Inverse Diffusion Curves using Shape Optimization

Shuang Zhao, Frédo Durand, and Changxi Zheng

Abstract—The inverse diffusion curve problem focuses on automatic creation of diffusion curve images that resemble user provided color fields. This problem is challenging since the 1D curves have a nonlinear and global impact on resulting color fields via a partial differential equation (PDE). We introduce a new approach complementary to previous methods by optimizing curve geometry. In particular, we propose a novel iterative algorithm based on the theory of shape derivatives. The resulting diffusion curves are clean and well-shaped, and the final image closely approximates the input. Our method provides a user-controlled parameter to regularize curve complexity, and generalizes to handle input color fields represented in a variety of formats.

Index Terms—Vector graphics, diffusion curves, inverse problem, shape optimization, Fréchet derivative

1 Introduction

Vector graphic images remain invaluable for a broad range of 2D applications because of their resolution independence, compactness of representation, and powerful editability. Diffusion curve images [1] further improve the expressiveness of vector graphics, providing flexible and easy-to-manipulate smooth gradients, and since then inspired a variety of novel applications [2], [3], [4], [5], [6]. Defined along the curves, colors are diffused across the image by a Poisson or Laplace reconstruction, and their smoothness can be further controlled by the curve's blurriness through post-processing. Although efficient rendering of diffusion curve images has been well explored, its inverse problem of creating diffusion curves automatically given desired target images remains challenging.

The *inverse diffusion curve problem* is difficult because even though the curves themselves are 1D, their impact on the final image is nonlinear and global over a 2D domain through a partial differential equation (PDE), Laplace's equation. The geometry of curves largely determines the reconstruction quality. Previous methods have used *local* heuristics to obtain curve geometry. They place curves at locations indicated by edge detectors applied to the target image [1] and its Laplacian or bi-Laplacian [7]. While these heuristics work well around sharp edges such as object boundaries, they have difficulty handling color variations in regions that are smooth yet visually rich.

We introduce a new approach to the inverse diffusion curve problem. Complementary to existing methods, our approach solves for curve geometry through a *global optimization* that takes into account the curves' full impact on the PDE-based color field. To achieve this, we characterize how modifications to a diffusion curve can reduce a global cost function determined by the solution of a PDE with the curves acting as boundary conditions.

- *S. Zhao is with the Department of Computer Science, University of California, Irvine. E-mail: shz@ics.uci.edu*
- *F. Durand is with the CSAIL, MIT. E-mail: fredo@mit.edu*
- *C. Zheng is with the Department of Computer Science, Columbia University. E-mail: cxz@cs.columbia.edu*

Our method is grounded on the theory of *shape optimization* [8]. Given a color field, it computes curve geometry by minimizing a measure of the color reconstruction residual. Starting from an initial set of curves, it iteratively evolves their shapes toward an optimal configuration. Mathematically, the curves are treated as continuous functionals. This allows them to deform arbitrarily, enabling full exploration of possible curve configurations. This iterative process is similar to a surface normal flow: our curve evolution at every iteration is guided by the Fréchet derivative of the residual function with respect to the curve's boundary velocity, leading to an efficient "gradient descent" of the residual. This method is mathematically clean and easy to implement: all computations at each iteration boil down to solving a Laplace equation and a Poisson equation.

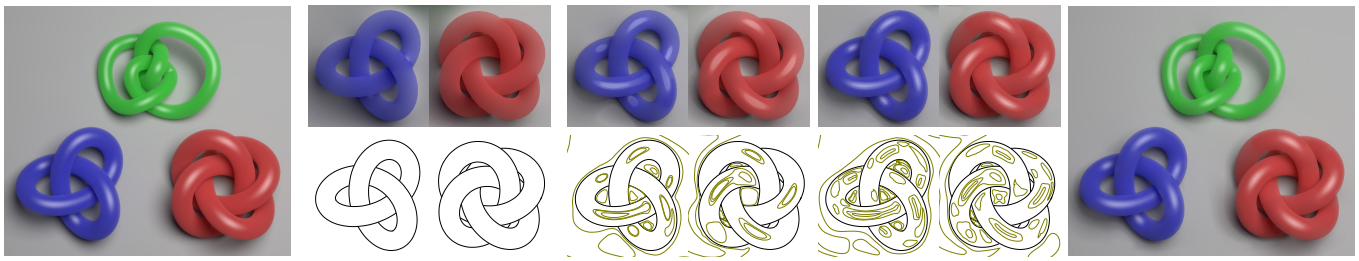
Building on our curve placement algorithm, we introduce a complete pipeline for solving the inverse diffusion curve problem. Our method generates curves in a clean and concise way, and the resulting images can accurately capture complex color variations of input color fields (see Figures 1, 14, 15, and 16 as well as the supplementary images).

We demonstrate that our method promises practical applications beyond pixel image vectorization. It enables automatic rendering of vector graphic images from 3D geometries, analogous to the traditional pixel image rendering. Further, using our algorithm, one can directly transform other formats of vector graphics, such as gradient meshes, into diffusion curve images without rasterizing the input.

2 Related Work

Diffusion curves [1] represent a color field by diffusing the colors defined along control curves over the entire image plane. The diffusion process is described by Laplace's equation solved using a finite volume method. Later, solving Laplace's equation was improved using a multigrid method [2], triangle mesh interpolation [9], Boundary Element method [4], 2D ray tracing [10], and Fast Multipole method [6].

Inverse diffusion curve problem: Our work focuses on the *inverse problem* of diffusion curves. Previously, Orzan et al. [1] proposed to place diffusion curves along edges



(a) Reference (b) Before optimization (c) Early opt. stage (d) Late opt. stage (e) Our final image

Fig. 1: We present a new approach to automatically build diffusion curve images approximating provided color fields (a). Starting from a set of boundary curves (black strokes) indicating color jump discontinuities (b), our method iteratively adds curves (dark yellow strokes) and refines their shapes in an optimized manner (c, d). The resulting image (e) accurately matches the input.

extracted from input images using the Canny detector [11]. Jeschke et al. [12] introduced a technique to improve curve colorings. Xie et al. [7] further improved this method by detecting edges in a Laplacian (and/or bi-Laplacian) domain and constructing curves hierarchically. They solve the Laplacian and bi-Laplacian weights using least-squares fitting. In all methods, diffusion curves are placed along the detected edges, and never moved or added in continuous color regions. These methods then rely on optimizing curve coloring for better accuracy.

We introduce a fundamentally complementary solution to the inverse diffusion curve problem. Instead of predetermining curve geometry and optimizing their coloring, we propose doing the opposite by first optimizing the geometry and then determining the coloring accordingly. We demonstrate that with a very simple coloring scheme, our method outperforms prior methods under many situations (§6.2). Furthermore, our approach accepts input color fields beyond pixel images.

Extensions of diffusion curves: Several methods have been proposed to extend the expressiveness of diffusion curves. Sun et al. [6] enabled fast diffusion curve cloning and multi-layer composition. Finch et al. [13] introduced a higher-order notion of smoothness: the colors are defined using a 4th-order linear elliptic PDE rather than a Laplace equation. To accelerate the color evaluation, Boyé et al. [14] developed a vectorial solver using the Finite Element Method, and Sun et al. [4] proposed a boundary element based formula, which was later improved in [5] to handle both Laplacian and bi-Laplacian curves in a unified framework. Higher-order curves offer greater flexibility than the standard diffusion curves, but their inverse problems are more difficult and remain unsolved. In this paper, we focus on the inverse problem for *original diffusion curves* and discuss potential extension to higher-order domains in §6.3.

Theory and applications of shape optimization: We build our curve optimization on the theoretical foundation of shape optimization [8], [15], a subfield of optimal control theory. Mathematically, it solves the problem of finding a bounded set Ω to minimize a continuous functional on Ω. The core idea of shape optimization has been used for image segmentation since the seminal work of [16], [17]. It is also related to surface gradient flow widely studied in geometry processing [18], [19]. In areas outside of computer graphics, shape optimization has been used to enhance mechanical structures such as airfoils [20] and photonic crystals [21]. It has also been used in computer vision for image segmentation (e.g., [22], [23]). To our knowledge, shape optimization has not yet been applied in vector graphics. In this paper, we solve a shape optimization problem with a PDE constraint (§5.1), which is significantly more challenging than a conventional shape optimization problem.

3 Background

We now briefly revisit the mathematical formulation of diffusion curve images and present the main focus of this work: the inverse diffusion curve problem.

Diffusion curve images: The color field u in a diffusion curve image [1], [2] is a harmonic function satisfying a Laplace equation with a Dirichlet boundary condition:

$$\begin{aligned} u(\boldsymbol{x}) &= \{C_\ell(\boldsymbol{x}), C_r(\boldsymbol{x})\}, & \boldsymbol{x} &\in \mathbb{B} \\ \Delta u(\boldsymbol{x}) &= 0, & &\text{otherwise}, \end{aligned} \tag{1}$$

where the boundary $\mathbb{B}$ consists of the entire set of diffusion curves; C_ℓ and C_r specify the colors on the *left* and *right* side of each curve, respectively. Typically, both the shapes of the curves and their left- and right-side colors are specified by the user, and the entire color field is uniquely determined by solving the Laplace equation (1).

