} \tag{G.4}$$

where $\mathcal{V}$ contains the vertices of the triangulation used for texture mapping, $\mathcal{P}$ is the set of positional constraints, $\mathcal{G}$ is a dense set of samples lying on the polygon P which is used for energy and gradient evaluation, and $\mathcal{H}$ is a subset of $\mathcal{G}$ used to evaluate the Hessian of the energy. Once the complex matrix $\mathfrak{D}(\mathcal{H})$ is constructed, we convert it to real form $\mathfrak{D} \in \mathbb{R}^{2|\mathcal{H}|\times 2n}$ (note the bold symbol to denote a real matrix) as follows:

$$\mathfrak{D} = \begin{bmatrix} \mathrm{Re}\,(\mathfrak{D}_1) & -\mathrm{Im}\,(\mathfrak{D}_1) \\ \mathrm{Im}\,(\mathfrak{D}_1) & \mathrm{Re}\,(\mathfrak{D}_1) \\ \mathrm{Re}\,(\mathfrak{D}_2) & -\mathrm{Im}\,(\mathfrak{D}_2) \\ \mathrm{Im}\,(\mathfrak{D}_2) & \mathrm{Re}\,(\mathfrak{D}_2) \\ \vdots & \vdots \\ \mathrm{Re}\,(\mathfrak{D}_{|\mathcal{H}|}) & -\mathrm{Im}\,(\mathfrak{D}_{|\mathcal{H}|}) \\ \mathrm{Im}\,(\mathfrak{D}_{|\mathcal{H}|}) & \mathrm{Re}\,(\mathfrak{D}_{|\mathcal{H}|}) \end{bmatrix} \in \mathbb{R}^{2|\mathcal{H}|\times 2n}, \tag{G.5}$$

where the subscripts indicates row numbers such that $\mathfrak{D}_i$ is the i^{th} row of the complex matrix $\mathfrak{D}$. We also store another closely related matrix with the same size which we denote as $\hat{\mathfrak{D}}$ and is obtained from $\mathfrak{D}$ by swapping each two pair of rows. That is, the 1^{st} row is swapped with the 2^{nd} one, the 3^{rd} with the 4^{th} and so on.

Next we compute in parallel a matrix with the coefficients $L_{i,j} = \frac{1}{2\pi d_i^2(z_j)}$ and $L^h_{i,j} = \frac{2}{d_i^3(\rho_{j-m})}$ (Section 8). These will be used in runtime to compute the Lipschitz constants of the Wirtinger derivatives. Unlike the Lipschitz constants, these depend only on the cage $\hat{\mathrm{P}}$ and the set $\mathcal{G}$, hence, can be precomputed. A simple formula for the distance $d_i(p)$ from a point p to the i^{th} segment is given in Appendix C of [Chen and Weber 2015].

The last step of the preprocessing computes the matrix $\mathfrak{Q}$ (see Equation (F.5) in Appendix F) and $\nabla^2 \mathrm{E}^f_{\mathrm{p2p}} = \mathfrak{Q}^{\mathsf{T}}\mathfrak{Q}$, which is a constant matrix since $\mathrm{E}^f_{\mathrm{p2p}}$ is a quadratic form, hence can be computed and stored once.

Next, we describe the computations that are needed to be carried out on each Newton iteration. The evaluation of the map f, its Wirtinger derivatives, energy, and gradient are computed using a matrix-matrix multiplication, where the right term in the product is a "thin" matrix (having many rows but very small number of columns). Such operation is known to be bandwidth limited, meaning that the computation time is dominated by the memory bandwidth of the GPU device. Hence, when performing the product we should strive to minimize the amount of access to the global memory of the device. To this end, we perform the computation using complex numbers and convert the result back to real numbers when necessary. During runtime, we first evaluate the Wirtinger derivatives at the samples of $\mathcal{G}$ using the product:

$$\underbrace{\begin{bmatrix} \mathfrak{f}_z & \overline{\mathfrak{f}_{\bar{z}}} \end{bmatrix}}_{|\mathcal{G}|\times 2} = \mathfrak{D}(\mathcal{G}) \underbrace{\begin{bmatrix} \varphi & \psi \end{bmatrix}}_{n\times 2}. \tag{G.6}$$

