Stripe Patterns on Surfaces

Felix Knöppel — TU Berlin
Keenan Crane — Columbia University
Ulrich Pinkall — TU Berlin
Peter Schröder — Caltech

Abstract

Stripe patterns are ubiquitous in nature, describing macroscopic phenomena such as stripes on plants and animals, down to material impurities on the atomic scale. We propose a method for synthesizing stripe patterns on triangulated surfaces, where singularities are automatically inserted in order to achieve user-specified orientation and line spacing. Patterns are characterized as global minimizers of a convex-quadratic energy which is well-defined in the smooth setting. Computation amounts to finding the principal eigenvector of a symmetric positive-definite matrix with the same sparsity as the standard graph Laplacian. The resulting patterns are globally continuous, and can be applied to a variety of tasks in design and texture synthesis.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems

Keywords: discrete differential geometry, digital geometry processing, texture synthesis, direction fields, singularities

1 Introduction

A diverse collection of natural phenomena exhibit the same characteristic pattern: unoriented stripes of uniform width that bifurcate at isolated points called *edge dislocations* [Kalpakjian and Schmid 2009, pp. 44–46]. Why does this bifurcation occur? Surfaces of revolution (Fig. 3) provide an instructive example: for any given stripe width there is a maximum integer number of stripes that can fit around the circumference at each height h. As the radius r grows or shrinks this number can suddenly jump—stripes must therefore branch or coalesce in order to maintain a continuous transition between neighboring regions.

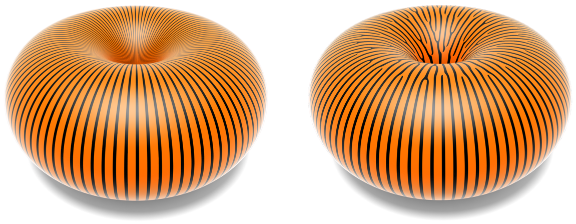
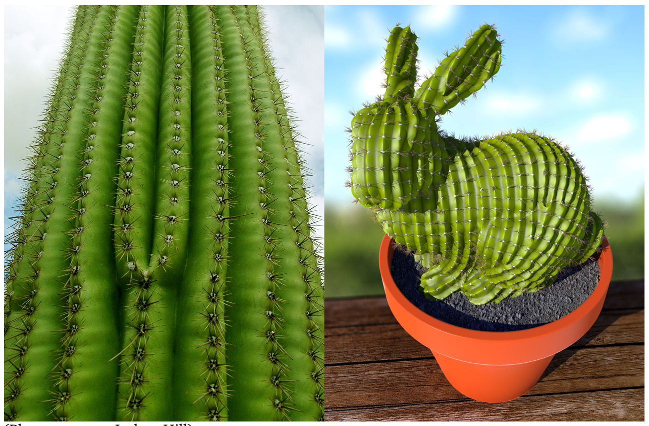


Figure 1: *In general, patterns without singularities exhibit uneven spacing, and patterns with even spacing must have singularities.*



(Photo courtesy Joshua Hill)

Figure 2: *Natural phenomena like cacti* (left) *exhibit characteristic branching patterns in an effort to maintain evenly-spaced features.* Right: *Our algorithm allows one to quickly and automatically synthesize similar patterns that seamlessly cover arbitrary surfaces.*

Our approach to stripe pattern synthesis is closely related to existing methods for field-aligned parameterization (Sec. 1.1). The main point of departure is that existing algorithms try to *prevent* singularities that were not specified as part of the input, whereas we intentionally *allow* new singularities, which are essential to modeling natural phenomena. In particular, we use a formulation akin to Ray *et al.* [2006], but omit both the curl correction step and the unit-norm constraint. As a result, we can formulate our problem via a simple convex quadratic energy; the practical payoff is an algorithm about an order of magnitude faster than those based on nonconvex constraints or mixed integer programming (Sec. 5.1). The final algorithm is simple to implement (App. C) requiring no global cutting or integer jumps, is robust to errors in the input (Fig. 16), and requires little work beyond factoring a single sparse matrix. The output is an angle α at each triangle corner which can be used to drive a periodic texture or shader. We also introduce a novel interpolation scheme (Sec. 4.3) that ensures the pattern is globally continuous (C^0) away from a finite collection of isolated singular points.

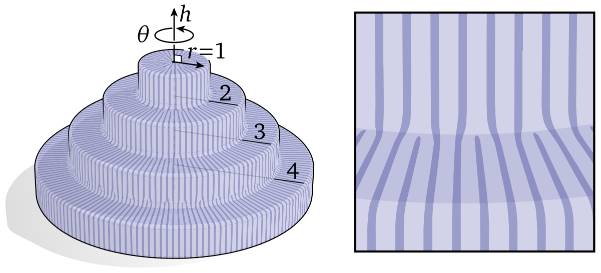


Figure 3: *To maintain constant stripe width, each "level" of this surface of revolution must exhibit a different number of stripes. Judiciously placed branch points are therefore essential to keeping the pattern continuous.* Inset: *each pair of stripes from the second level perfectly branches into three.*

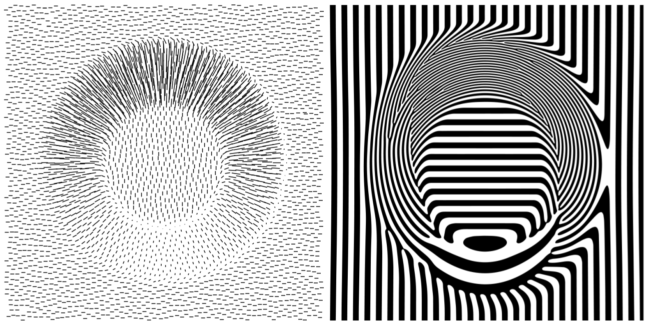


Figure 4: *An arbitrary vector field* (left) *drives stripe orientation and spacing* (right)*; there are no special conditions on the input.*

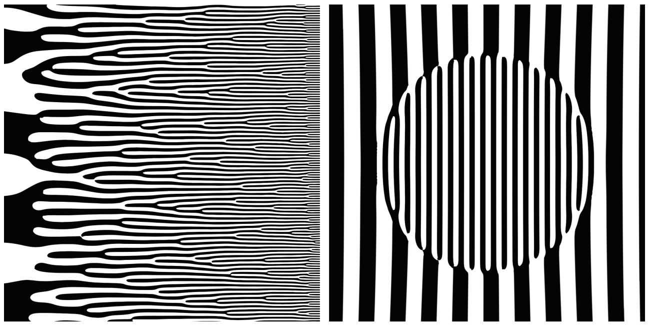


Figure 6: *Our method handles line spacing that varies spatially* (left)*, even if the target spacing function is discontinuous* (right).

1.1 Related Work

Synthesis of stripe patterns has a long history due to the prevalence of these patterns in nature (Figs. 2, 5, 7, 8, and 20). Placement of evenly-spaced stripes is also essential in vector field visualization [Jobard and Lefer 1997; Mebarki et al. 2005; Spencer et al. 2009] and nonphotorealistic rendering [Hertzmann and Zorin 2000]. Methods based on reaction-diffusion equations can generate stripe patterns aligned with nonorientable features [Turk 1991; Witkin and Kass 1991], but must solve the full time evolution of a nonlinear parabolic PDE whose parameters can be difficult to control; moreover, these methods solve directly for the color function rather than a parameterization, precluding more complex shaders like Fig. 2, right. Image-based texture synthesis can also produce patterns aligned with orientable [Praun et al. 2000; Lefebvre and Hoppe 2006] and nonorientable vector fields [Ying et al. 2001; Wei and Levoy 2001; Turk 2001], but features in such patterns result from, *e.g.*, statistical similarity of image features rather than geometric phenomena. Input to these methods can be generated using a variety of techniques [Hertzmann and Zorin 2000; Palacios and Zhang 2007; Fisher et al. 2007; Ray et al. 2008; Bommes et al. 2009; Crane et al. 2010; Knöppel et al. 2013; Diamanti et al. 2014].

Field-aligned parameterization is also a key component of global remeshing algorithms, where there are two popular representations: either angle-valued coordinate functions with discrete *period jumps* [Bommes et al. 2009], or vector-valued functions whose angular components provide the final coordinates [Ray et al. 2006; Zhang et al. 2010]. Both choices traditionally lead to difficult nonconvex problems: the former due to integer variables; the latter due to a quartic unit-norm constraint at each point. The unit constraint implicit in both representations also leads to energies that do not converge under refinement and are therefore unstable with respect to tessellation [Knöppel et al. 2013, Fig. 4]. Nonorientable features are handled using either *matchings* [Ray et al. 2006; Tong et al. 2006; Bommes et al. 2009], or a *branched covering* [Kälberer et al. 2007; Kälberer et al. 2010], though in retrospect these two approaches are essentially equivalent: matchings determine a covering space, and optimization on a covering is no more costly than on the original domain, as long as one respects *conjugate symmetry* (App. A). The advantage of the covering space perspective is that it allows for a meaningful formulation in the smooth setting (Sec. 2.3), helps to simplify implementation (Sec. 4.1), and clarifies interpolation—especially near branch points (Sec. 4.3). Finally, several quadrangulation methods automatically insert singularities, but cannot be used for stripe patterns since they do not align to a given field [Ling et al. 2014] or do not generalize to arbitrary *n*-direction fields [Myles and Zorin 2012; Myles and Zorin 2013]; moreover, these methods do not guarantee anything about global optimality.

Our method adopts the same energy as Ray *et al.* [2006], but differs in several important ways: (i) dropping quartic (unit-norm) constraints improves performance by about 10x (Sec. 5.1); (ii) permitting new singularities during parameterization allows us to directly handle nonintegrable input (Sec. 2.2); (iii) a new sub-triangle interpolation scheme (Sec. 4.3) means that pattern frequency is not limited by mesh resolution. We also provide an interpretation of the energy in the smooth setting (Sec. 2) and show new applications to texture synthesis and design (Sec. 5.3).

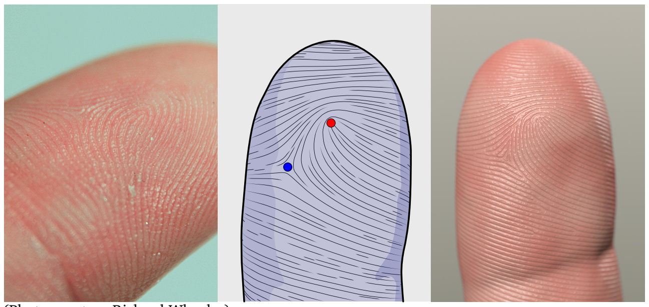

(Photo courtesy Richard Wheeler)

Figure 5: *We can mimic real fingerprints* (left) *by using a field with prescribed singularities* (center) *to guide our stripe pattern* (right).