Since its invention, diffusion curves have been augmented. Orzan et al. [1] proposed to apply per-pixel blurring to the rasterized image of u, the solution of (1). Finch et al. [13] further extended to diffuse colors using higher-order elliptic PDEs such as the biharmonic equations.

Inverse diffusion curve problem: While plenty of extensions of the *forward* diffusion curve problem have been proposed, largely under-explored is the *inverse problem*, one that computes a set of diffusion curves such that the resulting vector image closely resembles a user-provided 2D color field. In this paper, we address this inverse problem which in turn involves two subproblems:

- **Curve geometry:** To build a diffusion curve image, one needs to decide where to place the curves (namely, to determine $\mathbb{B}$).
- **Curve coloring:** Given the curve geometry, the colors on both sides of each curve (namely C_ℓ and C_r) need to be specified.

As discussed in §2, recent work [7], [12] has largely focused on optimizing curve coloring with their geometry predetermined (using edge detection). In contrast, we focus on the complementary problem of directly optimizing curve geometry. We demonstrate in §6 that curves with optimized

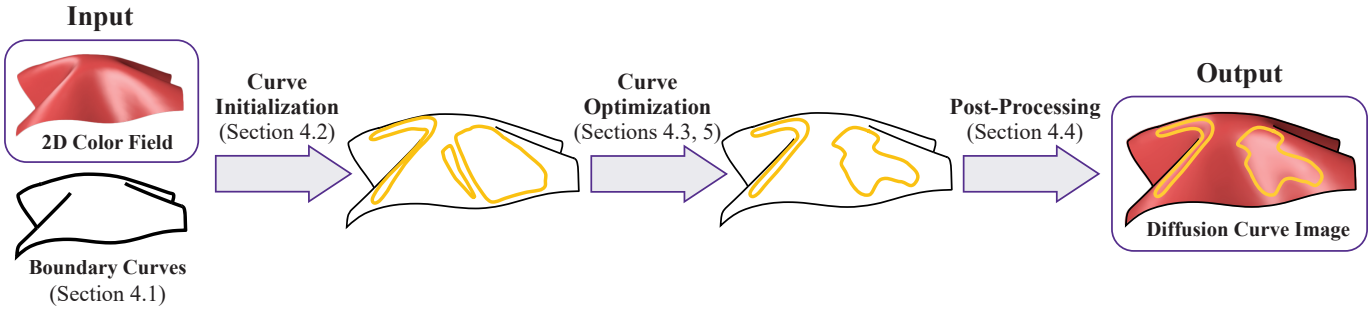


Fig. 2: **Our pipeline.** The input to our method is a 2D color field. After obtaining a set of boundary curves indicating color jump discontinuities (§4.1), our method constructs a set of initial curves (§4.2) and optimizes their shapes and trajectories (§4.3 and §5). Finally, we post-process the optimized curves and obtain the resulting diffusion curve image (§4.4).

geometries generally yield higher-quality of reconstructions, regardless of the curve coloring schemes.

4 Our Pipeline

We develop a complete pipeline, as outlined in Figure 2 and Algorithm 1, for automatic creation of diffusion curve images. Our method takes as input a color field I allowing to query for color values at for all $\boldsymbol{x}$ in the image domain Ω. Then, starting with extracting a set of boundary curves (§4.1) that indicate jump discontinuities in I, our method generates a set of curves as "initial guesses" (§4.2) which are then deformed by our core curve optimization algorithm (§5) to minimize reconstruction error (§4.3). Lastly, we post-process the deformed curves (§4.4) to generate final diffusion curve images.

As one of our main contributions, the key component of our pipeline (lines 7, 11, and 13 of Algorithm 1) is a *curve optimization algorithm* that deforms diffusion curves to minimize the reconstruction error. Detailed discussions and mathematical derivations on this algorithm are in §5. In the rest of this section, we provide more details for the remaining steps of Algorithm 1.

4.1 Boundary Curves

Provided an input color field I, we start the pipeline by obtaining a set of *boundary curves* $\partial\Omega$ indicating the outer boundary and jump discontinuities of I.[1] See Figure 3-ab for an example. In practice, we obtain the boundary curves $\partial\Omega$ depending on specific representation of the input color field I:

- **Pixel images.** A common way to represent color fields is using standard pixel images. The boundary curves, however, are not uniquely defined in this case. To obtain these curves in practice, we use Canny edge detection similar to Orzan et al.'s work [1].
- **3D renderings.** If the color field is defined by the rendering of a 3D scene, the boundary curves can be obtained by extracting object contours.
- **Other vector formats.** For input color fields represented in other vector formats (e.g., gradient mesh), $\partial\Omega$ can be determined directly based on the underlying vector representation (e.g., triangle edges).

Please refer to §6.2 for more details and experimental results on boundary curve computation.

1. The necessity of boundary curves is explained in §5.

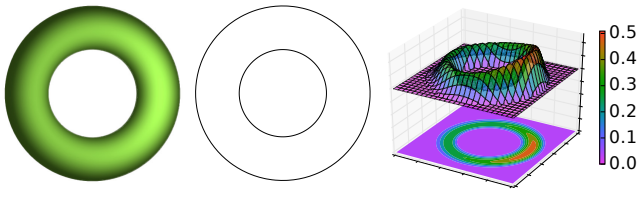


Fig. 3: **A sample color field:** (a) the color field representing a smoothly shaded torus viewed from the top; (b) the corresponding boundary curves; (c) a visualization of the color field.

Algorithm 1 Diffusion curve placement

Require: Color field I (defined on Ω)
1: **procedure** CurvePlacement(I, Ω, ϵ_0)
2: compute boundary curves $\partial\Omega$ ▷ §4.1
3: partition Ω into connected components
4: $\mathbb{B} \leftarrow \partial\Omega$
5: **for** each component $\mathcal{C}$ **do**
6: $\mathbb{D}_0 \leftarrow$ CurveInit('global', $\partial\mathcal{C}$, I, $\mathcal{C}$) ▷ Alg. 2
7: $\mathbb{D} \leftarrow$ CurveOpt($\mathbb{D}_0$, $\partial\mathcal{C}$, I, $\mathcal{C}$) ▷ Alg. 3
8: **while** $R(\mathcal{C}; \partial\mathcal{C} \cup \mathbb{D}) > \epsilon_0$ **do**
9: $\mathbb{D}'_0 \leftarrow$ CurveInit('local', $\partial\mathcal{C} \cup \mathbb{D}$, I, $\mathcal{C}$)
10: $\mathbb{D}' \leftarrow$ CurveOpt($\mathbb{D}'_0$, $\partial\mathcal{C} \cup \mathbb{D}$, I, $\mathcal{C}$)
11: $\mathbb{D} \leftarrow \mathbb{D} \cup \mathbb{D}'$
12: **end while**
13: $\mathbb{D} \leftarrow$ CurveOpt($\mathbb{D}$, $\partial\mathcal{C}$, I, $\mathcal{C}$)
14: post-process $\mathbb{D}$ ▷ §4.4
15: $\mathbb{B} \leftarrow \mathbb{B} \cup \mathbb{D}$
16: **end for**
17: **return** $\mathbb{B}$
18: **end procedure**

4.2 Curve Initialization

Desired properties: Similar to gradient descent methods, our curve optimization algorithm takes an initial guess to start with. For ensuring high-quality optimization results, there are a few properties required of the initial curves:

1) **Easy to compute:** The curve initialization step should not require intense computation: we rely on the optimization step to refine the shapes of these curves.
2) **Good coverage:** The initial curves should provide a good coverage to the full image domain Ω, so that the

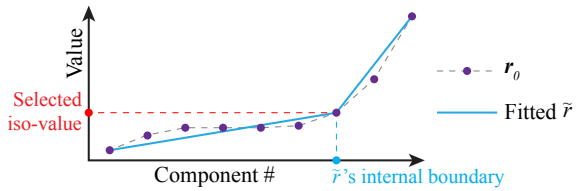


Fig. 4: **Fitting r_0** with 9 components (indicated with purple dots) using a piecewise linear function $\tilde{r}$ with 2 pieces ($m = 1$). The value of $\tilde{r}$ at its internal boundary is selected as iso-value.

optimization is less prone to local optima.

3) **Being well-shaped:** The initial curves need to be well-shaped. For example, they should have low complexities and not self-intersect or collide with the boundaries.