Gradient

Next, the following is being computed in a single dedicated kernel that accepts the matrix $[\mathfrak{f}_z\ \overline{\mathfrak{f}_{\bar{z}}}]$ as input. Each thread handles a row of $[\mathfrak{f}_z\ \overline{\mathfrak{f}_{\bar{z}}}]$ (which contains only two complex numbers). We compute the two real scalars $|f_z|^2$ and $|f_{\bar{z}}|^2$, and use it to compute the real scalars α_1, α_2 using (E.3). Finally, the kernel multiplies α_1, α_2 element-wise by the corresponding matrix entries, and store the result as a matrix $[\boldsymbol{\alpha_1}\mathfrak{f}_z\ \boldsymbol{\alpha_2}\overline{\mathfrak{f}_{\bar{z}}}] \in \mathbb{R}^{|\mathcal{G}|\times 2}$. We note that these are embarrassingly parallel operations, which are straightforward to implement in a kernel, where no synchronization is needed and the memory access pattern is perfectly regular.

The next step computes the following complex matrix-matrix product:

$$\underbrace{\begin{bmatrix} \mathfrak{g}_\varphi & \mathfrak{g}_\psi \end{bmatrix}}_{n\times 2} = \frac{2}{|\mathcal{G}|}\mathfrak{D}^{\mathsf{H}}(\mathcal{G}) \underbrace{[\boldsymbol{\alpha_1}\mathfrak{f}_z\ \boldsymbol{\alpha_2}\overline{\mathfrak{f}_{\bar{z}}}]}_{|\mathcal{G}|\times 2}, \tag{G.7}$$

where $(\cdot)^{\mathsf{H}}$ is the conjugate transpose (hermit) operator. We compute the multiplication in (G.7) by invoking the `gemm` cuBLAS procedure. Based on the summation in (32) and the expression for the pointwise gradient (E.7), it is easy to verify that the (real-valued) expression of the gradient of the energy is given by:

$$\nabla \mathrm{E}^f = \begin{bmatrix} \mathrm{Re}(\mathfrak{g}_\varphi) \\ \mathrm{Im}(\mathfrak{g}_\varphi) \\ \mathrm{Re}(\mathfrak{g}_\psi) \\ -\mathrm{Im}(\mathfrak{g}_\psi) \end{bmatrix} \in \mathbb{R}^{4n\times 1}. \tag{G.8}$$

The gradient of the P2P energy also needs to be computed (see Appendix F for the derivation):

$$\nabla \mathrm{E}^f_{\mathrm{p2p}} = \mathfrak{Q}^{\mathsf{T}}\left(\mathfrak{Q}\begin{bmatrix} \boldsymbol{\varphi} \\ \boldsymbol{\psi} \end{bmatrix} - \mathfrak{q}\right). \tag{G.9}$$

Note that $\mathfrak{Q} \in \mathbb{R}^{2|\mathcal{P}|\times 4n}$ was already computed in preprocessing, and the vector $\mathfrak{q} \in \mathbb{R}^{2|\mathcal{P}|\times 1}$ contains the target P2P positions (Equation (F.8) in Appendix F).

Hessian

We turn into the computation of the Hessian which is the most involved part of the implementation. First, we compute in a dedicated kernel the 4×4 matrix $\boldsymbol{K}_i^+$ that corresponds to the i^{th} sample in $\mathcal{H}$ as described in Section 7. We compute the real scalars $\alpha_1, \alpha_2, \beta_1, \beta_2, \beta_3$ (Table 1). Then, based on the energy type, we check for negative eigenvalues and decide whether to compute $\boldsymbol{K}_i$ (Equation (19)) or its modified version $\boldsymbol{K}_i^+$. In the case of $\mathrm{E}_{\mathrm{iso}}$, the matrix $\boldsymbol{K}_i^+$ can be computed directly using (25). Otherwise, (23) and (24) are used. The entries of all these 4×4 matrices are stored in a matrix $\mathfrak{K} \in \mathbb{R}^{16\times|\mathcal{H}|}$ where in each column we stack the 16 values of the corresponding matrix $\boldsymbol{K}_i^+$.

Unlike the computation of the gradient, which is bandwidth limited, the assembly of the Hessian is a ALU-bound operation, meaning that its runtime is governed by the amount of processing units (and their clock speed) rather than the speed of the global memory. This means that we can expect greater speedup compared to a CPU implementation. Yet, the amount of arithmetic operations involved is vast. Our strategy to use coarser sampling (Equation (33)) greatly reduces the Hessian assembly time. We now describe an efficient way to implement (33).

