Rectangular Surface Parameterization—Supplemental Information

ETIENNE CORMAN, Université de Lorraine, CNRS, LORIA, France
KEENAN CRANE, Carnegie Mellon University, USA

ACM Reference Format:
Etienne Corman and Keenan Crane. 2025. Rectangular Surface Parameterization—Supplemental Information. *ACM Trans. Graph.* 44, 4 (August 2025), 6 pages. https://doi.org/10.1145/3731176

B Discrete Poisson Equation

For completeness, we describe the standard *cotan* discretization of the Poisson equation [MacNeal 1949, Section 3.2], used to recover our final parameterization coordinates in Section 5.2.

B.1 Discrete Laplacian

The *cotan-Laplace* matrix L is then a sparse $|V| \times |V|$ matrix with nonzero entries

$$L_{i,j} = \begin{cases} -w_{ij}, & ij \in E, \\ \sum_{ik} w_{ik} & i = j. \end{cases} \tag{1}$$

B.2 Discrete Divergence

The *discrete divergence* operator is a sparse $|V| \times |E|$ matrix with nonzero entries

$$\mathrm{div}_{i,ij} = s_{ij} w_{ij}, \ \forall i \in V, \ ij \in E, \tag{2}$$

where $s_{ij} = +1$ when edge ij is oriented from i to j, and equals -1 otherwise.

B.3 Discrete Poisson Equation

Finally, we discretize the Poisson equation in Equation 13 as a scalar Poisson equation for each coordinate f^1, f^2 of the final rectangular parameterization $f : V \to \mathbb{R}^2$, namely

$$Lf^p = \mathrm{div}\mu^p, \quad p = 1, 2. \tag{3}$$

Here L, div, and μ are the discrete Laplacian, divergence, and differential given in Equations 1, 2, and 18, *resp.*.

C Underdetermined Newton Solver

To solve our main optimization problem (Equation 19), we need a method that can deal with underdetermined nonlinear constraints. We use the method of Polyak and Tremba [2020], detailed here. Similar to Newton's method, this method finds a zero of the first order optimality conditions by repeatedly solving linear systems, using a carefully-chosen line search.

Authors' Contact Information: Etienne Corman, Université de Lorraine, CNRS, LORIA, Vandœuvre-lès-Nancy, France; Keenan Crane, Carnegie Mellon University, Pittsburgh, PA, USA.

ACM 1557-7368/2025/8-ART
https://doi.org/10.1145/3731176

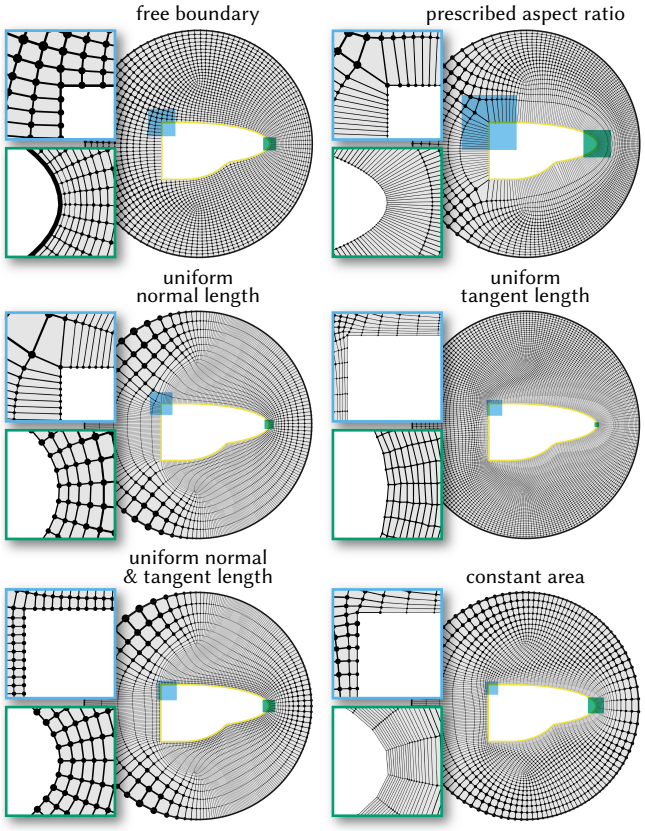

Fig. 32. Here we show boundary conditions like those from Figure 18, but without any cone singularities. This example illustrates both the robustness of our method to extreme constraints—and also emphasizes the utility of singularities for obtaining low distortion/good element quality.

More explicitly, let $x := (u, v, \theta) \in \mathbb{R}^{2|V|+|F|}$ be the primal variables and $\lambda \in \mathbb{R}^{|E|}$ be the Lagrange multiplier, the critical points of Equation 19 are solutions of the system of equations:

$$\begin{aligned} \nabla\Phi(x) + J_F(x)^\top \lambda &= 0 \\ F(x) &= 0, \end{aligned} \tag{4}$$

where $\nabla\Phi$ denote gradient of the objective chosen in Section 5.1.1, and J_F is the Jacobian of the constraint equation given in Section 4.1.