To achieve these properties, we use iso-contours of the residual field as the initial curves:

$$R_0(\partial\Omega; \boldsymbol{x}) = (u_0(\boldsymbol{x}) - I(\boldsymbol{x}))^2, \quad \forall \boldsymbol{x} \in \Omega. \tag{2}$$

In (2), u_0 is given by the diffusion curve image using only the input boundary curves $\partial\Omega$. These curves can be computed easily from a set of iso-values (Property 1). In addition, as long as the iso-values are distributed properly, the resulting iso-contours will provide a good coverage to the image domain while being well shaped (Properties 2 and 3). For example, to make the initial curves never intersecting with $\partial\Omega$, we can simply pick strictly positive iso-values as $R_0(\partial\Omega; \boldsymbol{x}) = 0$ for all $x \in \partial\Omega$.

Our approach: To choose a set of properly distributed iso-values, we start with sampling a set of points in Ω and stacking the residual values (2) at these points into a vector $\boldsymbol{r}_0$ in ascending order (lines 2 and 3 in Algorithm 2). The resulting vector $\boldsymbol{r}_0$ provides a picture on the distribution of residuals. We adopt two complementary schemes, *global* and *local*, to set iso-values using $\boldsymbol{r}_0$, and thereby obtain the iso-contours:

- **Global:** The global scheme constructs a relatively large set of initial curves over the entire domain Ω. Assume that the number of iso-values m is given. Ideally, we would like to find m values such that the consequent iso-contours optimally capture the structure of 2D residual field R_0. In practice, we solve this problem approximately and rely on our curve optimization algorithm to refine the curves. Particularly, we solve a well-studied 1D problem [24]: to fit a piecewise linear function $\tilde{r}$ with $m + 1$ pieces that closely describes $\boldsymbol{r}_0$ (interpreted as a polyline). Then, the values of $\tilde{r}$ at its m internal piece boundaries are used as iso-values (see Figure 4).
- **Local:** The local scheme, in contrast to the global one, adds curves locally in regions with high approximation error. In this case, we use only one iso-value determined based on the maximal sampled residual (line 8 of Algorithm 2).

In our curve placement algorithm (detailed in §4.3), we use the global scheme at the beginning to ensure that the initial curves provide a good coverage to the domain Ω (Property 2). Then, the local scheme is applied iteratively to add small sets of curves in high-residual areas. The combination of both schemes offers sufficient approximation accuracy without introducing unnecessarily complex curves (Property 3). We find that $m = 2$ works well in our experiments.

Algorithm 2 Diffusion curve initialization

Require: Color field I (defined on Ω) and boundary curves $\partial\Omega$

1: **procedure** CURVEINIT(scheme, $\partial\Omega$, I, Ω)
2: generate uniform point samples in Ω
3: form $\mathbf{r}_0$ by evaluating $R_0(\partial\Omega; \boldsymbol{x})$ on the sampled points
4: **if** scheme = 'global' **then** ▷ Global scheme
5: fit a piecewise function f to $\mathbf{r}_0$
6: let A to be the (internal) piece boundaries of f
7: **else** ▷ Local scheme
8: $A \leftarrow \{0.9 \max(\mathbf{r}_0)\}$
9: **end if**
10: **return** iso-contours with iso-values specified in A
11: **end procedure**

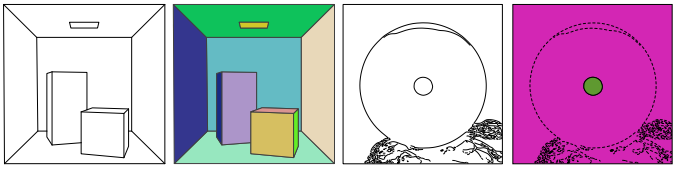


Fig. 5: Two examples of **boundary curves** and corresponding **partitioning** of domain Ω. For clean and well defined boundaries (a1), Ω can be divided into many well shaped components (a2); for messier boundaries often resulting from edge detections (b1), there are normally fewer components with more complex shapes (b2). Our approach works well for both cases.

4.3 Curve Placement

Given the initial curves generated by Algorithm 2, our curve optimization algorithm iteratively refines their trajectories to reduce reconstruction errors and finalize curve geometry. We postpone the details of this algorithm (Algorithm 3) and its derivations until §5 but present here the complete curve placement steps described in Algorithm 1.

The curve placement process is built on the algorithms of curve initialization and optimization schemes. It takes as input the target color field I defined on domain Ω, the previously obtained boundary curves $\partial\Omega$, and a tolerance ϵ_0 on reconstruction error. Based on $\partial\Omega$, we partition the domain Ω into a number of connected components and process them individually in parallel (line 3 of Algorithm 1). Figure 5 illustrates example boundaries and resulting partitionings. Notice that our approach allows boundary curves to exist inside individual components (e.g., Figure 5-b)—these curves will remain fixed throughout the entire pipeline.

For each connected component $\mathcal{C}$, our approach generates diffusion curves via several *passes*, each of which involves initializing a set of curves (Algorithm 2) and optimizing their shapes (Algorithm 3 in §5). In the first pass, we start with initial curves constructed using the *global* scheme (lines 6 and 7). After this pass, if the approximation error remains beyond a tolerance ϵ_0, additional passes are used in which new curves are initialized using the *local* scheme (lines 8 to 10). After the error drops below the threshold, we perform a final pass (line 13) in which all curves created in previous passes are optimized together. Finally, we post-process the resulting curves to remove redundant curve segments (line 14 and §4.4). An example of this curve

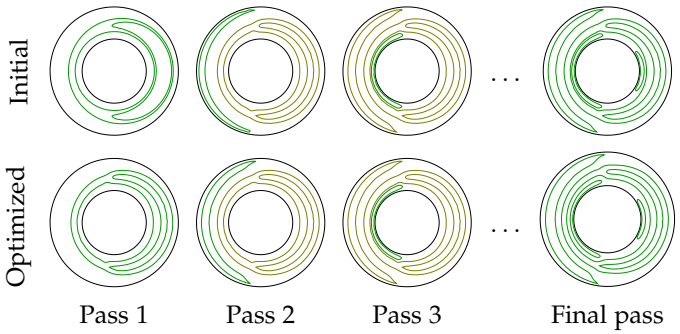


Fig. 6: An example of our **curve placement** process (Algorithm 1) using Figure 3-ab as input. New curves are constructed using the global scheme in Pass 1 and the local one in the following passes. The final pass (Pass 5) generates no new curves. Instead, it starts with those created in all previous passes. In each pass, active curves (those being added and/or optimized) and previously generated ones are drawn as green and dark yellow strokes, respectively.

placement process is illustrated in Figure 6.

4.4 Curve Post-Processing

Lastly, we post-process the curves $\mathbb{B}$ returned by Algorithm 1 and generate the final diffusion curve image.

Curve Coloring: Notice that Algorithm 1 returns optimized *curve geometry* instead of actual diffusion curves. Thus, to turn $\mathbb{B}$ into a set of diffusion curves, their *coloring*, namely colors on both sides of each curve, needs to be provided. This corresponds to specifying the values of C_ℓ and C_r in (1).

As aforementioned, this curve coloring step is completely orthogonal and complementary to our core technique (Algorithm 1). Thus, in the rest of this paper, we use a simple scheme which directly sample color values on both sides of each curve from the input color field I. That is, for any $\boldsymbol{x} \in \mathbb{B}$, we set

$$C_\ell(\boldsymbol{x}) = I(\boldsymbol{x} + \delta n_\ell) \quad \text{and} \quad C_r(\boldsymbol{x}) = I(\boldsymbol{x} + \delta n_r), \tag{3}$$

where n_ℓ and n_r respectively denote normal directions pointing left and right side of a point $\boldsymbol{x}$ on a curve (thus, $n_\ell = -n_r$) and δ is a small positive number that can be set to the size of one pixel when I is represented as a pixel image. Our experiments demonstrate that this simple scheme can yield high-quality results thanks to our optimized curve geometry (§6.2). In §6.3, we show that more advanced coloring techniques can further improve reconstruction accuracy.

Removing redundant curve segments: As mentioned in §5.4, we represent diffusion curves as polylines consisting of a number of line segments. Some of these segments, however, may be unnecessary. Note that the colors across a line segment are continuous because of the boundary condition (5) on $\mathbb{B}$. If the color gradient normal to a segment is also continuous across, then the segment as a boundary has no influence on the solution color field u. A mathematical explanation is in §3 of the supplementary document. Precisely, a normal gradient is continuous when

$$d_{\boldsymbol{n}}(\boldsymbol{x}) = \frac{\partial u(\boldsymbol{x})}{\partial n_\ell} + \frac{\partial u(\boldsymbol{x})}{\partial n_r}, \quad \boldsymbol{x} \in \mathbb{B}, \tag{4}$$

is zero. In practice, we solve u using the Finite Element Method (§5.4) and check if $|d_{\boldsymbol{n}}(\boldsymbol{x})|$ at the center point $\boldsymbol{x}$ of each segment is below a threshold. If so, we mark the segment as unnecessary. Lastly, for each curve output by Algorithm 1, we remove a largest set of connected redundant segments to avoid breaking the curve into many small disconnected components.