The assembly of the Hessian is done separately on each of its four $2n\times 2n$ blocks (Equation (E.8)). Below we explain how to compute the $\nabla^2_{\boldsymbol{\varphi}^2}\mathrm{E}^f$ block. The blocks for $\nabla^2_{\boldsymbol{\psi}^2}\mathrm{E}^f$ and $\nabla^2_{\boldsymbol{\varphi}\boldsymbol{\psi}}\mathrm{E}^f$ can be computed analogously. These three blocks are copied into the corresponding

position in the $4n\times 4n$ full Hessian matrix. Due to symmetry, there is no need to explicitly compute $\nabla^2_{\boldsymbol{\psi}\boldsymbol{\varphi}}\mathrm{E}^f$ as it is given by $\nabla^2_{\boldsymbol{\varphi}\boldsymbol{\psi}}\mathrm{E}^{f\,\mathsf{T}}$.

Let $\left(\begin{smallmatrix} a_i & b_i \\ c_i & d_i \end{smallmatrix}\right)$ be the 2×2 upper-left block of the matrix $\boldsymbol{K}_i^+$. Note that $\mathfrak{K}$ (computed earlier) contains the corresponding values for all samples in its rows. The entries of $\nabla^2_{\boldsymbol{\varphi}^2}\mathrm{E}^f$ can be computed using the following formula:

$$\nabla^2_{\boldsymbol{\varphi}^2}\mathrm{E}^f = \frac{1}{|\mathcal{H}|}\mathfrak{D}^{\mathsf{T}}\left[\mathfrak{K}\mathfrak{D} + \mathfrak{B}\hat{\mathfrak{D}}\right], \tag{G.10}$$

where $\mathfrak{K} \in \mathbb{R}^{2|\mathcal{H}|\times 2|\mathcal{H}|}$ is a diagonal matrix with diagonal entries $(a_1, d_1, a_2, d_2, \ldots, a_{|\mathcal{H}|}, d_{|\mathcal{H}|})$ and $\mathfrak{B} \in \mathbb{R}^{2|\mathcal{H}|\times 2|\mathcal{H}|}$ is a diagonal matrix with diagonal entries $(b_1, c_1, b_2, c_2, \ldots, b_{|\mathcal{H}|}, c_{|\mathcal{H}|})$. The advantage of performing the computation in such a way is the ability to use the highly-optimized cuBLAS library [cuBLAS 2017]. The products $\mathfrak{K}\mathfrak{D}$ and $\mathfrak{B}\hat{\mathfrak{D}}$ in (G.10) are computed using a diagonal matrix-matrix multiplication procedure `dgmm` followed by a matrix summation (`geam`) which result in a matrix $\mathfrak{M} = \mathfrak{K}\mathfrak{D} + \mathfrak{B}\hat{\mathfrak{D}}$. Finally, the result is obtained through a matrix-matrix product $\mathfrak{D}^{\mathsf{T}}\mathfrak{M}$ using `gemm`. This product dominates the runtime of (G.10).

Then, the gradient and the Hessian of $\mathrm{E}^f_{\mathrm{p2p}}$ are added to those of E^f (with the appropriate user-defined weight λ) to form the gradient and the Hessian of the full energy E^f (Equation (20)). We then eliminate the $10N + 2$ redundant real variables that correspond to the dimension of the null space of the harmonic subspace as explained in Section 5. This is done by removing the corresponding rows and columns from the gradient and the Hessian. After that, the Newton step is computed. We use the freely available cuSolverDN library [cuSOLVER 2017] to solve the linear system (1) using the dense GPU-accelerated direct Cholesky factorization.

Line search

The solution of the linear system provides the search direction for the Newton iteration. However, we do not take this step immediately. To ensure convergence to a local minimum which corresponds to a locally injective map, a backtracking line search is performed. This requires multiple evaluations of the energy and the local injectivity validation at various step sizes, starting from $t = 1$ and dividing it by 2 at each step until the resulted map is accepted. For a map to be accepted, we verify first that the energy sufficiently decreases (Algorithm 1, Line 17) and only then perform the local injectivity validation (Algorithm 1, Line 18). Both steps require evaluation of the Wirtinger derivatives f_z^t and $\overline{f_{\bar{z}}}^t$ at all samples, which varies depending on t. In general, evaluating these boils down to the same matrix product we used in (G.6) for the computation of the gradient. Evaluating it many times can easily become the algorithm's bottleneck. Luckily, since the derivatives are linear in the variables, we can achieve this with a single matrix product. The derivatives for step t are given by:

$$\begin{aligned}\begin{bmatrix} \mathfrak{f}_z^t & \overline{\mathfrak{f}_{\bar{z}}}^t \end{bmatrix} &= \mathfrak{D}\begin{bmatrix}\varphi^t & \psi^t\end{bmatrix} = \mathfrak{D}\begin{bmatrix}\varphi_k + t\Delta\varphi_k & \psi_k + t\Delta\psi_k\end{bmatrix} \\ &= \mathfrak{D}\begin{bmatrix}\varphi_k & \psi_k\end{bmatrix} + t\mathfrak{D}\begin{bmatrix}\Delta\varphi_k & \Delta\psi_k\end{bmatrix},\end{aligned} \tag{G.11}$$

where φ_k, ψ_k are the current state of the (complex) variables and $\begin{bmatrix}\Delta\varphi_k & \Delta\psi_k\end{bmatrix}$ is the Newton search direction. Note that the term on the left was already computed in (G.6). Hence, we only need to compute the product $\mathfrak{D}\begin{bmatrix}\Delta\varphi_k & \Delta\psi_k\end{bmatrix}$ once, and for each step t, (G.11) boils down to a scalar-matrix multiplication followed by a (thin) matrix summation. Both operations are very efficient. The energy is then evaluated in a dedicated kernel that accepts the matrix $\begin{bmatrix} \mathfrak{f}_z^t & \overline{\mathfrak{f}_{\bar{z}}}^t \end{bmatrix}$ as input. Each thread handles a row that contains two complex numbers: $f_z^t, \overline{f_{\bar{z}}}^t$, evaluate two real scalars: $\left|f_z^t\right|^2, |\overline{f_{\bar{z}}}^t|^2$, and use it to compute the scalar E using Table 1. Finally, the kernel performs a parallel reduction summation step to obtain E^f (Equation (31)). In order to avoid CPU-GPU communication overhead, we carried out this step on the GPU by using the dynamic parallelism feature which allows kernels to invoke other kernels, therefore allows the GPU to execute the line search autonomously.

We terminate the iterations when $t\,\|\begin{bmatrix}\Delta\varphi_k & \Delta\psi_k\end{bmatrix}\|$ is smaller than $\epsilon = 1\mathrm{e}{-12}$ (Algorithm 1, Line 9) and observe that the algorithm always converges. In contrast, for some inputs, the SLIM and AQP methods struggle to converge to machine precision error even after a thousand iterations. For fairness of comparison, we run all methods till we couldn't spot a change in the rendered image and report this number of iterations instead. Upon termination, the map itself is evaluated. This is done as explained in Appendix G (below Equation (G.2)). Since this product is also performed on the GPU, the result can be used by the GPU directly for rendering, avoiding expensive CPU-GPU transfer times.

H DERIVATION OF TIGHT LIPSCHITZ CONSTANTS

The idea behind our tighter constants is to first compute $L_{f_z'}$ and then use it to compute L_{f_z} as an upper bound on $\left|f_z'\right|$:

$$\left\|f_z'\right\|_\infty \le \frac{|f_z'(v_i)| + |f_z'(v_{i+1})|}{2} + \frac{L_{f_z'}l}{2} = L_{f_z}. \tag{H.1}$$

In order to get $L_{f_z'} \ge \left\|f_z''\right\|_\infty$, we start by expressing f_z' based on Equation (9) and the formula for the second derivative of the Cauchy coordinates (Equation (D.4)):