The nonlinear system of equations in Equation 4 is solved by taking Newton steps, using a line search that directly encourages satisfaction of this system. More explicitly, let x_k and λ_k denote the current guess at the kth step of optimization. Then we solve the linear system

$$\begin{bmatrix} \nabla^2\Phi(x_k) + \nabla(J_F(x_k)^\top \lambda_k) & J_F(x_k)^\top \\ J_F(x_k) & 0_{|E|\times|E|} \end{bmatrix} \begin{bmatrix} y_k \\ \delta_k \end{bmatrix} = \begin{bmatrix} \nabla\Phi(x_k) + J_F(x_k)^\top \lambda_k \\ F(x_k) \end{bmatrix} \tag{5}$$

for the descent directions y_k and δ_k, where $\nabla^2\Phi$ denote the Hessian the objective function. The updated guesses are $x_{k+1} = x_k - \tau y_k$ and $\lambda_{k+1} = \lambda_k - \tau\delta_k$, where $\tau > 0$ is a step size. The step size is computed by performing backtracking line search until

$$\begin{aligned}\|\nabla\Phi(x_k - \tau y_k) + J_F(x_k - \tau y_k)^\top(\lambda_k - \tau\delta_k)\| + \|F(x_k - \tau y_k)\| \\ \leq (1 - \tfrac{\tau}{2})\left(\|\nabla\Phi(x_k) + J_F(x_k)^\top\lambda_k\| + \|F(x_k)\|\right).\end{aligned}$$

This strategy guarantees that we converge to a local minimizer of the original optimization problem (Equation 19), at essentially the usual rate of Newton's method—see Polyak and Tremba [2020, Theorem 4.1]. In practice, we use a backtracking ratio of 0.9 for line search. Crucially, we find that Equation 5 always admits a solution and is only overconstrained when inadmissible boundary constraints are imposed (Section 5.1.3).

D Singularities and Cuts

In the case where either (i) the field has singularities and/or (ii) the domain M is not a topological disk, we must effectively cut along the curve Γ, plus additional topological cuts, before flattening the mesh. In practice, we compute topological cuts by considering the traversal tree used to construct Γ, and taking the complementary graph of primal edges not crossed by any edge in this dual tree. We then iteratively remove any degree-1 vertex from the primal graph, except for singular vertices, until no more vertices can be removed. The remaining primal edges define the cut. (This approach is similar in spirit to the *tree-cotree* strategy of Eppstein [2002].)

This problem of solving for a parameterization on a cut mesh comes up frequently in geometry processing, and is a perennial nuisance. Hence, we will first introduce a perspective that simplifies the situation both conceptually and in terms of implementation, before giving the details of our particular algorithm. We make just one simple change to our basic setup: rather than store values at vertices, we will store values at *all* triangle corners, using linear constraints to identify equivalent values as needed. In the absence of any cuts, our formulation ultimately yields the same solution as before, but with the additional flexibility to easily incorporate cuts.

In more detail, for any function f_i at vertices, we now store values f_i^{jk} at all triangle corners. A sparse matrix $U \in \mathbb{R}^{m\times|C|}$ encodes equivalence of values at different corners, where m is the number of identifications, and $|C|$ is the number of corners. Two corners around a common vertex are equivalent if and only if they are not separated by a cut edge. For the nth equivalence $f_i^{jk} = f_i^{kl}$ of consecutive corners, U will have nonzero entries $U_{\boldsymbol{n},{}_i^{jk}} = 1$ and $U_{\boldsymbol{n},{}_i^{kl}} = -1$, *i.e.*, the equation number determines the row index, and the corners are specified via the column indices.

Next, suppose we want to solve any linear equation $Af = b$, which in general may be under- or over-determined. We can express this system in terms of values on corners, plus the identification matrix U, as a least-squares problem subject to linear constraints:

$$\begin{aligned}\min_{f\in\mathbb{R}^{|C|}} \quad & \|Af - b\|_M^2 \\ \text{s.t.} \quad & Uf = 0.\end{aligned} \tag{6}$$

Here $|A|_M^2 := A^\top MA$ denotes the ℓ-2 norm with a mass matrix $M \in \mathbb{R}^{|C|\times|C|}$. We use in particular a diagonal matrix with nonzero entries

$$M_{{}_i^{jk},{}_i^{jk}} = \tfrac{1}{2}\cot\alpha_i^{jk},$$

i.e., we use the cotan weight associated with each corner. In the case where there is no cut, our problem amounts to solving the exact same cotan-Poisson problem in Equation 3.

Equation 6 can be solved using any available method—for instance, the method of Lagrange multipliers, we find the solution to this problem is given by the block linear system

$$\begin{bmatrix} A^\top MA & U^\top \\ U & 0 \end{bmatrix}\begin{bmatrix} f \\ \lambda \end{bmatrix} = \begin{bmatrix} A^\top Mb \\ 0 \end{bmatrix},$$

where $\lambda \in \mathbb{R}^m$ are the Lagrange multipliers. By formulating the problem this way, rather than directly applying least-squares to the combined system $[AU]^\top f = [b0]$, we ensure that corner values are identified *exactly*, and only the original equations are approximated. Note that in general there may be more efficient ways to solve this kind of problem—we find that this approach is simple, and helps avoid implementation errors. For instance, one could directly eliminate variables by hand (though modern direct solvers are often quite good at effectively performing this same elimination); a general-purpose QP solver might also work fine depending on the use case.

In the case of our particular problem, we also have to modify Equation 3, splitting it into two different equations on the two sides of every edge. In particular, for each edge ij in Γ we modify the right-hand side so that both vectors μ_{ij}^k, μ_{ji}^l are in the same coordinate system. In particular, consider the angles β_{ij} from Section 5.3.1, which give the rotation between frames (after parallel transport) across any cut edge ij. Hence, to evaluate Equation 3 for an edge from corner ${}_i^{jk}$ to corner ${}_j^{ki}$, we use

$$f_j^{ki} - f_i^{jk} = \tfrac{1}{2}(\mu_{ij}^k + R_{-\zeta_{ij}}\mu_{ji}^l), \tag{7}$$

where $\zeta_{ij} := \frac{\pi}{2}\lceil(\beta - \pi/4)/(\pi/2)\rceil$ is the closest quarter-rotation taking the frame in triangle ijk to the frame in triangle jil. In other words, we account for the jump in the frame across the cut. Likewise, on the other side of the edge, with endpoints at corners ${}_i^{lj}$ and ${}_j^{il}$, we have

$$f_j^{il} - f_i^{lj} = \tfrac{1}{2}(R_{\zeta_{ij}}\mu_{ij}^k + \mu_{ji}^l). \tag{8}$$

References

David Eppstein. 2002. Dynamic generators of topologically embedded graphs. *arXiv preprint cs/0207082* (2002).

