Repulsive Curves

CHRIS YU, Carnegie Mellon University
HENRIK SCHUMACHER, RWTH Aachen University
KEENAN CRANE, Carnegie Mellon University

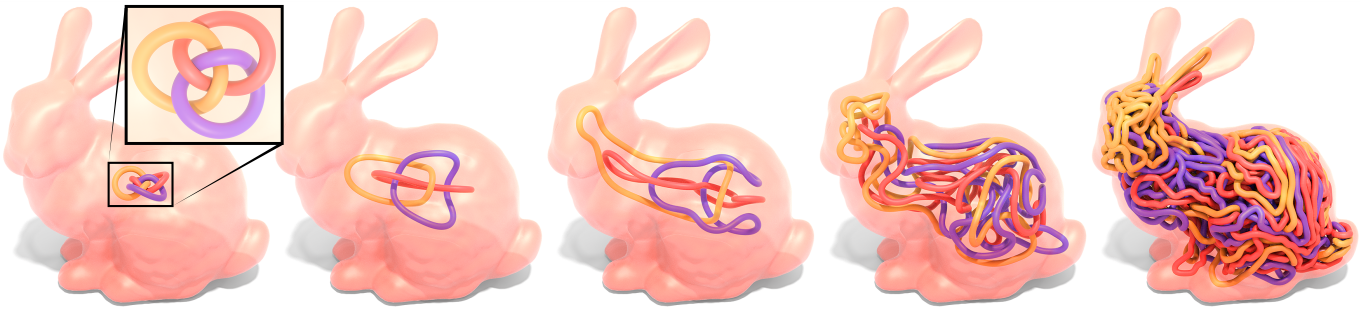


Fig. 1. We develop an efficient strategy for optimizing curves while avoiding self-collisions. Here for instance, interwoven curves of increasing length are confined inside a fixed domain, resulting in an intricate "curve packing." Replacing ordinary gradient descent with a specially-tailored *fractional Sobolev gradient* lets us take very large steps toward the solution, enabling rapid design exploration.

Curves play a fundamental role across computer graphics, physical simulation, and mathematical visualization, yet most tools for curve design do nothing to prevent crossings or self-intersections. This paper develops efficient algorithms for (self-)repulsion of plane and space curves that are well-suited to problems in computational design. Our starting point is the so-called *tangent-point energy*, which provides an infinite barrier to self-intersection. In contrast to local collision detection strategies used in, *e.g.*, physical simulation, this energy considers interactions between all pairs of points, and is hence useful for global shape optimization: local minima tend to be aesthetically pleasing, physically valid, and nicely distributed in space. A reformulation of gradient descent, based on a *Sobolev-Slobodeckij inner product* enables us to make rapid progress toward local minima—independent of curve resolution. We also develop a hierarchical multigrid scheme that significantly reduces the per-step cost of optimization. The energy is easily integrated with a variety of constraints and penalties (*e.g.*, inextensibility, or obstacle avoidance), which we use for applications including curve packing, knot untangling, graph embedding, non-crossing spline interpolation, flow visualization, and robotic path planning.

CCS Concepts: • **Computing methodologies** → **Shape modeling**; • **Mathematics of computing** → **Continuous optimization**.

Additional Key Words and Phrases: Computational design, shape optimization, curves, knots

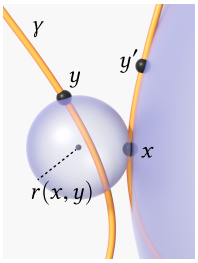
Authors' addresses: Chris Yu, Carnegie Mellon University; Henrik Schumacher, RWTH Aachen University, Templergraben 55, Aachen, Germany, 52062; Keenan Crane, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA, 15213.

0730-0301/XXXX/-1-ARTXX $15.00
https://doi.org/XX

1 INTRODUCTION

Shape optimization plays a role in a broad range of tasks ranging from variational data fitting to computational design. However, for many tasks it is essential to design *in context*, *i.e.*, relative to the geometry of the surrounding environment. Hard boundary conditions (*e.g.*, fixing the endpoints of a cable) provide a basic mechanism for providing context, but do not account for another fundamental requirement: physical objects cannot penetrate solid objects in the environment, nor can they intersect themselves. In some contexts, self-intersection can be avoided by detecting and resolving collisions at the moment of impact. However, forward simulation is not particularly effective at guiding shape optimization toward an intelligent design—for example, untangling a complicated knot via forward physical simulation is just as hard as trying to untangle it by hand. In this paper we instead explore how a global variational approach to curve self-avoidance provides new opportunities for computational design.

Our starting point is the *tangent-point energy* of Buck and Orloff [1995], which for an arc-length parameterized curve $\gamma : M \to \mathbb{R}^3$ can be expressed as an integral over all pairs of points $(x, y) \in M^2 := M \times M$:



$$\mathcal{E} := \iint_{M^2} \frac{1}{r(\gamma(x), \gamma(y))^\alpha}\, dxdy. \quad (1)$$

Here $r(x, y)$ is the radius of the smallest sphere tangent to $\gamma(x)$ and passing through $\gamma(y)$, and $\alpha \in \mathbb{R}$ is a parameter controlling the strength of repulsion. This energy approaches infinity for points $\gamma(y)$ that are close to $\gamma(x)$ in space but far from $\gamma(x)$ along the curve itself—preventing self-collision. For points $\gamma(y')$ close to $\gamma(x)$ *along* the curve, the radius r is very large—keeping forces bounded, and making the integral well-defined.

Although this energy has a simple definition, its gradient involves high-order *fractional* derivatives. Hence, classic optimization techniques must take extremely small steps, and standard techniques from shape optimization are not well-suited to handle the nonlocal nature of the energy. Our approach is to develop a preconditioner that *exactly* matches the fractional order of the differential (Section 4). In doing so, we obtain a gradient descent equation involving no spatial derivatives, permitting large time steps that make rapid progress toward local minima (Figure 2). In practice, this method is orders of magnitude more efficient than the simple untangling schemes often used in the knot literature (Figure 15), and offers substantial improvements over general-purpose optimization techniques from geometry processing (Section 7). Algorithms of this flavor have proven effective for problems such as finding minimal surfaces [Pinkall and Polthier 1993], integrating Willmore flow [Schumacher 2017], and computing surface parameterizations [Kovalsky et al. 2016]. However, little work has been done in the more challenging setting of nonlocal, "all-pairs" energies.

Contributions. Though knot energies have received significant attention in mathematics, there has been little work on the numerical and algorithmic tools needed to apply such energies to the computational design of curves. In this paper we develop:

- a principled discretization of the tangent-point energy,
- a novel preconditioner based on the *Sobolev-Slobodeckij* inner product,
- a numerical solver that easily incorporates constraints needed for design, and
- a Barnes-Hut strategy and hierarchical multigrid scheme for the tangent-point energy that greatly improve scalability.

We also explore a collection of constraints and potentials that enable us to apply this machinery to a broad range of applications in visualization and computational design (Section 8).

2 RELATED WORK

We briefly review topics related to computational design of curves; Section 3 gives more detailed background on curve energies. At a high level, computational design of free-form curves has generally focused on specific domains such as road networks [Hassan et al. 1998; McCrae and Singh 2009], telescoping structures [Yu et al. 2017], or rod assemblies [Pérez et al. 2015; Zehnder et al. 2016]; Moreton [1992, Chapter 3] gives a history of traditional design via spline curves. Our goal is to develop tools that can be applied to a wide range of multi-objective design scenarios, as explored in Section 8.

2.1 Curve Simulation

One natural idea is to avoid collision via physics-based simulation of elastic rods [Bergou et al. 2008]. However, the paradigm of collision detection and response is "too local": for computational design, one aims to globally optimize a variety of design criteria, rather than simulate the behavior of a given curve. *Sensitivity analysis*, which provides sophisticated *local* improvement of an initial design, has been successfully applied to several rod design problems [Pérez et al. 2015; Zehnder et al. 2016; Pérez et al. 2017]. This technique can be seen as complementary to global repulsion-based form-finding, helping to incorporate, *e.g.*, nonlinear mechanical phenomena into a final design. Curves also arise naturally as filaments or field lines in continuum phenomena like fluids, plasmas, and superfluids [Angelidis and Neyret 2005; Weißmann and Pinkall 2010; Padilla et al. 2019; Kleckner et al. 2016; Chern et al. 2016; DeForest and Kankelborg 2007]. However, using such phenomena for curve design is challenging since (i) initial conditions are hard to construct, and (ii) these systems naturally exhibit *reconnection events* where distinct pieces of a curve merge [Maucher and Sutcliffe 2016].

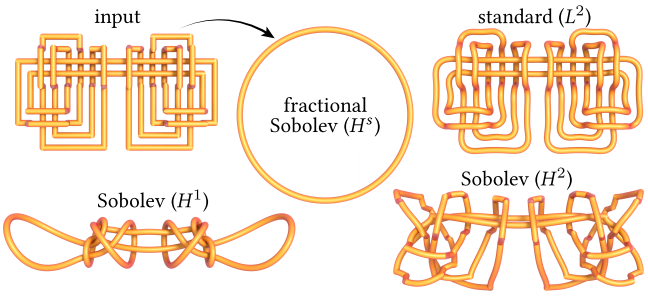


Fig. 2. Untangling the *Freedman unknot (top left)* to the unit circle. For the same wall clock time, standard L^2 gradient descent makes almost no progress, whereas conventional Sobolev descent fails to smooth out low (H^1) or high (H^2) frequencies. By carefully matching the inner product to the energy, our fractional H^s descent quickly flows to the circle.

2.2 Knot Energies

Motivated by questions in mathematics, biology, and physics [Calvo et al. 2002], there is a significant body of work on the *unknot problem*: can a closed loop be continuously deformed into a circle without passing through itself (*i.e.*, via *isotopy*)? Solving this decision problem is not our goal—so far it is not clear it can even be done in polynomial time [Lackenby 2014]. Yet knot untangling energies (discussed in Section 3) provide a valuable starting point for computational design. Numerically, simple ad-hoc methods that repel all pairs of vertices can yield inconsistent, unreliable behavior and slow convergence (Figure 15, *right*). Starting with more principled discretizations, *KnotPlot* [Scharein 1998] uses a simple relaxation scheme, and Kusner and Sullivan [1998] apply a standard conjugate gradient method via *SurfaceEvolver* [Brakke 1992], both evaluating all $O(n^2)$ interactions between the n vertices. Other, adjacent methods have been developed for *tightening* a given knot [Pierański 1998; Ashton et al. 2011], simulating the knot tying process [Brown et al. 2004; Kubiak et al. 2007; Harmon et al. 2009], or untangling knots without optimizing their shape [Ladd and Kavraki 2004]; more recent methods apply L^2 [Walker 2016] or integer Sobolev (H^2) descent [Bartels et al. 2018]. Octrees have been used to evaluate the ropelength of a static knot [Ashton and Cantarella 2005], but Barnes-Hut/multipole schemes have not yet been developed for energy minimization. Likewise, little has been said about fractional preconditioners, and treatment of general constraints.

Our approach builds on careful analysis of the fractional Sobolev spaces associated with the tangent point energy [Blatt 2012, 2013; Blatt and Reiter 2015]. Whereas this work focuses on, *e.g.*, the existence of local minimizers and short-time existence of gradient flows in the smooth setting, we use it to develop numerical algorithms.

2.3 Geometric Optimization

Optimization of curve and surface energies can be greatly accelerated by "Sobolev-like" preconditioning. The idea is to replace the ordinary L^2 inner product with one that is better matched to the energy, yielding a gradient flow that is easier to integrate (Section 4.1 gives a didactic example). Such flows make more rapid progress toward minimizers (Figure 2), since energy is reduced uniformly across all spatial frequencies. Crucially, Sobolev preconditioners are most effective when the *order of the preconditioner is perfectly matched to the order of spatial derivatives in the energy.* A preconditioner whose order is too high or too low can slow down convergence—see for instance Figure 5, *bottom-right.*

Sobolev-type preconditioners have seen some prior use in geometry processing and scientific computing. For example, the minimal surface algorithm of Pinkall and Polthier [1993] effectively performs Sobolev descent [Brakke 1994, Section 16.10], but was not originally framed in these terms; Renka and Neuberger [1995] give an algorithm directly formulated via a (variable) Sobolev inner product. Later work adopts Sobolev-like strategies for surface fairing and filtering [Desbrun et al. 1999; Eckstein et al. 2007; Martin et al. 2013; Crane et al. 2013; Schumacher 2017]. More recently, Sobolev-like descent has become popular for minimizing elastic energies, such as those arising in surface parameterization or shape deformation [Kovalsky et al. 2016; Claici et al. 2017; Zhu et al. 2018]; see Section 7 for in-depth comparisons.

Previous work on shape optimization does not consider the challenging *fractional* case, which differs significantly from standard Sobolev preconditioning. From an analytical point of view, we must first determine the order of derivatives arising in the differential—achieved by reasoning about the associated function spaces (Appendix A). We use this knowledge to formulate a novel preconditioner in the smooth setting which carefully considers lower-order terms (Section 4), which we then translate into the discrete setting via a principled discretization of the tangent-point energy (Section 5). From a computational point of view, the machinery needed to apply a fractional preconditioner is also different from ordinary Sobolev preconditioners: one cannot simply solve a sparse linear system, and must instead construct an efficient hierarchical scheme for (approximately) inverting a dense nonlocal operator. None of these pieces appear in the previous optimization work discussed above, though fractional operators and multigrid methods have been studied in other contexts, such as finite element simulation [Ainsworth and Glusa 2017] and multiphysics systems [Bærland et al. 2019; Bærland 2019]. Further, previously studied Sobolev preconditioners (such as those based on the Laplacian) and standard optimization strategies (such as Newton descent) are not as effective for our problem—as we show via extensive numerical experiments (Section 7).

3 CURVE ENERGIES

We first give a detailed discussion of the tangent-point energy, which we optimize in Section 4. Throughout we will use single bars $|X|$ and brackets $\langle X, Y\rangle$ to denote the Euclidean inner product on vectors in $\mathbb{R}^3$, and reserve double bars $\|f\|$ and brackets $\langle\!\langle f, g\rangle\!\rangle$ for norms and inner products on functions. We also use $\cdot|_f$ to indicate that a quantity (*e.g.*, an energy) is evaluated at a function f.

3.1 Background

Consider a collection of curves given by a parameterization $\gamma : M \to \mathbb{R}^3$, where M is comprised of intervals and/or loops. How can we formulate an energy that prevents self-intersection of γ? In general we will consider energies of the form

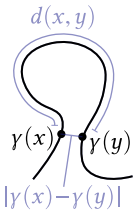
$$\mathcal{E}(\gamma) = \iint_{M^2} k(x, y)\, dx_\gamma\, dy_\gamma,$$

where the kernel $k : M \times M \to \mathbb{R}$ captures the interaction between two points on the curve, and dx_γ denotes the length element on γ.

3.1.1 Electrostatic Potential. One natural idea for defining k is to imagine that there is electric charge distributed along γ that pushes it away from itself, producing the Coulomb-like potential

$$k_{\text{Coulomb}}(x, y) := \frac{1}{|\gamma(x)-\gamma(y)|^\alpha}, \tag{2}$$

where the parameter α controls the strength of repulsion. Unfortunately this simple energy does not work for a continuous curve: for $\alpha < 2$ it is not strong enough to prevent collisions, allowing the curve to pass through itself—yet for $\alpha \geq 1$ the integral does not exist, resulting in unpredictable and unreliable behavior when discretized.



3.1.2 Möbius Energy. To obtain a well-defined energy, one can *regularize* the integrand in regions where x approaches y. One such regularization, proposed by O'Hara [1991], is the *Möbius energy*, with kernel

$$k_{\text{Möbius}}(x, y) := \frac{1}{|\gamma(x)-\gamma(y)|^2} - \frac{1}{d(x, y)^2},$$

where $d(x, y)$ denotes the shortest distance between x and y *along* the curve (*e.g.*, the smaller of two arcs along a circle). Intuitively: if two points are both close in space and close along the curve, we remove the singular energy; if they are close in space but *distant* along the curve, they continue to repel each other (see inset). This energy is invariant to Möbius transformations [Freedman et al. 1994], which can be attractive from the perspective of knot theory—but causes problems for computational design, since near-intersections may not be penalized in a natural way (Figure 3).

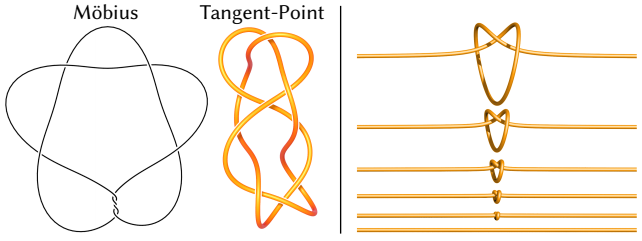


Fig. 3. *Left:* Since the Möbius energy is scale-invariant, it allows "tight spots" where the curve nearly touches itself; such features are avoided by the tangent-point energy. *Right:* The Möbius energy can likewise artificially eliminate knots by pulling them tight at no energetic cost. (Leftmost image from Kusner and Sullivan [1998].)