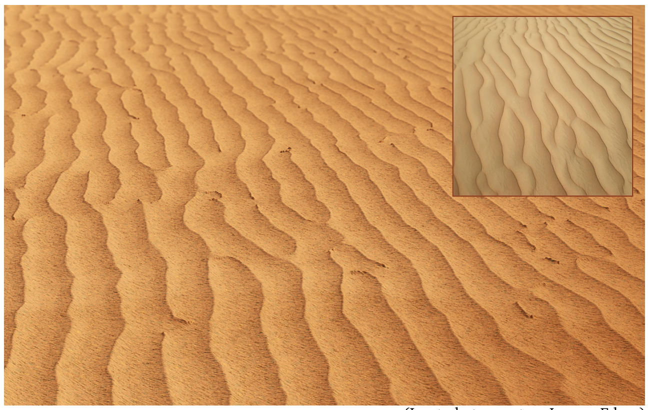
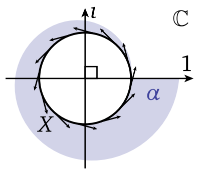

(Inset photo courtesy Lauren Edgar)

Figure 7: *Randomly perturbing a constant vector field yields a pattern reminiscent of real* aeolian wind ripples (inset)*, here rendered as a displacement map over a flat plane.*

2 Smooth Formulation

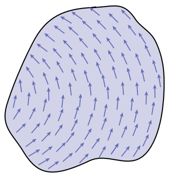
Given a surface M, we seek a scalar function α that varies along a given unit vector field X at a given rate ν. There are two principal challenges having to do with (i) the *integrability* of the vector field $Z := \nu X$ and (ii) the representation of α itself. Roughly speaking, Z is integrable if it is locally expressible as the gradient of a scalar potential, but if we simply adopt a *real*-valued representation then we may not be able to find a potential that is globally continuous. Later, we also address *nonorientable* patterns by formulating the same problem on a *double cover* of M (Sec. 2.3). Throughout we use $\langle\cdot,\cdot\rangle$ and $|\cdot|$ to denote the real inner product and norm on vectors, $||\cdot||$ for the L^2 norm on functions, dA for the usual area measure, and $\imath$ for the imaginary unit. We also make occasional use of (discrete) differential forms—see Crane *et al.* [2013, Chapter 3] for a brief introduction.

2.1 Periodic Functions



A key challenge in computing globally continuous parameterizations is choosing a suitable representation for the coordinate functions. As a motivating example, consider a smooth unit tangent field X on the circle $\alpha \mapsto (\cos\alpha, \sin\alpha)$. Locally we can think of X as the derivative of the angle α, but globally it cannot be expressed as the derivative of a smooth function since any such function would have to increase indefinitely as we walk multiple times around the circle. We therefore adopt an alternative representation: rather than a real function α, we work with a complex function ψ whose argument $\arg\psi := \operatorname{atan2}(\operatorname{Im}\psi, \operatorname{Re}\psi) \in [-\pi,\pi)$ serves as a proxy for α. Although $\arg\psi$ is not globally continuous when viewed as a real-valued function, it still has a well-defined rate of change when viewed as a map to the unit circle, namely $d\arg\psi := \langle d\psi, i\psi\rangle/|\psi|^2$. Hence, we obtain a representation of angle that is globally continuous and differentiable away from zeros (in the case of the circle, we have just $\psi = \cos\alpha + \imath\sin\alpha$). For surfaces, nothing changes except that the domain of ψ is now a surface rather than a curve. Importantly, we do not need to cut the surface into a topological disk since coordinates are globally continuous *by construction*.

2.2 Integrability



Vector fields with nonzero curl are *nonintegrable*, *i.e.*, even on small, simply-connected patches they cannot be expressed as the gradient of any scalar function. A common technique is to reduce or eliminate curl prior to parameterization via *curl correction* [Ray et al. 2006] or *Helmholtz-Hodge decomposition* [Kälberer et al. 2007]. We instead work directly with nonintegrable input by allowing new singularities during parameterization—empirically, these singularities tend to appear in regions where there is significant curl, or where stripe spacing changes quickly (see Figs. 1, 3, 4, and 6). More explicitly, consider the complex function ψ (Sec. 2.1)—since we care only about its argument $\alpha := \arg\psi$ (and not its magnitude), we initially consider optimization over functions $\varphi := e^{\imath\alpha}$ with unit norm at each point. Conceptually, we want the angle α to change at rate ν along the direction X, and remain constant along the orthogonal direction. In other words, we want $d\alpha(Y) = \nu\langle X, Y\rangle$ for all vector fields Y, or equivalently, $d\alpha = \omega$ for $\omega := \langle \nu X, \cdot\rangle = \langle Z, \cdot\rangle$. Differentiating φ we get $d\varphi = \imath d\alpha\varphi$, which means we can achieve the desired angular velocity $d\alpha$ by solving

$$d\varphi = \imath\omega\varphi. \tag{1}$$

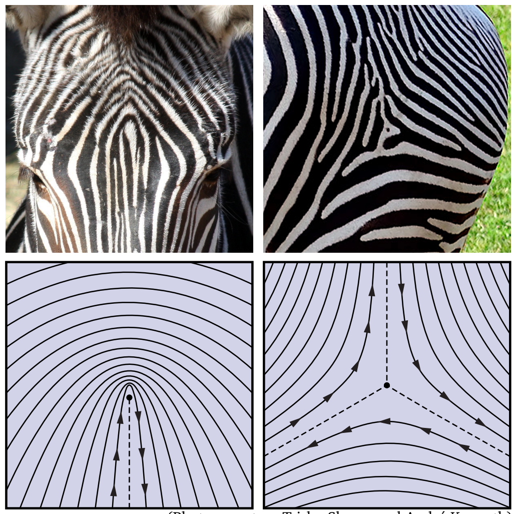

(Photos courtesy Trisha Shears and André Karwath)

Figure 8: *Naturally-occurring stripe patterns* (top) *often include* nonorientable singularities*,* i.e.*, features that do not follow any continuously varying vector field* (bottom).

Unless Z is exactly integrable, this equation admits only the trivial solution $\varphi \equiv 0$. We therefore consider its L^2 residual

$$\mathcal{E}(\varphi) := \int_M |(d - \imath\omega)\varphi|^2. \tag{2}$$

Since the operator $\tilde{\nabla} := d - \imath\omega$ defines a *connection* on M, $\mathcal{E}$ can be viewed as a *Dirichlet energy* on complex functions. However, this energy is not well-defined for unit functions φ that encode the type of features we observe in natural stripe patterns (*e.g.*, Fig. 2, left), since the angular velocity $d\alpha$ approaches infinity as we approach an edge dislocation. Following the approach advocated in Knöppel *et al.* [2013, Sec. 3], we therefore define the energy of a *unit* function φ as the minimum Dirichlet energy among all possible rescalings by a real-valued function $a \geq 0$:

$$\hat{\mathcal{E}}(\varphi) := \min_{a\geq 0, ||a||=1} \int_M |\tilde{\nabla}(a\varphi)|^2 \, dA.$$

The constraint $||a|| = 1$ prevents the solution $a \equiv 0$. Minimizing this energy over unit functions φ is then equivalent to minimizing Dirichlet energy over *all* complex functions ψ:

$$\min_{|\varphi|=1} \hat{\mathcal{E}} \quad = \min_{|\varphi|=1, a\geq 0, ||a||=1} \mathcal{E}(a\varphi) \quad = \min_{||\psi||=1} \mathcal{E}(\psi). \tag{3}$$

In the case of parameterization, it is precisely the ability to scale ψ down to zero that allows us to form new singularities, thereby improving alignment in nonintegrable regions. Just as with standard Dirichlet energy, minimizers of this energy among functions ψ with unit L^2-norm are eigenfunctions associated with the smallest eigenvalue of the corresponding Laplace operator [Mullen et al. 2008]—discretization of this operator is discussed in Sec. 3.

2.3 Nonorientable Patterns

The formulation above is suitable in the orientable case, but in general we want to produce stripe patterns with *nonorientable* features, which often appear in nature (Fig 8). Another common geometric example is a *principal direction field*—see for instance Fig. 17, left. In general, such features can be encoded as a *line field*, *i.e.*, a choice of unoriented line in each tangent plane. A

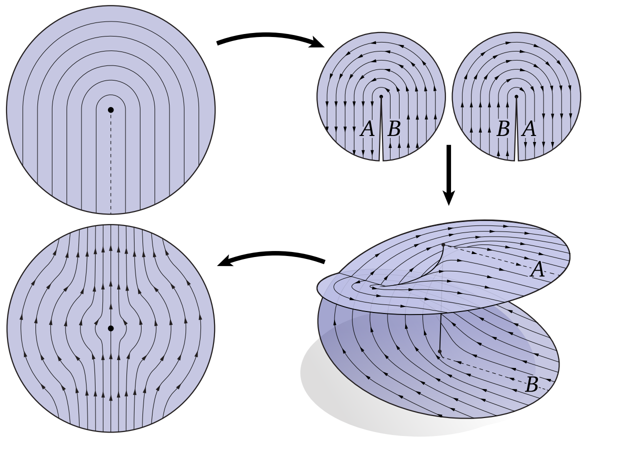


Figure 9: *To deal with a nonorientable singularity* (top left)*, one can make two oppositely oriented copies* (top right)*, and cut and glue along opposite edges to obtain a branched double cover* (bottom right)*. Untwisting this figure into the plane reveals that the new vector field has a globally consistent orientation* (bottom left).

useful way to work with such fields, pioneered in geometry processing by Kälberer *et al.* [2007; 2010], is as a vector field on a *branched double cover* of the original surface. Consider for instance Fig. 9 (top left)—here it is impossible to give the innermost stripe an orientation that is consistent across the dashed line. Yet by gluing together two oppositely-oriented copies of the surface, we obtain a consistently-oriented vector field $\widetilde{Z}$ on a new surface $\widetilde{M}$ called the *double cover*. Away from singularities, a double cover looks like two identical copies of the original surface M (Fig. 10). In practice we never explicitly construct the cover, but will use this mental image to turn algorithms which are naturally expressed on $\widetilde{M}$ into algorithms that can be implemented using standard data structures on the original surface M.