Richard Henri MacNeal. 1949. *The solution of partial differential equations by means of electrical networks.* Ph.D. Dissertation. California Institute of Technology.

Boris Polyak and Andrey Tremba. 2020. New versions of Newton method: step-size choice, convergence domain and under-determined equations. *Optimization Methods and Software* 35, 6 (2020).

B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd. 2020. OSQP: an operator splitting solver for quadratic programs. *Mathematical Programming Computation* 12, 4 (2020), 637–672.

E Pseudocode

Here we give detailed pseudocode for our rectangular parameterization algorithm. We omit pre-processing (namely, generation of the reference field) and post-processing (namely, quantization and contouring of the parameterization in the case of meshing), since these steps are handled by existing algorithms. All other methods are given explicitly in terms of basic operations on vectors and a standard halfedge mesh data structure, and should be easily translatable to real code. The only subroutines not spelled out here are:

- $\textsc{Unit}(v)$ — given a nonzero vector $v \in \mathbb{R}^3$, returns the unit vector in the same direction.
- $\textsc{Objective}(M, A, u, v, \theta)$, $\textsc{gradObjective}(M, A, u, v, \theta)$, $\textsc{hessObjective}(M, A, u, v, \theta)$ – implements one of the objective functions from Section 5.1.1, and its gradient/Hessian, given the mesh connectivity M, the triangle areas A, log scale factors u, v per vertex, frame rotation angles θ per face.
- $\textsc{Solve}(A, b)$ — returns x solution of $Ax = b$
- $\textsc{SolveQP}(H, g, J, F)$ — solves a quadratic program of the form $\min_y \frac{1}{2} y^T H y + y^T g$ subject to $Jy + F = 0$. (We use *OSQP* [Stellato et al. 2020].)
- $\textsc{SparseFromTriplets}((I, J, S), n, m)$ — returns a sparse matrix A of size $n \times m$ whose coefficient $A_{I_i J_i}$ is equal to S_i for all i. If a coefficient appears several times in the lists (I, J), their corresponding values in S are summed.
- $\textsc{Append}(L, a)$ — returns a new list appending the element a to the list L.
- $\textsc{Angle}(x, y, z)$ — returns the angle between the vector $y - x \in \mathbb{R}^3$ and $z - x \in \mathbb{R}^3$.

For simplicity we assume M is without boundary.

Algorithm 1 $\textsc{FlattenMesh}(M, x, W)$

Input: A manifold, orientable, simply-connected triangle mesh $M = (V, E, F)$ with corners C, coordinates $x_i \in \mathbb{R}^3$ at each vertex $i \in V$, and a cross field given by any one of four representative unit vectors $W_{ijk} \in \mathbb{R}^3$ in each triangle $ijk \in F$

Output: Coordinates $f : C \to \mathbb{R}^2$ describing the flattening of each triangle into the 2D plane. Note that these values are stored at triangle corners rather than vertices to accommodate possible cuts.

for each $ij \in E$ **do** $\ell_{ij} \leftarrow |x_i - x_j|$ ▷ *edge lengths*
for each $i^{jk} \in C$ **do** $\alpha_i^{jk} \leftarrow \textsc{Angle}(x_i, x_j, x_k)$ ▷ *corner angles*
for each $ijk \in F$ **do** ▷ *cross field angles*
▷ *compute a basis (N, T_1, T_2) for triangle ijk adapted to edge ij*
  $N_1 \leftarrow (x_j - x_i) \times (x_k - x_i)$ ▷ *unnormalized normal*
  $A_{ijk} \leftarrow |N_1|/2$ ▷ *triangle area*
  $N_1 \leftarrow N_1/(2A_{ijk})$ ▷ *normalize*
  $T_1 \leftarrow \textsc{Unit}(x_j - x_i)$
  $T_2 \leftarrow N_1 \times T_1$
  $\xi_{ijk} \leftarrow \text{atan2}(\langle T_2, W_{ijk} \rangle, \langle T_1, W_{ijk} \rangle)$ ▷ *angle relative to ij*
$\Gamma, \zeta, s, \varphi, \omega^0 \leftarrow \textsc{ComputeCutJumpData}(M, \alpha, \xi)$
$u, v, \theta \leftarrow \textsc{SolveOptimizationProblem}(M, A, \alpha, \varphi, \omega^0, s)$
$f \leftarrow \textsc{RecoverParameterization}(M, \Gamma, \zeta, \ell, \alpha, \varphi, \theta, u, v)$
return f

Algorithm 2 $\textsc{ComputeCutJumpData}(M, \alpha, \xi)$

Input: A manifold, orientable triangle mesh $M = (V, E, F)$ with angles α_i^{jk} at each corner i of each triangle $ijk \in F$, angles $\xi : F \to \mathbb{R}$ describing one of the four directions of the cross field in each triangle $ijk \in F$ as a rotation relative to the direction of the first edge ij.