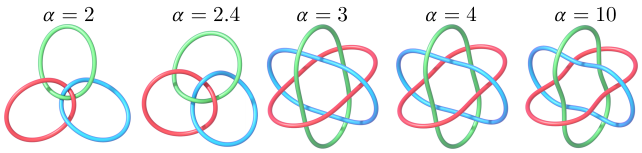


Fig. 4. Local minimizers of the tangent-point energy $\mathcal{E}^{\alpha}_{2\alpha}$. When $\alpha = 2$ the tangent-point energy is scale-invariant and can exhibit "tight spots"; for larger values of α local interactions are penalized more than distant ones.

3.2 Tangent Point Energy

Instead, we will use the tangent point energy introduced in Section 1. We can write this energy more explicitly by noting that (up to a constant factor)

$$r(x, y) = \frac{|\gamma(x) - \gamma(y)|^2}{|T(x) \times (\gamma(x) - \gamma(y))|}$$

where $T(x)$ is the unit tangent of γ at x. This expression leads to a generalized tangent-point energy [Blatt and Reiter 2015], given by

$$\mathcal{E}^{\alpha}_{\beta}(\gamma) := \iint_{M^2} k^{\alpha}_{\beta}(\gamma(x), \gamma(y), T(x))\, dx_{\gamma}\, dy_{\gamma},$$

where k^{α}_{β} is the *tangent-point kernel*

$$k^{\alpha}_{\beta}(p, q, T) := \frac{|T \times (p - q)|^{\alpha}}{|p - q|^{\beta}}. \tag{3}$$

In the case $\beta = 2\alpha$, this energy agrees with Equation 1; as shown by Blatt [2013] it is well-defined for any α, β satisfying $\alpha > 1$ and $\beta \in [\alpha + 2, 2\alpha + 1)$ (Lemma A.1). Most importantly, it tends toward infinity as the curve approaches itself, preventing self-intersection. In particular, when $\beta - \alpha > 2$ it is not scale-invariant, and hence avoids the pull-tight phenomenon. (We set (α, β) to $(2, 4.5)$ in Figures 14–19, and $(3, 6)$ elsewhere.)

This energy is also attractive for design since it provides natural regularization, akin to bending energy. The reason is that the integrand can vanish only for a straight line (where the radius r is infinite at every point). The powers β and α have an impact on this bending behavior—for instance, if $\beta = 2\alpha$, then a higher α gives a more repulsive energy where curves are willing to bend more in order to avoid collision (Figure 4).

4 OPTIMIZATION

Consider an energy $\mathcal{E}$ that depends on a function f. A typical starting point for optimization is to integrate the gradient flow

$$\frac{d}{dt} f = -\operatorname{grad} \mathcal{E}(f), \tag{4}$$

i.e., to move in the direction of "steepest descent." As mentioned in Section 2, however, the efficiency of this flow depends critically on the *inner product* used to define the gradient—in other words, there are many different notions of what it means to be "steepest." Recall in particular that the *differential* $d\mathcal{E}$ describes the change in $\mathcal{E}$ due to any small perturbation u of f:

$$d\mathcal{E}|_f(u) = \lim_{\varepsilon \to 0} \frac{1}{\varepsilon} \left(\mathcal{E}(f + \varepsilon u) - \mathcal{E}(f) \right).$$

The *gradient* of $\mathcal{E}$ is then the unique function $\operatorname{grad} \mathcal{E}$ whose inner product with any function u gives the differential in that direction:

$$\langle\!\langle \operatorname{grad} \mathcal{E}, u \rangle\!\rangle_V = d\mathcal{E}(u). \tag{5}$$

Traditionally, the inner product $\langle\!\langle \cdot, \cdot \rangle\!\rangle_V$ is just the L^2 inner product

$$\langle\!\langle u, v \rangle\!\rangle_{L^2} := \int_M \langle u(x), v(x) \rangle\, dx.$$

More generally, however, one can try to pick a so-called *Sobolev inner product* $\langle\!\langle u, v \rangle\!\rangle_{H^k}$ that yields an easier gradient flow equation. Examples include the H^1 and H^2 inner products, which for a domain without boundary can be written as

$$\langle\!\langle u, v \rangle\!\rangle_{H^1} := \langle\!\langle \operatorname{grad} u, \operatorname{grad} v \rangle\!\rangle_{L^2} = -\langle\!\langle \Delta u, v \rangle\!\rangle_{L^2}, \tag{6}$$

and

$$\langle\!\langle u, v \rangle\!\rangle_{H^2} := \langle\!\langle \Delta u, \Delta v \rangle\!\rangle_{L^2} = \langle\!\langle \Delta^2 u, v \rangle\!\rangle_{L^2}, \tag{7}$$

which measure first and second derivatives (*resp.*) rather than function values. In general, if we write our inner product as $\langle\!\langle u, v \rangle\!\rangle_{H^k} = \langle\!\langle Au, v \rangle\!\rangle_{L^2}$ for some linear operator A, then we can express the new gradient direction g as the solution to

$$Ag = \operatorname{grad}_{L^2} \mathcal{E}. \tag{8}$$

This transformation is akin to the preconditioning provided by Newton's method, except that we replace the Hessian with an operator A that is always positive-definite, and often easier to invert. In particular, when A comes from a carefully-designed Sobolev inner product, it will eliminate spatial derivatives, avoiding the stringent time step restriction typically associated with numerical integration of gradient flow (Figure 6).

4.1 Warm-up: Dirichlet energy

Since analysis of the tangent-point energy is quite involved, we begin with a standard "toy" example that helps sketch out the main ideas of our approach. In particular, consider the *Dirichlet energy*

$$\mathcal{E}_D(f) := \tfrac{1}{2} \int_{\Omega} |\operatorname{grad} f(x)|^2\, dx, \tag{9}$$

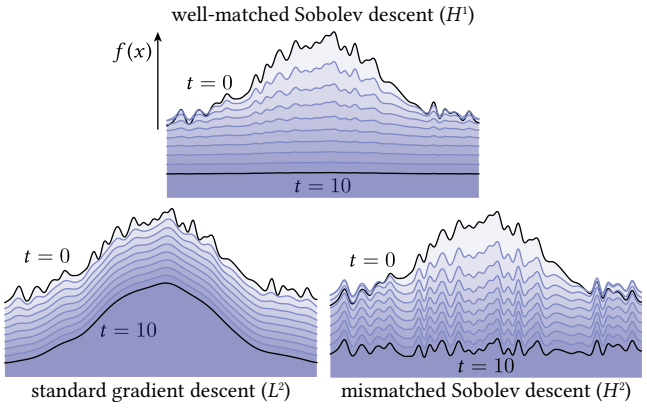


Fig. 5. For Dirichlet energy, which penalizes variations in a function $f(x)$, standard L^2 gradient descent mostly smooths out local features *(bottom left)*, whereas an inner product that is too high-order has trouble removing high frequencies *(bottom right)*. A Sobolev descent that is well-matched to the order of the energy yields rapid progress toward a local minimizer *(top)*. We apply a similar strategy to quickly optimize the shape of curves.

which penalizes variation in a function $f : \Omega \to \mathbb{R}$. If the domain Ω has no boundary, then we can use integration by parts to write this energy as

$$\mathcal{E}_D(f) = \tfrac{1}{2}\langle\!\langle \operatorname{grad} f, \operatorname{grad} f \rangle\!\rangle_{L^2} = -\tfrac{1}{2}\langle\!\langle \Delta f, f \rangle\!\rangle_{L^2},$$

where Δ denotes the Laplace operator. The differential is then

$$d\mathcal{E}_D|_f(u) = -\langle\!\langle \Delta f, u \rangle\!\rangle_{L^2},$$

and from Equation 5, we see that the L^2 gradient of $\mathcal{E}_D$ is given by $\operatorname{grad}_{L^2} \mathcal{E}_D|_f = -\Delta f$. Hence, L^2 gradient descent yields the *heat flow*

$$\tfrac{d}{dt} f = \Delta f, \qquad (L^2 \text{ gradient flow})$$

which involves second-order derivatives in space [Andrews et al. 2020, Section 1.2]. If we try to solve this equation using, say, explicit finite differences with grid spacing h, we will need a time step of size $O(h^2)$ to remain stable—significantly slowing down computation as the grid is refined. To lift this time step restriction, we can use a different inner product to define the gradient. In particular, replacing $\langle\!\langle \cdot, \cdot \rangle\!\rangle_V$ with the H^1 inner product in Equation 5 yields

$$\langle\!\langle \Delta \operatorname{grad}_{H^1} \mathcal{E}_D, u \rangle\!\rangle_{L^2} = \langle\!\langle \Delta f, u \rangle\!\rangle_{L^2}. \tag{10}$$

This equation can be satisfied by letting $\operatorname{grad}_{H^1} \mathcal{E}_D := f$, in which case Equation 4 defines an H^1 gradient flow

$$\tfrac{d}{dt} f = -f. \qquad (H^1 \text{ gradient flow})$$

This flow involves no spatial derivatives, and hence comes with no time step restriction. In effect, rather than a PDE, we now have a system of independent ODEs, which is far easier to integrate numerically. As shown in Figure 5, the character of this flow is quite different: it makes progress by simultaneously flattening all spatial frequencies, rather than just performing local smoothing. While this approach is not appropriate for dynamical simulation, it is quite useful for finding local minima, as needed in geometric design. In general, however, Sobolev descent is not as simple as just uniform scaling—instead, one must solve a linear PDE (Equation 8) for the new descent direction.

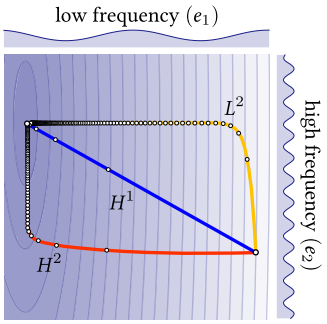


Fig. 6. Gradient flows projected onto a low- and high-frequency mode e_1, e_2, *resp.* Notice that poor preconditioning leads to slow convergence.

Note that we should not use an inner product with *too many* derivatives. For example, if we use the H^2 inner product (Equation 7) we get a gradient $\operatorname{grad}_{H^2} \mathcal{E}_D|_f = -\Delta^{-1} f$, and a flow

$$\tfrac{d}{dt} f = \Delta^{-1} f. \qquad (H^2 \text{ gradient flow})$$

This flow is again hard to integrate, and has trouble smoothing out high frequencies (Figure 5, *bottom-right*). In general, one cannot achieve good behavior by blindly picking a Sobolev inner product, but must instead *carefully match the inner product to the energy.*

Low-Order Terms. One remaining issue is that Equation 10 determines the H^1 gradient only up to functions in the null space of Δ. This situation is problematic, since it means we cannot obtain a gradient by solving Equation 8 directly (with $A = -\Delta$). Instead, we must include *low-order terms* that make the overall operator A invertible. For instance, we could let $A := -\Delta + \mathrm{id}$, where id denotes the identity. But if we uniformly scale the domain by a factor $c > 0$, the new operator looks like $-\frac{1}{c^2}\Delta + \mathrm{id}$ and the character of the flow changes substantially: when c is small it looks like the H^1 flow; when c is large, it looks more like the L^2 flow. Careful treatment of regularization and scaling is therefore an important consideration in the development of our curve flow (Section 4.2.3).

4.2 Fractional Sobolev Gradient

In the case of a nonlocal energy like the tangent-point energy $\mathcal{E}^\alpha_\beta$, one can no longer use a standard Sobolev inner product—instead, an inner product of *fractional* order is needed, in order to match fractional derivatives that appear in the differential. Construction of a suitable inner product for the tangent-point energy is fairly technical—in a nutshell, we begin with a known expression for the *fractional Laplacian* on Euclidean $\mathbb{R}^n$, and formulate an analogous operator for embedded curves. Taking additional (integer) derivatives yields a differential operator B_σ of the same order as the differential $d\mathcal{E}^\alpha_\beta$. We then add a lower-order operator B^0_σ that makes the overall operator $A_\sigma := B_\sigma + B^0_\sigma$ more well-behaved. Our *Sobolev-Slobodeckij inner product* is then defined as

$$\langle\!\langle u, v \rangle\!\rangle_{H^s_\gamma} := \langle\!\langle A_\sigma u, v \rangle\!\rangle_{L^2}.$$

Details are given in Appendix A—here we give only the most essential definitions needed to derive our discrete algorithm (Section 5).

4.2.1 Derivative Operator. To define the inner product, we will need the first derivative operator $\mathcal{D}$ given by

$$\mathcal{D}u := du\, d\gamma^{\mathsf{T}} / |d\gamma|^2. \tag{11}$$

This operator just takes the usual derivative of u along M and expresses it as a vector in $\mathbb{R}^3$ tangent to γ; the factor $1/|d\gamma|^2$ accounts for the fact that the curve is not in general arc-length parameterized.

4.2.2 High-Order Term. As discussed in Appendix A.3, the differential $d\mathcal{E}^\alpha_\beta$ of the tangent-point energy has order $2s$, where $s = (\beta - 1)/\alpha$. To build an inner product of the same order, we first define the fractional differential operator B_σ, given by

$$\langle\!\langle B_\sigma u, v \rangle\!\rangle := \iint_{M^2} \frac{\mathcal{D}u(x) - \mathcal{D}u(y)}{|\gamma(x) - \gamma(y)|^\sigma} \frac{\mathcal{D}v(x) - \mathcal{D}v(y)}{|\gamma(x) - \gamma(y)|^\sigma} \frac{dx_\gamma\, dy_\gamma}{|\gamma(x) - \gamma(y)|} \tag{12}$$

for all sufficiently regular $u, v : M \to \mathbb{R}$, where $\sigma = s - 1$. This operator also has order $2s$ (Appendix A.4), and plays a role analogous to the Laplacian in Section 4.1. Yet just like the Laplacian, B_σ is only *semidefinite*, since it vanishes for functions that are constant over each component of the domain M. Hence, it is not invertible, and cannot be used directly to solve for a descent direction—instead we must "regularize" B_σ by adding an additional, lower-order term.

4.2.3 Low-Order Term. A naïve approach to regularization, like adding some small $\epsilon > 0$ times the identity, yields undesirable behavior—ϵ must be sufficiently large to have an effect, but if ϵ is too large, motion is significantly damped. Moreover, an inner

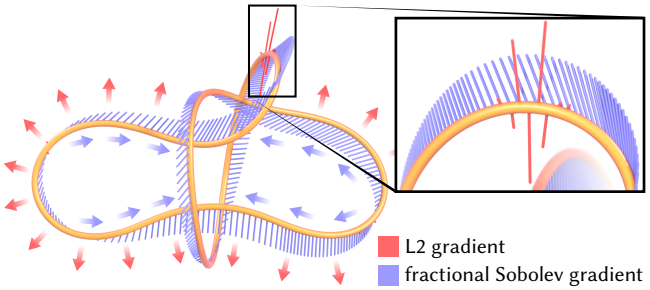


Fig. 7. Since an L^2 gradient flow is always perpendicular to the curve *(red)*, it fails to resolve even simple cases like the one shown above, where a large near-tangential motion is needed to untangle a knot. The fractional Sobolev gradient *(blue)* permits such motions, yielding a far more efficient flow.

product constructed this way will no longer exhibit predictable scaling behavior, *i.e.*, rescaling the input will actually change the *direction* of the gradient rather than just its magnitude—and hence can change the solution obtained by a designer. Instead, we carefully choose an additional, low-order term B^0_σ that not only provides the right scaling behavior, but also enables us to steer the flow more quickly toward self-avoiding configurations (Figure 7). In particular, we add the term $\langle\!\langle B^0_\sigma u, v\rangle\!\rangle$, given by

$$\iint_{M^2} k_4^2(\gamma(x),\gamma(y),T(x))\frac{(u(x)-u(y))(v(x)-v(y))}{|\gamma(x)-\gamma(y)|^{2\sigma+1}}\,dx_\gamma\,dy_\gamma, \quad (13)$$

where k^α_β is the tangent-point kernel given in Equation 3. See Appendix A.4 for further discussion.

4.2.4 Sobolev-Slobodeckij Gradient. Following Equation 5, our final gradient $\operatorname{grad}_{H^s_\gamma}$ is defined via the fractional inner product:

$$\langle\!\langle \operatorname{grad}_{H^s_\gamma}\mathcal{E}^\alpha_\beta, X\rangle\!\rangle_{H^s_\gamma} = d\mathcal{E}^\alpha_\beta|_\gamma(X), \quad \text{for all } X : M \to \mathbb{R}^3. \quad (14)$$

Since $\operatorname{grad}_{H^s_\gamma}\mathcal{E}^\alpha_\beta$ and X are vector- rather than scalar-valued, we apply the inner product componentwise. In other words,

$$\operatorname{grad}_{H^s_\gamma}\mathcal{E}^\alpha_\beta = \bar{A}^{-1}_\sigma \operatorname{grad}_{L^2}\mathcal{E}^\alpha_\beta|_\gamma, \quad (15)$$

where $\bar{A}_\sigma$ denotes componentwise application of A_σ. Note that the combined operator $A_\sigma = B_\sigma + B^0_\sigma$ still has *globally* constant functions in its kernel, corresponding to global translations. To make Equation 15 well-defined, we can simply add any constraint that fixes the translation of the curve (Section 5.3). In practice, we never need a closed-form expression for the gradient, nor do we explicitly invert the operator A_σ; instead, we solve Equation 8 numerically.