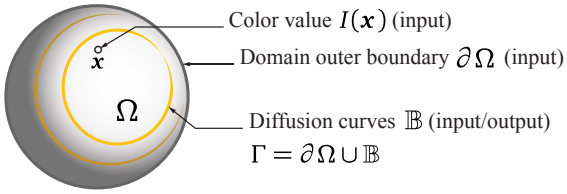


Fig. 7: **Input** and **output** of our curve optimization algorithm. **Input:** color field I, domain Ω and its outer boundary $\partial\Omega$, initial diffusion curves $\mathbb{B}$; **output:** refined curves $\mathbb{B}$.

To transform the final polyline into a standard diffusion curve made from end-to-end connected Bézier curves, we adopt the Potrace algorithm [25], which was also used in prior work [1].

Per-pixel blurring (optional): The curves placed in a smooth color region have continuous color values across the curves. However, since these curves serve as boundaries in the Laplace solve, color gradients may not necessarily remain continuous across curves generated by Algorithm 1. Such gradient discontinuities can sometimes lead to noticeable artifacts [13]. Thus, our pipeline includes an optional step following the original framework of diffusion curves [1] to perform per-pixel blurring on the rasterized image. The size of blur kernel at each pixel is determined by another Laplace equation:

$$\begin{aligned} K(\boldsymbol{x}) &= K_0(\boldsymbol{x}), && x \in \Gamma \\ \Delta K(\boldsymbol{x}) &= 0, && \text{otherwise,} \end{aligned}$$

where $K_0(\boldsymbol{x})$ gives the desired kernel size along the curves. In particular, we set $K_0(\boldsymbol{x}) = 0$ for all $\boldsymbol{x} \in \partial\Omega$ since the boundaries and discontinuities in the input color field should never be blurred. For $\boldsymbol{x} \in \mathbb{B}$, the value $d_{\boldsymbol{n}}(\boldsymbol{x})$ indicates the magnitude of the gradient domain discontinuity. Thus, we set $K_0(\boldsymbol{x}) = b\,|d_{\boldsymbol{n}}(\boldsymbol{x})|^a$ for all $\boldsymbol{x} \in \mathbb{B}$, where a and b are two global parameters. In our implementation, we set $a = 0.2$ and b to 5% of the longest axis of Ω's bounding box.

More advanced curve coloring techniques (e.g., [7]) may optimize color gradients across the curves, largely removing gradient discontinuity artifacts. In this case, per-pixel blurring is unnecessary (§6.3).

5 Diffusion Curve Optimization

We now detail the core of our pipeline, the optimization of diffusion curve geometries to approximate a given 2D color field. We first describe an algorithm minimizing the approximation error of diffusion curve images (§5.1, §5.2) and then extend it to balance accuracy against curve length (§5.3). Lastly, we provide implementation details (§5.4) followed by the discussions of further extensions (§5.5).

We introduce *Shape Optimization* [8] to formulate the inverse diffusion curve problem. While building our approach on existing shape optimization concepts and theories (§5.2), we also develop a new formula for regularizing curve length (§5.3). Please refer to the supplementary document for complete derivations and a review of related background.

Curve Optimization in a Nutshell: The major steps of our approach are outlined in Algorithm 3. Its input includes

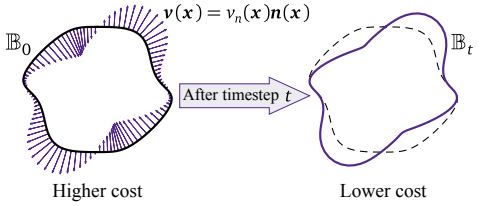


Fig. 8: **Curve optimization.** Given a set of curves $\mathbb{B}_0$, we construct a velocity field $\boldsymbol{v}$ so that if one deforms $\mathbb{B}_0$ according to $\boldsymbol{v}$, the resulting curves $\mathbb{B}_t$ provide a lower cost.

the color field I, a 2D closed domain Ω over which I is defined, and a set of initial curves $\mathbb{B}$ in Ω (Figure 7). In this section, the color field I is treated as a black box, allowing $I(\boldsymbol{x})$ and $\nabla I(\boldsymbol{x})$ to be evaluated for any $\boldsymbol{x} \in \Omega$. Our curve optimization algorithm then iteratively refines the curves by changing their shapes (i.e., the trajectories) and topologies to obtain better approximation. The resulting diffusion curve image consists of the optimized curves $\mathbb{B}$ and the domain boundary $\partial\Omega$. During the optimization process, the colors along both sides of these curves (i.e., C_ℓ and C_r of the Laplace equation (1)) are sampled from the given color filed I, and the approximated color value $u(\boldsymbol{x})$ for all $\boldsymbol{x} \in \Omega$ is determined according to the equation (1) with the Dirichlet boundary condition

$$u(\boldsymbol{x}) = I(\boldsymbol{x}), \quad \forall \boldsymbol{x} \in \mathbb{B} \cup \partial\Omega. \tag{5}$$

We note that rather than sampling color values along the curves, prior methods [7], [12] post-optimize color values after the curves are determined. We will discuss the extension of our method to incorporate their post-optimization later (§5.5) and examine it in our experiments (§6.3).

5.1 PDE-Constrained Optimization Problem

Formally, our iterative curve optimization process minimizes a *cost functional* defined as the L_2 residual of the color approximation,

$$R(\Omega; \mathbb{B}) = \frac{1}{2} \int_\Omega (u(\boldsymbol{x}) - I(\boldsymbol{x}))^2 \, \mathrm{d}\Omega, \tag{6}$$

where u is the color field determined by diffusion curves. We write $\mathbb{B}$ as a parameter of R to emphasize the dependence of the residual on $\mathbb{B}$ through the Dirichlet boundary condition (5). Since u is the solution of the Laplace equation (1), we are concerned with an optimization problem with a PDE constraint,

$$\min_{\mathbb{B}} R(\Omega; \mathbb{B}) \;\text{ s.t. } u \text{ satisfies the Laplace eqn. (1).} \tag{7}$$

PDE-constrained optimization problems are known to be challenging in general [26]. In our problem (7), the optimization variables are the shapes of diffusion curves, that is, the spatial trajectories and topologies of the curves. Ideally, a curve can have an arbitrarily continuous trajectory, and therefore needs to be represented using a continuous functional rather than using individual and discrete parameters. More importantly, the error residual $R(\Omega; \mathbb{B})$ depends on the optimization variables (the curves) through the Laplace equation (1) in a complex manner: any local change to the curves $\mathbb{B}$ has a global impact, one that changes u over the entire domain Ω, which further affects the residual via (6).

5.2 Gradient-Descent Solver

We propose a new approach for solving the curve optimization problem (7), following the general spirit of gradient descent. Starting from a set of initial curves, our approach iteratively decreases the residual (6) by adjusting their shapes. Throughout, a fundamental difficulty we need to address is the computation of the residual's "gradient" with respect to the shapes of the curves, as the conventional gradient in terms of continuous curves is undefined.

We develop our method from the perspective of functional analysis: in each gradient-descent step, we first construct a velocity field $\boldsymbol{v}$ on the curves, specifying $\boldsymbol{v}(\boldsymbol{x})$ for all $\boldsymbol{x} \in \mathbb{B}$ (Figure 8). We then use $\boldsymbol{v}(\boldsymbol{x})$ to deform the curves, analogous to a (2D) surface flow in geometry processing [27], [28]. In other words, we evolve the curves via a single step of the forward Euler method of integrating $\dot{\boldsymbol{x}} = \boldsymbol{v}(\boldsymbol{x})$, $\forall \boldsymbol{x} \in \mathbb{B}$.

We now present the details of computing such a $\boldsymbol{v}$ that after deforming the curves accordingly, the residual is guaranteed to decrease (lines 5–7 of Algorithm 3). Briefly speaking, we will first assume that $\boldsymbol{v}(\boldsymbol{x})$ is known and analytically express how much the residual would change if the curve is deformed according to $\boldsymbol{v}$. This analytical expression allows us to formulate the condition of $\boldsymbol{v}$ resulting in a decrease of the residual, and thereby provides us a recipe for computing $\boldsymbol{v}$.