Output: The curve $\Gamma : E \to \{0, 1\}$ cutting M to a topological disk, the closest quarter-rotation $\zeta : E \to \mathbb{R}$ between two frame in adjacent triangle, sign bits $s : C \to \pm 1$ giving the sign of v at each corner relative to the values stored at vertices, angles $\varphi : H \to \mathbb{R}$ which give the reference frame relative to each halfedge, and angles $\omega^0 : E \to \mathbb{R}$ describing the rotation of the reference frame X^0 along each oriented dual edge.

for each $ijk \in F$ **do** ▷ *for convenience, express cross directions relative to halfedges*
  $\xi_{\overrightarrow{ij}} \leftarrow \xi_{ijk}$
  $\xi_{\overrightarrow{jk}} \leftarrow \xi_{\overrightarrow{ij}} - (\pi - \alpha_j^{ki})$
  $\xi_{\overrightarrow{ki}} \leftarrow \xi_{\overrightarrow{jk}} - (\pi - \alpha_k^{ij})$
▷ *Perform breadth-first traversal (Section 5.3.1)*
$ijk \leftarrow \textsc{First}(F)$ ▷ *get the first triangle in the mesh*
for each $ijk \in F$ **do** $\varphi_{ijk} \leftarrow \infty$ ▷ *frame not yet set*
$\varphi_{ijk} \leftarrow \xi_{\overrightarrow{ij}}$ ▷ *reference frame gets one of the four cross directions*
$\textsc{Push}(Q, \overrightarrow{ij})$ ▷ *push any halfedge of ijk onto a queue*
while $\textsc{NotEmpty}(Q)$ **do**
  $\overrightarrow{ij} \leftarrow \textsc{Pop}(Q)$ ▷ *go from ijk to jil*
  $\varphi_{ij \to ji} \leftarrow \varphi_{ij} + \pi$ ▷ *parallel transport*
  $n^* \leftarrow \textsc{Mod}(\textsc{Round}(2\frac{\varphi_{ij \to ji} - \xi_{ji}}{\pi}), 4)$ ▷ *closest cross index*
  $\zeta_{ij} \leftarrow \frac{\pi}{2} n^*$ ▷ *jump angle across ij*
  $\xi^* \leftarrow \zeta_{ij} + \xi_{ji}$ ▷ *closest cross*
  $\omega_{ij}^0 \leftarrow \varphi_{ij} - \xi^* + \pi$ ▷ *smallest rotation to neighboring cross*
  if $\varphi_{ji} == \infty$ **then** ▷ *neighboring frame not yet set*
    $\Gamma_{ij} \leftarrow 0$ ▷ *primal edge not in cut*
    $s_{ij} \leftarrow +1$ ▷ *no sign change*
    $\varphi_{ji} \leftarrow \xi^*$ ▷ *set reference frame in triangle*
    $\varphi_{\overrightarrow{jk}} \leftarrow \varphi_{\overrightarrow{ij}} - (\pi - \alpha_j^{ki})$
    $\varphi_{\overrightarrow{ki}} \leftarrow \varphi_{\overrightarrow{jk}} - (\pi - \alpha_k^{ij})$
    $\textsc{Push}(Q, \overrightarrow{il})$ ▷ *visit other two neighbors*
    $\textsc{Push}(Q, \overrightarrow{lj})$
  else ▷ *neighboring frame already set*
    $\Gamma_{ij} = 0$ ▷ *primal edge could be in cut*
    if $\textsc{IsOdd}(n^*)$ **then** $s_{ij} = -1$
    else $s_{ij} = +1$
end while
▷ *Compute relative signs at corners*
for each $i \in V$ **do** ▷ *iterate over vertices*
  $S = 1$ ▷ *cumulative product (Equation 20)*
  for $p \leftarrow 0, \ldots, \textsc{Degree}(M, i) - 1$ **do**
    $s_i^{j_p j_{p+1}} \leftarrow S$
    $S \leftarrow s_{ij_{p+1}} S$
▷ *compute angle defect*

$K_i \leftarrow 0$
for each $ijk \in F$ **do**
  $K_i \leftarrow K_i + \alpha_i^{jk}$
  $K_j \leftarrow K_j + \alpha_j^{ki}$
  $K_k \leftarrow K_k + \alpha_k^{ij}$
$K \leftarrow 2\pi - K$
▷ *compute cones indices at vertices*
$c_i \leftarrow K$
for each $i \in V$ **do**
  for each $j \in V$ such that $ij \in E$ **do** ▷ *edge incident to i*
    $w \leftarrow \omega_{ij}^0$
    if $i > j$ **then** $w \leftarrow -w$
    $c_i \leftarrow c_i + w$ ▷ *accumulate all rotations*
▷ *compute the cut*
$r \leftarrow$ True
while r **do**
  $d \leftarrow 0$ ▷ *initialize degree of unvisited edges*
  for each $\Gamma_{ij} == 1$ **do** ▷ *iterate over unvisited edges*
    $d_i \leftarrow d_i + 1$
    $d_j \leftarrow d_j + 1$
  $r \leftarrow$ False
  for each $i \in V$ **do** ▷ *iterate over vertices*
    if $d_i == 1$ and $c_i == 0$ **then** ▷ *if degree-1 and not a cone*
      for each $ij \in E$ **do** $\Gamma_{ij} = 0$ ▷ *remove edge*
      $r \leftarrow$ True
end while
return $(\Gamma, \zeta, s, \varphi, \omega^0)$

Algorithm 3 SolveOptimizationProblem$(M, A, \alpha, \varphi, \omega^0, s)$

Input: A manifold, orientable, simply-connected triangle mesh $M = (V, E, F)$, areas A_{ijk} for each triangle $ijk \in F$, angles α_i^{jk} at each corner $_i^{jk} \in C$, angles $\varphi_{\vec{ij}}$ describing the angle of the reference frame relative to each halfedge $\vec{ij} \in H$, angles ω_{ij}^0 describing the rotation of the reference frame across each edge $ij \in E$, and values $s_i^{jk} \in \{\pm 1\}$ giving the sign of the value at each corner $_i^{jk}$ relative to a value stored at vertex i.
Output: Scale factors $u_i, v_i \in \mathbb{R}$ at each vertex $i \in V$, and angles $\theta_{ijk} \in \mathbb{R}$ for each triangle $ijk \in F$.