The key idea behind this construction is that features in a given line field determine the topology of a corresponding covering space. In particular, suppose that lines vary continuously except at a collection of isolated *branch points* $\{p_1, \ldots, p_k\}$ around which the field is nonorientable (Fig. 8). At each such point the "trick" demonstrated in Fig. 9 can be used to produce an orientable vector field $\widetilde{Z}$ on the double cover $\widetilde{M}$ of $\widehat{M} := M \setminus \{p_1, \ldots, p_k\}$. Away from these points the double cover has two points $x_1, x_2 \in \widetilde{M}$ "above" each point $x \in \widehat{M}$ which are mapped back "down" to the base by the *projection map* $\pi(x_1) = \pi(x_2) = x$; the *sheet interchange* function $\tau(x_1) = x_2$ swaps these two points (Fig. 10).

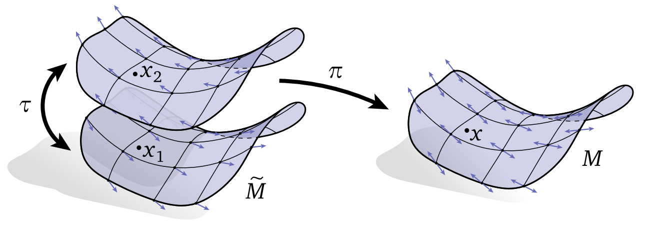


Figure 10: *Away from singularities, an unoriented* line field *on a patch of M can be expressed as a pair of oppositely and consistently-oriented vector fields on two identical copies of this patch in $\widetilde{M}$. The map τ switches between the two copies, and the map π takes us back down to the original surface.*

Conjugate Symmetry We now have an oriented vector field $\widetilde{Z}$ on a surface $\widetilde{M}$ which—at least conceptually—could be parameterized precisely as described in Sec. 2.3, since we can simply forget that this data comes from a double cover. However, we ultimately need texture coordinates on the base surface M. Suppose for a moment that the minimizer ψ on $\widetilde{M}$ is *conjugate symmetric*, meaning that it gets conjugated under sheet interchange:

$$\psi \circ \tau = \overline{\psi}. \tag{4}$$

The coordinate function $\alpha = \arg \psi$ is then *antisymmetric* with respect to sheet interchange, *i.e.*, $\alpha \circ \tau = -\alpha$, and plugging it into any *even* 2π-periodic function (like cosine) hence yields the same color value whether we use $\arg(\psi_{x_1})$ or $\arg(\psi_{x_2})$. An arbitrary minimizer ψ will not in general satisfy Eq. 4, but as shown in App. A, *one of the energy minimizers will always be conjugate-symmetric.* We can therefore always obtain a stripe pattern that is globally continuous away from isolated branch points. Moreover, the algorithm can be implemented on the base mesh without doubling the memory or computation cost, since we need only store a single value to represent both ψ and $\overline{\psi}$.

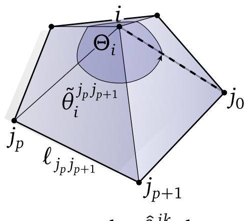
3 Discretization

As in the smooth setting we begin with oriented vector fields, then use a double cover to reduce the nonorientable case to the orientable one. The ultimate goal is to compute a value of $\alpha \in \mathbb{R}$ at each triangle corner—plugging interpolated values of α into any even periodic function will then produce a pattern of oscillating stripes orthogonal to X. The input to our algorithm is a manifold simplicial 2-complex K and positive edge lengths ℓ_{ij} satisfying the triangle inequality on each face, along with a unit tangent vector X_i and target line frequency $\nu_i \in \mathbb{R}^+$ which together define a (not necessarily unit) vector $Z_i := \nu_i X_i$ at each vertex $i \in V$. Overall, the algorithm consists of four basic steps:

1. Project Z onto each edge to get angular displacements ω.
2. Build a Laplace-like matrix A with entries determined by ω.
3. Find an eigenvector ψ of A with smallest eigenvalue.
4. In each face, compute texture coordinates α from ψ and ω.

Notation For any simplicial complex K, we use V, E, F to denote its vertices, edges, and faces, respectively. Individual simplices are expressed as a list of incident vertices, *e.g.*, $ijk \in F$ denotes the face with vertices $i, j, k \in V$. The orientation of a simplex is determined by the order of its indices, and an edge $ij \in E$ is *canonically oriented* if $i < j$. An expression of the form $x_i = \sum_{ij \in E} y_{ij}$ indicates that a value x_i associated with vertex i is obtained by summing the quantity y_{ij} over all edges incident on i. Likewise, $x_i = \sum_{ijk \in F} y_{ijk}$ denotes a sum over all incident faces.

3.1 Discrete Tangent Spaces



Rather than thinking of tangent vectors X_i as elements of $\mathbb{R}^3$, we view them as angles $\phi_i \in [-\pi, \pi)$ relative to a reference edge $ij_0 \in E$ chosen at each vertex $i \in V$, effectively adopting polar coordinates. This intrinsic setup simplifies the rest of our algorithm, and does not require us to define normals at vertices. Moreover, let $\hat{\theta}_i^{jk}$ denote the angle at corner i of triangle ijk, and let $\Theta_i := \sum_{ijk \in F} \hat{\theta}_i^{jk}$ be the sum of all angles incident on vertex i. For each oriented edge ij_a incident on i, we store the angle

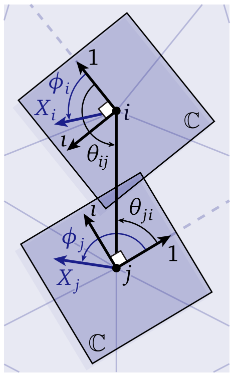
$$\theta_{ij_a} := \tfrac{2\pi}{\Theta_i}\sum_{p=0}^{a-1}\hat\theta_i^{j_p j_{p+1}},$$

where the j_p are indexed counter-clockwise around vertex i, and $p+1$ is taken modulo the degree of the vertex. These rescaled angles are effectively the polar coordinates of the edges, starting at $\theta_{ij_0} = 0$ (see Knöppel *et al.* [2013, Sec. 6] for further interpretation). Hence, given any unit vector X_i encoded by an angle ϕ_i, the parallel vector at a neighboring vertex j can be computed via

$$\phi_i \mapsto \phi_i \underbrace{-\theta_{ij} + (\theta_{ji}+\pi)}_{=:\rho_{ij}}, \tag{5}$$

i.e., by using the shared edge ij as a common frame of reference (see inset below). For convenience, we store the values ρ_{ij}, noting that $\rho_{ji} = -\rho_{ij}$.

Complex Representation In many cases, we will instead encode tangent vectors as complex numbers to avoid the difficulty of comparing angles modulo 2π. An expression like Eq. 5 can then be written as $X_i \mapsto e^{\imath\rho_{ij}}X_i$, which in complex arithmetic expresses a counter-clockwise rotation of X_i by the angle ρ_{ij}. It is important to note, however, that these complex values still only carry meaning relative to the reference edge chosen at each vertex, which we identify with the real axis $1 + 0\imath \in \mathbb{C}$.



3.2 Discrete Energy

As discussed in Sec. 2.2, we ideally want an angle-valued function α such that $d\alpha = \omega$, *i.e.*, such that the angular velocity $d\alpha$ matches our input field. To discretize this relationship, we linearly interpolate Z and integrate along each edge $ij \in E$, yielding

$$\alpha_j - \alpha_i = \int_{ij} d\alpha = \int_{ij}\omega = \tfrac{1}{2}(\langle e_{ij}, Z_i\rangle + \langle e_{ij}, Z_j\rangle) =: \omega_{ij}. \tag{6}$$

The resulting quantity $\omega_{ij} = -\omega_{ji} \in \mathbb{R}$ determines a *discrete 1-form* [Crane et al. 2013, Sec. 3.6] that describes the target angular displacement as we move from i to j; we will use

$$P_{ij}(X) := e^{\imath\omega_{ij}}X$$

to denote the transport of a vector X from i to j via ω. If extrinsic data is available (*i.e.*, vectors and edges in $\mathbb{R}^3$) then this quantity can be computed as above. If only intrinsic data is available (*i.e.*, lengths and angles), one can just as easily write ω as

$$\omega_{ij} = \tfrac{1}{2}\ell_{ij}(\nu_i\cos(\phi_i - \theta_{ij}) + \nu_j\cos(\phi_j - \theta_{ji})). \tag{7}$$

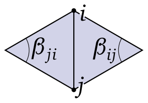
Recalling that $\arg\psi$ is used as a proxy for α (Sec. 2.1), we therefore ask that the rotated value of ψ_i agrees exactly with ψ_j, providing a discrete version of Eq. (1):

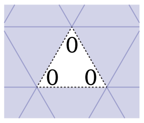
$$e^{\imath\omega_{ij}}\psi_i = \psi_j \quad \forall ij \in E.$$

As in the smooth setting we cannot expect to satisfy this relationship exactly, since not every discrete vector field is integrable. Hence, we instead minimize the L^2 residual

$$\mathcal{E}(\psi) := \sum_{ij\in E} w_{ij}|\psi_j - e^{\imath\omega_{ij}}\psi_i|^2, \tag{8}$$

where $w_{ij} := (\cot\beta_{ij} + \cot\beta_{ji})/2$ and β_{ij}, β_{ji} are the two angles opposite edge ij; these *cotangent weights* account for the shape of mesh elements [MacNeal 1949, Sec. 3.2]. For boundary edges we set the unknown cotan to zero, corresponding to zero-Neumann boundary conditions. Likewise, we effectively omit branch triangles (Sec. 3.3) by simply setting their weights to zero.





3.3 Discrete Double Cover

To handle nonorientable input, we now define the double cover $\widetilde{K} = (\widetilde{V}, \widetilde{E}, \widetilde{F})$ induced by vectors Z_i at the vertices of our original mesh K, as well as an *orientable* vector field $\widetilde{Z}$ on $\widetilde{K}$ (Fig. 11). Note that at no point do we actually *build* $\widetilde{K}$—this description merely serves to clarify implementation.

Vertices: For each vertex $i \in V$, $\widetilde{V}$ contains a pair of vertices i_1, i_2, which we associate with vectors Z_i and $-Z_i$, respectively.