5 DISCRETIZATION

We now use the inner product from the previous section to derive an efficient numerical scheme for minimizing the tangent-point energy. Our discretization operates on polygonal curves. While in principle splines could be used *à la* Bartels et al. [2018], in practice this makes little difference due to the use of numerical quadrature in both cases. The description given here assumes a naïve implementation using dense matrices and an $O(n^2)$ evaluation of the energy and its differential; hierarchical acceleration is described in Section 6.

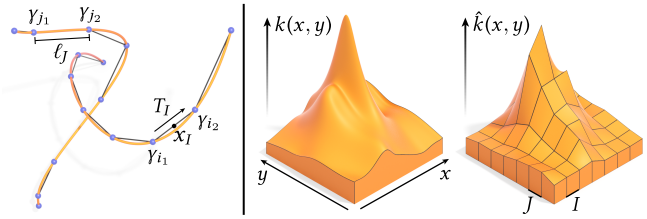


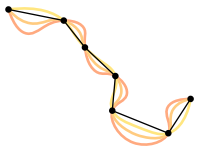
Fig. 8. *Left:* notation used for discrete curves. *Right:* Our discrete energy is obtained by applying the trapezoidal rule to the smooth energy for each edge pair I, J.

Notation. In the discrete setting, we will model any collection of curves and loops (including several curves meeting at a common point) as a graph $G = (V, E)$ with vertex coordinates $\gamma : V \to \mathbb{R}^3$ (Figure 8); we use $|V|$ and $|E|$ to denote the number of vertices and edges, *resp.* For each edge $I \in E$ with endpoints i_1, i_2, we use

$$\ell_I := |\gamma_{i_1} - \gamma_{i_2}|, \quad T_I := (\gamma_{i_2} - \gamma_{i_1})/\ell_I, \quad \text{and} \quad x_I := (\gamma_{i_1} + \gamma_{i_2})/2$$

to denote the edge length, unit tangent, and midpoint, *resp.* For any quantity $u : V \to \mathbb{R}$ on vertices we use $u_I := (u_{i_1} + u_{i_2})/2$ to denote the average value on edge $I = (i_1, i_2)$, and $\mathrm{u}[I] := [u_{i_1}\ u_{i_2}]^\mathsf{T}$ to denote the 2×1 column vector storing the values at its endpoints. Finally, we refer to any pair $(T, x) \in \mathbb{R}^6$ as a *tangent-point*.

5.1 Discrete Energy



Since the tangent-point energy is infinite for polygonal curves [Strzelecki and von der Mosel 2017, Figure 2.2], we assume that γ is inscribed in some (unknown) smooth curve, and apply numerical quadrature to the smooth energy $\mathcal{E}^\alpha_\beta$. The resulting discrete energy then approximates the energy of any sufficiently smooth curve passing through the vertices γ_i. We start by integrating k^α_β over all pairs of edges:

$$\sum_{I\in E}\sum_{J\in E}\int_{\bar{I}}\int_{\bar{J}} k^\alpha_\beta(\gamma(x),\gamma(y),T_I)\,dx_\gamma\,dy_\gamma. \quad (16)$$

Here $\bar{I}$ denotes the interval along edge I. As stated, this expression is ill-defined since any two edges with a common endpoint contribute infinite energy. One idea is to replace any such term with one proportional to the curvature of the circle passing through the three distinct endpoints (in the spirit of Equation 1). However, such terms would contribute nothing to the energy in the limit of regular refinement (Figure 9)—hence, we simply omit neighboring edge pairs. Applying the (2D) trapezoidal rule to Equation 16 then yields a discrete energy

$$\hat{\mathcal{E}}^\alpha_\beta(\gamma) = \sum_{I,J\in E, I\cap J=\varnothing}\sum (\hat{k}^\alpha_\beta)_{IJ}\ell_I\ell_J, \quad (17)$$

where $\hat{k}$ is the discrete kernel

$$(\hat{k}^\alpha_\beta)_{IJ} := \tfrac{1}{4}\textstyle\sum_{i\in I}\sum_{j\in J} k^\alpha_\beta(\gamma_i,\gamma_j,T_I). \quad (18)$$

The discrete differential is then simply the partial derivatives of this energy with respect to the coordinates of all the curve vertices:

$$d\hat{\mathcal{E}}^\alpha_\beta|_\gamma = \begin{bmatrix} \partial\mathcal{E}^\alpha_\beta/\partial\gamma_1 & \cdots & \partial\mathcal{E}^\alpha_\beta/\partial\gamma_{|V|} \end{bmatrix} \in \mathbb{R}^{3|V|}.$$

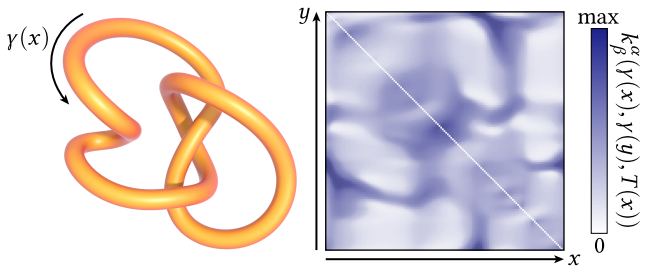


Fig. 9. The tangent-point energy is a double integral of the kernel k_β^α *(right)* over the curve γ *(left)*. Since this kernel is only weakly singular, omitting diagonal terms has an insignificant effect on the overall energy.

These derivatives can be evaluated via any standard technique (*e.g.*, by hand, or using symbolic or automatic differentiation).

5.2 Discrete Inner Product

As in the smooth setting, we define our inner product matrix as a sum $\mathsf{A} = \mathsf{B} + \mathsf{B}^0$ of high-order and low-order terms $\mathsf{B}, \mathsf{B}^0 \in \mathbb{R}^{|V|\times|V|}$ (as defined below). For $\mathbb{R}^3$-valued functions, we also define a corresponding $3|V| \times 3|V|$ matrix

$$\overline{\mathsf{A}} = \begin{bmatrix} \mathsf{A} & & \\ & \mathsf{A} & \\ & & \mathsf{A} \end{bmatrix}. \tag{19}$$

Mirroring Equation 8, the discrete (fractional) Sobolev gradient $\mathsf{g} \in \mathbb{R}^{3|V|}$ is then defined as the solution to the matrix equation

$$\overline{\mathsf{A}}\mathsf{g} = d\hat{\mathcal{E}}_\beta^\alpha. \tag{20}$$

5.2.1 Discrete Derivative Operator. For each edge $I \in E$ we approximate the derivative $\mathcal{D}u$ of a function $u : M \to \mathbb{R}$ (Equation 11) via the finite difference formula $\frac{1}{\ell_I}(\mathsf{u}_{i_2} - \mathsf{u}_{i_1})T_I$, where u_i denotes the value of u sampled at vertex i. The corresponding derivative matrix $\mathsf{D} \in \mathbb{R}^{3|E|\times|V|}$ can be assembled from local 3×2 matrices

$$\mathsf{D}_I = \tfrac{1}{\ell_I}\begin{bmatrix} -T_I & T_I \end{bmatrix}.$$

5.2.2 Discrete High-Order Term. We approximate the high-order part of the inner product $\langle\langle B_\sigma u, v\rangle\rangle$ as

$$\mathsf{u}^\mathsf{T}\mathsf{B}\mathsf{v} = \sum_{I,J\in E, I\cap J=\varnothing} w_{IJ}\langle \mathsf{D}_I\mathsf{u}[I] - \mathsf{D}_J\mathsf{u}[J], \mathsf{D}_I\mathsf{v}[I] - \mathsf{D}_J\mathsf{v}[J]\rangle, \tag{21}$$

where the weights w_{IJ} arise from applying trapezoidal quadrature to the denominator in Equation 25:

$$w_{IJ} := \tfrac{1}{4}\ell_I\ell_J \sum_{i\in I}\sum_{j\in J} \tfrac{1}{|\gamma_i - \gamma_j|^{2\sigma+1}}.$$

The entries of the corresponding Gram matrix $\mathsf{B} \in \mathbb{R}^{|V|\times|V|}$ are obtained by differentiating Equation 21 with respect to the entries of u and v. More explicitly, starting with the zero matrix one can build B by making the following increments for all pairs of disjoint edges $I \cap J = \varnothing$, and all pairs of values $a, b \in \{1, 2\}$:

$$\mathsf{B}_{i_a i_b} \mathrel{+}= (-1)^{a+b} w_{IJ}/\ell_I^2, \quad \mathsf{B}_{i_a j_b} \mathrel{-}= (-1)^{a+b} w_{IJ}\,\langle T_I, T_J\rangle/(\ell_I\ell_J),$$
$$\mathsf{B}_{j_a j_b} \mathrel{+}= (-1)^{a+b} w_{IJ}/\ell_J^2, \quad \mathsf{B}_{j_a i_b} \mathrel{-}= (-1)^{a+b} w_{IJ}\,\langle T_J, T_I\rangle/(\ell_J\ell_I).$$

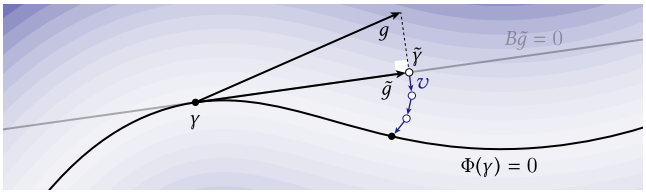


Fig. 10. To enforce constraints $\Phi(\gamma) = 0$ on the curve, we both project the gradient g onto the tangent of the constraint set, and also apply an iterative procedure to project the curve itself back onto the constraint set. In both cases, the fractional Sobolev norm provides the definition of closeness.

5.2.3 Discrete Low-Order Term. To discretize the low-order term B_σ^0 (Section 4.2.3), we use a different discrete weight

$$w_{IJ}^0 := \tfrac{1}{4}\ell_I\ell_J \sum_{i\in I}\sum_{j\in J} \frac{k_4^2(\gamma_i, \gamma_j, T_I)}{|\gamma_i - \gamma_j|^{2\sigma+1}},$$

and define a matrix $\mathsf{B}^0 \in \mathbb{R}^{|V|\times|V|}$, given by the relationship

$$\mathsf{u}^\mathsf{T}\mathsf{B}^0\mathsf{v} = \sum_{I,J\in E, I\cap J=\varnothing} w_{IJ}^0(\mathsf{u}_I - \mathsf{u}_J)(\mathsf{v}_I - \mathsf{v}_J).$$

Following a similar derivation as above, this matrix can be constructed via the following increments:

$$\mathsf{B}^0_{i_a i_b} \mathrel{+}= \tfrac{1}{4}w_{IJ}^0, \qquad \mathsf{B}^0_{i_a j_b} \mathrel{-}= \tfrac{1}{4}w_{IJ}^0,$$
$$\mathsf{B}^0_{j_a i_b} \mathrel{-}= \tfrac{1}{4}w_{IJ}^0, \qquad \mathsf{B}^0_{j_a j_b} \mathrel{+}= \tfrac{1}{4}w_{IJ}^0.$$

5.3 Constraints

For design applications, we will need to impose a variety of scalar constraints $\Phi_i(\gamma) = 0$, $i = 1, \ldots, k$, which we encode as a single constraint function $\Phi : \mathbb{R}^{3|V|} \to \mathbb{R}^k$ (Section 8.1). To enforce these constraints, we project the gradient onto a valid descent direction (Section 5.3.1); after taking a step in this direction, we also project the result onto the constraint set (Section 5.3.2).

5.3.1 Gradient Projection. Let $\mathsf{C} := d\Phi(\gamma)$ be the Jacobian matrix of the constraint, and let $\mathsf{g} := \operatorname{grad}_{H_\gamma^s} E \in \mathbb{R}^{3|V|}$ denote the unconstrained energy gradient. We seek the descent direction $\tilde{\mathsf{g}}$ that is closest to g with respect to the fractional Sobolev norm, but which is also tangent to the constraint set:

$$\min_{\tilde{\mathsf{g}}} \tfrac{1}{2}||\tilde{\mathsf{g}} - \mathsf{g}||^2_{H_\gamma^s} \quad \text{s.t.} \quad \mathsf{C}\tilde{\mathsf{g}} = 0.$$

Writing $||v||^2_{H_\gamma^s}$ as $v^\mathsf{T}\overline{\mathsf{A}}v$ (Section 5.2), we can apply the method of Lagrange multipliers to obtain the usual first-order optimality conditions, given by the saddle point system

$$\begin{bmatrix} \overline{\mathsf{A}} & \mathsf{C}^\mathsf{T} \\ \mathsf{C} & 0 \end{bmatrix}\begin{bmatrix} \tilde{\mathsf{g}} \\ \lambda \end{bmatrix} = \begin{bmatrix} d\mathcal{E}_\beta^\alpha|_\gamma^\mathsf{T} \\ 0 \end{bmatrix}, \tag{22}$$

where $\lambda \in \mathbb{R}^k$ are the Lagrange multipliers, and we have applied the identity $\overline{\mathsf{A}}\mathsf{g} = d\mathcal{E}_\beta^\alpha|_\gamma^\mathsf{T}$ (Equation 20).

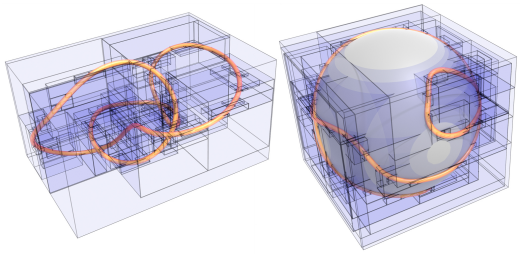


Fig. 11. To accelerate evaluation of the tangent-point energy, we build a bounding volume hierarchy that partitions both positions *(left)* and tangent directions *(right)*, here drawn as a curve on the unit sphere.

5.3.2 Constraint Projection. Suppose that we take a small step of size τ along the projected gradient direction $\tilde{\mathrm{g}}$ to get a new candidate curve $\tilde{\gamma} := \gamma - \tau\tilde{\mathrm{g}}$. To project this curve back onto the constraint set, we will apply an approximation of Newton's method that is faster to evaluate. In particular, to find a displacement $\mathrm{x} \in \mathbb{R}^{3|V|}$ that takes us from $\tilde{\gamma}$ back toward the constraint set $\Phi(\mathrm{g}) = 0$, we solve the problem

$$\min_{\mathrm{x}} \tfrac{1}{2}\mathrm{x}^{\mathsf{T}}\overline{\mathrm{A}}\mathrm{x} \quad \text{s.t.} \quad \mathrm{C}\mathrm{x} = -\Phi(\tilde{\gamma}).$$

We then update our guess via $\tilde{\gamma} \leftarrow \tilde{\gamma} + \mathrm{x}$ and repeat until the constraint violation $\Phi(\tilde{\gamma})$ is numerically small (10^{-4} in our experiments). In practice, this process rarely takes more than three iterations. At each iteration, x is obtained by solving the saddle point problem

$$\begin{bmatrix} \overline{\mathrm{A}} & \mathrm{C}^{\mathsf{T}} \\ \mathrm{C} & 0 \end{bmatrix} \begin{bmatrix} \mathrm{x} \\ \mu \end{bmatrix} = \begin{bmatrix} 0 \\ -\Phi(\tilde{\gamma}) \end{bmatrix}, \tag{23}$$

where $\mu \in \mathbb{R}^k$ are Lagrange multipliers.

5.4 Time Stepping

A judicious choice of time step can significantly improve the efficiency of the flow. One strategy is to use the first time step $\tau_{\max}$ at which a collision occurs as the starting point for a line search, which guarantees that the curve remains in the same isotopy class. (Similar approaches have been used in, *e.g.*, *KnotPlot* [Scharein 1998] for knot untangling, and by Smith and Schaefer [2015] for surface parameterization.) Computing this time step via standard techniques [Redon et al. 2002] costs about as much as a single energy evaluation, *i.e.*, significantly less than the overall cost of a single time step. From here we apply standard backtracking line search [Boyd and Vandenberghe 2004, Algorithm 9.2]; as a heuristic, we start this search at $\frac{2}{3}\tau_{\max}$. We use this strategy throughout Section 7.

An even simpler strategy that works well in practice (but comes with no collision guarantees) is to just normalize the gradient and perform backtracking line search starting with $\tau = 1$, until both (i) the Armijo condition is satisfied and (ii) constraint projection succeeds (Section 5.3.2). We use this latter strategy for all application examples in Section 8. We stop when the L^2 norm of the fractional Sobolev gradient goes below a user-specified tolerance ε. In our examples we use $\varepsilon = 10^{-4}$, though of course for design applications one can also stop whenever the results are aesthetically pleasing.

6 ACCELERATION

Computational design problems can entail large collections of curves with many thousands of vertices (Section 8). Optimization hence becomes expensive since it involves not only an all-pairs energy (Section 5.1), but also inverting a dense inner product (Section 5.2). However, since the kernel falls off rapidly in space, we can use hierarchical approximation to avoid a $\Omega(|V|^2)$ time and storage cost. Though our high-level approach is reasonably standard, careful consideration of the tangent-point energy is needed to develop a scheme that is efficient, easy to implement, and handles general nonlinear constraints. To streamline exposition, we reserve the details of this scheme for Appendix B; at a high level it consists of three main parts, outlined below. Note that since we care only about finding a good descent direction—and not accurately simulating a dynamical trajectory—we are free to use low-order schemes, which still provide good preconditioning. Empirically, the overall strategy exhibits near-linear scaling in both time and memory (Figure 20).