$u, v \leftarrow 0 \in \mathbb{R}^{|V|}$ ▷ *initialize scale factors*
$\theta \leftarrow 0 \in \mathbb{R}^{|F|}$ ▷ *initialize frame angles*
$\lambda \leftarrow 0 \in \mathbb{R}^{|E|}$ ▷ *initialize Lagrange multiplier*
while True **do**
  $H \leftarrow$ hessObjective(M, A, u, v, θ)
  $g \leftarrow$ gradObjective(M, A, u, v, θ)
  $D \leftarrow$ BuildHessian$(M, \alpha, u, v, \lambda, s, \varphi, \theta)$ ▷ *constraint Hessian*
  $J \leftarrow$ BuildJacobian$(M, \alpha, u, v, s, \varphi, \theta)$ ▷ *constraint Jacobian*
  $F \leftarrow$ BuildSystem$(M, \alpha, u, v, s, \varphi, \theta, \omega^0)$ ▷ *constraint value*
  $x \leftarrow (u, v, \theta, \lambda)$ ▷ *current guess*
  $A \leftarrow \begin{bmatrix} H + D & J^\top \\ J & 0_{|E|\times|E|} \end{bmatrix}$ ▷ *linear system*
  $b \leftarrow \begin{bmatrix} g + J^\top \lambda \\ F \end{bmatrix}$ ▷ *right hand side*
  $y \leftarrow$ Solve(A, b)
  $\tau \leftarrow$ LineSearch$(M, \alpha, s, \varphi, \omega^0, J, F, g, x, y)$
  $(u, v, \theta, \lambda) \leftarrow x + \tau y$ ▷ *update guess*
  if $\|g + J^\top \lambda\| + \|F\| < \varepsilon$ **then**
    break
end while
return (u, v, θ)

Algorithm 4 LineSearch$(M, \alpha, s, \varphi, \omega^0, J, F, g, x, y)$

Input: A manifold, orientable triangle mesh $M = (V, E, F)$ with angles α_i^{jk} at each corner i of each triangle $ijk \in F$, sign bits $s : C \to \pm 1$ giving the sign of v at each corner relative to the values stored at vertices, angles $\varphi : H \to \mathbb{R}$ giving the angle of the reference frame relative to each halfedge, angles $\omega^0 : E \to \mathbb{R}$ describing the rotation of the reference frame X^0 across each edge, a sparse matrix $J \in \mathbb{R}^{|E|\times(2|V|+|F|)}$ giving the Jacobian of the constraint, values $F : E \to \mathbb{R}$ giving the discrete residual of Equation 17, a vector $g \in \mathbb{R}^{2|V|+|F|}$ giving the gradient of the objective function, two vectors $x, y \in \mathbb{R}^{2|V|+|F|}$ describing the current estimate and the descent direction
Output: Step size $\tau > 0$ decreasing the Lagrangian gradient norm

$(u, v, \theta, \lambda) \leftarrow x$
$\mathcal{E} \leftarrow \|g + J^\top \lambda\| + \|F\|$ ▷ *norm of gradient Lagrangian*
$\tau \leftarrow 1$ ▷ *step size*
while True **do**
  $(u, v, \theta, \lambda) \leftarrow x + \tau y$ ▷ *update estimate*
  $g \leftarrow$ gradObjective(M, A, u, v, θ) ▷ *gradient of objective*
  $J \leftarrow$ BuildJacobian$(M, \alpha, u, v, s, \varphi, \theta)$ ▷ *constraint Jacobian*
  $F \leftarrow$ BuildSystem$(M, \alpha, u, v, s, \varphi, \theta, \omega^0)$ ▷ *constraint value*
  $\mathcal{E}_{\text{new}} \leftarrow \|g + J^\top \lambda\| + \|F\|$ ▷ *update gradient norm*
  if $\mathcal{E}_{\text{new}} \leq \left(1 - \frac{\tau}{2}\right) \mathcal{E}$ **then**
    break ▷ *stop when sufficient decrease*
  else
    $\tau \leftarrow 0.9\tau$ ▷ *decrease step size*
end while
return τ

Algorithm 5 BuildSystem$(M, \alpha, u, v, s, \varphi, \theta, \omega^0)$

Input: A manifold, orientable triangle mesh $M = (V, E, F)$ with angles α_i^{jk} at each corner i of each triangle $ijk \in F$, values $u, v : V \to \mathbb{R}$ at each vertex giving the log scale factors, sign bits $s : C \to \pm 1$ giving the sign of v at each corner relative to the values stored at vertices, angles $\varphi : H \to \mathbb{R}$ giving the angle of the reference frame relative to each halfedge, angles $\theta : F \to \mathbb{R}$ describing the current frame X_1^θ as a rotation of the reference frame, angles $\omega^0 : E \to \mathbb{R}$ describing the rotation of the reference frame X^0 across each edge.
Output: Values $\rho : E \to \mathbb{R}$ giving the residual of our discrete equation (Equation 17) on each edge.

for each $ij \in E$ **do** ▷ *left-hand side of Equation 10*
  $\rho_{ij} \leftarrow (\theta_{ijk} - \theta_{jil}) - \omega_{ij}^0$ ▷ *k and l are left/right of ij*