Edges: For each edge $ij \in E$, let the quantity

$$s_{ij} := \begin{cases} +1 & \text{if } \phi_j - (\phi_i + \rho_{ij}) \in [-\pi/2, \pi/2), \\ -1 & \text{otherwise} \end{cases}$$

encode the relative orientation of Z_i and Z_j. If $s_{ij} = +1$ then $\widetilde{E}$ contains edges i_1j_1 and i_2j_2; otherwise, it contains i_1j_2 and i_2j_1.

Faces: For each face $ijk \in F$, let $s_{ijk} := s_{ij}s_{jk}s_{ki}$. If $s_{ijk} = +1$ then ijk is *regular*, and the six corresponding edges in $\widetilde{E}$ form two distinct triangles which we include in $\widetilde{F}$. Otherwise, ijk is a *branch triangle* and these edges form a hexagon which we omit from $\widetilde{F}$. We will use $B \subset F$ to denote the set of branch triangles.

We also define a 1-form $\widetilde{\omega}$ on $\widetilde{K}$ corresponding to $\widetilde{Z}$, given by

$$\begin{aligned} \widetilde{\omega}_{i_1j_2} &= -\widetilde{\omega}_{i_2j_1} = \tfrac{1}{2}(\langle e_{ij}, Z_i\rangle - \langle e_{ij}, Z_j\rangle) & \text{if } s_{ij} = -1, \text{ and} \\ \widetilde{\omega}_{i_1j_1} &= -\widetilde{\omega}_{i_2j_2} = \tfrac{1}{2}(\langle e_{ij}, Z_i\rangle + \langle e_{ij}, Z_j\rangle) & \text{if } s_{ij} = +1, \end{aligned}$$

along with a corresponding parallel transport map $\widetilde{P}_{pq}(Z) := e^{\imath\widetilde{\omega}_{pq}}Z$. Notice that $\widetilde{\omega}\circ\tau = -\widetilde{\omega}$, *i.e.*, switching sheets flips the sign of $\widetilde{\omega}$. For convenience, we define

$$\widehat{\omega}_{ij} = \begin{cases} \widetilde{\omega}_{i_1j_2}, & \text{if } s_{ij} = -1, \\ \widetilde{\omega}_{i_1j_1}, & \text{if } s_{ij} = +1, \end{cases}$$

and use $\hat{P}_{ij}(Z) := e^{\imath\widehat{\omega}_{ij}}Z$ to denote parallel transport by $\widehat{\omega}$.

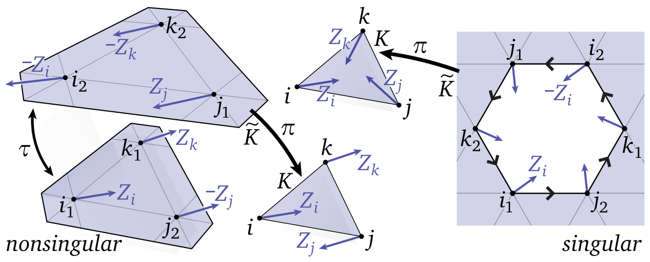


Figure 11: *The double cover of a triangle is determined by the relative orientation of vectors Z along each edge.* Left: *If the vector field can be locally oriented, we simply make two copies of the original triangle.* Right: *Otherwise we have a* branch triangle, *and use the six corresponding edges to form a hexagonal boundary cycle.*

3.4 Conjugate Symmetric Energy

We now formulate an expression for the discrete energy (Eq. 8) on $\widetilde{K}$, restricted to conjugate-symmetric functions $\widetilde{\psi}$ (Sec. 2.3). To do so, we store a single value $\psi_i \in \mathbb{C}$ at each vertex $i \in V$ of our original mesh, and define (but do not explicitly store)

$$\widetilde{\psi}_{i_1} := \psi_i, \qquad \widetilde{\psi}_{i_2} := \overline{\psi}_i.$$

For a given edge $ij \in E$, one then observes that the two corresponding edges on the double cover make identical contributions to the energy. For example, when $s_{ij} = +1$, we have terms

$$w_{ij}|\psi_j - e^{\iota\widetilde{\omega}_{i_1j_1}}\psi_i|^2 \qquad \text{and} \qquad w_{ij}|\overline{\psi}_j - e^{\iota\widetilde{\omega}_{i_2j_2}}\overline{\psi}_i|^2,$$

which are identical in magnitude since $e^{\iota\widetilde{\omega}_{i_2j_2}} = e^{-\iota\widetilde{\omega}_{i_1j_1}} = \overline{e^{\iota\widetilde{\omega}_{i_1j_1}}}$, and conjugation does not affect the norm (likewise for $s_{ij} = -1$). The energy associated with a given edge $ij \in E$ is therefore

$$\tilde{\mathcal{E}}_{ij} := \begin{cases} 2w_{ij}|\overline{\psi}_j - e^{\iota\widehat{\omega}_{ij}}\psi_i|^2, & s_{ij} = -1, \\ 2w_{ij}|\psi_j - e^{\iota\widehat{\omega}_{ij}}\psi_i|^2, & s_{ij} = +1, \end{cases} \tag{9}$$

yielding a total energy $\tilde{\mathcal{E}} := \sum_{ij\in E}\tilde{\mathcal{E}}_{ij}$. Notice that this energy is defined entirely in terms of a single value ψ per vertex and a single value $\widehat{\omega}$ per edge of our original mesh K: we do not need to double the degrees of freedom.

4 Implementation

We now describe essential aspects of implementation; detailed pseudocode can be found in App. C.

4.1 Matrix Representation

Although $\tilde{\mathcal{E}}$ is expressed using complex variables, it must be encoded as a real matrix in order to represent conjugation of the stored values of ψ, which is not a complex-linear operation. In particular, for complex numbers $u = a + b\iota$, $v = c + d\iota$ and $z = x + y\iota$ the following relations are easily verified:

$$\langle zu, v\rangle = (a\ b)[z](c\ d)^\mathsf{T}, \quad \langle zu, \overline{v}\rangle = (a\ b)\overline{[z]}(c\ d)^\mathsf{T},$$

where

$$[z] := \begin{pmatrix} x & y \\ -y & x \end{pmatrix} \qquad \text{and} \qquad \overline{[z]} := \begin{pmatrix} x & -y \\ -y & -x \end{pmatrix}.$$

Expanding Eq. 9 and applying these expressions then allow us to express $\tilde{\mathcal{E}}$ as a matrix $\mathsf{A} \in \mathbb{R}^{2|V|\times 2|V|}$ with diagonal blocks

$$\mathsf{A}_{ii} = \sum_{ij\in E}[w_{ij}]$$

for each vertex $i \in V$, and off-diagonal blocks

$$\mathsf{A}_{ij} = \begin{cases} -w_{ij}\,\overline{[e^{\iota\widehat{\omega}_{ij}}]}, & s_{ij} = -1 \\ -w_{ij}\,[e^{\iota\widehat{\omega}_{ij}}], & s_{ij} = +1 \end{cases}, \qquad \mathsf{A}_{ji} = \mathsf{A}_{ij}^\mathsf{T}$$

for each canonically oriented edge $ij \in E$. The resulting matrix is symmetric positive-definite with the same structure as the usual cotan-Laplace matrix. Finally, letting $\mathcal{A}_{ijk}$ denote the area of triangle ijk, we encode the usual L^2 norm on functions via the block diagonal lumped mass matrix $\mathsf{B} \in \mathbb{R}^{2|V|\times 2|V|}$ with entries

$$\mathsf{B}_{ii} = \tfrac{1}{3}\sum_{ijk\in F}\mathcal{A}_{ijk},$$

i.e., one-third the total area of triangles incident on vertex i.

4.2 Global Optimization

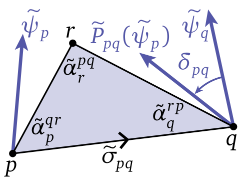
We now seek values $\psi_i = a_i + b_i\iota$ that minimize $\tilde{\mathcal{E}}$, which we encode as a vector $\mathsf{x} \in \mathbb{R}^{2|V|}$ of interleaved values $a_i, b_i \in \mathbb{R}$. To avoid the trivial solution $\psi \equiv 0$, we ask that ψ have unit L^2 norm—using the matrices above, this problem can be stated as

$$\min_{\mathsf{x}\in\mathbb{R}^{2|V|}} \mathsf{x}^\mathsf{T}\mathsf{A}\mathsf{x} \quad \text{s.t.} \quad \mathsf{x}^\mathsf{T}\mathsf{B}\mathsf{x} = 1. \tag{10}$$

Differentiating the Lagrangian of this problem yields the generalized eigenvalue problem $\mathsf{A}\mathsf{x} = \lambda\mathsf{B}\mathsf{x}$ for an eigenvector x associated with the smallest eigenvalue λ, which we solve by factoring A and applying the inverse power method (Alg. 6). Hence, the total cost of our algorithm is dominated by a single matrix factorization.

4.3 Texture Coordinates

After computing ψ, we still need to extract our final texture coordinates α. However, simply using the values $\arg(\psi_i)$ is less than ideal since (i) we will experience aliasing if the target frequency ν is greater than the mesh spacing (Fig. 12) and (ii) linear interpolation will produce discontinuities along the boundaries of triangles containing zeros or branch points (Fig. 13, 14). We therefore adjust α as described below. On $\widetilde{K}$ these coordinates are antisymmetric with respect to sheet interchange and encode a globally continuous map to the unit circle, as shown in App. B. Hence, for any given base triangle we can use coordinates from either of the two covering triangles to obtain a globally continuous stripe pattern. Note that for sufficiently fine meshes one can often ignore singular triangles and simply use linear interpolation everywhere, as done in Figs. 2, 5, 19, 20, 21, and 7.



The Spinning Form Along each edge $pq \in \widetilde{E}$ we had a target angular displacement $\widetilde{\omega}_{pq}$, which in general we cannot achieve due to nonintegrability. The *spinning form* $\widetilde{\sigma}$ is the angular displacement closest to $\widetilde{\omega}$ that agrees with the obtained minimizer of $\widetilde{\mathcal{E}}$, *i.e.*, for which $\arg(e^{\iota\widetilde{\sigma}_{pq}}\widetilde{\psi}_p) = \arg(\widetilde{\psi}_q)$. These values will be used to adjust our texture coordinates. In particular, let

$$\delta_{pq} := \arg(\widetilde{P}_{pq}(\widetilde{\psi}_p)/\widetilde{\psi}_q)$$

be the smallest angular distance from the obtained vector $\widetilde{\psi}_q$ at vertex q to the desired vector $\widetilde{P}_{pq}(\widetilde{\psi}_p)$. Then

$$\widetilde{\sigma}_{pq} := \widetilde{\omega}_{pq} - \delta_{pq}.$$

Note that, like $\widetilde{\omega}$, $\widetilde{\sigma}$ is a discrete 1-form on $\widetilde{K}$, *i.e.*, $\widetilde{\sigma}_{pq} = -\widetilde{\sigma}_{qp}$.