Fréchet derivative as a linear form: Given a domain Ω and a set of initial curves $\mathbb{B}_0$, we consider a general cost functional,

$$C(\Omega; \mathbb{B}_0) = \int_{\Omega_0} y(\boldsymbol{x}; \mathbb{B}_0) \, \mathrm{d}\Omega, \tag{8}$$

where y is continuous on Ω and may depend on the choice of $\mathbb{B}_0$. Our residual (6) takes the form $y(\boldsymbol{x}; \mathbb{B}_0) = \frac{1}{2}(u(\boldsymbol{x}) - I(\boldsymbol{x}))^2$ and depends on $\mathbb{B}_0$ via the Laplace solution u. Assuming a known $\boldsymbol{v}$, we introduce the *Fréchet derivative* [29] of C with respect to $\boldsymbol{v}$. Let $\mathbb{B}_t$ denote the curves evolved according to $\boldsymbol{v}$ after an infinitesimal time period of t, that is, $\boldsymbol{x} \mapsto \boldsymbol{x} + \boldsymbol{v}(\boldsymbol{x})\, t$ for all $\boldsymbol{x} \in \mathbb{B}_0$ (Figure 8). The Fréchet derivative of C is a linear form of $\boldsymbol{v}$ satisfying that

$$\mathrm{d}C(\Omega; \mathbb{B}_0) = \lim_{t \downarrow 0} \frac{1}{t} (C(\Omega; \mathbb{B}_t) - C(\Omega; \mathbb{B}_0)).$$

Conceptually, this derivative measures how quickly the cost functional C changes as we deform the curves using $\boldsymbol{v}$ infinitesimally According to *Hadamard-Zeloésio Structure Theorem* [30], such a linear form always exists when Ω, $\mathbb{B}_0$ and $\boldsymbol{v}$ are sufficiently regular, which is usually the case in practice. For our cost functional (6), we further reduce the Fréchet derivative into a linear form expressed as a

Algorithm 3 Gradient-descent diffusion curve optimization

Require: initial curves $\mathbb{B}_0$, color field I on Ω with boundary $\partial\Omega$

1: **procedure** CURVEOPT($\mathbb{B}_0$, $\partial\Omega$, I, Ω)
2: $\quad\Delta R \leftarrow \infty$; $\quad\mathbb{B} \leftarrow \mathbb{B}_0$ $\quad\triangleright\ \Delta R$ tracks residual change
3: $\quad$**while** $\Delta R > \epsilon$ **do**
4: $\quad\quad$triangulate Ω using $\partial\Omega \cup \mathbb{B}$ as boundaries $\quad\triangleright$ §5.4
5: $\quad\quad$solve the Laplace equation (1) for $u(\boldsymbol{x})$
6: $\quad\quad$solve the Poisson equation (11) for $p(\boldsymbol{x})$ $\quad\triangleright$ §5.2
7: $\quad\quad$compute $v_n(\boldsymbol{x}) = -B_R(\boldsymbol{x})$ using (12)
8: $\quad\quad$forward-Euler curve advancement, $\mathbb{B} \leftarrow \mathbb{B} + v_n t$
9: $\quad\quad$evaluate R using (6), and update its change ΔR
10: $\quad$**end while**
11: $\quad$**return** current curves $\mathbb{B}$
12: **end procedure**

boundary integral

$$\mathrm{d}C(\Omega;\mathbb{B}_0) = L[\boldsymbol{v}(\boldsymbol{x})] := \int_{\Gamma_0} B(\boldsymbol{x}) v_n(\boldsymbol{x})\,\mathrm{d}\Gamma, \tag{9}$$

where $v_n(\boldsymbol{x}) := \boldsymbol{v}(\boldsymbol{x}) \cdot \boldsymbol{n}(\boldsymbol{x})$ denotes the normal velocity on the curves (Figure 8), $\Gamma_0 = \partial\Omega \cup \mathbb{B}_0$ includes both the domain boundary $\partial\Omega$ and all the inner curves $\mathbb{B}_0$ (see Figure 7), and B is another function independent from $\boldsymbol{v}$ but related to the specific integrand y. In the rest of this subsection, we aim to derive a formula to evaluate $B(\boldsymbol{x})$ for any $\boldsymbol{x} \in \mathbb{B}_0$.

Once B is known, setting

$$v_n(\boldsymbol{x}) = \begin{cases} -B(\boldsymbol{x}) & \text{if } \boldsymbol{x} \in \mathbb{B}_0, \\ 0 & \text{if } \boldsymbol{x} \in \partial\Omega \end{cases} \tag{10}$$

guarantees a negative derivative value in (9) (assuming that B does not vanish everywhere on $\mathbb{B}_0$). This provides a formula of constructing v_n, which we then apply to deform the curves $\mathbb{B}_0$. With a sufficiently small timestep size t, the deformed curves $\mathbb{B}_t$, computed by $\boldsymbol{x} + v_n(\boldsymbol{x})\,t$ for all $\boldsymbol{x} \in \mathbb{B}_0$, is guaranteed by construction to yield a smaller residual value and thus a better approximation of I.

Computational Recipe: Shape Optimization Theory has provided a simple recipe of computing B for our particular cost functional (6). Here we simply present the formulas. Please see §3 of the supplementary document a detailed derivation.

We first solve the Laplace equation (1) to compute $u(\boldsymbol{x})$, which in turn allows us to construct a Poisson equation with a Dirichlet boundary condition,

$$\begin{aligned} \Delta p(\boldsymbol{x}) &= u(\boldsymbol{x}) - I(\boldsymbol{x}) \\ p(\boldsymbol{x}) &= 0, \qquad\qquad \forall \boldsymbol{x} \in \Gamma_0. \end{aligned} \tag{11}$$

Next, the solution p of this equation, together with u, allows the computation of $B(\boldsymbol{x})$ in a simple form

$$B(\boldsymbol{x}) = \frac{\partial p(\boldsymbol{x})}{\partial n}\left(\frac{\partial I(\boldsymbol{x})}{\partial n} - \frac{\partial u(\boldsymbol{x})}{\partial n}\right). \tag{12}$$

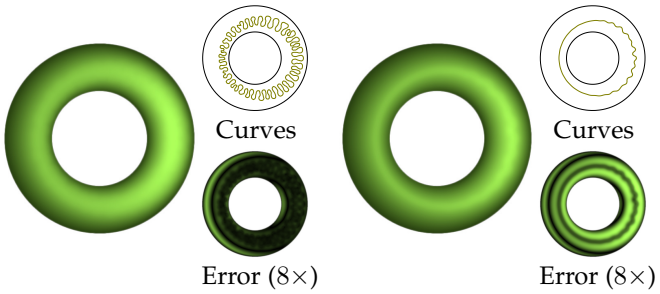
Combining (12) and (10) computes the normal velocity v_n, the velocity that can deform the curves $\mathbb{B}_0$ and lead to a decrease of the approximation residual (6). This computation is performed at each gradient-decent step, and the optimization process stops when the residual change drops below a threshold ϵ. Figure 10 illustrates the optimization process with synthetic examples.

5.3 Regularizing Curve Complexity

So far, our optimization problem (7) focuses solely on minimizing the L_2 residual (6). However, because the L_2 error along a curve is always zero due to the boundary condition (5), one simple way to yield a very low residual is to use space-filling curves. Indeed, if we start with one curve in a complex color region, it becomes zigzag after running the optimization for many iterations (Figure 9-a). While the numerical residual is low for such curves, their largely increased geometric complexity may be undesirable for certain applications (such as vector graphics editing). Thus, we propose a simple extension to the cost functional (6), providing users the flexibility to trade approximation accuracy for simpler curves. To this end, we add a regularization term to (6) to penalize the total length of the curves:

$$\tilde{R}(\Omega;\mathbb{B}) = \frac{1}{2}\int_{\Omega} (u(\boldsymbol{x}) - I(\boldsymbol{x}))^2\,\mathrm{d}\Omega + \alpha \int_{\mathbb{B}} \mathrm{d}\Gamma, \tag{13}$$

where α is a user-specified scalar controlling the strength of regularization. It can be shown that similar to (9), the



(a) Weak regularization (b) Strong regularization

Fig. 9: Our method allows the user to **regularize** curve complexity. Subfigures (a) and (b) show two optimization results using the color field illustrated in Figure 3 as input. Both results are generated using Algorithm 3 with identical initial configurations (a circle) but varying α values. The resulting curves and error images (scaled by $8\times$) are shown to the right of the final images. See Figures 6 and 14 for results created using our full pipeline (Algorithm 1).

Fréchet derivative of the second term is also a linear form of v_n. Let $R_L(\mathbb{B}_0) = \int_{\mathbb{B}_0} d\Gamma$. Then, its derivative is

$$\mathrm{d}R_L(\mathbb{B}_0) = \int_{\mathbb{B}_0} \kappa(\boldsymbol{x}) v_n(\boldsymbol{x})\,\mathrm{d}\Gamma, \tag{14}$$

where $\kappa(\boldsymbol{x})$ measures the curvature of a point $\boldsymbol{x}$ on the curves. This formula has been used to derive the mean curvature flow [31] in geometry processing. It is also a special case of the Fréchet derivative of a general boundary integral (see §1.3 of the supplementary document). Following the derivation of B in §5.2, we obtain the normal velocity for decreasing $\tilde{R}(\Omega;\mathbb{B})$, that is, $v_n(\boldsymbol{x}) = -B_R(\boldsymbol{x}) - \alpha\kappa(\boldsymbol{x})$. With this slightly different velocity formula, the entire optimization algorithm remains the same as before. In addition, the user is able to control the complexity of resulting curves by adjusting the strength of regularization (Figure 9).