3: **for each** $ijk \in F$ **do** ▷*right-hand side of Equation 10* ▷*add contribution to residual from dual edges inside this triangle*
4: $\rho_{ij} \leftarrow \rho_{ij} + \textsc{Residual}(i, j, k, \alpha, u, v, s, \varphi_{\vec{ij}} + \theta_{ijk})$
5: $\rho_{jk} \leftarrow \rho_{jk} + \textsc{Residual}(j, k, i, \alpha, u, v, s, \varphi_{\vec{jk}} + \theta_{ijk})$
6: $\rho_{ki} \leftarrow \rho_{ki} + \textsc{Residual}(k, i, j, \alpha, u, v, s, \varphi_{\vec{ki}} + \theta_{ijk})$
7: return ρ

Algorithm 6 $\textsc{Residual}(i, j, k, \alpha, u, v, s, \eta)$

Input: Three vertices i, j, k of a triangle, the angles α at triangle corners, the log scale factors u, v at vertices, the signs s at triangle corners, and the angle η of the current frame X_1^θ relative to a coordinate system aligned with edge ij.
Output: The contribution to the residual of equation Equation 17 due to the piece of the edge dual to ij within triangle ijk.

1: $(v_i^{jk}, v_j^{ki}, v_k^{ij}) \leftarrow (s_i^{jk} v_i, s_j^{ki} v_j, s_k^{ij} v_k)$ ▷*values at corners*
2: $\rho \leftarrow u_j - u_i$ ▷*conformal part*
3: $\rho \leftarrow \rho + \cos(2\eta)(v_j^{ki} - v_i^{jk})$ ▷*non-conformal part*
4: $\rho \leftarrow \rho + \sin(2\eta) \cot \alpha_j^{ki} (v_k^{ij} - v_i^{jk})$
5: $\rho \leftarrow \rho + \sin(2\eta) \cot \alpha_i^{jk} (v_k^{ij} - v_j^{ki})$
6: $\rho \leftarrow \frac{1}{2} \cot \alpha_k^{ij} \rho$
7: **if** $i > j$ **then**
8: $\rho \leftarrow -\rho$ ▷*account for edge orientation*
9: return ρ

Algorithm 7 $\textsc{BuildJacobian}(M, \alpha, u, v, s, \varphi, \theta)$

Input: A manifold, orientable triangle mesh $M = (V, E, F)$ with angles α_i^{jk} at each corner i of each triangle $ijk \in F$, values $u, v : V \to \mathbb{R}$ at each vertex giving the log scale factors, signs $s : C \to \pm 1$ describing the values of v at corners relative to the values stored at vertices, angles $\varphi : F \to \mathbb{R}$ describing the reference frame X_1^0 in each triangle $ijk \in F$ as a rotation relative to the direction of the first edge ij, and angles $\theta : F \to \mathbb{R}$ describing the current frame X_1^θ as a rotation of the reference frame.
Output: A sparse matrix $J \in \mathbb{R}^{|E| \times (2|V| + |F|)}$ giving the Jacobian of the constraint equations computed by $\textsc{BuildSystem}$.

1: $L_v \leftarrow ()$ ▷*list of nonzeros of Jacobian with respect to v*
2: $L_\theta \leftarrow ()$ ▷*list of nonzeros of Jacobian with respect to θ*
3: **for each** $ij \in E$ **do** ▷*Jacobian of left-hand side of Equation 10*
4: $L_\theta \leftarrow \textsc{Append}(L_\theta, (ij, ijk, +1))$
5: $L_\theta \leftarrow \textsc{Append}(L_\theta, (ij, jil, -1))$
6: **for each** $ijk \in F$ **do** ▷*Jacobian of right-hand side of Equation 10*
7: $\eta_{ij} \leftarrow \varphi_{ijk} + \theta_{ijk}$ ▷*angle of X_1^θ relative to each edge*
8: $\eta_{jk} \leftarrow \eta_{ij} - (\pi - \alpha_j^{ki})$
9: $\eta_{ki} \leftarrow \eta_{jk} - (\pi - \alpha_k^{ij})$
10: ▷*append contribution to Jacobian from dual edges in this triangle*
11: $(L_u, L_v, L_\theta) \leftarrow \textsc{Jacobian}(i, j, k, \alpha, u, v, s, \eta_{ij}, L_u, L_v, L_\theta)$
12: $(L_u, L_v, L_\theta) \leftarrow \textsc{Jacobian}(j, k, i, \alpha, u, v, s, \eta_{jk}, L_u, L_v, L_\theta)$
13: $(L_u, L_v, L_\theta) \leftarrow \textsc{Jacobian}(k, i, j, \alpha, u, v, s, \eta_{ki}, L_u, L_v, L_\theta)$
14: ▷*Build Jacobian blocks from lists of nonzeros*
15: $J_u \leftarrow \textsc{SparseFromTriplets}(L_u, |E|, |V|)$
16: $J_v \leftarrow \textsc{SparseFromTriplets}(L_v, |E|, |V|)$
17: $J_\theta \leftarrow \textsc{SparseFromTriplets}(L_\theta, |E|, |F|)$
18: return $[J_u\ J_v\ J_\theta]$ ▷*assemble overall $|E| \times (2|V| + |F|)$ Jacobian*

Algorithm 8 $\textsc{Jacobian}(i, j, k, \alpha, u, v, s, \eta, L_u, L_v, L_\theta)$

Input: The same inputs as $\textsc{Residual}$, plus the cumulative lists of entries L_u, L_v, L_θ of the three Jacobian blocks built so far.
Output: Modified lists L_u, L_v, L_θ that account for the derivatives (with respect to u, v and θ) of the residual due to the piece of the edge dual to ij within triangle ijk.