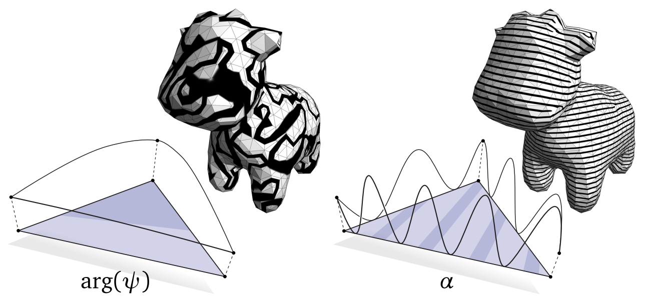


Figure 12: Left: *for coarse meshes or high frequency stripes, naïve interpolation of angles leads to severe aliasing.* Right: *by augmenting the angles according to the target frequency, we can render stripes far above the resolution of the mesh.*

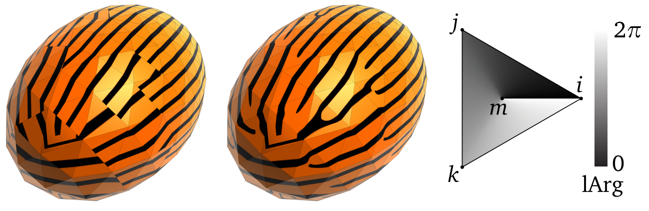


Figure 13: *In triangles containing zeros, piecewise linear interpolation* (left) *can be substantially improved* (middle) *by using a closed-form, nonlinear interpolant* (right).

Frequency Adjustment We first adjust the texture coordinates locally in each triangle to properly account for the target frequency ν. More explicitly, let $\tilde{\alpha}_p^{qr}$ denote the coordinate at corner p of a triangle $pqr \in \widetilde{F}$. We let $\tilde{\alpha}_p^{qr} := \arg(\widetilde{\psi}_p)$ at the first corner, and use the spinning form to define coordinates at the other two corners: $\tilde{\alpha}_q^{rp} := \tilde{\alpha}_p^{qr} + \widetilde{\sigma}_{pq}$ and $\tilde{\alpha}_r^{pq} := \tilde{\alpha}_p^{qr} + \widetilde{\sigma}_{pr}$. To render a given base triangle $ijk \in F$ we need only extract the coordinates from one of its two covering triangles, which we can do using the values ψ and $\widehat{\omega}$ stored on our original mesh K. In particular, if we let

$$\hat{\delta}_{ij} := \begin{cases} \arg(\hat{P}_{ij}(\psi_i)/\overline{\psi}_j), & s_{ij} = -1, \\ \arg(\hat{P}_{ij}(\psi_i)/\psi_j), & s_{ij} = +1, \end{cases}$$

and $\hat{\sigma} := \hat{\omega} + \hat{\delta}$, then our local coordinates are just

$$\begin{aligned} \alpha_i^{jk} &:= \arg(\psi_i), \\ \alpha_j^{ki} &:= \arg(\psi_i) + \hat{\sigma}_{ij}, \\ \alpha_k^{ij} &:= \arg(\psi_i) + \hat{\sigma}_{ik}. \end{aligned}$$

In nonsingular regions, α can then be interpolated linearly over each triangle. In general, however, we must account for singularities due to either (i) zeros of the function ψ or (ii) branch points arising from the field Z; each of these cases is treated below.

Zeros As we walk around the boundary of any regular triangle $ijk \in F \setminus B$ covered by a triangle $i_1qr \in \widetilde{F}$, the spinning form describes a net change in angle of

$$\widetilde{\sigma}_{i_1q} + \widetilde{\sigma}_{qr} + \widetilde{\sigma}_{ri_1} =: 2\pi n_{ijk}$$

for some integer $n_{ijk} \in \mathbb{Z}$ which we refer to as the *index* of ψ. If $n_{ijk} \neq 0$, this means that the angle α on the base domain must "jump" somewhere and hence cannot be interpolated by any continuous function (including a linear one). We instead use an interpolant that mimics the behavior of the complex function $z \mapsto \arg(z)$, but remains linear along the boundary so that it agrees with interpolants in adjacent triangles. Explicitly, let

$$\mathrm{lArg}_n(t_i,t_j,t_k) := \begin{cases} \frac{\pi n}{3}\left(1 + \frac{t_j - t_i}{1-3t_k}\right), & t_k \le t_i \text{ and } t_k \le t_j, \\ \frac{\pi n}{3}\left(3 + \frac{t_k - t_j}{1-3t_i}\right), & t_i \le t_j \text{ and } t_i \le t_k, \\ \frac{\pi n}{3}\left(5 + \frac{t_i - t_k}{1-3t_j}\right), & t_j \le t_k \text{ and } t_j \le t_i, \end{cases} \quad (11)$$

where $t_i, t_j, t_k \in \mathbb{R}$ are barycentric coordinates (Fig. 13, right). One can easily check that the function lArg is piecewise linear along the boundary, and describes a continuous map to the unit circle away from an isolated singularity at the barycenter m. To get our final interpolant, we first subtract the value of $\mathrm{lArg}_{n_{ijk}}$ from our texture coordinates α_i^{jk} at all three triangle corners (Alg. 7, lines 22–24). Texture coordinates are then computed by evaluating Eq. 11 in a fragment shader and adding the result to the usual linearly interpolated values of α.

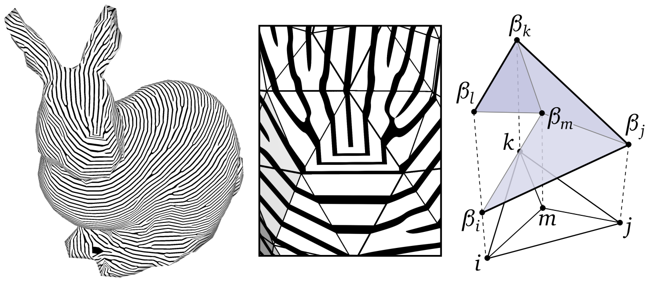


Figure 14: Left: *stripe pattern on a coarse mesh of 708 vertices.* Center: *tip of the nose with a branch point in the middle and nearby triangles containing zeros of index* 1 *and* 3. Right: *piecewise linear texture coordinates used to draw branch points.*

Branch Points Finally, we extend our texture coordinates to any branch triangle $ijk \in B$, which on the double cover $\widetilde{K}$ corresponds to a hexagonal boundary cycle (p,q,r,s,t,u) starting at $p = i_1$ (Fig. 11, right). Conceptually, we imagine triangulating the hexagon by inserting a new vertex $\widetilde{m}$ at the middle and extending coordinates linearly over each face. In practice, we simply draw the barycentric subdivision of the base triangle ijk (Fig. 14, right) using texture coordinates

$$\begin{aligned} \beta_i &:= \arg(\psi_i) & \beta_j &:= \beta_i + \widetilde{\sigma}_{pq} \\ \beta_m &:= \beta_i + (\widetilde{\sigma}_{pq} + \widetilde{\sigma}_{qr} + \widetilde{\sigma}_{rs})/2 & \beta_k &:= \beta_j + \widetilde{\sigma}_{qr} \\ & & \beta_l &:= \beta_k + \widetilde{\sigma}_{rs} \end{aligned}$$

with linear interpolation in each sub-triangle. (See App. B for further discussion; an explicit implementation is given in Alg. 7.)

5 Results

5.1 Performance

We implemented our method in C++ using SuiteSparse [Chen et al. 2008]; performance was measured on a single core of a 2.6 GHz Intel Core i7. Typical run times were less than a second; for example, we required 591ms, 14ms, 804ms, and 386ms to generate patterns in Figs. 2, 12, 18, and 21 on meshes with 28k, 1k, 44k, and 39k faces (resp.). This level of performance makes it possible to interactively edit or even animate patterns. We also compared the performance of our eigenvector scheme to the quartic penalty scheme suggested by Ray *et al.* [2006, Sec. 2.5] using the same matrix for the quadratic term and their suggested solver parameters. Across a broad range of examples our eigenvector method performed over an order of magnitude faster (Fig. 15); moreover, the penalty scheme sometimes gets trapped in local minima, yielding unwanted distortion.

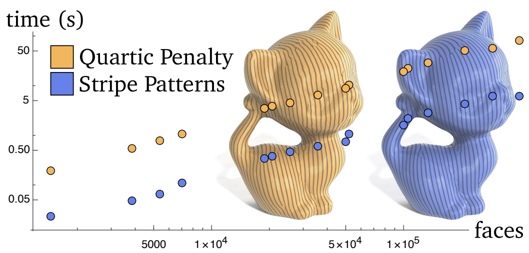


Figure 15: *Our method* (blue) *produces results of comparable quality to expensive, nonlinear optimization* (yellow)*, yet is on average 10.54 times faster across a broad collection of examples.*

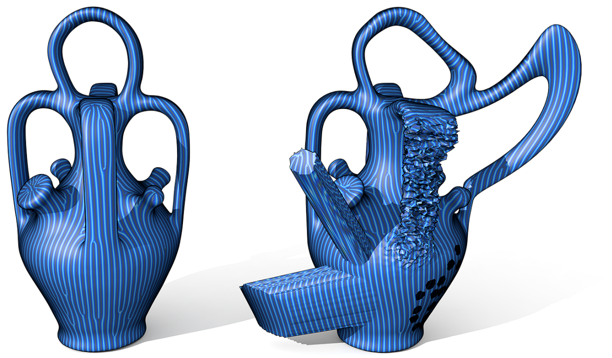


Figure 16: *Due to the elliptic nature of our problem, the method is robust to noise, holes, and other severe errors in the input.*

5.2 Robustness

In Fig. 16 we severely distort the input mesh, yet still obtain a well-behaved stripe pattern; this type of behavior is a hallmark of elliptic variational problems based on Laplace-like operators. The method also has no trouble handling sharp discontinuities in the input vector field Z (Fig. 6, right); likewise, outliers and noise in Z are handled gracefully (Fig. 7.)