$$\begin{aligned} f_z'(z) &= \sum_{j=1}^{m}\frac{1}{2\pi\mathrm{i}}\left(\frac{1}{B_{j-1}B_j} - \frac{1}{B_jB_{j+1}}\right)\varphi_j - \sum_{j=m+1}^{n}\frac{\varphi_j}{(z-\rho_{j-m})^2} \\ &= \frac{1}{2\pi\mathrm{i}}\sum_{j=1}^{m}\left(\frac{\varphi_j - \varphi_{j-1}}{B_{j-1}B_j}\right) - \sum_{j=m+1}^{n}\frac{\varphi_j}{(z-\rho_{j-m})^2} \\ &= \frac{1}{2\pi\mathrm{i}}\sum_{j=1}^{m}\frac{\varphi_j - \varphi_{j-1}}{z_j - z_{j-1}}\left(\frac{1}{B_j} - \frac{1}{B_{j-1}}\right) - \sum_{j=m+1}^{n}\frac{\varphi_j}{(z-\rho_{j-m})^2} \\ &= \frac{1}{2\pi\mathrm{i}}\sum_{j=1}^{m}s_j\left(\frac{1}{B_j} - \frac{1}{B_{j-1}}\right) - \sum_{j=m+1}^{n}\frac{\varphi_j}{(z-\rho_{j-m})^2} \\ &= \frac{1}{2\pi\mathrm{i}}\sum_{j=1}^{m}\frac{s_j - s_{j+1}}{B_j} - \sum_{j=m+1}^{n}\frac{\varphi_j}{(z-\rho_{j-m})^2}, \end{aligned}$$

where $s_j = \frac{\varphi_j - \varphi_{j-1}}{z_j - z_{j-1}}$. Next, we differentiate $f_z'(z)$ with respect to z:

$$f_z''(z) = \frac{1}{2\pi\mathrm{i}}\sum_{j=1}^{m}\frac{s_j - s_{j+1}}{B_j^2(z)} + \sum_{j=m+1}^{n}\frac{2\varphi_j}{(z-\rho_{j-m})^3}.$$

We then compute an upper bound:

$$\left\|f_z''\right\|_\infty \le \frac{1}{2\pi}\sum_{j=1}^{m}\frac{|s_j - s_{j+1}|}{d^2(z_j)} + \sum_{j=m+1}^{n}\frac{2|\varphi_j|}{d^3(\rho_{j-m})} = L_{f_z'}$$

or more compactly:

$$L_{f_z'} = \sum_{j=1}^{m} L_j|s_j - s_{j+1}| + \sum_{j=m+1}^{n} L_j^h|\varphi_j|,$$

where $L_j = \frac{1}{2\pi d^2(z_j)}$ and $L_j^h = \frac{2}{d^3(\rho_{j-m})}$ can be computed in preprocessing. Finally, we can substitute the above expression in (H.1):

$$L_{f_z} = \frac{|f_z'(v_i)| + |f_z'(v_{i+1})|}{2} + \frac{l}{2}\left(\sum_{j=1}^{m} L_j|s_j - s_{j+1}| + \sum_{j=m+1}^{n} L_j^h|\varphi_j|\right).$$

I THE NULL SPACE OF THE MODIFIED HESSIAN

Let $f(z)$ be a harmonic map parameterized by $\tilde{\varphi}$ and $\tilde{\psi}$, $f(z) = C(z)\tilde{\varphi} + \overline{C(z)\tilde{\psi}}$. Denote by $\boldsymbol{H}(z) = \nabla^2 \mathrm{E}^+_{\mathrm{iso}}(z)$, the modified Hessian of the Symmetric Dirichlet isometric energy of f at a single point z. The Hessian of the total isometric energy is:

$$\boldsymbol{H}^f = \oint_{\partial\Omega} \boldsymbol{H}(w)ds \quad \in \mathbb{R}^{4n\times 4n}.$$

Proposition I.1. *For any vector* $\boldsymbol{v} = \begin{bmatrix}\boldsymbol{\varphi}\\ \boldsymbol{\psi}\end{bmatrix} \in \mathbb{R}^{4n\times 1}$ *in the kernel of* $\boldsymbol{H}^f$*, there exists* $s \in \mathbb{R}$ *such that* $\forall w \in \partial\Omega$,

$$D(w)\begin{bmatrix}\varphi - s\tilde{\varphi}\mathrm{i} & \psi\end{bmatrix} = 0,$$

where the vectors φ, ψ *are related to* $\boldsymbol{\varphi}, \boldsymbol{\psi}$ *via Equation* (E.1).

Proof. By definition, $\boldsymbol{H}^f\boldsymbol{v} = \oint_{\partial\Omega}\boldsymbol{H}(w)\boldsymbol{v}\,ds = 0$.