5.4 Implementation Details

We now present implementation details of Algorithm 3, wherein two major steps are solving the Laplace equation (1) and the Poisson's equation (11). Both PDEs have Dirichlet boundary conditions defined on the boundary of Ω and the optimized curves $\mathbb{B}$ (recall (5)). Since we also need to evaluate the domain integral over Ω during the iterations (line 9 of Algorithm 3), we triangulate the entire domain of Ω and use the Finite Element Method [32] for both solves, while other numerical solvers (e.g., the Boundary Element Method) could also be applied.

Finite element discretization: We discretize the boundary and optimized curves into piecewise linear segments and represent them using polylines. The velocity v_n is discretized and stored at every vertex along the polylines. We use the package Triangle [33] to triangulate the domain Ω (line 4 of Algorithm 3). The resulting triangle mesh is then used in the finite element solves. The computation of curves' normal velocity in (12) involves boundary normal derivatives of the finite element solutions (i.e., $\partial p/\partial n$ and $\partial u/\partial n$). We choose the second-order finite element basis, as it offers higher accuracy especially near the boundary (see §4 of the supplementary document).

Curve tracking: Advancing the curves using the computed normal velocity (i.e., computing $\mathbb{B}_t$ given $\mathbb{B}_0$ and v_n in

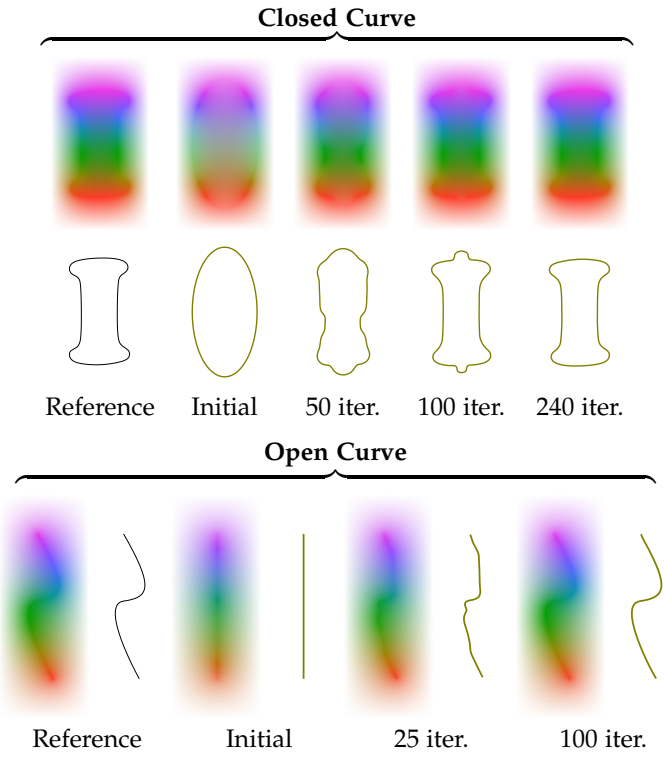


Fig. 10: **Synthetic validation** of our *diffusion curve optimization* algorithm. The left-most column contains input color fields given by one closed curve (top) and one open curve (bottom). The following columns respectively show diffusion curve images after Algorithm 3 finishing a varying number of iterations. Our resulting curves precisely match the original.

Figure 8) is a typical yet nontrivial surface tracking problem. We use a recently developed explicit tracking approach [34], which advances the vertices on curve polylines using explicit forward Euler method, and then carefully remeshes the polylines to ensure correct topology changes and a collision-free state.

Timestep size t: To ensure robust curve tracking, we dynamically set the timestep size t for the forward-Euler curve advancement (line 8 in Algoirthm 3). We start with choosing a t value such that the vertex displacement $v_n(\boldsymbol{x})\,t, \forall \boldsymbol{x} \in \mathbb{B}$ would not collapse any polyline segment on $\mathbb{B}$. This ensures that possible topology changes can be robustly processed. From this starting value, we iteratively halve t until the residual value (after a step of curve deformation) decreases.

5.5 Discussions

Measuring geometric complexities: In Eq. (13) and the rest of this paper, we use the *total length* of all diffusion curves to measure their geometric complexity. Depending on specific applications, there may exist other metrics more suitable to user needs. As an example, in §2 of the supplementary document, we discuss another possible measure which can also be incorporated in our curve optimization framework.

Coloring schemes: As described at the beginning of this section, given the curve geometry $\mathbb{B}$, we specify colors on both sides of each curve by directly sampling color values from the input color field I. Alternatively, prior

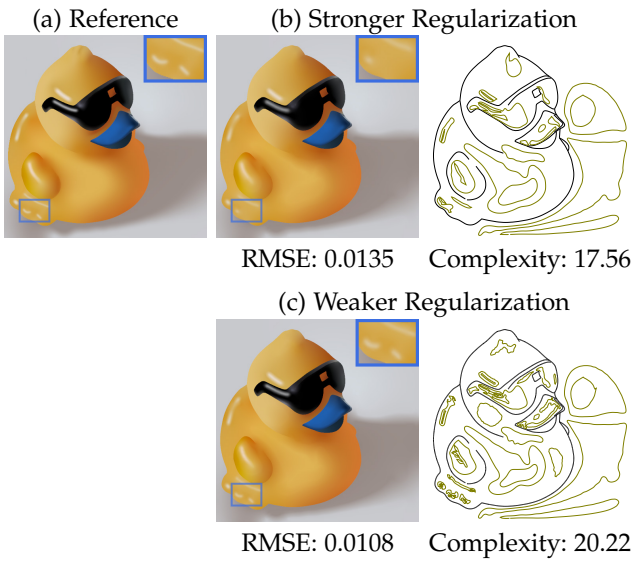


Fig. 11: Our method allows the user to **balance** resulting accuracy with curve complexity by varying the strength of regularization.

work [7], [12] propose to post-optimize curve colors for better reconstruction accuracy. Our method can easily adopt this approach, post-optimizing the colors after the curves are optimized. We implemented this approach and present the results in §6.3.

Higher-order domains and curves: While our approach focuses on solving the inverse problem of the standard (first-order) diffusion curves, it can be also applied to higher-order domains. For instance, as demonstrated in §6.3, we can feed ∇I instead of I to Algorithm 3 to compute curves offering a higher order of smoothness.

In principle, one can use the cost function (6) to optimize $u(\boldsymbol{x})$ resulted from higher-order (e.g., biharmonic diffusion) curves. However, in our straightforward derivation following that in §5.1 (for biharmonic curves), we found that this derivation dramatically complicates the form of the Fréchet derivative (9) and thus that of the velocity field (10). As a result, the velocity field to evolve higher-order curves are much more difficult to evaluate numerically. We thus leave the extension of (6) to higher-order curves as a future work.

6 Results

In §6.1, we show experimental results demonstrating the validity of our curve optimization algorithm as well as how the regularization behaves in practice. Then, in §6.2, we show reconstructed diffusion curve images using input color fields represented in three forms: pixel image, 3D renderings, and gradient meshes. In addition, we show preliminary results motivating possible future applications.

6.1 Experimental Results

Synthetic validations: We design two synthetic tests (Figure 10) to validate our diffusion curve optimization algorithm (Algorithm 3). In these tests, the input color fields I are themselves diffusion curve images with continuous colors. In this case, the optimal $\mathbb{B}$ is simply the set of diffusion curves used to generate I. Although the shape optimization problem is in general non-convex, our method successfully finds the optimal solutions for both closed and

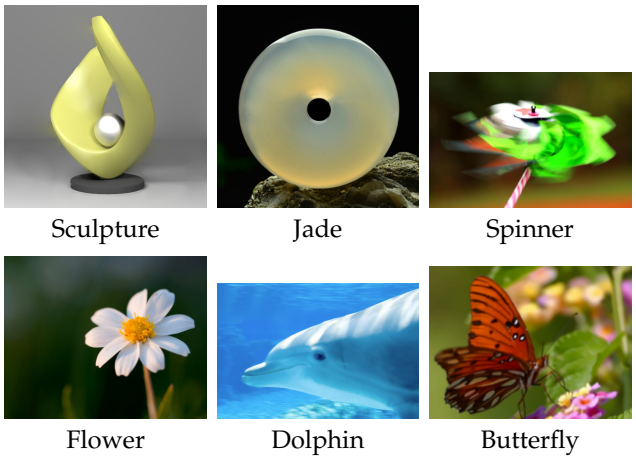


Fig. 12: **Input pixel images** for generating diffusion curve results in Figures 13 and 14.

open curves. Please see the accompanying video for curve deformation animations.