5.3 Applications

Figures throughout show applications of stripe patterns to design and texture synthesis. For real-time hatching [Rost 2005], a precomputed stripe pattern pattern circumvents the need to trace integral curves, *etc.* (Fig. 17). In Fig. 21 we used marching triangles to generate geometric stripes. In Figs. 2 and 18, we combined two orthogonal stripe patterns to drive a 2D displacement map with bilateral symmetry across each axis. In Fig. 19 we mimic the artistic style of a ceramic mug. All of the preceding examples used optimally smooth or curvature-aligned fields as input [Knöppel et al. 2013], but in principle any technique can be applied—Fig. 4 shows an example where Z is described by artist-driven input; Fig. 5 uses the smoothest field with prescribed singularities [Crane et al. 2010] to emulate fingerprints.

6 Conclusion

Remarkably, many disparate natural phenomena are well-captured by one simple energy (Eq. 2), making stripe patterns a versatile tool for design. More broadly, stripe patterns contribute to a growing collection of tasks in geometry processing and simulation where singular features and sparse approximations are efficiently obtained by minimizing Dirichlet energy on a particular *Hermitian line bundle*—other recent examples include the design of smooth vector fields [Knöppel et al. 2013], extraction of smoke rings [Weißmann et al. 2014], and conformal volume deformation [Chern et al. 2015]. A better understanding of this phenomenon will no doubt lead to valuable future developments.

Acknowledgments

This work was supported by the DFG Collaborative Research Center TRR 109, "Discretization in Geometry and Dynamics," an NSF Mathematical Sciences Postdoctoral Research Fellowship (Award #1304254), and Intel. Meshes in Figs. 2, 14, 18, 22 are courtesy the Stanford Computer Graphics Lab; in Figs. 5, 15, 16, 17 courtesy AIM@Shape; and in Fig. 21 courtesy Martin Newell.

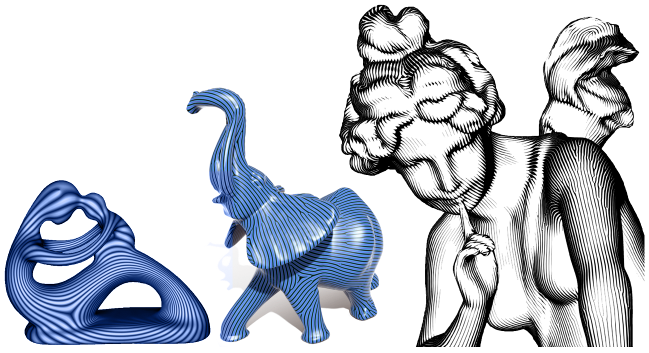


Figure 17: *Minimum principal curvature directions with either nonuniform* (left) *or uniform spacing* (center) *provide a natural pattern orientation.* Right: *this kind of pattern can be used to automatically drive a real-time hatching shader, requiring no user input apart from a single global scale parameter.*

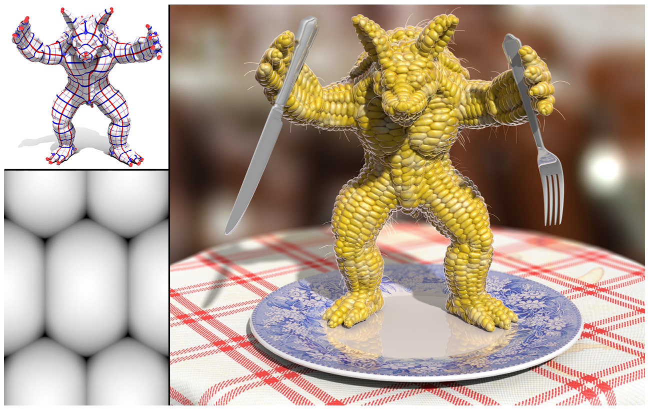
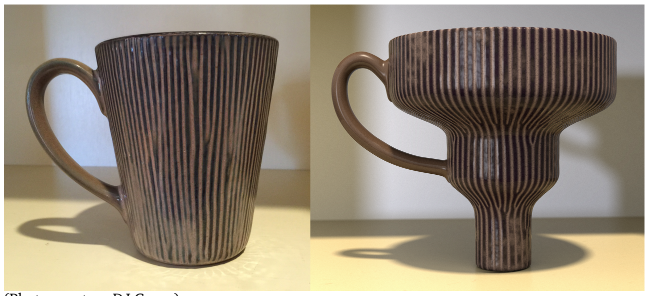
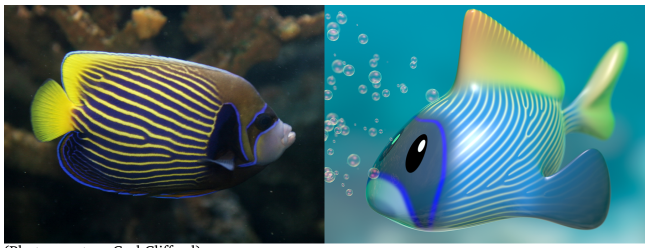


Figure 18: *Two orthogonal stripe patterns (red and blue) are computed separately and combined to drive a displacement map with bilateral symmetry across both axes.*



(Photo courtesy DJ Crane)

Figure 19: Left: *Photograph of a real mug with uniform, branching stripes.* Right: *Virtual mug synthesized using our method.*



(Photo courtesy Carl Clifford)

Figure 20: *Stripe patterns are ubiquitous in nature—here we caricature a real angelfish* (left) *using a stripe pattern* (right).

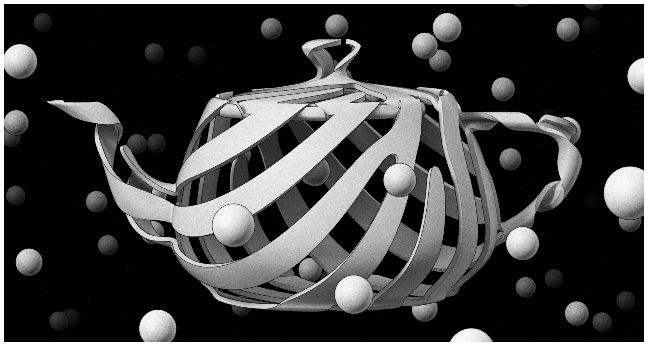


Figure 21: *Stripe patterns can also be used to achieve a variety of artistic effects, here inspired by the work of M.C. Escher.*

References

BOMMES, D., ZIMMER, H., AND KOBBELT, L. 2009. Mixed-Integer Quadrangulation. *ACM Trans. Graph. 28*, 3.

CHEN, Y., DAVIS, T. A., HAGER, W. W., AND RAJAMANICKAM, S. 2008. CHOLMOD, Supernodal Sparse Cholesky Factorization, and Update/Downdate. *ACM Trans. Math. Softw. 35*.

CHERN, A., PINKALL, U., AND SCHRÖDER, P. 2015. Close-to-Conformal Deformations of Volumes. *ACM Trans. Graph. 34*.

CRANE, K., DESBRUN, M., AND SCHRÖDER, P. 2010. Trivial Connections on Discrete Surfaces. *Comp. Graph. Forum 29*, 5.

CRANE, K., DE GOES, F., DESBRUN, M., AND SCHRÖDER, P. 2013. Digital Geometry Processing with Discrete Exterior Calculus. In *ACM SIGGRAPH 2013 courses*.

DIAMANTI, O., VAXMAN, A., PANOZZO, D., AND SORKINE-HORNUNG, O. 2014. Designing N-PolyVector Fields with Complex Polynomials. *Comp. Graph. Forum 33*, 5.

FISHER, M., SCHRÖDER, P., DESBRUN, M., AND HOPPE, H. 2007. Design of Tangent Vector Fields. *ACM Trans. Graph. 26*, 3.

HERTZMANN, A., AND ZORIN, D. 2000. Illustrating Smooth Surfaces. *Proc. ACM/SIGGRAPH Conf*.

JOBARD, B., AND LEFER, W. 1997. Creating Evenly-Spaced Streamlines of Arbitrary Density. In *Vis. Sci. Comp.*, W. Lefer and M. Grave, Eds.

KÄLBERER, F., NIESER, M., AND POLTHIER, K. 2007. QuadCover: Surface Parameterization using Branched Coverings. *Comp. Graph. Forum 26*, 3.

KÄLBERER, F., NIESER, M., AND POLTHIER, K. 2010. Stripe Parameterization of Tubular Surfaces. *Top. Meth. Data Anal. Vis*.

KALPAKJIAN, S., AND SCHMID, S. R. 2009. *Manufacturing Engineering and Technology*, 6th Ed. Prentice Hall, New York.

KNÖPPEL, F., CRANE, K., PINKALL, U., AND SCHRÖDER, P. 2013. Globally Optimal Direction Fields. *ACM Trans. Graph. 32*, 4.

LEFEBVRE, S., AND HOPPE, H. 2006. Appearance-Space Texture Synthesis. *ACM Trans. Graph. 25*, 3.

LING, R., HUANG, J., JÜTTLER, B., SUN, F., BAO, H., AND WANG, W. 2014. Spectral Quadrangulation with Feature Curve Alignment and Element Size Control. *ACM Trans. Graph. 34*, 1.

MACNEAL, R. 1949. *The Solution of Partial Differential Equations by means of Electrical Networks*. PhD thesis, Caltech.

MEBARKI, A., ALLIEZ, P., AND DEVILLERS, O. 2005. Farthest Point Seeding for Efficient Placement of Streamlines. *Proc. IEEE Vis*.

MULLEN, P., TONG, Y., ALLIEZ, P., AND DESBRUN, M. 2008. Spectral Conformal Parameterization. In *Proc. Symp. Geom. Proc.*

MYLES, A., AND ZORIN, D. 2012. Global Parameterization by Incremental Flattening. *ACM Trans. Graph. 31*, 4.

MYLES, A., AND ZORIN, D. 2013. Controlled-Distortion Constrained Global Parameterization. *ACM Trans. Graph. 32*, 4.

PALACIOS, J., AND ZHANG, E. 2007. Rotational Symmetry Field Design on Surfaces. *ACM Trans. Graph. 26*, 3.