With a slight abuse of notations, in the following we will use the real vectors $\boldsymbol{\varphi}, \boldsymbol{\psi}$ and the complex vectors φ, ψ interchangeably. Since $\boldsymbol{H}(w)$ is PSD, we have:

$$\boldsymbol{H}^f\boldsymbol{v} = 0 \Rightarrow \boldsymbol{v}^\mathsf{T}\boldsymbol{H}^f\boldsymbol{v} = 0 \Rightarrow \forall w \in \partial\Omega,\ \boldsymbol{v}^\mathsf{T}\boldsymbol{H}(w)\boldsymbol{v} = 0.$$

Recall that $\boldsymbol{H}(w) = \boldsymbol{B}^\mathsf{T}\boldsymbol{K}^+(w)\boldsymbol{B}$ where $\boldsymbol{B} = \begin{bmatrix}\boldsymbol{D}(w) & 0\\ 0 & \boldsymbol{D}(w)\end{bmatrix} \in \mathbb{R}^{4\times 4n}$.

Denote $\boldsymbol{u}(w) = \boldsymbol{B}\boldsymbol{v} = \begin{bmatrix}\boldsymbol{D}(w)\boldsymbol{\varphi}\\ \boldsymbol{D}(w)\boldsymbol{\psi}\end{bmatrix} \in \mathbb{R}^{4\times 1}$, then:

$$\boldsymbol{v}^\mathsf{T}\boldsymbol{H}(w)\boldsymbol{v} = \boldsymbol{v}^\mathsf{T}\boldsymbol{B}^\mathsf{T}\boldsymbol{K}^+(w)\boldsymbol{B}\boldsymbol{v} = \boldsymbol{u}^\mathsf{T}\boldsymbol{K}^+(w)\boldsymbol{u} = 0 \Rightarrow \boldsymbol{K}^+(w)\boldsymbol{u} = 0.$$

Note that there are two possibilities for $\boldsymbol{K}(w)$:

(1) $\boldsymbol{K}(w)$ is positive definite, hence $\boldsymbol{K}^+(w)$ is also positive definite and has an empty null space, therefore:

$$\boldsymbol{u}(w) = 0 \ \Rightarrow\ \boldsymbol{D}(w)\boldsymbol{\varphi} = 0,\ \boldsymbol{D}(w)\boldsymbol{\psi} = 0.$$

(2) $\boldsymbol{K}(w)$ is not positive definite, then by construction, $\boldsymbol{K}^+(w)$ has a one dimensional null space spanned by:

$$\boldsymbol{u} = (-\mathrm{Im}\,(f_z), \mathrm{Re}\,(f_z), 0, 0).$$

$(-\mathrm{Im}\,(f_z), \mathrm{Re}\,(f_z))$ is simply the complex number $f_z\mathrm{i}$ in vector form. Therefore:

$$\boldsymbol{u} \parallel \boldsymbol{v} \Rightarrow \begin{cases} D(w)\varphi = s(w)f_z(w)\mathrm{i} \\ D(w)\psi = 0 \end{cases}$$

where $s(w) \in \mathbb{R}$. Notice that for simplicity, we write these two equalities in complex numbers.

In either case, the following holds,

$$D(w)\varphi = s(w)f_z(w)\mathrm{i}, \tag{I.1}$$

$$D(w)\psi = 0, \tag{I.2}$$

where $s(w) \in \mathbb{R}$ and $s(w) = 0$ if $\boldsymbol{K}(w)$ is positive definite. Note that $D(z)\varphi$ is a holomorphic function in z, as a consequence, $s(z)$ is also holomorphic and hence must be constant, $s(z) \equiv s$. Then Equation (I.1) implies that $D(w)(\varphi - s\tilde{\varphi}\mathrm{i}) = 0$. Together with Equation (I.2), we have the following characterization for φ and ψ,

$$D(w)\begin{bmatrix}\varphi - s\tilde{\varphi}\mathrm{i} & \psi\end{bmatrix} = 0.$$

□

Ignoring the redundancies in the harmonic map representation (see Section 5), $\mathfrak{D}(\partial\Omega)$ has a null space that includes only the constant vectors, which corresponds to the constant map $f(z) = a + b\mathrm{i}$. Therefore, the null space of $\boldsymbol{H}^f$ is spanned by $\varphi = \psi = \vec{1}$, $\varphi = \psi = \vec{\mathrm{i}}$, and additionally $\varphi = \tilde{\varphi}\mathrm{i}, \psi = \vec{0}$ in the case that $\boldsymbol{K}^+(w)$ is singular for every $w \in \partial\Omega$, hence the co-rank of $\boldsymbol{H}^f$ is either 2 or 3. A simple example of map with singular $\boldsymbol{K}^+(w), \forall w \in \partial\Omega$ is the rigid motion: $|f_z| = 1, f_{\bar{z}} = 0$.