Regularizing curve complexity: As discussed in §5.3, our method is able to balance resulting accuracy and curve complexity by varying the strength α of regularization in Eq. (13). Figure 11 shows how α influences resulting curves generated with our full pipeline (Algorithm 1). Figure 11-a has simpler curves (due to greater α), but some highlights at the bottom-left of the image are absent. Figure 11-b, on the other hand, provides lower approximation error but at the cost of greater curve complexity (resulting from a lower α). In all our results, the complexity is numerically defined as total curve length normalized, so that the longest axis of each image's bounding box has unit length.

6.2 Main Results

We now show diffusion curve images generated using our method (Algorithm 1). All our results utilize the per-pixel blurring described in §4.4. Please refer to the supplemental material for unblurred versions.

Theoretically, our approach does not require the input color field I to have any particular representation or discretization. In practice, we demonstrate such flexibility using three types of input: pixel images, 3D renderings, and gradient meshes. The execution time for generating each of these results is summarized in Table 1. The supplemental video contains animations demonstrating the creation process for these results.

Pixel images: One common way to represent a color field I is to use standard *pixel images*. In this case, $I(\boldsymbol{x})$ is evaluated using bilinear interpolation, and $\nabla I(\boldsymbol{x})$ using finite difference. As stated in §4.1, we perform edge detection to obtain the boundary curves $\partial\Omega$ required by Algorithm 1. Although color discontinuities are not well defined for standard pixel images, our method in practice is robust on the choice of boundary curves. Figure 13 shows three examples with boundary curves detected using Canny detector with three thresholds. Notice how missing boundaries (when increasing the threshold) are handled by additional curves generated by Algorithm 1. All our results for standard pixel input used thresholds between 0.1 and 0.2.

Figure 14 contains diffusion curve images reconstructed from pixel input (Figures 3-a and 12) using Algorithm 1 as well as previous edge detection based methods [1], [7].[2] The parameters for each method are selected such that the resulting curves have approximately *identical complexities* (measured with their total lengths). Our method outperforms previous ones when handling smoothly varying color fields. Notice that, in the bottom two rows (i.e., Flower and Jade), Xie et al.'s approach [7] has slightly higher approximation errors (measured in RMSE), because it requires higher curve complexities to work properly in these cases.

TABLE 1: Optimization time (in seconds) for generating results in Figures 14, 15, and 16 using our approach on a Linux machine with an 8-core Intel Xeon E5 CPU.

Input Format	Scene	Time	Scene	Time
Pixel images (Figure 14)	Torus	14.5	Sculpture	48.3
	Jade	37.6	Spinner	62.7
	Flower	64.2	Dolphin	43.7
	Butterfly	505.3		
3D renderings (Figure 15)	Cornell Box	27.8	Carved	97.2
	Twill	13.5	Knots	295.6
	Wobble Chess	460.3		
Grad. meshes (Figure 16)	Apple	44.2	Tomato	16.7
	Mango	16.1	Candle	11.6

3D renderings: Another kind of color field common to computer graphics applications is renderings of 3D scenes. Our approach can be used to approximate these color fields with diffusion curve images. In this case, we represent I as high-resolution pixel images and obtain the boundary curves $\partial\Omega$ directly using object contours.[3] Figure 15 demonstrates diffusion curve images generated using our method from 3D renderings, where slightly higher curve complexities (compared to Figure 14) are permitted to ensure low reconstruction errors. Our method successfully captures detailed appearances: from glossy surfaces to smooth shadow boundaries.

Gradient meshes: Our approach can also generate diffusion curve images directly from input color fields I represented using other vector formats. We demonstrate this using input color fields represented as gradient meshes in SVG format [35] where each mesh grid is a Coons Patch [36]. In this case, the color $I(\boldsymbol{x})$ and gradient $\nabla I(\boldsymbol{x})$ can be evaluated analytically for any $\boldsymbol{x} \in \Omega$, and the boundary curves $\partial\Omega$ are simply the mesh boundaries. As shown in Figure 16, our method directly creates diffusion curve images closely approximating the gradient meshes, without having to rasterize the input into pixel formats.

6.3 Additional Results

Coloring optimization: As discussed in §5.5, our technique is orthogonal and complementary to coloring optimization techniques which take diffusion curve geometry as input and optimizes colors (and color gradients) at both sides of the curves. These techniques can be used to replace our simple coloring scheme which samples color values directly (§5.1) and performs per-pixel blurring (§4.4). Figure 18 demonstrates that our optimized curve geometry can be coupled with the coloring optimization scheme introduced by Xie et al. [7] to further reduce reconstruction errors. Since

2. We thank the authors of [7] for confirming the correctness of results in Figure 14 generated with their approach.

3. In this example, we assume all objects to be homogeneous. To handle heterogeneity, $\partial\Omega$ needs to include color jumps across object surfaces.

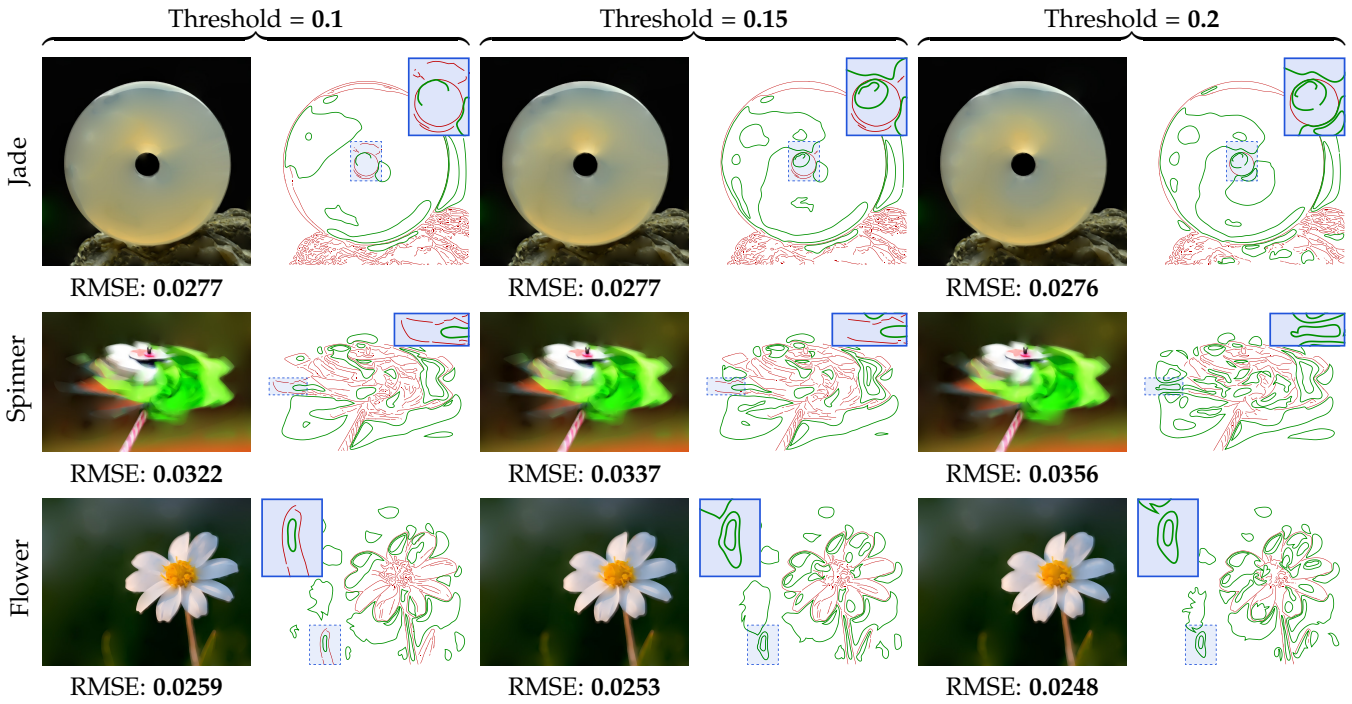


Fig. 13: Our approach, when handling standard pixel images, is **insensitive to threshold values** used for detecting initial boundary edges. For the examples above, three different thresholds varying from 0.1 to 0.2 have been used. Boundary curves and additional ones generated by Algorithm 1 are shown in red and green, respectively. The resulting reconstructions under roughly identical curve complexities offer similar qualities.

this technique explicitly optimizes color gradients across each curve, we did not perform per-pixel blurring.

Higher-order domain: Our approach can be applied to higher-order domains for generating curves with higher-order smoothness (as discussed in §5.5). Figure 19 shows an example where our pipeline (Algorithm 1) is applied to color gradients rather than original color values. In other words, given a RGB image I, we can use ∇I, a six-channel image, as the input. Given our reconstructed gradient image, we solve an additional least square problem to recover the final image.

However, as observed by Xie et al. [7], we found that for natural images, solving the optimization at higher-order domains generally does not lead to better approximation accuracy under similar curve complexities. This is because higher-order domains are usually filled with significantly more high-frequency contents that require complex (almost space-filling) curve geometry to accurately reconstruct.