6.1 Energy and Differential Evaluation

To accelerate evaluation of the energy $\hat{\mathcal{E}}_{\beta}^{\alpha}$ and its differential, we apply the *Barnes-Hut algorithm* from N-body simulation [Barnes and Hut 1986]. The basic idea is to approximate distant energy contributions by aggregating values in a spatial hierarchy. In our case, this hierarchy must have six dimensions rather than three, since $\hat{\mathcal{E}}_{\beta}^{\alpha}$ depends on both positions $\gamma \in \mathbb{R}^3$ and tangents $T \in \mathbb{R}^3$. In lieu of a standard octree we therefore use an axis-aligned *bounding volume hierarchy (BVH)*, for which additional dimensions do not incur significant cost (Figure 11). Appendix B.1 gives further details.

6.2 Hierarchical Matrix-Vector Product

For optimization we need to solve linear systems involving so-called *kernel matrices*. Any such matrix $\mathrm{K} \in \mathbb{R}^{|E|\times|E|}$ has a special form

$$\mathrm{K}_{IJ} = k(p_I, p_J)\ell_I\ell_J,$$

where the kernel k maps a pair of tangent-points to a real value (Section 3). If k is a sufficiently regular, then K is well-approximated by a *hierarchical matrix* [Hackbusch 2015], *i.e.*, a matrix of low-rank blocks (Figure 12). Encoding this matrix as a *block cluster tree (BCT)* enables fast matrix-vector multiplication via the *fast multipole method* [Greengard and Rokhlin 1997]. Like the BVH, our BCT involves both positions *and* tangents; in fact, each BCT block corresponds to a pair of BVH nodes. See Appendix B.2 for details.

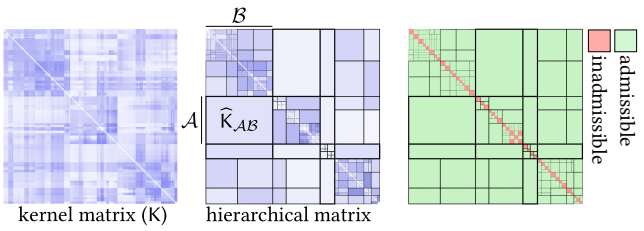


Fig. 12. *Left:* A kernel matrix K encodes interactions between all pairs of edges. *Center:* To accelerate multiplication, this matrix is approximated by rank-1 blocks $\widehat{K}_{\mathcal{AB}}$, corresponding to pairs $(\mathcal{A}, \mathcal{B})$ of distant BVH nodes. *Right:* For pairs that are too close, this approximation is *inadmissible*, and we must use the original matrix entries.

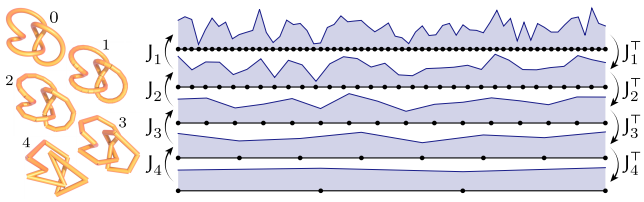


Fig. 13. We accelerate linear solves using multigrid on a hierarchy of curves.

6.3 Multigrid Solver

Since the hierarchical matrix-vector multiply does not build an explicit matrix, we use an iterative method to solve our linear systems. Empirically, off-the-shelf methods such as *GMRES* and *BiCGStab* are not well-suited for our problem. Instead, we use geometric multigrid (Figure 13), since (i) it is straightforward to coarsen a curve network, and (ii) the low frequency modes of our Laplace-like operators are well-captured on a coarse mesh. In the Euclidean case, this type of approach has been used successfully by Ainsworth and Glusa [2017]. Appendix B.3 describes our geometric coarsening/prolongation operators, as well as our multigrid strategy for both Sobolev gradient evaluation and constraint projection.

7 EVALUATION AND COMPARISONS

We performed extensive evaluation and comparisons of our fractional Sobolev descent strategy relative to other methods. Here we give an overview of results; a detailed account of how these evaluations were performed can be found in supplemental material.

7.1 Dataset

We created two datasets of difficult knot embeddings: KNOT128, which contains random embeddings of 128 distinct isotopy classes from *KnotPlot*'s "knot zoo," and TREFOIL100, which contains 100 random embeddings of the trefoil knot (Figure 14). We also used the *Freedman unknot* (Figure 2, *top left*), which is a standard "challenge problem" from the knot energy literature [Scharein 1998, Section 3.3]. To examine scaling under refinement, we performed regular refinement on knots from each of these sets.

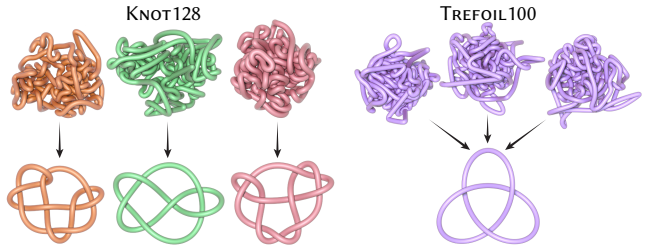
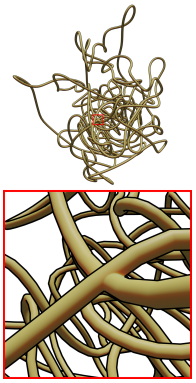


Fig. 14. To evaluate performance, we built a "stress test" dataset of 128 random embeddings of different knot classes *(left)* and 100 random embeddings of the trefoil knot *(right)*. The tangent point energy drives these curves toward much simpler embeddings, as shown here.

7.2 Performance Comparisons

We compared our fractional Sobolev descent strategy to a variety of methods from optimization and geometry processing. Overall, methods that use our fractional preconditioner performed best, especially as problem size increases. We first ran all methods on several resolutions of a small set of test curves (Figure 18); we then took the fastest methods, and ran them on all 228 curves from our two datasets (Figure 19). For simplicity we did not use hierarchical acceleration in our method (and instead just solve dense systems), which gave a *significant* performance advantage to alternative methods (which are based on sparse solves). Even with this handicap, the fractional approach outperformed all other methods; as indicated in Figure 20, hierarchical acceleration would widen this gap even further.



Importantly, previous methods also have a much higher failure rate at untangling difficult curves such as those in our dataset (Figure 19). Further, cases on which the fractional approach itself fails generally contain near-intersections in the initial configuration (inset), which also lead to failures in most or all other methods.

Note that some previous methods do not directly handle hard nonlinear constraints; for these methods we perform an apples-to-apples comparison by replacing—in *all* methods—hard edge length constraints with a soft elastic penalty (see supplemental material for further details).

Knot untangling methods. We first compared to two well-known methods for knot untangling (Figure 15): *KnotPlot*, based on the so-called *symmetric energy*, and *shrink on no overlaps (SONO)* [Pierański 1998] which performs a local iterative projection in the spirit of contemporary *position-based dynamics* [Müller et al. 2007]. Both methods successfully untangle the Freedman knot, but only after tens of thousands of iterations [Scharein 1998, Figure 7.6]. The basic reason is that, like L^2 descent, such methods focus on reduction of local error, making global convergence quite slow.

1st-order methods. Figure 16 indicates that basic 1st-order schemes like ordinary L^2 gradient descent, L-BFGS using 10, 30, or 100 vectors, and nonlinear conjugate gradients *à la* Fletcher and Reeves [1964] exhibit poor performance relative to our fractional scheme

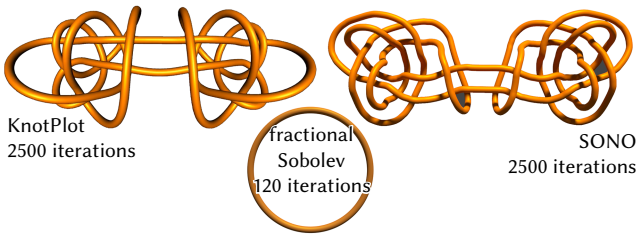


Fig. 15. Our fractional Sobolev strategy is dramatically more efficient than previous methods for knot untangling—here we untangle the unknot from Figure 2. Neither KnotPlot nor SONO converged after several hours.

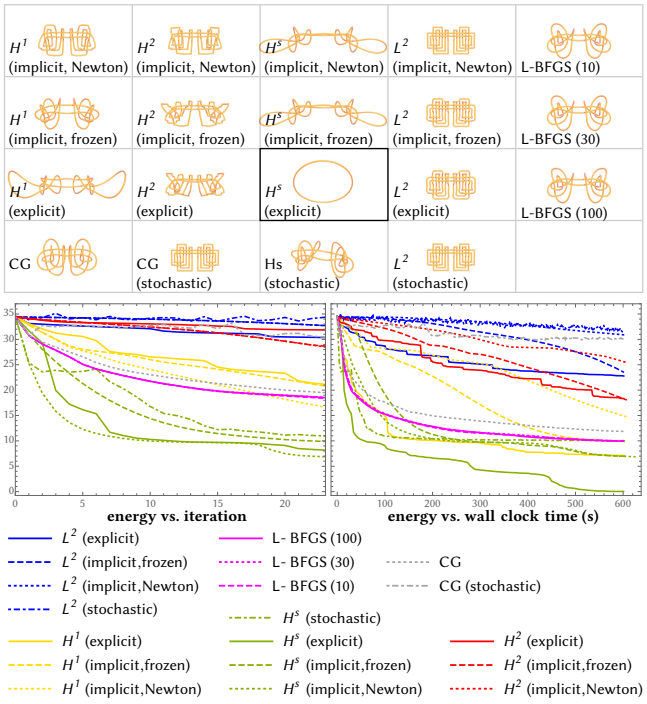


Fig. 16. Across a wide variety of descent methods and inner products, our fractional Sobolev approach does significantly better both in terms of energy reduction per iteration *(middle left)* and real-world run time *(middle right)*. At top we show results for an equal amount of compute time.

in terms of both wall clock time and number of iterations. This example also indicates that for $1 < s < 2$, the next smallest or largest *integer* Sobolev preconditioners (H^1 and H^2) underperform the fractional H^s preconditioner, whether using explicit (forward) or implicit (backward) Euler. We solve the backward Euler update equation using Newton's method, either by updating the Hessian for each Newton step ("Newton"), or "freezing" the Hessian at the beginning of the time step ("frozen"). If Newton's method fails to converge within a few (10) iterations, the step size is halved and the solve is reattempted. We also tried stochastic gradient descent (SGD) with respect to the L^2 inner product, implemented by subsampling a fixed proportion (25% in our trials) of edge pairs (u, v) for each edge u in each iteration for energy and gradient evaluation. This method did far worse than any other scheme we tried. SGD with respect to H^s works better, but the speedup from stochastic evaluation does not compensate for the poor quality of the descent direction.

2nd-order methods. Second-order schemes like Newton's method can be adapted to nonconvex problems by projecting the Hessian onto a nearby positive-semidefinite matrix. Since a global projection is prohibitively expensive, a heuristic sometimes used in geometric optimization is to project and sum up the Hessians of each local energy term [Teran et al. 2005]; in our case we can decompose the energy into the edge-edge terms from Equation 17. Though this heuristic can work well for, *e.g.*, elastic energies, it does not appear to work very well for the tangent-point energy, and for larger examples had among the slowest run times of any scheme we tried (Figure 18).

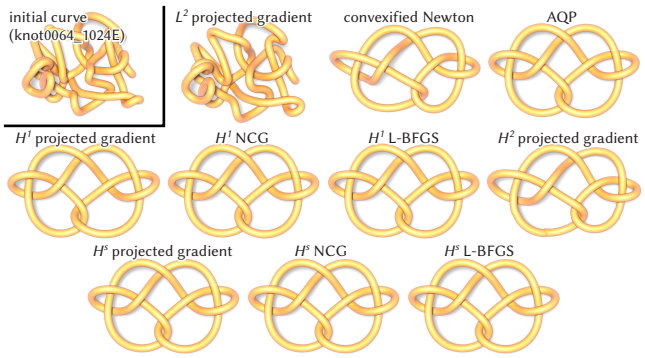


Fig. 17. The tangent point energy appears to have relatively few local minima; hence, different descent strategies tend to find the same local minimizers (though some, like L^2, do not find solutions in a reasonable amount of time). See supplemental material for several hundred more examples.

Quasi-Newton methods. Several recent methods from geometry processing apply Sobolev-like preconditioning to elastic energies, such as those used for shape deformation or surface parameterization [Kovalsky et al. 2016; Claici et al. 2017; Zhu et al. 2018]. Since the highest-order term in such problems often looks like a Dirichlet energy, H^1 preconditioning via the Laplacian Δ can be an effective starting point for optimization (as discussed in Section 4.1). However, such preconditioners do not perform as well as our fractional preconditioner, since they are not as well-matched to the order of the differential $d\mathcal{E}^{\alpha}_{\beta}$. For instance, as seen in Figure 18, the AQP strategy of Kovalsky et al. [2016] significantly underperforms our preconditioner when the Laplacian is used as the quadratic proxy; using our fractional operator as the quadratic proxy improves performance—but of course requires the machinery introduced in this paper. Another possibility is to use Laplacian-initialized L-BFGS (in the spirit of Zhu et al. [2018]); we found this strategy works a bit better than AQP, but again not as well as the fractional preconditioner. We also considered several variants of these strategies, such as applying Nesterov acceleration, and combining nonlinear conjugate gradients (NCG) *à la* Polak and Ribiere [1969] or L-BFGS with our fractional preconditioner. For hard constraints we advocate the use of our fractional (H^s) projected gradient scheme (as detailed in Section 5); if soft constraint enforcement is acceptable, then L-BFGS or H^s-preconditioned NCG are both good options: the former converges faster near minima; the latter gets stuck less often.

7.3 Local minimizers

As seen in Figure 17, the local minimizers found via our fractional descent strategy generally appear to be the same as with other schemes, up to rigid motions. Hundreds more such examples can be found in the supplemental material. Very rarely, two different methods produced local minimizers that were identical up to a *reflection*; such *amphichiral* pairs exist in some knot classes [Liang and Mislow 1994], but of course have the same energy.

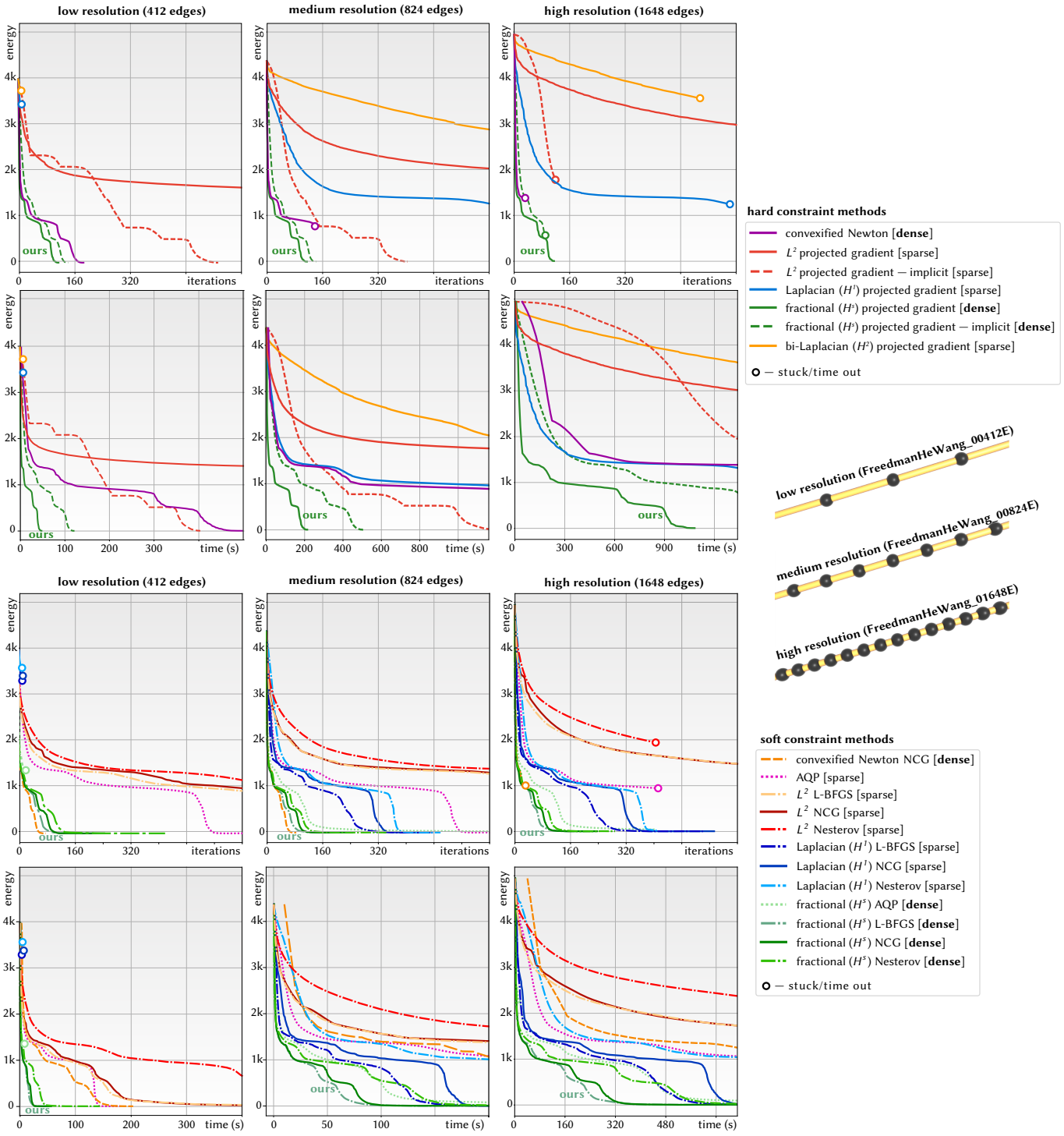


Fig. 18. We compared our descent strategy to a variety of 1st-order, 2nd-order, and quasi-Newton strategies, using both hard constraints (top) and a soft penalty (bottom) to preserve length. Here we show energy versus both time and iteration count for several resolutions of the initial curve from Figure 2; tests on additional curves yield very similar results (see supplemental material). Note that we achieve the best real-world clock time—even though we compare a dense implementation of our method (without hierarchical acceleration) to sparse versions of other schemes.