Note that similar analysis is applicable to the exponential symmetric Dirichlet energy $\mathrm{E}_{\mathrm{exp}}$.

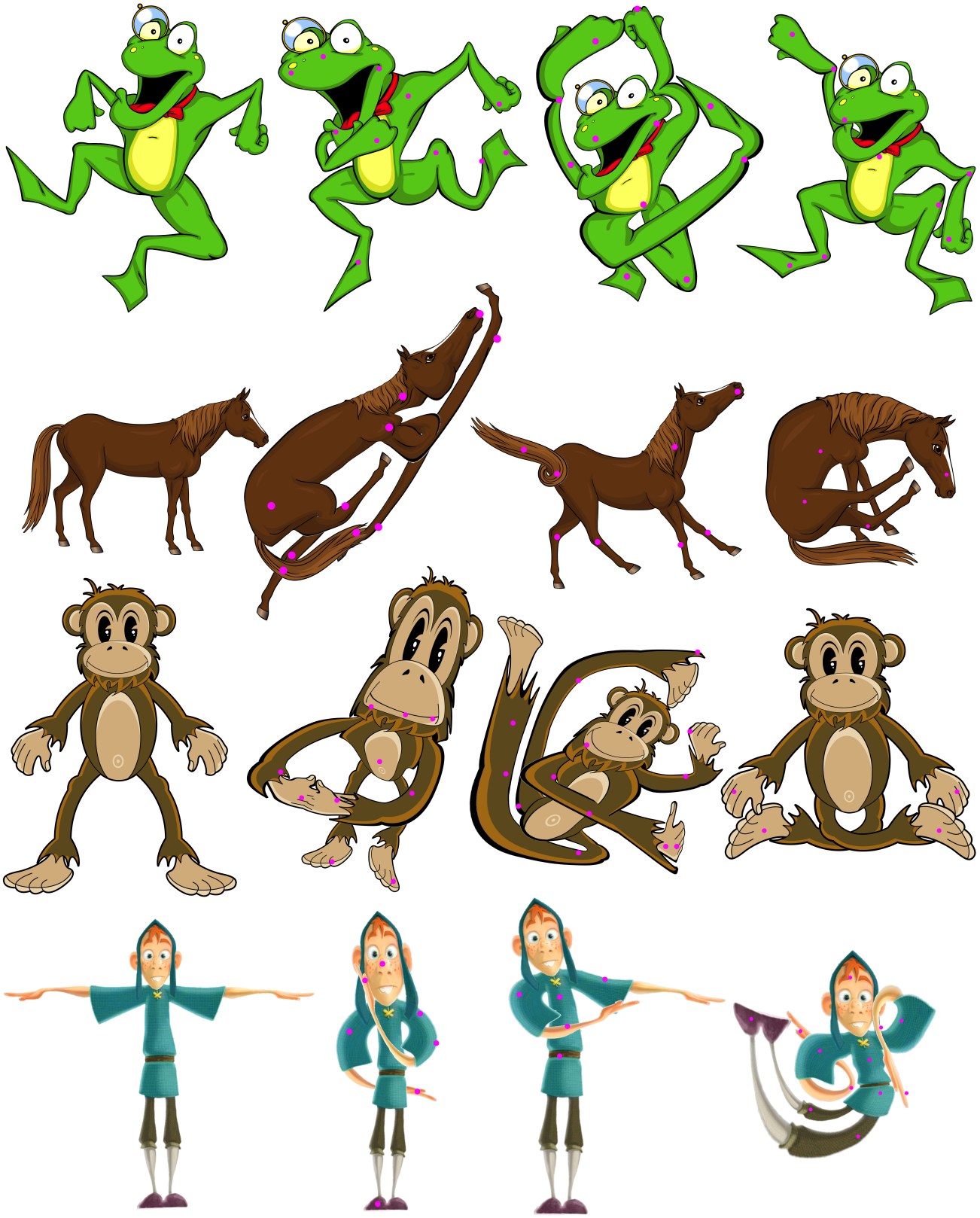

Fig. 12. Additional results minimizing E_{iso}^{f} rendered at high-resolution. Some of these deformations are intentionally radical and are used to stress test the method and to demonstrate robustness. Our algorithm quickly converges on these extreme deformations and certifies the maps as locally injective.