PRAUN, E., FINKELSTEIN, A., AND HOPPE, H. 2000. Lapped Textures. In *Proc. ACM/SIGGRAPH Conf*.

RAY, N., LI, W. C., LÉVY, B., SHEFFER, A., AND ALLIEZ, P. 2006. Periodic Global Parameterization. *ACM Trans. Graph. 25*, 4.

RAY, N., VALLET, B., LI, W. C., AND LÉVY, B. 2008. N-Symmetry Direction Field Design. *ACM Trans. Graph. 27*, 2.

ROST, R. J. 2005. *OpenGL(R) Shading Language (2nd Edition)*. Addison-Wesley Professional.

SPENCER, B., LARAMEE, R. S., CHEN, G., AND ZHANG, E. 2009. Evenly Spaced Streamlines for Surfaces: An Image-Based Approach. *Comp. Graph. Forum 28*, 6.

TONG, Y., ALLIEZ, P., COHEN-STEINER, D., AND DESBRUN, M. 2006. Designing Quadrangulations with Discrete Harmonic Forms. In *Proc. Symp. Geom. Proc.*

TURK, G. 1991. Generating Textures on Arbitrary Surfaces using Reaction Diffusion. *Proc. ACM/SIGGRAPH Conf*.

TURK, G. 2001. Texture Synthesis on Surfaces. In *Proc. ACM/SIGGRAPH Conf.*

WEI, L.-Y., AND LEVOY, M. 2001. Texture Synthesis over Arbitrary Manifold Surfaces. In *Proc. ACM/SIGGRAPH Conf.*

WEISSMANN, S., PINKALL, U., AND SCHRÖDER, P. 2014. Smoke Rings from Smoke. *ACM Trans. Graph. 33*, 4.

WITKIN, A., AND KASS, M. 1991. Reaction-Diffusion Textures. *Proc. ACM/SIGGRAPH Conf*.

YING, L., HERTZMANN, A., BIERMANN, H., AND ZORIN, D. 2001. Texture and Shape Synthesis on Surfaces. *Proc. EG W. Rend.*

ZHANG, M., HUANG, J., LIU, X., AND BAO, H. 2010. Wave-Based Anisotropic Quadrangulation. *ACM Trans. Graph. 29*, 4.

A Conjugate Symmetry

Here we show that the energy $\mathcal{E}$ on $\widetilde{M}$ always has a global minimizer ψ that is *conjugate symmetric*, *i.e.*, such that

$$\psi = T\psi := \overline{\psi} \circ \tau.$$

Notice that $T(\iota\psi) = -\iota T\psi$, in other words, T is *complex antilinear*. Hence, we can decompose ψ as

$$\psi = \underbrace{\tfrac{1}{2}(\mathrm{Id}+T)\psi}_{=:\psi_1} + \tfrac{1}{2}(\mathrm{Id}-T)\psi = \psi_1 + \iota\,\underbrace{\left(\tfrac{1}{2}(\mathrm{Id}+T)(-\iota\psi)\right)}_{=:\psi_2},$$

i.e., $\psi = \psi_1 + \iota\psi_2$. Since $T^2 = \mathrm{Id}$, one easily checks that both ψ_1 and ψ_2 are conjugate symmetric and $T\psi = \psi_1 - \iota\psi_2$. Moreover, since ψ_1 and $\iota\psi_2$ are orthogonal ($(\mathrm{Id}-T)(\mathrm{Id}+T) = 0$) we get

$$\|\psi\|^2 = \|\psi_1\|^2 + \|\psi_2\|^2.$$

Further, $\mathcal{E}(\psi) = \mathcal{E}(T\psi)$, *i.e.*, conjugation does not change the energy. Applying the polarization identity then yields

$$\begin{aligned}\mathcal{E}(\psi) &= \tfrac{1}{2}(\mathcal{E}(\psi) + \mathcal{E}(T\psi)) \\ &= \tfrac{1}{2}(\mathcal{E}(\psi_1 + \iota\psi_2) + \mathcal{E}(\psi_1 - \iota\psi_2)) \\ &= \mathcal{E}(\psi_1) + \mathcal{E}(\psi_2).\end{aligned}$$

Suppose now that ψ with $\|\psi\|^2 = 1$ solves (Eq. (3)). Then for any other nonzero function ϕ we have $\mathcal{E}(\phi)/\|\phi\|^2 \geq \mathcal{E}(\psi)$, which when applied to ψ_1 (resp. ψ_2) implies

$$\mathcal{E}(\psi_1) \geq \mathcal{E}(\psi)\|\psi_1\|^2 \qquad \mathcal{E}(\psi_2) \geq \mathcal{E}(\psi)\|\psi_2\|^2.$$

But these inequalities must actually be *equalities*, otherwise summing them would yield a contradiction. Hence, there will always be some unit, conjugate symmetric function $\psi_1/\|\psi_1\|$ or $\psi_2/\|\psi_2\|$ *with the same energy as* ψ. In other words, restricting optimization to conjugate symmetric functions will still yield a global minimizer.

B Texture Coordinates

We now show that the coordinate function $\tilde{\alpha} : \widetilde{M} \to \mathbb{R}/2\pi\mathbb{Z}$ on the double cover is (i) antisymmetric with respect to sheet interchange, *i.e.*,

$$\tilde{\alpha} \circ \tau = -\tilde{\alpha}, \tag{12}$$

and (ii) encodes a globally continuous map to the unit circle, away from isolated singularities. Together, these properties guarantee that $\tilde{\alpha}$ can be used to draw a continuous stripe pattern. More formally, they ensure that for any even 2π-periodic function $u : \mathbb{R} \to \mathbb{R}$, there exists a function $c : K \setminus B \to \mathbb{R}$ such that away from branch triangles B,

$$u \circ \tilde{\alpha} = c \circ \pi,$$

i.e., $u(\tilde{\alpha})$ descends to some function c on the base. We will first define $\tilde{\alpha}$ on edges, and then extend it to triangle interiors.

To begin, let $\widetilde{\psi}$ be a minimizer of our discrete energy (Eq. 9), which is conjugate-symmetric by construction, and let $\varphi := \widetilde{\psi}/|\widetilde{\psi}|$. If $\widetilde{\sigma}$ is the corresponding *spinning form* (Sec. 4.3), one easily checks that $e^{\iota\widetilde{\sigma}_{pq}}\varphi_p = \varphi_q$ along any edge $pq \in \widetilde{E}$. Hence, for any path $\tilde{\gamma}$ from p to q along edges of $\widetilde{K}$, we have

$$e^{\iota\int_{\tilde{\gamma}}\widetilde{\sigma}}\varphi_p = \varphi_q, \tag{13}$$

where the integral is just the sum of $\widetilde{\sigma}$ over all the oriented edges of $\tilde{\gamma}$. From Eq. (13) it follows that the integral of $\widetilde{\sigma}$ over any closed path ($p = q$) has a value in $2\pi\mathbb{Z}$. Thus, if we integrate $\widetilde{\sigma}$ over the whole domain (starting at an arbitrary vertex), we obtain a piecewise-linear function $\tilde{\alpha}$ on edges such that

$$e^{\iota\tilde{\alpha}_p} = \varphi_p. \tag{14}$$

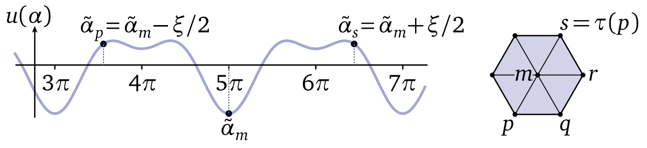
at each vertex p. By conjugate symmetry of $\widetilde{\psi}$, and since $\widetilde{\omega}$ changes sign under τ, we see that $\widetilde{\sigma}$ also changes sign under τ. Together with Eq. (14) this yields Eq. (12).

We now extend $\tilde{\alpha}$ to triangle interiors. For triangles $pqr \in \widetilde{F}$ where the index $\tilde{n}_{pqr} := (\widetilde{\sigma}_{pq} + \widetilde{\sigma}_{qr} + \widetilde{\sigma}_{rp})/2\pi$ equals zero (*i.e.*, where $\widetilde{\psi}$ is regular), we can simply extend $\tilde{\alpha}$ linearly—clearly Eq. (12) stays valid in this case. When $\tilde{n}_{pqr}$ is nonzero, we interpolate boundary values using the nonlinear function lArg described in Sec. 4.3. Here Eq. 12 follows from the behavior of $\widetilde{\sigma}$ under τ: changing sheets flips the sign of $\tilde{n}_{pqr}$ and thus of $\mathrm{lArg}_{\tilde{n}_{pqr}}$.

Finally, consider any branch triangle $ijk \in B$, which corresponds to a hexagonal boundary cycle on the double cover $\widetilde{K}$ (Sec. 3.3). We will extend $\tilde{\alpha}$ over a triangulation connecting each boundary vertex to a "middle" vertex m. In particular, consider a path $\tilde{\gamma} = (p, q, r, s)$ on $\widetilde{K}$ between vertices $\tau(p) = s$ that projects to the path $\gamma = (i, j, k, i)$ on K. Let $\xi := \int_{\tilde{\gamma}}\widetilde{\sigma}$, and assume $\tilde{\gamma}$ is oriented such that $\tilde{\alpha}_s = \tilde{\alpha}_p + \xi$. From Eq. (13) we have

$$e^{\iota\int_{\tilde{\gamma}}\widetilde{\sigma}}\varphi_p = \varphi_s = \overline{\varphi_p},$$

which implies that $\xi + \tilde{\alpha}_s = -\tilde{\alpha}_s + 2a\pi$ for some $a \in \mathbb{Z}$, or equivalently that the texture coordinate $\tilde{\alpha}_m := \tilde{\alpha}_s + \xi/2$ equals $a\pi$. Hence, $u(\tilde{\alpha}_p) = u(a\pi - \xi/2) = u(a\pi + \xi/2) = u(\tilde{\alpha}_s)$, which means that the piecewise linear extension of $\tilde{\alpha}$ satisfies Eq. 12.