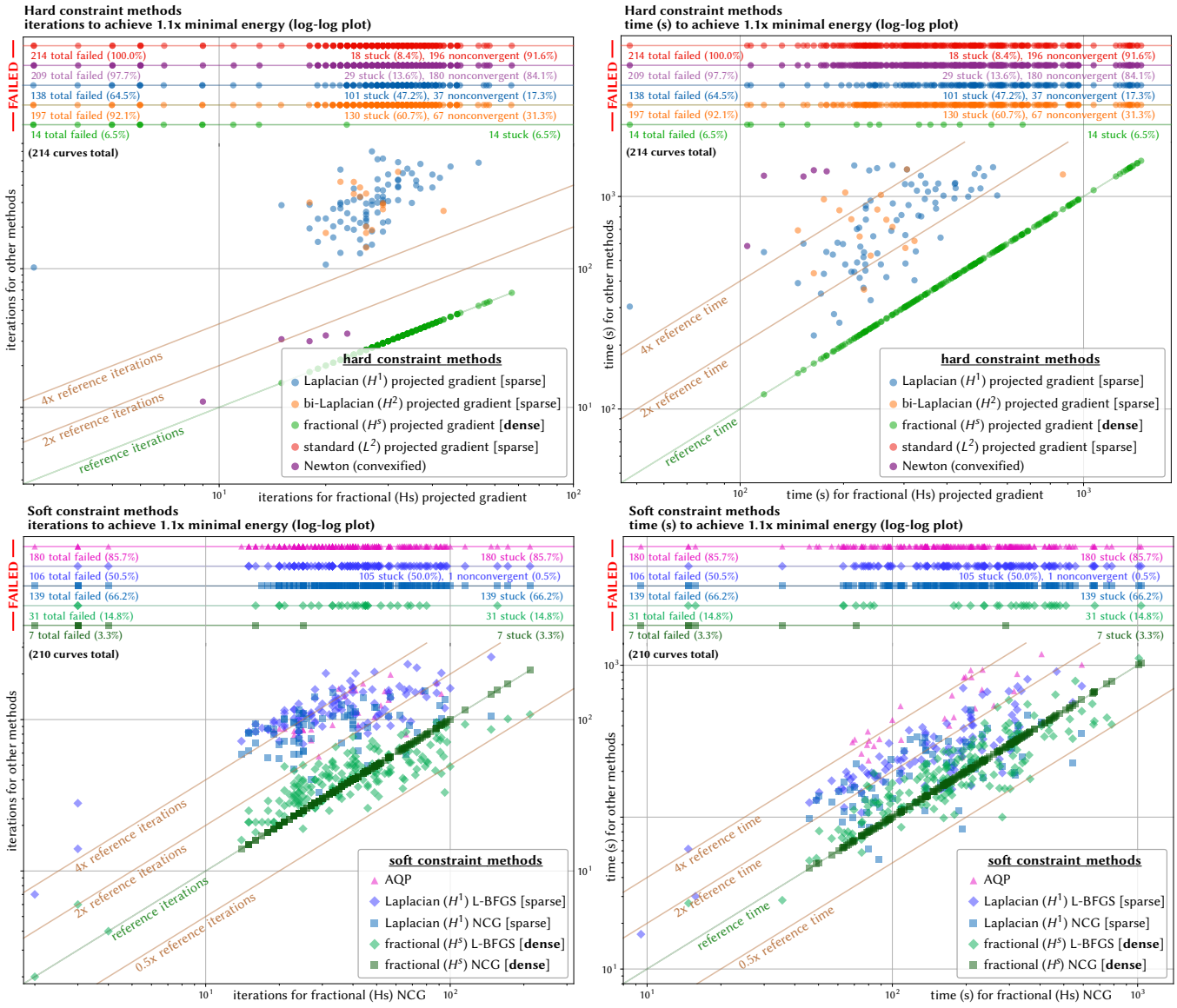


Fig. 19. We used a dataset of about two hundred difficult knot embeddings to evaluate the performance of our strategy compared to the next most competitive methods. Even without hierarchical acceleration, our fractional strategy was significantly faster—and succeeded at untangling a much larger fraction of knots. Here we plot the time it took for each method to get within 1.1x of the reference energy, against the time taken by our fractional strategy. Results have been split into hard/soft constraint enforcement (top/bottom rows), and iteration count/wall clock time (left/right columns). At the top of each plot we show the number of failures after 24 minutes of compute time—*stuck* indicates a failure of line search to make progress due to collisions; *nonconvergent* means the method failed to get below 1.1x of the reference energy.

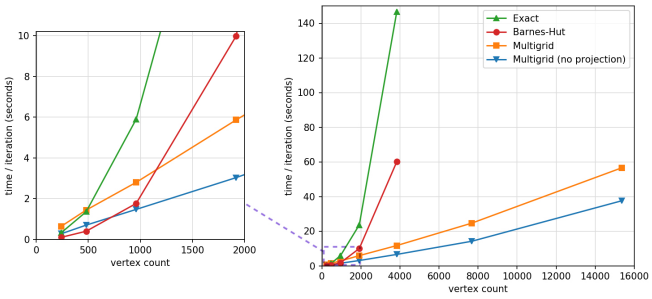


Fig. 20. A comparison of runtime per iteration on samplings of the same curve with increasing resolution. "Exact" indicates no acceleration, "Barnes-Hut" indicates accelerated gradients only, and "Multigrid" indicates all accelerations enabled, with and without constraint projection. Reported numbers are averages over up to 500 iterations or until convergence.

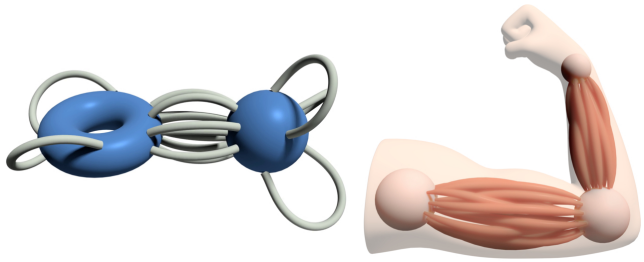


Fig. 21. Allowing curves to slide freely over constraint surfaces *(left)* enables design tasks like arranging networks of muscles or muscle fibers *(right)*.

7.4 Scaling behavior

We compared per-iteration costs of the unaccelerated scheme, a scheme using only Barnes-Hut (Section 6.1), and the full acceleration scheme described in Section 6—see Figure 20. With full acceleration we observe near-linear scaling, whereas schemes that directly solve the dense system exhibit super-quadratic scaling and quickly run out of memory. Note that constraint projection with direct solvers comes nearly for free, since a factorization of Equation 23 can be reused to solve Equation 22. In contrast, no reuse is possible in the fully accelerated scheme, making constraint projection relatively expensive. Disabling this step further speeds up the accelerated scheme, but leads to constraint drift over time. Alternative methods for constraint enforcement (such as soft penalties, as noted above) might hence provide further improvement.

8 RESULTS AND APPLICATIONS

Given how ubiquitous plane and space curves are in areas like geometry, graphics, robotics, and visualization—and how natural it is to want to avoid collision of such curves—our method provides a useful computational framework for a wide variety of tasks. Here we explore some preliminary applications that we hope will inspire future work. All other examples in this section completed within a few minutes, except for the 3D curve packing example where we allowed curves to grow longer for several hours as a stress test. We first describe constraints and potentials used for these examples.

8.1 Constraints and Potentials

A key feature of our optimization framework is that it not only efficiently minimizes knot energies, but that it can do so in conjunction with fairly arbitrary user-defined constraints and penalties (Section 5.3). This opens the door to a rich variety of computational design applications beyond the basic "knot untangling" that has been the focus of previous work. For the applications that will be explored in Section 8, we consider the following constraints:

- **Barycenter.** This fixes the barycenter of the curve to a point x_0 via the constraint $\Phi_{\text{barycenter}}(\gamma) := \sum_{I\in E} \ell_I(x_I - x_0)$. In the absence of other constraints, this eliminates the null space of globally constant functions discussed in Appendix A.
- **Length.** The repulsive curve energy naturally wants to make the curve longer and longer. A simple way to counteract this is via a total length constraint $\Phi_{\text{length}}(\gamma) := L^0 - \sum_{I\in E} \ell_I$, where L^0 is the target length.
- **Edge Length.** We can also constrain the lengths of each individual edge, allowing only isometric motions. This entails a constraint $\Phi_{\text{length},I}(\gamma) := \ell_I^0 - \ell_I$ for each edge I, where ℓ_I^0 is the target edge length.
- **Point Constraint.** To fix the position of a vertex i to the point $x_i \in \mathbb{R}^3$, we can add the constraint $\Phi_{\text{point},i}(\gamma) := \gamma_i - x_i$.
- **Surface Constraint.** To keep a point of the curve constrained to an implicit surface $f(x) = 0$, we can add the constraint $\Phi_{\text{surface},i}(\gamma) := f(\gamma_i)$.
- **Tangent Constraint.** We can force the tangent T_I of an edge I to match a unit vector $X \in \mathbb{R}^3$ via the constraint $\Phi_{\text{tangent},I}(\gamma) := T_I - X$.

In several applications, we progressively increase or decrease the target length values L_0 or l_I^0; the next constraint projection step then enforces the new length. We also consider the following penalties:

- **Total length.** A simple energy is the total curve length, which provides a "soft" version of the total length constraint. Discretely, this energy is given by $\hat{\mathcal{E}}_{\text{length}}(\gamma) := \sum_{I\in E} \ell_I$.
- **Length difference.** This energy penalizes differences in adjacent edge lengths, given by $\hat{\mathcal{E}}_{\text{diff}}(\gamma) = \sum_{v\in V_{\text{int}}} (\ell_{I_v} - \ell_{J_v})^2$, where V_{int} denotes the set of "interior" vertices with degree 2, and I_v and J_v are the indicent edges to v.
- **Surface potential.** Given a surface $M \subset \mathbb{R}^3$, we use the energy $\mathcal{E}_M(\gamma) := \int_\gamma \int_M 1/|x_M - \gamma(x_\gamma)|^{\beta-\alpha} dx_M\, dx_\gamma$ to avoid collisions. This is effectively a Coulomb potential of the same order as $\mathcal{E}_\beta^\alpha$ on M. In the discrete setting, M is a triangulated surface, and we use a BVH on M to accelerate the evaluation of $\mathcal{E}_M$ and its differential, in a similar fashion to $\mathcal{E}_\beta^\alpha$.
- **Field potential.** Given a fixed unit vector field X on $\mathbb{R}^3$, the energy $\mathcal{E}_X(\gamma) := \int_0^L |T(x) \times X(\gamma(x))|^2\, dx_\gamma$ encourages γ to run parallel (or anti-parallel) to X. We discretize this as $\hat{\mathcal{E}}_X(\gamma) := \sum_{I\in E} \ell_I |T_I \times X(x_I)|^2$.

Note that the energies considered here involve lower-order derivatives than those in $\mathcal{E}_\beta^\alpha$, and do not therefore have a major effect on the stiffness of the overall system. Hence, we can continue to use the fractional Sobolev inner product without modification to define an efficient gradient flow.

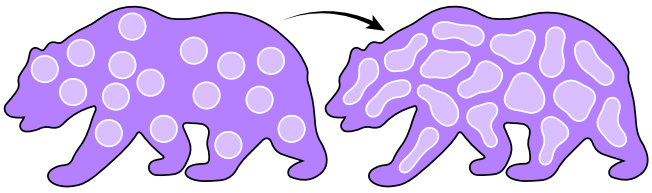


Fig. 22. Just as repulsive potentials are commonly used to find equally-distributed points, we can compute collections of equally-spaced curves (here constrained to a region via a fixed curve potential).

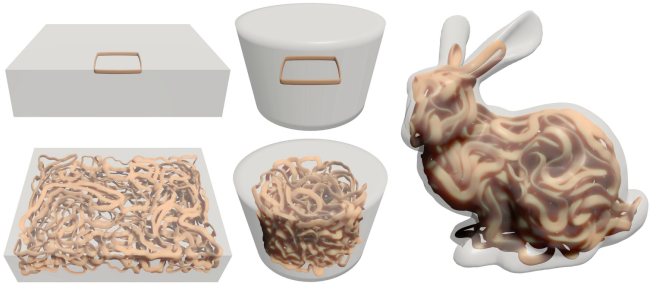


Fig. 23. By penalizing proximity to a fixed surface, we can pack curves into any domain. Progressively increasing edge length forces curves to maintain a balance between surface avoidance and self-avoidance. (Here we render curves with a non-circular cross section, which is not modeled by the energy.)

8.2 Curve Packing

Packing problems (such as *bin packing*) appear throughout geometry and computer graphics, playing an important role in, *e.g.*, 2D layouts for manufacturing or UV atlas generation. An adjacent problem is generation of regular sampling patterns, *e.g.*, blue noise sampling via Poisson disk rejection. The ability to optimize large families of repulsive curves enables us to solve analogous "curve packing" problems—for instance, in Figure 22, we use a fixed boundary curve to pack disks of increasing length; likewise, in Figures 1 and 23, we use a surface penalty to pack increasingly long curves into a target region. Figure 24 likewise packs increasingly long curves on a surface. Going the opposite direction, we can also *decrease* length while encouraging repulsion to generate clean illustrations that are difficult to draw by hand (Figure 25). Finally, by constraining only parts of curves to lie on surfaces, we can design biologically-inspired curve networks such as muscle fibers (Figure 21), which are attached to objects at their endpoints but are otherwise free.

8.3 Graph Drawing

A basic problem in data visualization is drawing *graphs*; a typical approach is to use a force-based layout that seeks to avoid, *e.g.*, collisions between nodes, or over/under-extension of edges [Fruchterman and Reingold 1991]. Our framework makes it easy to optimize the geometry of the edges themselves, opening the door to graph layouts that are both more compact and more legible (Figure 26). We can also use this machinery to obtain legible drawings of nonplanar graphs, by perturbing a planar embedding (Figure 27); here, the ability to preserve lengths conveys information about edge weights. A particularly interesting graph embedding problem is the design of synthetic hydrogel *vascular networks* [Grigoryan et al. 2019]; Figure 28 shows a simple example where we optimize a multivascular network (starting from subgraphs of a tet mesh and its dual).

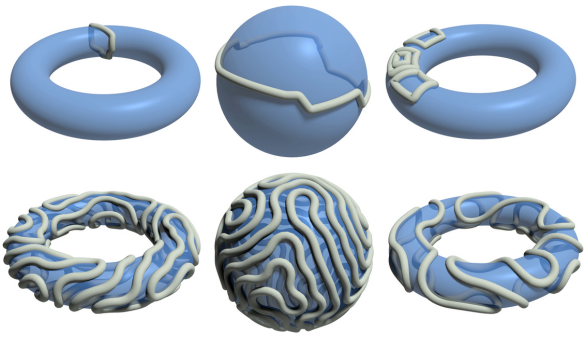


Fig. 24. Patterns obtained by constraining a collection of repulsive curves to a surface and increasing their lengths (initial states shown above their final configurations).

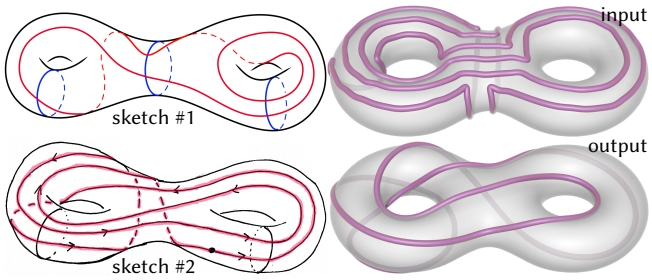


Fig. 25. Loops arising in topology can be difficult to draw by hand—the sketches at left were done by Nathan Dunfield to illustrate *Dehn-Thurston* coordinates. At right we generate an equispaced version of this curve by flowing a rough sketch, subject to an implicit surface constraint.