1: $w \leftarrow \frac{1}{2} \cot \alpha_k^{ij}$ ▷*cotan weight*
2: **if** $i > j$ **then**
3: $w \leftarrow -w$ ▷*account for edge orientation*
4: ▷*conformal part w.r.t. u*
5: $L_u \leftarrow \textsc{Append}(L_u, (ij, i, -w))$
6: $L_u \leftarrow \textsc{Append}(L_u, (ij, j, +w))$
7: ▷*non-conformal part w.r.t v*
8: $L_v \leftarrow \textsc{Append}(L_v, (ij, i, -w s_i^{jk} (\cos(2\eta) + \sin(2\eta) \cot \alpha_j^{ki})))$
9: $L_v \leftarrow \textsc{Append}(L_v, (ij, j, +w s_j^{ki} (\cos(2\eta) - \sin(2\eta) \cot \alpha_i^{jk})))$
10: $L_v \leftarrow \textsc{Append}(L_v, (ij, k, +w s_k^{ij} \sin(2\eta)(\cot \alpha_j^{ki} + \cot \alpha_i^{jk})))$
11: ▷*non-conformal part w.r.t θ*
12: $(v_i^{jk}, v_j^{ki}, v_k^{ij}) \leftarrow (s_i^{jk} v_i, s_j^{ki} v_j, s_k^{ij} v_k)$ ▷*values at corners*
13: $L_\theta \leftarrow \textsc{Append}(L_\theta, (ij, ijk, -2w \sin(2\eta)(v_j^{ki} - v_i^{jk})))$
14: $L_\theta \leftarrow \textsc{Append}(L_\theta, (ij, ijk, +2w \cos(2\eta) \cot \alpha_j^{ki} (v_k^{ij} - v_i^{jk})))$
15: $L_\theta \leftarrow \textsc{Append}(L_\theta, (ij, ijk, +2w \cos(2\eta) \cot \alpha_i^{jk} (v_k^{ij} - v_j^{ki})))$
16: return (L_u, L_v, L_θ)

Algorithm 9 $\textsc{BuildHessian}(M, \alpha, u, v, \lambda, s, \varphi, \theta)$

Input: A manifold, orientable triangle mesh $M = (V, E, F)$ with angles α_i^{jk} at each corner i of each triangle $ijk \in F$, values $u, v : V \to \mathbb{R}$ at each vertex giving the log scale factors, values $\lambda : E \to \mathbb{R}$ at each dual-edge giving the Lagrange multiplier associated to the constraint, signs $s : C \to \pm 1$ describing the values of v at corners relative to the values stored at vertices, angles $\varphi : F \to \mathbb{R}$ describing the reference frame X_1^0 in each triangle $ijk \in F$ as a rotation relative to the direction of the first edge ij, and angles $\theta : F \to \mathbb{R}$ describing the current frame X_1^θ as a rotation of the reference frame.
Output: A sparse matrix $H \in \mathbb{R}^{(2|V|+|F|) \times (2|V|+|F|)}$ giving the Jacobian of the matrix computed by $\textsc{Jacobian}$ multiplied by the Lagrange multiplier λ.

1: $L_u \leftarrow ()$ ▷*list of nonzeros of Jacobian with respect to u*
2: $L_v \leftarrow ()$ ▷*list of nonzeros of Jacobian with respect to v*
3: $L_\theta \leftarrow ()$ ▷*list of nonzeros of Jacobian with respect to θ*
4: **for each** $ijk \in F$ **do** ▷*Jacobian of right-hand side of Equation 10*
5: $\eta_{ij} \leftarrow \varphi_{ijk} + \theta_{ijk}$ ▷*angle of X_1^θ relative to each edge*
6: $\eta_{jk} \leftarrow \eta_{ij} - (\pi - \alpha_j^{ki})$
7: $\eta_{ki} \leftarrow \eta_{jk} - (\pi - \alpha_k^{ij})$
8: ▷*append contribution to Jacobian from dual edges in this triangle*
9: $(L_v, L_\theta) \leftarrow \textsc{Hessian}(i, j, k, \alpha, u, v, \lambda, s, \eta_{ij}, L_v, L_\theta)$
10: $(L_v, L_\theta) \leftarrow \textsc{Hessian}(j, k, i, \alpha, u, v, \lambda, s, \eta_{jk}, L_v, L_\theta)$

$(L_v, L_\theta) \leftarrow$ Hessian$(k, i, j, \alpha, u, v, \lambda, s, \eta_{ki}, L_v, L_\theta)$

▷ *Build Jacobian blocks from lists of nonzeros*

$H_v \leftarrow$ SparseFromTriplets$(L_v, |V|, |F|)$

$H_\theta \leftarrow$ SparseFromTriplets$(L_\theta, |F|, |F|)$

return $\begin{bmatrix} 0_{|V|\times|V|} & 0_{|V|\times|V|} & 0_{|V|\times|F|} \\ 0_{|V|\times|V|} & 0_{|V|\times|V|} & H_v^\top \\ 0_{|F|\times|V|} & H_v & H_\theta \end{bmatrix}$ ▷ *assemble overall* $(2|V| + |F|) \times (2|V| + |F|)$ *Jacobian*

Algorithm 10 Hessian$(i, j, k, \alpha, u, v, \lambda, s, \eta, L_v, L_\theta)$

Input: The same inputs as BuildHessian, plus the cumulative lists of entries L_v, L_θ of the two Hessian blocks built so far.

Output: Modified lists L_v, L_θ that account for the derivatives (with respect to v and θ) of the Jacobian due to the piece of the edge dual to ij within triangle ijk.