$u(\alpha)$
$\tilde{\alpha}_p = \tilde{\alpha}_m - \xi/2$
$\tilde{\alpha}_s = \tilde{\alpha}_m + \xi/2$
3π
4π
5π
6π
7π
$\tilde{\alpha}_m$
$s = \tau(p)$
m
r
p
q

C Pseudocode

The input to the main algorithm StripePattern is a simplicial surface $K = (V, E, F)$, a collection of edge lengths $\ell \in \mathbb{R}^{|E|}$ satisfying the triangle inequality in each face, unit vectors $X \in \mathbb{C}^{|V|}$ describing the desired pattern orientation, and positive values $\nu \in \mathbb{R}^{|V|}$ giving the target line frequency. The output is a collection of coordinates $\alpha \in \mathbb{R}^{3|F|+2|B|}$ (one for each triangle corner, and two more for the midpoint m and duplicate vertex l of each branch triangle), along with indices $n_{ijk}, S_{ijk} \in \mathbb{Z}^{|F|}$ for each triangle indicating whether one should apply nonlinear interpolation or draw the barycentric subdivision (resp.).

Algorithm 1 The main stripe pattern algorithm (Sec. 2).

```
1: procedure StripePattern($K, \ell, X, \nu$)                ▷ $K = (V, E, F)$
2:     $\theta \leftarrow$ VertexAngles($K, \ell$)
3:     $\omega, s \leftarrow$ EdgeData($K, \ell, \theta, X, \nu$)
4:     $A \leftarrow$ EnergyMatrix($K, \ell, \omega, s$)
5:     $B \leftarrow$ MassMatrix($K, \ell$)
6:     $\psi \leftarrow$ PrincipalEigenvector($A, B$)
7:     $\alpha, n, S \leftarrow$ TextureCoordinates($K, \psi, \omega, s$)
8:     return $\alpha, n, S$
9: end procedure
```

Algorithm 2 Computes polar coordinates of outgoing halfedges around each vertex (Sec. 3.1).

```
1: procedure VertexAngles($K, \ell$)
2:     for each $i \in V$ do
3:         $\Theta_i \leftarrow 0$                              ▷ cumulative angle
4:         for $p \leftarrow \{0, \ldots, \text{Degree}(K, i) - 1\}$ do
5:             $\theta_{ij_p} \leftarrow \Theta_i$
6:             $\Theta_i \leftarrow \Theta_i + \text{TipAngle}(K, \ell, i, j_p, j_{p+1})$   ▷ returns $\hat{\theta}_i^{jk}$
7:         end for
8:         for $p \leftarrow \{0, \ldots, \text{Degree}(K, i) - 1\}$ do
9:             $\theta_{ij_p} \leftarrow 2\pi\theta_{ij_p}/\Theta_i$
10:        end for
11:    end for
12:    return $\theta$
13: end procedure
```

Algorithm 3 Initializes basic edge data (Sec. 3.1).

```
procedure EdgeData($K, \ell, \theta, X, \nu$)
    for each $ij \in E$ do                          ▷ canonically oriented ($i < j$)
        $\rho_{ij} \leftarrow -\theta_{ij} + \theta_{ji} + \pi$
        $s_{ij} \leftarrow \operatorname{sgn}(\langle e^{\iota\rho_{ij}} X_i, X_j \rangle)$
        $\phi_i \leftarrow \arg(X_i)$
        $\phi_j \leftarrow \arg(s_{ij} X_j)$
        $\omega_{ij} \leftarrow \frac{\ell_{ij}}{2}(\nu_i \cos(\phi_i - \theta_{ij}) + \nu_j \cos(\phi_j - \theta_{ji}))$
    end for
    return $\omega, s$
end procedure
```

Algorithm 4 Builds the matrix defining the energy (Sec. 4.1).

```
procedure EnergyMatrix($K, \ell, \omega, s$)
    $\mathsf{A} \leftarrow 0 \in \mathbb{R}^{2|V| \times 2|V|}$                ▷ start with all zeros
    for each $ij \in E$ do                          ▷ canonically oriented ($i < j$)
        $\beta_i, \beta_j \leftarrow$ OppositeAngles($K, \ell, ij$)
        $w_{ij} \leftarrow \frac{1}{2}(\cot\beta_{ij} + \cot\beta_{ji})$
        $\mathsf{A}_{ii} \leftarrow \mathsf{A}_{ii} + [w_{ij}]$
        $\mathsf{A}_{jj} \leftarrow \mathsf{A}_{jj} + [w_{ij}]$
        if $s_{ij} \geq 0$ then
            $\mathsf{A}_{ij} \leftarrow -w_{ij}[e^{\iota\omega_{ij}}]$
        else
            $\mathsf{A}_{ij} \leftarrow -w_{ij}\overline{[e^{\iota\omega_{ij}}]}$
        end if
        $\mathsf{A}_{ji} \leftarrow \mathsf{A}_{ij}^T$
    end for
    return A
end procedure
```

Algorithm 5 Builds the mass matrix associated with vertices (Sec. 4.1).

```
procedure MassMatrix($K, \ell$)
    $\mathsf{B} \leftarrow 0 \in \mathbb{R}^{2|V| \times 2|V|}$                ▷ start with all zeros
    for each $ijk \in F$ do
        $\mathcal{A}_{ijk} \leftarrow$ TriangleArea($K, \ell, ijk$)
        $\mathsf{B}_{ii} \leftarrow \mathsf{B}_{ii} + \mathcal{A}_{ijk}/3$
        $\mathsf{B}_{jj} \leftarrow \mathsf{B}_{jj} + \mathcal{A}_{ijk}/3$
        $\mathsf{B}_{kk} \leftarrow \mathsf{B}_{kk} + \mathcal{A}_{ijk}/3$
    end for
    return B
end procedure
```

Algorithm 6 Computes an eigenvector corresponding to the smallest eigenvalue via the inverse power method (Sec. 4.2).

```
procedure PrincipalEigenvector(A, B)
    L ← CholeskyFactor(A)
    x ← UniformRand(Size(A))
    for $i = 1, \ldots, N$ do
        x ← BackSolve(L, Bx)
        $\mathsf{x} \leftarrow \mathsf{x}/\sqrt{\mathsf{x}^\mathsf{T}\mathsf{B}\mathsf{x}}$
    end for
    return x
end procedure
```

Algorithm 7 Computes final texture coordinates. (Sec. 4.3).

```
procedure TextureCoordinates($K, \psi, \omega, s$)
    for each $ijk \in F$ do
        $c_{ij} \leftarrow i < j\ ?\ 1 : -1$                  ▷ is each edge canonical?
        $c_{jk} \leftarrow j < k\ ?\ 1 : -1$
        $c_{ki} \leftarrow k < i\ ?\ 1 : -1$
        $z_i, z_j, z_k \leftarrow \psi_i, \psi_j, \psi_k$           ▷ get local copies of edge data
        $\upsilon_{ij}, \upsilon_{jk}, \upsilon_{ki} \leftarrow c_{ij}\omega_{ij}, c_{jk}\omega_{jk}, c_{ki}\omega_{ki}$
        $S_{ijk} \leftarrow s_{ij}s_{jk}s_{ki}$                     ▷ compute branch index
        if $S_{ijk} < 0$ then                               ▷ branch triangle
            $\upsilon_{ki} \leftarrow -\upsilon_{ki}$               ▷ want transport to $\tau(i_1)$, not $i_1$
        end if
        if $s_{ij} < 0$ then               ▷ make values at $j$ consistent w/ $i$
            $z_j \leftarrow \bar{z}_j$
            $\upsilon_{ij} \leftarrow \ \ c_{ij}\upsilon_{ij}$
            $\upsilon_{jk} \leftarrow -c_{jk}\upsilon_{jk}$
        end if
        if $S_{ijk}s_{ki} < 0$ then        ▷ make values at $k$ consistent w/ $i$
            $z_k \leftarrow \bar{z}_k$
            $\upsilon_{ki} \leftarrow -c_{ki}\upsilon_{ki}$
            $\upsilon_{jk} \leftarrow \ \ c_{jk}\upsilon_{jk}$
        end if
        $\alpha_i^{jk} \leftarrow \arg(z_i)$          ▷ compute angles at triangle corners
        $\alpha_j^{ki} \leftarrow \alpha_i^{jk} + \upsilon_{ij} - \arg(e^{\iota\upsilon_{ij}} z_i/z_j)$
        $\alpha_k^{ij} \leftarrow \alpha_j^{ki} + \upsilon_{jk} - \arg(e^{\iota\upsilon_{jk}} z_j/z_k)$
        $\alpha_l^{ijk} \leftarrow \alpha_k^{ij} + \upsilon_{ki} - \arg(e^{\iota\upsilon_{ki}} z_k/z_i)$
        $\alpha_m^{ijk} \leftarrow \alpha_i^{jk} + (\alpha_l^{ijk} - \alpha_i^{jk})/2$      ▷ midpoint coordinate
        $n_{ijk} \leftarrow \frac{1}{2\pi}(\alpha_l^{ijk} - \alpha_i)$               ▷ compute zero index
        $\alpha_j^{ki} \leftarrow \alpha_j^{ki} - 2\pi n/3$                        ▷ adjust zeros
        $\alpha_k^{ij} \leftarrow \alpha_k^{ij} - 4\pi n/3$
    end for
    return $\alpha, n, S$
end procedure
```

The remaining routines are standard, but are defined here for completeness:

- Degree(K, i) - returns the degree of vertex $i \in V$.
- TipAngle(K, ℓ, i, j, k) - returns the angle at vertex $i \in V$ in triangle $ijk \in F$.
- OppositeAngles(K, ℓ, ij) - returns the two angles opposite edge $ij \in E$ (Sec. 3.2).
- TriangleArea(K, ℓ, ijk) - returns area of triangle $ijk \in F$.
- CholeskyFactor(A) - returns Cholesky factor of matrix A.
- UniformRand(n) - returns a vector of n numbers in the interval $[-1, 1]$, picked uniformly at random.
- Size(A) - returns the dimension of a square matrix A.
- BackSolve(L, b) - if L is a Cholesky factor of A, solves the linear system Ax = b.

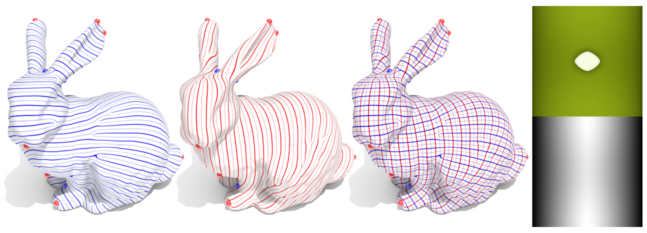


Figure 22: *Two independent stripe patterns* (left, center left) *oriented along orthogonal fields are combined* (center right) *to drive the periodic texture and displacement maps* (right) *used in Fig. 2.*