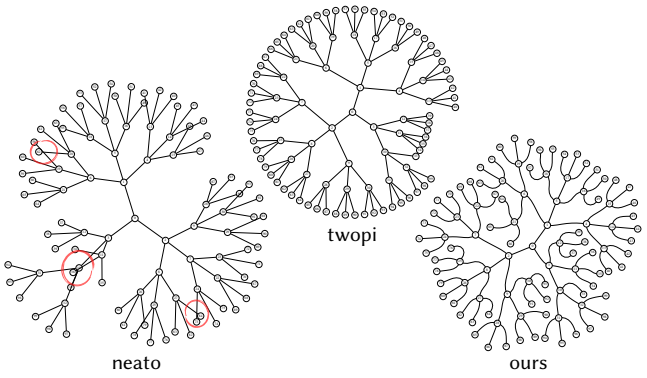


Fig. 26. Traditional 2D graph drawing algorithms based on nodal proximity may cause edges to cross *(left)* or position nodes extremely close together *(center)*; these layouts were produced by the popular *Graphviz* library [Ellson et al. 2001]. By treating edges as repulsive curves, we can obtain graph drawings that are both more compact and more legible *(right)*.

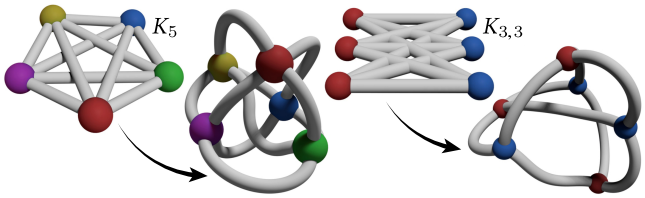


Fig. 27. Isometric embedding: by jittering 2D drawings of non-planar graphs (which necessarily have crossings), curve repulsion with length constraints yields nicely spaced embeddings in R^3 with prescribed edge lengths.

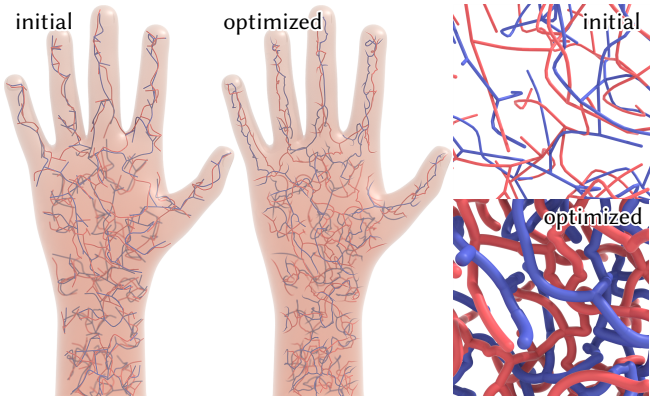
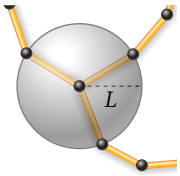


Fig. 28. *Left:* a crude initial topology for a synthetic vascular network *(left)* is optimized to achieve more uniform delivery of nutrients throughout a volume. *Right:* plotting the maximum collision-free thickness helps to visualize the improvement in uniformity.



Note that at junctures between more than two edges, the tangent-point energy will always be large (since three or more edges cannot be collinear), rapidly forcing vertices away from each other. This can be counteracted by constraining their edge lengths, forcing the vertices to lie on spheres of constant radii around the junctures.

8.4 Self-Avoiding Splines

Beyond standard Bézier input, sophisticated tools have been developed for drawing spline curves—but do not consider the basic constraint of ensuring that curves do not cross themselves (which is often desirable for physical or aesthetic reasons). For instance, Figure 30 (*center*) shows the interpolation of a set of control points by *k-curves* [Yan et al. 2017], which underpin one of the basic drawing

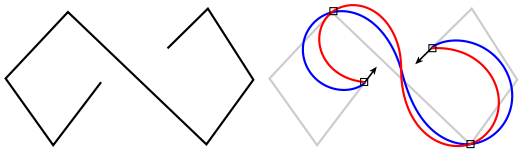


Fig. 29. As with Bézier curves, we can also control curve tangents at both interior and endpoints. Here we flow a polygonal curve *(left)*, to a smooth interpolant with fixed points *(red)*, and fixed points and tangents *(blue)*.

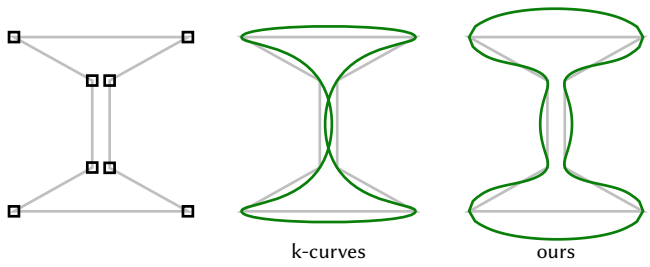


Fig. 30. Standard curve interpolation methods in 2D drawing programs can cause curves to self-intersect *(center)*, even when the control polygon *(left)* does not. By starting from the control polygon and constraining the control points, we obtain a smooth, non-intersecting interpolant *(right)*.

tools in Adobe Illustrator (the *Curvature Tool*). By simply applying point constraints at the control points, and letting the length increase under our repulsive flow, we obtain a nice interpolating curve without self-intersection (Figure 30, *right*). In this context we can also use our tangent constraint to control the behavior of such a curve at open endpoints (Figure 29).

8.5 Multi-agent Path Planning

In robotics, numerous algorithms have been developed for the problem of multi-agent path planning [de Wilde et al. 2013], wherein multiple agents must travel from fixed start to end locations without colliding with the environment or each other. Many algorithms operate on a discrete grid or graph [Yu and LaValle 2013], which quantizes the solution space and does not penalize near-collisions; such trajectories may therefore not be robust to sensing or control error. By treating path planning as a space-time optimization of continuous curves with fixed endpoints, we can use curve repulsion to find (or refine) trajectories that maximize collision avoidance, making them more resilient to error (Figure 31). Finding such trajectories in n dimensions is equivalent to optimizing a braid in $n + 1$ dimensions; since neither the size of the curve nor the cost

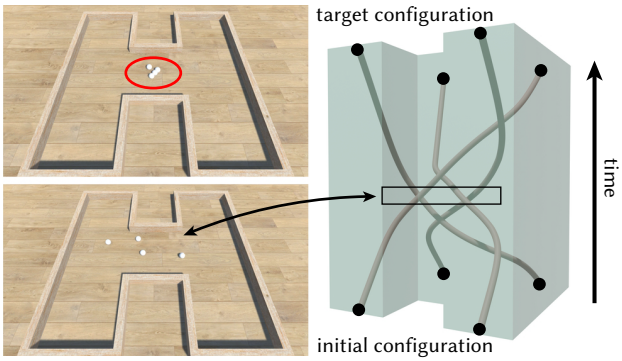


Fig. 31. *Top-left:* In this path planning scenario, an initial trajectory brings the four agents dangerously close together. *Bottom-left:* By treating trajectories as curves in space-time, our system provides solutions that maximally avoid collisions, making them more robust to control errors. *Right:* Finding 2D trajectories is equivalent to optimizing a 3D braid with fixed endpoints constrained to an extrusion of the given environment. This same construction can easily be generalized to 3D environments.

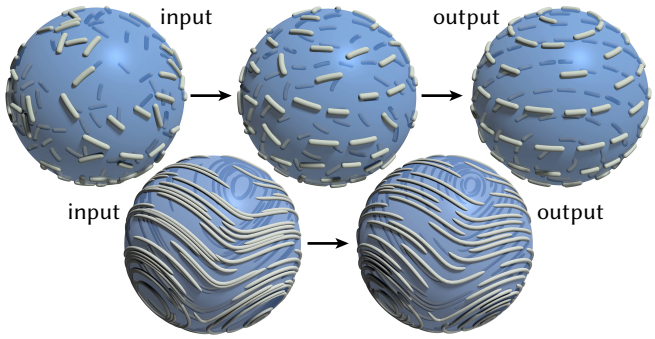


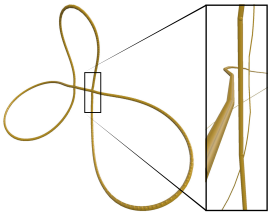
Fig. 32. Encouraging curve tangents to align with a given vector field improve the quality of streamline visualization. Here, a random set of curve segments *(top)* aligns itself with a rotational vector field; we can also optimize randomly sampled streamlines *(bottom)* to improve their spacing.

of a BVH/BCT depends strongly on dimension, this strategy easily generalizes to three (or more) dimensions.

8.6 Streamline Visualization

A common way to visualize vector fields is by tracing integral curves or *streamlines*; significant effort has gone into algorithms that provide uniform spacing (*e.g.*, by incrementally constructing a Delaunay triangulation [Mebarki et al. 2005]), though such methods can be difficult to generalize to 3D volumes or vector fields on surfaces. We can generate nicely-spaced streamlines by adding a field alignment potential to the tangent-point energy—for instance, in Figure 32 we start with a set of random curve segments, which automatically coalesce into streamlines.

9 LIMITATIONS AND CONCLUSION



Since we approximate the tangent-point energy via numerical quadrature, it is possible for a very coarse curve to pass through the energy barrier. However, crossings can be prevented via continuous time collision detection (Section 5.4); to maintain accuracy one could also try adding more quadrature points at the previous time step if any collisions occur. For the design tasks in this paper, we did not find such strategies necessary. Also on very coarse meshes, edges that are extremely close together can temporarily get stuck in a near-crossing configuration (see inset). In this situation, the term k_4^2 from the low-order term (Equation 13) is very large, causing the inverse of A—and hence the Sobolev gradient—to be very small. One idea is to use adaptive quadrature for edge pairs that are close in space, which would better resolve the near-infinite high-order term and hence push the curve apart. Given the scalability of our approach, another pragmatic solution is simply to increase the overall resolution.

There are many ways to further accelerate our solver. For instance, we did not vectorize our code, parallelized only the matrix-vector multiply in non-well-separated leaves of the BCT, and did not make use of the GPU. For small time steps one might re-fit rather than re-build the BVH; likewise, it may be beneficial to incrementally update the BCT. Better line search or descent direction heuristics may also reduce the overall number of steps.

Fig. 33. Untangling a pair of earbuds via repulsion (see supplemental video).

Finally, a natural question is how to extend these techniques to *surface* repulsion. The tangent-point energy seems attractive here since (unlike Möbius energy) it needs only Euclidean rather than geodesic distances. One now has double integrals over *surfaces*, but might still achieve efficiency via hierarchical acceleration. In general, we are hopeful our investigation will provide valuable insight into using repulsive energies for computational design.

ACKNOWLEDGMENTS

Thanks to Stelian Coros for early discussion of these ideas. The bunny mesh is used courtesy of the Stanford Computer Graphics Laboratory. This work was supported by a Packard Fellowship, NSF awards 1717320 and 1943123, and gifts from Autodesk, Adobe, Activision Blizzard, Disney, and Facebook. The second author was supported by a postdoc fellowship of the German Academic Exchange Service and by DFG-Project 282535003: *Geometric curvature functionals: energy landscape and discrete methods.*

REFERENCES

M. Ainsworth and C. Glusa. 2017. Aspects of an Adaptive Finite Element Method for the Fractional Laplacian. *Comput. Methods Appl. Mech. Eng.* 327 (2017).
B. Andrews, B. Chow, C. Guenther, and M. Langford. 2020. *Extrinsic Geometric Flows.* Graduate Studies in Mathematics, Vol. 206.
A. Angelidis and F. Neyret. 2005. Simulation of smoke based on vortex filament primitives. In *Symp. Comp. Anim.* 87–96.
T. Ashton and J. Cantarella. 2005. A fast octree-based algorithm for computing rope-length. In *Physical And Numerical Models In Knot Theory.* 323–341.
T. Ashton, J. Cantarella, M. Piatek, and E. Rawdon. 2011. Knot tightening by constrained gradient descent. *Experimental Mathematics* 20, 1 (2011), 57–90.
Trygve Bærland. 2019. An Auxiliary Space Preconditioner for Fractional Laplacian of Negative Order. *arXiv preprint arXiv:1908.04498* (2019).
Trygve Bærland, Miroslav Kuchta, and Kent-Andre Mardal. 2019. Multigrid methods for discrete fractional Sobolev spaces. *SIAM J. Sci. Comput.* 41, 2 (2019), A948–A972. https://doi.org/10.1137/18M1191488
J. Barnes and P. Hut. 1986. A hierarchical O(N log N) force-calculation algorithm. *Nature* 324, 6096 (1986), 446–449.
S. Bartels, P. Reiter, and J. Riege. 2018. A simple scheme for the approximation of self-avoiding inextensible curves. *IMA J. Num. Anal.* 38, 2 (2018), 543–565.
M. Bergou, M. Wardetzky, S. Robinson, B. Audoly, and E. Grinspun. 2008. Discrete elastic rods. In *ACM Trans. Graph.*, Vol. 27. ACM, 63.
S. Blatt. 2012. Boundedness and Regularizing Effects of O'hara's Knot Energies. *Journal of Knot Theory and Its Ramifications* 21, 01 (2012).
S. Blatt. 2013. The Energy Spaces of the Tangent Point Energies. *Journal of Topology and Analysis* 5, 3 (2013), 261–270.
S. Blatt and P. Reiter. 2015. Regularity Theory for Tangent-point Energies: the Non-degenerate Sub-critical Case. *Adv. Calc. Var.* 8, 2 (2015), 93–116.
S. Boyd and L. Vandenberghe. 2004. *Convex Optimization.* Cambridge University Press.
D. Braess and R. Sarazin. 1997. An Efficient Smoother for the Stokes Problem. *Applied Numerical Mathematics* 23, 1 (1997), 3–19.
K. Brakke. 1992. The surface evolver. *Experimental mathematics* 1, 2 (1992).
K. Brakke. 1994. Surface evolver manual.
J. Brown, J. Latombe, and K. Montgomery. 2004. Real-time Knot-tying Simulation. *The Visual Computer* 20, 2 (01 May 2004), 165–179.
G. Buck and J. Orloff. 1995. A simple energy function for knots. *Top. Appl.* 61, 3 (1995).
J. Calvo, K. Millett, and E. Rawdon. 2002. *Physical Knots: Knotting, Linking, and Folding Geometric Objects in* $\mathbb{R}^3$. Vol. 304. American Mathematical Society.
A. Chern, F. Knöppel, U. Pinkall, P. Schröder, and S. Weißmann. 2016. Schrödinger's smoke. *ACM Trans. Graph.* 35, 4 (2016), 77.
S. Claici, M. Bessmeltsev, S. Schaefer, and J. Solomon. 2017. Isometry-aware preconditioning for mesh parameterization. In *Comp. Graph. Forum*, Vol. 36.