$w \leftarrow \frac{1}{2}\cot \alpha_k^{ij}$ ▷ *cotan weight*

if $i > j$ **then**

  $w \leftarrow -w$ ▷ *account for edge orientation*

▷ *non-conformal part w.r.t* v

$L_v \leftarrow$ Append$(L_v, (i, ijk, 2s_i^{jk}\lambda_{ij} w(\sin(2\eta) - \cos(2\eta)\cot\alpha_j^{ki})))$

$L_v \leftarrow$ Append$(L_v, (j, ijk, 2s_j^{ki}\lambda_{ij} w(-\sin(2\eta) - \cos(2\eta)\cot\alpha_i^{jk})))$

$L_v \leftarrow$ Append$(L_v, (k, ijk, 2s_k^{ij}\lambda_{ij} w\cos(2\eta)(\cot\alpha_j^{ki} + \cot\alpha_i^{jk})))$

▷ *non-conformal part w.r.t* θ

$(v_i^{jk}, v_j^{ki}, v_k^{ij}) \leftarrow (s_i^{jk} v_i, s_j^{ki} v_j, s_k^{ij} v_k)$ ▷ *values at corners*

$L_\theta \leftarrow$ Append$(L_\theta, (ijk, ijk, -4\lambda_{ij} w\cos(2\eta)(v_j^{ki} - v_i^{jk})))$

$L_\theta \leftarrow$ Append$(L_\theta, (ijk, ijk, -4\lambda_{ij} w\sin(2\eta)\cot\alpha_j^{ki}(v_k^{ij} - v_i^{jk})))$

$L_\theta \leftarrow$ Append$(L_\theta, (ijk, ijk, -4\lambda_{ij} w\sin(2\eta)\cot\alpha_i^{jk}(v_k^{ij} - v_j^{ki})))$

return (L_v, L_θ)

Algorithm 11 RecoverParameterization$(M, \Gamma, \zeta, \ell, \alpha, \varphi, \theta, u, v)$

Input: The mesh connectivity $M = (V, E, F)$, the cut curve $\Gamma : E \to \{0, 1\}$ cutting the mesh to disk topology, the closest quarter-rotation $\zeta : E \to \mathbb{R}$ between two frame in adjacent triangle, the length $\ell_{ij} \in \mathbb{R}_{\geq 0}$ of each edge $ij \in E$, angles α_i^{jk} at each triangle corner, the angle $\varphi_{ijk} \in \mathbb{R}$ of the reference frame relative to edge ij in each triangle ijk, the angle θ_{ijk} of the optimized frame relative to the reference frame, and the log scale/stretch factors u_i, v_i at each vertex $i \in V$.

Output: $f : C \to \mathbb{R}^2$ giving the uv-coordinates per triangle corner.

for each $ij \in E$ **do** ▷ *edge scale from log-scale*

  $a_{ij} \leftarrow e^{(u_i + u_j + v_i + v_j)/2}$

  $b_{ij} \leftarrow e^{(u_i + u_j - v_i - v_j)/2}$

for each $ijk \in F$ **do**

  $\eta_{ij} \leftarrow \varphi_{ijk} + \theta_{ijk}$ ▷ *angle of* X_1^θ *relative to each edge*

  $\eta_{jk} \leftarrow \eta_{ij} - (\pi - \alpha_j^{ki})$

  $\eta_{ki} \leftarrow \eta_{jk} - (\pi - \alpha_k^{ij})$

  ▷ *target edge vector*

  $\mu_{ij}^k \leftarrow \ell_{ij}(+a_{ij}\cos(\eta_{ijk}), +b_{ij}\sin(\eta_{ijk}))$

  $\mu_{jk}^i \leftarrow \ell_{jk}(-a_{jk}\cos(\eta_{ijk} + \alpha_j^{ki}), -b_{jk}\sin(\eta_{ijk} + \alpha_j^{ki}))$

  $\mu_{ki}^j \leftarrow \ell_{ki}(-a_{ki}\cos(\eta_{ijk} - \alpha_i^{jk}), -b_{ki}\sin(\eta_{ijk} - \alpha_i^{jk}))$

  ▷ *right hand-side in Equation ??*

for each $\frac{ij}{k} \in C$ **do** $b_k^{ij} \leftarrow \frac{1}{2}(R_{\zeta_{ij}}\mu_{ij}^k + \mu_{ji}^l)$

$L_A \leftarrow ()$

$L_U \leftarrow ()$

$p \leftarrow 0$

for each $\frac{ij}{k} \in C$ **do**

  ▷ *build matrix* A *in Equation 6*

  $L_A \leftarrow$ Append$(L_A, (\frac{ij}{k}, \frac{jk}{i}, +1))$

  $L_A \leftarrow$ Append$(L_A, (\frac{ij}{k}, \frac{ki}{j}, -1))$

  ▷ *build matrix* U *in Equation 6*

  if $\Gamma_{ij} == 0$ **then** ▷ *if* ij *not in the cut, add corner constraints*

    $L_U \leftarrow$ Append$(L_U, (p, \frac{jk}{i}, +1))$

    $L_U \leftarrow$ Append$(L_U, (p, \frac{lj}{i}, -1))$

    $L_U \leftarrow$ Append$(L_U, (p + 1, \frac{ki}{j}, +1))$

    $L_U \leftarrow$ Append$(L_U, (p + 1, \frac{il}{j}, -1))$

    $p \leftarrow p + 2$

▷ *Build Jacobian blocks from lists of nonzeros*

$A \leftarrow$ SparseFromTriplets$(L_A, |C|, |C|)$

$U \leftarrow$ SparseFromTriplets$(L_U, p, |C|)$

$f \leftarrow$ SolveQP$(A^\top W A, -A^\top W b, U, 0)$ ▷ *corner coordinates*