K. Crane, U. Pinkall, and P. Schröder. 2013. Robust Fairing via Conformal Curvature Flow. *ACM Trans. Graph.* 32, 4 (2013).
B. de Wilde, A. ter Mors, and C. Witteveen. 2013. Push and rotate: cooperative multi-agent path planning. In *Proc. Conf. Auton. Agents and Multi-agent Sys.*
C. DeForest and C. Kankelborg. 2007. Fluxon Modeling of Low-beta Plasmas. *J. Atm. Sol.-Terr. Phys.* 69, 1-2 (2007), 116–128.
M. Desbrun, M. Meyer, P. Schröder, and A. Barr. 1999. Implicit Fairing of Irregular Meshes Using Diffusion and Curvature Flow. In *Proc. ACM SIGGRAPH*. 8.
E. Di Nezza, G. Palatucci, and E. Valdinoci. 2012. Hitchhiker's guide to the fractional Sobolev spaces. *Bull. Sci. Math.* 136, 5 (2012), 521–573.
I. Eckstein, J. Pons, Y. Tong, C. Kuo, and M. Desbrun. 2007. Generalized Surface Flows for Mesh Processing. In *Geometry Processing*, Alexander Belyaev and Michael Garland (Eds.). The Eurographics Association.
J. Ellson, E. Gansner, L. Koutsofios, S. North, and G. Woodhull. 2001. Graphviz: Open Source Graph Drawing Tools. In *Int. Symp. on Graph Drawing*. 483–484.
R. Fletcher and C. Reeves. 1964. Function minimization by conjugate gradients. *The computer journal* 7, 2 (1964), 149–154.
M. Freedman, Z. He, and Z. Wang. 1994. Mobius Energy of Knots and Unknots. *Annals of Mathematics* 139, 1 (1994), 1–50.
T. Fruchterman and E. Reingold. 1991. Graph drawing by force-directed placement. *Software: Practice and experience* 21, 11 (1991), 1129–1164.
L. Greengard and V. Rokhlin. 1997. A new version of the fast multipole method for the Laplace equation in three dimensions. *Acta numerica* 6 (1997).
B. Grigoryan, S. Paulsen, et al. 2019. Multivascular networks and functional intravascular topologies within biocompatible hydrogels. *Science* 364, 6439 (2019), 458–464.
W. Hackbusch. 2015. *Hierarchical matrices: algorithms and analysis*. Vol. 49. Springer.
D. Harmon, E. Vouga, B. Smith, R. Tamstorf, and E. Grinspun. 2009. Asynchronous Contact Mechanics. In *ACM Trans. Graph.*, Vol. 28. ACM, 87.
Y. Hassan, S. Easa, and A. Abd El Halim. 1998. State-of-the-art of Three-dimensional Highway Geometric Design. *Can. J. Civ. Eng.* 25, 3 (1998), 500–511.
D. Kleckner, L. Kauffman, and W. Irvine. 2016. How Superfluid Vortex Knots Untie. *Nature Physics* 12, 7 (2016), 650.
S. Kovalsky, M. Galun, and Y. Lipman. 2016. Accelerated quadratic proxy for geometric optimization. *ACM Trans. Graph.* 35, 4 (2016), 1–11.
B. Kubiak, N. Pietroni, F. Ganovelli, and M. Fratarcangeli. 2007. A Robust Method for Real-time Thread Simulation. In *Proc. ACM Symp. Virt. Real. Soft. Tech.* ACM, 85–88.
R. Kusner and J. Sullivan. 1998. Möbius-invariant knot energies. *Ideal knots* 19 (1998).
M. Kwaśnicki. 2017. Ten equivalent definitions of the fractional Laplace operator. *Fract. Calc. Appl. Anal.* 20, 1 (2017), 7–51.
M. Lackenby. 2014. Elementary Knot Theory. *Clay Mathematics Institute* (2014).
A. Ladd and L. Kavraki. 2004. *Motion Planning for Knot Untangling*. 7–23.
C. Liang and K. Mislow. 1994. On amphicheiral knots. *J. Math. Chem.* 15, 1 (1994).
T. Martin, P. Joshi, M. Bergou, and N. Carr. 2013. Efficient Non-linear Optimization via Multi-scale Gradient Filtering. In *Comp. Grap. Forum*, Vol. 32. 89–100.
F. Maucher and P. Sutcliffe. 2016. Untangling knots via reaction-diffusion dynamics of vortex strings. *Physical review letters* 116, 17 (2016), 178101.
J. McCrae and K. Singh. 2009. Sketching Piecewise Clothoid Curves. *Computers & Graphics* 33, 4 (2009), 452–461.
A. Mebarki, P. Alliez, and O. Devillers. 2005. Farthest point seeding for efficient placement of streamlines. In *IEEE Visualization*. 479–486.
H. Moreton. 1992. *Minimum curvature variation curves, networks, and surfaces for fair free-form shape design*. Ph.D. Dissertation. University of California, Berkeley.
M. Müller, B. Heidelberger, M. Hennix, and J. Ratcliff. 2007. Position based dynamics. *J. Vis. Comm. and Im. Repr.* 18, 2 (2007), 109–118.
J. O'Hara. 1991. Energy of a knot. *Topology* 30, 2 (1991), 241–247.
M. Padilla, A. Chern, F. Knöppel, U. Pinkall, and P. Schröder. 2019. On bubble rings and ink chandeliers. *ACM Trans. Graph.* 38, 4 (2019), 129.
J. Pérez, M. Otaduy, and B. Thomaszewski. 2017. Computational design and automated fabrication of kirchhoff-plateau surfaces. *ACM Trans. Graph.* 36, 4 (2017), 62.
J. Pérez, B. Thomaszewski, et al. 2015. Design and Fabrication of Flexible Rod Meshes. *ACM Trans. Graph.* 34, 4 (2015).
P. Pierański. 1998. In Search of Ideal Knots. In *Ideal Knots*, A. Stasiak, V. Katritch, and L. Kauffman (Eds.). Vol. 19. World Scientific.
U. Pinkall and K. Polthier. 1993. Computing discrete minimal surfaces and their conjugates. *Experimental mathematics* 2, 1 (1993), 15–36.
E. Polak and G. Ribiere. 1969. Note sur la convergence de méthodes de directions conjuguées. *ESAIM: Math. Model. Num. Anal.* 3, R1 (1969), 35–43.
S. Redon, A. Kheddar, and S. Coquillart. 2002. Fast continuous collision detection between rigid bodies. In *Comp. Graph. Forum*, Vol. 21. 279–287.
R. Renka and J. Neuberger. 1995. Minimal Surfaces and Sobolev Gradients. *SIAM Journal on Scientific Computing* 16, 6 (1995), 1412–1427.
R. Scharein. 1998. *Interactive Topological Drawing*. Ph.D. Dissertation. University of British Columbia.
H. Schumacher. 2017. On H^2-gradient Flows for the Willmore Energy. *arXiv e-prints* (Mar 2017).
J. Smith and S. Schaefer. 2015. Bijective parameterization with free boundaries. *ACM Trans. Graph.* 34, 4 (2015), 1–9.
P. Strzelecki and H. von der Mosel. 2017. Geometric curvature energies: facts, trends, and open problems. In *New Directions in Geometric and Applied Knot Theory*.
J. Teran, E. Sifakis, G. Irving, and R. Fedkiw. 2005. Robust quasistatic finite elements and flesh simulation. In *Symp. Comp. Anim.* 181–190.
H. Triebel. 1983. *Theory of function spaces*. Monographs in Mathematics, Vol. 78.
S. Walker. 2016. Shape optimization of self-avoiding curves. *J. Comp. Phys.* 311 (2016).
S. Weißmann and U. Pinkall. 2010. Filament-based smoke with vortex shedding and variational reconnection. In *ACM Trans. Graph.*, Vol. 29.
Z. Yan, S. Schiller, G. Wilensky, N. Carr, and S. Schaefer. 2017. k-curves: interpolation at local maximum curvature. *ACM Trans. Graph.* 36, 4 (2017).
C. Yu, K. Crane, and S. Coros. 2017. Computational Design of Telescoping Structures. *ACM Trans. Graph.* 36, 4 (2017).
J. Yu and S. LaValle. 2013. Multi-agent path planning and network flow. In *Algorithmic Foundations of Robotics X*. Springer, 157–173.
J. Zehnder, S. Coros, and B. Thomaszewski. 2016. Designing structurally-sound ornamental curve networks. *ACM Trans. Graph.* 35, 4 (2016), 99.
Y. Zhu, R. Bridson, and D. Kaufman. 2018. Blended cured quasi-newton for distortion optimization. *ACM Trans. Graph.* 37, 4 (2018), 1–14.

A SOBOLEV-SLOBODECKIJ GRADIENT

How do we obtain an ideal gradient flow for the tangent-point energy (*i.e.*, one that behaves like an ODE)? Unlike standard energies (elastic energy, Willmore energy, *etc.*), an answer to this question has not yet been worked out formally. However, we can make an educated guess based on past wisdom about curve energies.

In general, suppose an energy $\mathcal{E}$ has a (Fréchet) differential $d\mathcal{E}$. To determine the highest-order derivatives, it is not necessary to derive an explicit expression for $d\mathcal{E}$ as we did for the Dirichlet energy (Section 4.1). Instead, we can reason about the associated function spaces: as long as we know the *order* of $d\mathcal{E}$, we can "cancel" spatial derivatives by constructing an inner product of the same order.

For the tangent-point energy, existing analysis gives the maximum order of derivatives in $\mathcal{E}^\alpha_\beta$ (Appendix A.2), from which we deduce the order of $d\mathcal{E}^\alpha_\beta$ (Appendix A.3). What is unusual here is that the number of derivatives is *fractional* (Appendix A.1.2); to build an inner product of appropriate order, we therefore start with the *fractional Laplacian* (Section A.1), and formulate an analogous operator for embedded curves. Taking further (integer) derivatives then yields an operator of the same order as $d\mathcal{E}^\alpha_\beta$ (Appendix A.4). From there, we use additional heuristics (inspired by numerical experiments) to choose a low-order term that makes this operator well-behaved and invertible (Appendix A.4.2), allowing us to use it in the definition of a fractional Sobolev gradient (Section 4.2).

A.1 Fractional Analysis

We begin with a brief discussion of Sobolev spaces of *fractional* order $k \notin \mathbb{Z}$; for further background, see [Di Nezza et al. 2012].

A.1.1 Fractional Differential Operators. Whereas standard differential operators L are purely *local* (*i.e.*, the value of $(Lu)(x)$ depends only on an arbitrarily small neighborhood of $u(x)$), fractional differential operators are *nonlocal* ($(Lu)(x)$ can depend on the value of u at any point y). Since the tangent-point energy is nonlocal, it will also have nonlocal derivatives. Hence, finding an inner product well-matched to

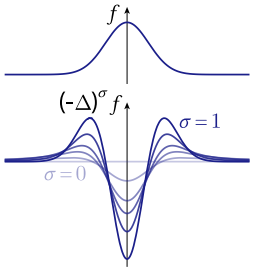


Fig. 34. Fractional Laplacian of f for several values of σ.

its gradient flow entails constructing
an appropriate fractional differential operator—an important example in our setting is the *fractional Laplacian* $(-\Delta)^\sigma$ on $\mathbb{R}^n$, which is commonly defined by taking powers of the eigenvalues in the spectral expansion. For $\sigma \in (0,1)$ and all sufficiently regular $u, v : \mathbb{R}^n \to \mathbb{R}$, the operator can also be expressed via the integral

$$\langle\langle (-\Delta)^\sigma u, v \rangle\rangle = C \iint_{\mathbb{R}^n \times \mathbb{R}^n} \frac{u(x)-u(y)}{|x-y|^\sigma} \frac{v(x)-v(y)}{|x-y|^\sigma} \frac{dxdy}{|x-y|^n}, \quad (24)$$

where the constant $C \in \mathbb{R}$ depends only on n and σ [Kwaśnicki 2017]. The behavior of this operator is illustrated in Figure 34.

A.1.2 Fractional Sobolev Spaces. There are two common ways to understand Sobolev spaces of fractional order. One is to consider the Fourier transform of the Laplacian Δ, leading to the *Bessel potential spaces* $H^{s,p} := (-\Delta)^{-s/2}(L^p)$ [Triebel 1983, Section 2.2.2]. For us, however, this viewpoint helps only to understand the case $W^{s,2}$. The other, essential for studying the tangent-point energy, is via the *Sobolev-Slobodeckij spaces* $W^{k+\sigma,p}$. Functions u in these spaces look like functions in an ordinary Sobolev space, but with a nonlocal regularity condition on the highest-order derivative $u^{(k)}$. In particular, suppose we write $s = k + \sigma$ for $k \in \mathbb{Z}_{\geq 0}$ and $\sigma \in (0,1)$. Then, on an n-dimensional Riemannian manifold M, one defines

$$W^{k+\sigma,p}(M) := \left\{ u \in W^{k,p}(M) \;\middle|\; [u^{(k)}]_{W^{\sigma,p}} < \infty \right\}.$$

The expression in square brackets is the (Gagliardo) semi-norm

$$[u]_{W^{\sigma,p}} := \left(\iint_{M^2} \left| \frac{u(x)-u(y)}{d(x,y)^\sigma} \right|^p \frac{dx\, dy}{d(x,y)^n} \right)^{1/p},$$

where $d(x,y)$ is the shortest distance between x and y in M. Just as a Lipschitz function is more regular than an arbitrary continuous function without being differentiable, a function in $W^{k+\sigma,p}$ is more regular than one in $W^{k,p}$, without getting a whole additional derivative (*i.e.*, $W^{k+1,p} \subsetneq W^{k+\sigma,p}$). Figure 35 shows an example.

Dual Space. Just as the dual of the classical Sobolev space $W^{k,p}$ is $W^{-k,q}$ (where $1/p + 1/q = 1$), the dual of the Sobolev-Solobdeckij space $W^{s,p}$ can be characterized as a space with "$-s$ derivatives" in the sense that the fractional Laplacian $(-\Delta)^s$ identifies $W^{s,p}$ with $W^{-s,q} := (W^{s,p})^*$ [Di Nezza et al. 2012, Remark 2.5].

A.2 Energy Space

To determine the order of the tangent-point differential $d\mathcal{E}^\alpha_\beta$, we first consider the biggest space of functions for which the energy $\mathcal{E}^\alpha_\beta$ is well-defined. Blatt [2013] gives the following condition on the differentiability of the curve γ (see also Blatt and Reiter [2015]):

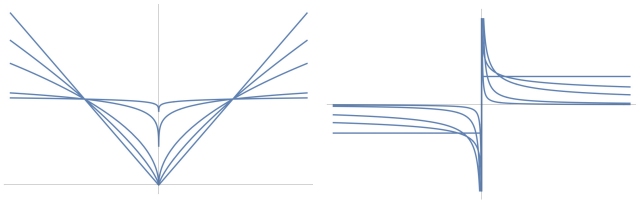


Fig. 35. The curves $(x, |x|^\sigma)$ are examples of curves in $W^{\sigma,p}$ *(left)*. Their 1st derivatives are not L^p integrable *(right)*.

Lemma A.1. *Suppose $\alpha > 1$ and $\beta \in [\alpha + 2, 2\alpha + 1)$, let $s := \frac{\beta}{\alpha} - \frac{1}{\alpha}$, and consider an embedded curve $\gamma \in C^1(S^1; \mathbb{R}^3)$. Then γ has finite tangent point energy $\mathcal{E}^\alpha_\beta(\gamma)$ if and only if, up to reparameterization, $\gamma \in W^{s,\alpha}(S^1; \mathbb{R}^3)$.*

In other words, the tangent point energy is well-defined only for curves that have an sth derivative, and for which the αth power of that derivative is integrable—for example, it will not be finite for a polygonal curve. The somewhat unusual situation is that s is not an integer: instead, it is a fractional value in the interval $(1,2)$.

A.3 Order of the Differential

In general, if an energy $\mathcal{E}$ is defined for functions in a space X, then its differential $d\mathcal{E}^\alpha_\beta$ will have the prototype $d\mathcal{E} : X \to X^*$, where X^* is the dual space. For instance, the Dirichlet energy $\mathcal{E}_D$ operates only on functions $f \in H^1$. Hence, its differential is a map $d\mathcal{E}_D : H^1 \to (H^1)^*$, which we saw explicitly in Section 4.1: given a function $f \in H^1$, $d\mathcal{E}_D|_f$ produces a linear map $\langle\langle -\Delta f, \cdot \rangle\rangle$ from functions in H^1 to real numbers, *i.e.*, an element of $(H^1)^*$.

In the case of the tangent point energy, then, we get that $d\mathcal{E}^\alpha_\beta$ is a map from $W^{s,p}$ to the dual space $(W^{s,p})^* = W^{-s,q}$ (Section A.1.1). Hence, $d\mathcal{E}^\alpha_\beta$ is a "differential operator" of order $2s$, *i.e.*, it reduces the differentiability of its argument by $2s$. To get a well-behaved flow, we should therefore pick an inner product of the same order, and (for computational purposes) is reasonably easy to invert.

A.4 Fractional Inner Product

Just as one uses the Laplace operator Δ to define integer Sobolev inner products, we use a fractional operator to define a fractional Sobolev inner product. For an embedded curve $\gamma : M \to \mathbb{R}^3$, one idea is to start with the 1D fractional Laplacian $(-\Delta)^\sigma$. Alternatively, we can define an analogous operator by replacing the intrinsic distance $|x - y|$ on the right-hand side of Equation 24 with the extrinsic distance $|\gamma(x) - \gamma(y)|$ between points in the embedding. This latter construction yields an operator L_σ defined by the relationship

$$\langle\langle L_\sigma u, v \rangle\rangle := \iint_{M^2} \frac{u(x)-u(y)}{|\gamma(x)-\gamma(y)|^\sigma} \frac{v(x)-v(y)}{|\gamma(x)-\gamma(y)|^\sigma} \frac{dx_\gamma\, dy_\gamma}{|\gamma(x)-\gamma(y)|} \quad (25)$$

for all sufficiently regular $u, v : M \to \mathbb{R}$. For any $\sigma \in (0,1)$, both $(-\Delta)^\sigma$ and L_σ are fractional operators of order 2σ. But the benefit of L_σ is that it requires only Euclidean distances—which for embedded curves are easier to evaluate than geodesic distances. Moreover, building a fractional Laplacian via an explicit Fourier transform is prohibitively expensive, requiring a full eigendecomposition of a discrete Laplace matrix. In contrast, integral expressions like Equations 24 and 25 can easily be evaluated *à la* Section 5.2.3, and accelerated using hierarchical techniques *à la* Section 6.

A.4.1 High-Order Term. To get an inner product of the same order as $d\mathcal{E}^\alpha_\beta$, we compose the operator L_σ with further (integer) derivatives $\mathcal{D}$. In particular, Lemma A.1 implies that $s = 1 + \sigma$ for $\sigma \in (0,1)$. Hence, to define an operator B_σ of order $2s = 2\sigma + 2$, we apply two additional derivatives to L_σ, *i.e.*, we say that

$$\langle\langle B_\sigma u, v \rangle\rangle := \langle\langle L_\sigma \mathcal{D}u, \mathcal{D}v \rangle\rangle$$

for all sufficiently regular $u, v : M \to \mathbb{R}$. This relationship provides the definition of B_σ in Equation 12.

A.4.2 Low-Order Term. As discussed in Section 4.2.2, the operator B_σ is not invertible. We hence add the low-order term B^0_σ from Equation 13. Since B_σ and B^0_σ exhibit the same scaling behavior under a rescaling of γ, the behavior of the resulting gradient flow will not depend on the global scale. To see why, consider a rescaling of the curve $\gamma \mapsto c\gamma$ by a factor $c > 0$. Then $\mathcal{D}$ scales by a factor $1/c$, the term $1/|\gamma(x) - \gamma(y)|^{2s+1}$ scales by $1/c^{2s+1}$, and the measure $dx_\gamma\, dy_\gamma$ scales by c^2. Then B_σ scales by $c^2/(c^2 c^{2s+1}) = 1/c^{2s+1}$, and L_σ scales by just c^2/c^{2s+1}. Hence, to get B^0_σ we multiply L_σ by k^2_4, which scales like $1/c^2$ (since it has c^2 in the numerator, and c^4 in the denominator). More generally, one could use k^α_β for any α, β such that $\alpha - \beta = -2$. This low-order term also tends to accelerate the evolution of the flow by preserving near-constant motions that slide near-tangentially and do not tend toward collision (Figure 7).

B ACCELERATION SCHEME

B.1 Energy and Differential Evaluation

B.1.1 Bounding Volume Hierarchy. To build the BVH we first construct tangent-points $p_I := (T_I, x_I) \in \mathbb{R}^6$ for each edge $I \in E$. We then cycle through all six coordinates, choosing a splitting plane that minimizes the sum of squared diameters of the two child bounding boxes. Below a user-specified threshold, all remaining tangent-points are placed in a single leaf node. In each node $\mathcal{N}$ we also store data needed for Barnes-Hut. Specifically,

$$L_\mathcal{N} := \sum_{I \in \mathcal{N}} \ell_I, \qquad \overline{x}_\mathcal{N} := \sum_{I \in \mathcal{N}} \ell_I x_I / L_\mathcal{N}, \qquad \overline{T}_\mathcal{N} := \sum_{I \in \mathcal{N}} \ell_I T_I / L_\mathcal{N},$$

give the total mass, center of mass, and (length-weighted) average tangent, *resp.*; we will use $\overline{p}_\mathcal{N} := (\overline{T}_\mathcal{N}, \overline{x}_\mathcal{N})$ to denote the corresponding tangent-point. We also store the bounding box radii $r^\mathcal{N}_x$ and $r^\mathcal{N}_T$ with respect to spatial and tangential coordinates, *resp.*

B.1.2 Barnes-Hut Approximation. To evaluate the energy for a tangent-point $p_I = (T_I, x_I) \in \mathbb{R}^6$ with mass $\ell_I \in \mathbb{R}$, we traverse the BVH from the root, checking at each node if a local approximation is *admissable* (see below). If so, we evaluate the approximation

$$(\mathcal{E}^\alpha_\beta)_{I\mathcal{B}} := \frac{|T_I \times (x_I - \overline{x}_\mathcal{B})|^\alpha}{|x_I - \overline{x}_\mathcal{B}|^\beta} \ell_I L_\mathcal{B}. \tag{26}$$

and terminate traversal; otherwise, we sum the energy of the two children. If we reach a leaf node $\mathcal{B}$, we directly add up the contributions of the edges contained in this node, *i.e.*,

$$\sum_{J \in \mathcal{B}} \frac{|T_I \times (x_I - x_J)|^\alpha}{|x_I - x_J|^\beta} \ell_I \ell_J.$$

Admissibility. A simple Taylor series analysis of Equation 26 indicates that to keep approximation error below a user-specified threshold $\varepsilon > 0$, it is sufficient to ensure that

$$r^\mathcal{B}_x / |x_I - \overline{x}_\mathcal{B}| \lesssim \varepsilon \qquad \text{and} \qquad r^\mathcal{B}_T \lesssim \varepsilon. \tag{27}$$

Intuitively, if $\mathcal{B}$ is far from the query point p_I relative to its size, *and* contains tangents that are close together, then the "lumped" energy is a good approximation of the total energy between edge I and the edges in $\mathcal{B}$.

Differential. Rather than differentiate our Barnes-Hut approximation of $\hat{\mathcal{E}}^\alpha_\beta$, we approximate the differential of the (full) discrete energy directly. Starting with the zero vector $d\hat{\mathcal{E}}^\alpha_\beta = 0 \in \mathbb{R}^{3|V|}$, we perform a BVH traversal for the tangent point p_I associated with each edge $I \in E$. At each admissible node $\mathcal{B}$ and for each endpoint i_a, $a = 1, 2$ of I we increment the differential via

$$(d\hat{\mathcal{E}}^\alpha_\beta)_{i_a} \mathrel{+}= L_\mathcal{B} \frac{\partial}{\partial \gamma_{i_a}} \left(\ell_I (\hat{k}^\alpha_\beta(x_I, \overline{x}_\mathcal{B}, T_I) + \hat{k}^\alpha_\beta(\overline{x}_\mathcal{B}, x_I, \overline{T}_\mathcal{B})) \right).$$

Here, $\hat{k}^\alpha_\beta$ is the discrete kernel defined in Equation 18; note that $L_\mathcal{B}$, $\overline{x}_\mathcal{B}$, and $\overline{T}_\mathcal{B}$ do not depend on γ_{i_1} or γ_{i_2}, since I is not contained in any admissible node $\mathcal{B}$. At any leaf node $\mathcal{B}$ we add the corresponding derivatives for all edges $J \in \mathcal{B}$.

B.2 Hierarchical Matrix-Vector Product

B.2.1 Block Cluster Tree (BCT). A BCT partitions a matrix into low-rank *blocks* that approximate the original entries (Figure 12). It is like a quadtree, except that the matrix ordering is not fixed *a priori*. The basic idea is that the edges in a BVH node $\mathcal{N}$ correspond to a subset of BCT rows/columns. A block of the BCT is hence specified by a pair of nodes $(\mathcal{A}, \mathcal{B})$ from the BVH. To construct a BCT, we recursively split the root block $(\mathcal{R}, \mathcal{R})$, where $\mathcal{R}$ is the root of the BVH. A block $(\mathcal{A}, \mathcal{B})$ is a leaf if and only if (i) it is *well-separated*, *i.e.*, it provides a good approximation of the local double sum, or (ii) $\mathcal{A}$ or $\mathcal{B}$ contains just a few edges. Otherwise, this block is given four children $(\mathcal{A}_1, \mathcal{B}_1)$, $(\mathcal{A}_1, \mathcal{B}_2)$, $(\mathcal{A}_2, \mathcal{B}_1)$, $(\mathcal{A}_2, \mathcal{B}_2)$, where $\mathcal{A}_1, \mathcal{A}_2$ are the children of $\mathcal{A}$ in the BVH (and likewise for $\mathcal{B}$). The conditions for being well-separated are similar to Equation 27:

$$\frac{\max(r^\mathcal{A}_x, r^\mathcal{B}_x)}{|x_\mathcal{A} - x_\mathcal{B}|} \lesssim \varepsilon \qquad \text{and} \qquad \max(r^\mathcal{A}_T, r^\mathcal{B}_T) \lesssim \varepsilon, \tag{28}$$

where $r^\mathcal{N}_x$ and $r^\mathcal{N}_T$ are the spatial and tangential radii of node $\mathcal{N}$.

B.2.2 Matrix-Vector Product. The BCT is used to accelerate a matrix-vector product $\varphi = \mathsf{K}\psi$ via the *fast multipole method*. We adopt the lowest (0th) order version of this method, which is accurate enough for preconditioning. In particular, for any admissible leaf node $(\mathcal{A}, \mathcal{B})$, the midpoints and tangents of edges in $\mathcal{A}$ and $\mathcal{B}$ are quite coherent relative to the distance between them. Since the kernel k is regular, the restriction of K to rows $I \in \mathcal{A}$ and columns $J \in \mathcal{B}$ is hence well-approximated by

$$\widehat{\mathsf{K}}_{\mathcal{A}\mathcal{B}} := \ell[\mathcal{A}]\, k(\overline{p}_\mathcal{A}, \overline{p}_\mathcal{B})\, \ell[\mathcal{B}]^\mathsf{T},$$

where $\ell[\mathcal{N}] \in \mathbb{R}^{|\mathcal{N}|}$ is the vector of edge lengths in $\mathcal{N}$. Using this rank-1 approximation, matrix-vector multiplication amounts to a single dot product (with $\ell[\mathcal{B}]$), followed by a scalar-vector product.

To perform a multiplication, we start with the zero vector $\varphi = 0 \in \mathbb{R}^{|E|}$ and iterate over all BCT leaves. For each admissible leaf $(\mathcal{A}, \mathcal{B})$ (*i.e.*, one which satisfies Equation 28) we perform an update

$$\varphi[\mathcal{A}] \leftarrow \varphi[\mathcal{A}] + \widehat{\mathsf{K}}_{\mathcal{A}\mathcal{B}}\, \psi[\mathcal{B}].$$

For inadmissible leaves, we simply sum over all edge pairs:

$$\varphi_I \leftarrow \varphi_I + \sum_{J \in \mathcal{B}} \mathsf{K}_{IJ}\, \psi_J$$

for all $I \in \mathcal{A}$. To accelerate evaluation, we percolate these sums up and down the BVH, following a standard fast multipole strategy.

B.3 Multigrid Solver

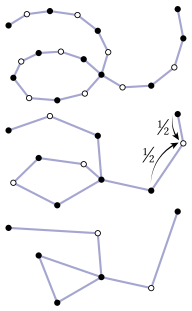
We first sketch out a generic multigrid strategy for saddle-point problems on a curve network; the specific solves needed for the tangent-point energy are detailed in Appendix B.3.4.

B.3.1 Geometric Multigrid. Suppose we want to solve a linear equation $Ax = b$. The basic idea of geometric multigrid is to use a coarser mesh to reduce the residual of an equation on the finer mesh. Consider a simple two-level hierarchy—in particular, let $\mathsf{A}_0 \in \mathbb{R}^{|V_0|\times|V_0|}$ and $\mathsf{A}_1 \in \mathbb{R}^{|V_1|\times|V_1|}$ be discretizations of A on a fine and coarse mesh, *resp.*, and let b_0 be a discretization of the function b onto the finest mesh. Also let $\mathsf{J}_1 \in \mathbb{R}^{|V_0|\times|V_1|}$ be a so-called *prolongation operator*, which interpolates data from the coarse mesh onto the fine mesh. Starting with any initial guess $\mathsf{x}_0 \in \mathbb{R}^{|V_0|}$, we first apply a *smoothing procedure* S to the system $\mathsf{A}_0\mathsf{x}_0 = \mathsf{b}_0$, *i.e.*, a fixed number of iterations of any iterative linear solver to get an improved guess $\tilde{\mathsf{x}}_0 \leftarrow S(\mathsf{A}_0, \mathsf{x}_0, \mathsf{b}_0)$. We then compute the residual $\mathsf{r}_0 \leftarrow \mathsf{A}_0\tilde{\mathsf{x}}_0 - \mathsf{b}_0$, and transfer it to the coarse mesh via $\mathsf{b}_1 \leftarrow \mathsf{J}_1^\mathsf{T}\mathsf{r}_0$. On the coarse mesh we solve the system $\mathsf{A}_1\mathsf{x}_1 = \mathsf{b}_1$ directly, and transfer the result back to the fine mesh via $\mathsf{y}_0 \leftarrow \mathsf{J}_1\mathsf{x}_1$. These values are used to update our guess via $\tilde{\mathsf{x}}_0 \leftarrow \tilde{\mathsf{x}}_0 + \mathsf{y}_0$, and smoothed again. If the residual is small enough, we stop; otherwise, we repeat another such *V-cycle* until convergence. More generally, one can apply this two-level strategy to solve the linear system on the coarser level, yielding a multi-level strategy. The size of the coarsest level is chosen so that a direct solve at this level is more efficient than continuing to apply multigrid.

Initialization. We get an initial guess x_0 by first coarsening the fine right-hand side b_0 down to the coarsest mesh. We then perform a direct solve and prolong the solution all the way to the finest mesh, applying smoothing after each refinement. In practice this strategy works much better than starting with the zero vector.

Implementation Details. In practice we use a standard conjugate gradient smoother, and typically need 6 or fewer V-cycles to achieve a relative residual of order 10^{-3}. Making the residual smaller via further cycles (and a more accurate BCT) yields diminishing returns: we need only a reasonable intermediate descent direction. Note that although we build a BCT at each level, overall construction cost is only about twice the cost at the finest level.

B.3.2 Curve Coarsening and Prolongation. To build a multigrid hierarchy on a curve network, we apply a simple coarsening scheme. We mark alternating vertices as "black" and "white", and mark all endpoints and junctures where two or more curves meet as black. The next coarsest curve is obtained by removing white vertices; we stop when we reach a target size or when there are no more white nodes. The prolongation operator J preserves values at black vertices, and at white vertices takes the average of the two neighboring black vertices. In our experience, using linear interpolation based on edge lengths did not improve multigrid performance. Coarsening need not preserve the isotopy class of the curve network to provide useful preconditioning for the next level of the hierarchy.



B.3.3 Multigrid for Saddle Point Problems. Our constraint scheme entails solving *saddle point problems* of the form

$$\begin{bmatrix} \overline{\mathsf{A}} & \mathsf{C}^\mathsf{T} \\ \mathsf{C} & 0 \end{bmatrix} \begin{bmatrix} \mathsf{x} \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathsf{a} \\ 0 \end{bmatrix}, \tag{29}$$

where $\overline{\mathsf{A}}$ is the inner product (for vector-valued functions) (see Equation 19), and C is the constraint matrix (Section 5.3.1); the data $\mathsf{a} \in \mathbb{R}^{3|V|}$ depends on the problem being solved. We follow the approach of Braess and Sarazin [1997], who note that for the structurally identical *Stokes' problem* (where $\overline{\mathsf{A}}$ and C are replaced by the Laplace and divergence operators, *resp.*), applying multigrid to the whole matrix does not work well. Instead, let $\mathsf{P} \in \mathbb{R}^{3|V|\times 3|V|}$ be a projection onto the null space of C, *i.e.*, $\mathsf{CP} = 0$ and $\mathsf{P}^2 = \mathsf{P}$. Then by construction, any solution y to the equation

$$\mathsf{P}^\mathsf{T}\overline{\mathsf{A}}\mathsf{P}\mathsf{y} = \mathsf{P}^\mathsf{T}\mathsf{a} \tag{30}$$

yields a vector $\mathsf{x} = \mathsf{P}\mathsf{y}$ within the constraint space $\mathsf{C}\mathsf{x} = 0$ that satisfies our original equation. Equation 30 is therefore the system that we actually solve via multigrid. In particular, we use the projection $P := \mathsf{C}\mathsf{C}^\dagger$, where † denotes the (Moore-Penrose) pseudoinverse

$$\mathsf{C}^\dagger := (\mathsf{C}\mathsf{C}^\mathsf{T})^{-1}\mathsf{C}^\mathsf{T}.$$

Since our constraints are typically sparse, we can factorize the inner term $\mathsf{C}\mathsf{C}^\mathsf{T}$ (once per time step) to further accelerate computation. Note that one must build a constraint matrix C_i and projection matrix P_i at each level i of the multigrid hierarchy.

B.3.4 Gradient Solve and Constraint Projection. With these pieces in place, we can apply multigrid to compute the constrained gradient (Equation 22), and perform constraint projection (Equation 23).

Gradient. To compute the gradient, recall that $\mathsf{A} = \mathsf{B}^0 + \mathsf{B}$. A matrix-vector product $\mathsf{B}^0\mathsf{u}$ can be expressed as

$$\mathsf{B}^0\mathsf{u} = \mathsf{E}^\mathsf{T}(\mathrm{diag}(\mathsf{K}\mathbf{1}) - \mathsf{K})\mathsf{E}\mathsf{u} \tag{31}$$

where diag(v) is a diagonal matrix with entries v, $\mathsf{E} \in \mathbb{R}^{|E|\times|V|}$ averages values from vertices to edges (*i.e.*, $(\mathsf{E}\mathsf{u})_I = \frac{1}{2}(u_{i_1} + u_{i_2})$), and

$$\mathsf{K}_{IJ} = (k^2_{2\sigma+5}(x_I, x_J, T_I) + k^2_{2\sigma+5}(x_J, x_I, T_J))\ell_I\ell_J. \tag{32}$$

We use the method from Appendix B.2 to efficiently perform the products $\mathsf{K}\mathbf{1}$ and $\mathsf{B}^0\mathsf{u}$, and ordinary sparse matrix multiplication for E. The high-order part B is expressed exactly as in Equation 31, except that (i) we replace the averaging operator E with the difference operator D, (ii) we define a different kernel matrix K by replacing $k^2_{2\sigma+5}$ with $k^0_{2\sigma+1}$ in Equation 32, and (iii) just like A, K acts blockwise on the three components of vector-valued data $\mathsf{x} \in \mathbb{R}^{3|E|}$ (*à la* Equation 19).

Constraint Projection. To use our multigrid solver for constraint projection, we apply a simple transformation to Equation 23 that gives it the same form as Equation 29. In particular, we solve

$$\begin{bmatrix} \overline{\mathsf{A}} & \mathsf{C}^\mathsf{T} \\ \mathsf{C} & 0 \end{bmatrix} \begin{bmatrix} \mathsf{y} \\ \mu \end{bmatrix} = \begin{bmatrix} \overline{\mathsf{A}}\mathsf{z} \\ 0 \end{bmatrix},$$

where $\mathsf{z} := \mathsf{C}^\dagger\mathsf{b}$, and b is the lower block of the right-hand side of Equation 23. The final result is then given by

$$\mathsf{x} = \mathsf{z} - \mathsf{y}. \tag{33}$$