Animated result: Lastly, we show preliminary results to motivate future applications of our approach. Since our method optimizes the shape of diffusion curves iteratively, it is suitable for generating animated results from a sequence of gradually changing input color fields. The basic idea is *curve reusing*: taking optimized curve geometry from one frame as the initial configuration to "warm start" the next.

Figure 20 and the accompanying video show a proof-of-concept example. The input is the relighting (i.e., the object stays static while the light source moves) of a shiny torus knot. In this case, the boundary curves keep unchanged throughout all frames, and optimized curve geometry from one frame remains valid for all other frames. Previous methods [1], [7] cannot easily enforce curve coherence across different frames, leading to temporally noisy animations. By modifying the curve initialization step in Algorithm 1 to reuse optimized curve geometry, we are able to accelerate the optimization process by 2.3×, and the resulting animation has lower approximation error and little noise. Please see the supplementary video for full animations.

7 Conclusion

This paper introduces a novel solution to the inverse diffusion curve problem. The key component of our approach is a curve optimization algorithm that iteratively deforms a set of diffusion curves in a way that guarantees the reduction of approximation error. Based upon the core algorithm, we develop a full pipeline that takes an input color field as well as a set of boundary curves, and produces an image with well-shaped and clean curves that closely matches the input. Our approach offers the generality to take input presented in different formats, which we demonstrate using three types: pixel images, 3D renderings, and gradient meshes.

Limitations and Future Work: Our approach has a few limitations that can inspire future work. Firstly, it requires the color field to be C^0 continuous everywhere except at the given boundaries. Robustly finding clean boundary curves, however, can be challenging. Secondly, if the color field contains spatially high-frequency features, very fine triangulation may be needed to fully resolve them. Not meeting the required level of resolution can lead to results with poor accuracy. Furthermore, our reconstructed curves, despite being simpler than those produced with previous methods [1], [7], are still too complex for many manual editing applications. This is mainly due to our error metrics (6) and (13) being focused on reconstruction error but

not the curve editability. In the future, more sophisticated error metrics and regularization schemes can be developed in order to enable easy editing of automatically generated diffusion curves. Lastly, the diffusion curves considered in this work are mainly the first-order curves, ones that are described by the diffusion equation. While we demonstrated a simple extension to handle high-order curves (in §6.3), it remains open how to derive a shape optimization method for biharmonic diffusion curves.

REFERENCES

[1] A. Orzan, A. Bousseau, H. Winnemöller, P. Barla, J. Thollot, and D. Salesin, "Diffusion curves: A vector representation for smooth-shaded images," *ACM Trans. Graph.*, vol. 27, no. 3, pp. 92:1–92:8, 2008.

[2] S. Jeschke, D. Cline, and P. Wonka, "A GPU Laplacian solver for diffusion curves and Poisson image editing," *ACM Trans. Graph.*, vol. 28, no. 5, 2009.

[3] K. Takayama, O. Sorkine, A. Nealen, and T. Igarashi, "Volumetric modeling with diffusion surfaces," *ACM Trans. Graph.*, vol. 29, no. 6, pp. 180:1–180:8, 2010. [Online]. Available: http://dx.doi.org/10.1145/1882261.1866202

[4] X. Sun, G. Xie, Y. Dong, S. Lin, W. Xu, W. Wang, X. Tong, and B. Guo, "Diffusion curve textures for resolution independent texture mapping," *ACM Trans. Graph.*, vol. 31, no. 4, 2012.

[5] P. Ilbery, L. Kendall, C. Concolato, and M. McCosker, "Biharmonic diffusion curve images from boundary elements," *ACM Trans. Graph.*, vol. 32, no. 6, 2013.

[6] T. Sun, P. Thamjaroenporn, and C. Zheng, "Fast multipole representation of diffusion curves and points," *ACM Trans. Graph.*, vol. 33, no. 4, pp. 53:1–53:12, 2014.

[7] G. Xie, X. Sun, X. Tong, and D. Nowrouzezahrai, "Hierarchical diffusion curves for accurate automatic image vectorization," *ACM Trans. Graph.*, vol. 33, no. 6, pp. 230:1–230:11, 2014.

[8] J. Sokolowski and J.-P. Zolésio, *Introduction to shape optimization*. Springer, 1992.

[9] W.-M. Pang, J. Qin, M. Cohen, P.-A. Heng, and K.-S. Choi, "Fast rendering of diffusion curves with triangles," *IEEE Computer Graphics and Applications*, vol. 32, no. 4, pp. 68–78, 2012.

[10] R. Prévost, W. Jarosz, and O. Sorkine-Hornung, "A vectorial framework for ray traced diffusion curves," in *Computer Graphics Forum*, 2014.

[11] J. Canny, "A computational approach to edge detection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, no. 6, pp. 679–698, 1986.

[12] S. Jeschke, D. Cline, and P. Wonka, "Estimating color and texture parameters for vector graphics," in *Computer Graphics Forum*, vol. 30, no. 2, 2011, pp. 523–532.

[13] M. Finch, J. Snyder, and H. Hoppe, "Freeform vector graphics with controlled thin-plate splines," *ACM Trans. Graph.*, vol. 30, no. 6, pp. 166:1–166:10, 2011.

[14] S. Boyé, P. Barla, and G. Guennebaud, "A vectorial solver for free-form vector gradients," *ACM Trans. Graph.*, vol. 31, no. 6, pp. 173:1–173:9, 2012.

[15] J. Haslinger *et al.*, *Introduction to shape optimization: theory, approximation, and computation*. Siam, 2003, vol. 7.

[16] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *International journal of computer vision*, vol. 1, no. 4, pp. 321–331, 1988.

[17] D. Mumford and J. Shah, "Optimal approximations by piecewise smooth functions and associated variational problems," *Communications on pure and applied mathematics*, vol. 42,5, 1989.

[18] R. Schneider and L. Kobbelt, "Geometric fairing of irregular meshes for free-form surface design," *Computer aided geometric design*, vol. 18, no. 4, pp. 359–379, 2001.

[19] K. Crane, U. Pinkall, and P. Schröder, "Robust fairing via conformal curvature flow," *ACM Trans. Graph.*, vol. 32, no. 4, 2013.

[20] B. Mohammadi, O. Pironneau, B. Mohammadi, and O. Pironneau, *Applied shape optimization for fluids*. Oxford University Press Oxford, 2001, vol. 28.

[21] M. Burger, S. J. Osher, and E. Yablonovitch, "Inverse problem techniques for the design of photonic crystals," *IEICE transactions on electronics*, vol. 87, no. 3, pp. 258–265, 2004.

[22] A. Herbulot, S. Jehan-Besson, S. Duffner, M. Barlaud, and G. Aubert, "Segmentation of vectorial image features using shape gradients and information measures," *Journal of Mathematical Imaging and Vision*, vol. 25, no. 3, pp. 365–386, 2006.

[23] M. Jung, G. Peyré, and L. D. Cohen, "Nonlocal active contours," *SIAM Journal on Imaging Sciences*, vol. 5, no. 3, pp. 1022–1054, 2012.

[24] U. Ramer, "An iterative procedure for the polygonal approximation of plane curves," *Computer Graphics and Image Processing*, vol. 1, no. 3, pp. 244–256, 1972.

[25] P. Selinger, "Potrace: a polygon-based tracing algorithm," *Potrace (online)*, 2003.

[26] R. Pinnau and M. Ulbrich, *Optimization with PDE constraints*. Springer, 2008, vol. 23.

[27] J. A. Sethian *et al.*, "Level set methods and fast marching methods," *Journal of Computing and Information Technology*, vol. 11, no. 1, pp. 1–2, 2003.

[28] K. A. Brakke, "The surface evolver," *Experimental mathematics*, vol. 1, no. 2, pp. 141–165, 1992.

[29] R. Coleman, *Calculus on Normed Vector Spaces*. Springer, 2012.

[30] M. C. Delfour and J.-P. Zolésio, *Shapes and geometries: metrics, analysis, differential calculus, and optimization*. Siam, 2011, vol. 22.

[31] C. Mantegazza, *Lecture notes on mean curvature flow*. Springer, 2011, vol. 290.

[32] O. C. Zienkiewicz and P. Morice, *The finite element method in engineering science*. McGraw-hill London, 1971, vol. 1977.

[33] J. R. Shewchuk, "Triangle: Engineering a 2d quality mesh generator and delaunay triangulator," in *Applied computational geometry towards geometric engineering*. Springer, 1996, pp. 203–222.

[34] T. Brochu and R. Bridson, "Robust topological operations for dynamic explicit surfaces," *SIAM Journal on Scientific Computing*, vol. 31, no. 4, pp. 2472–2493, 2009.

[35] T. Bah, "Advanced gradients for SVG (online)," 2011.

[36] S. A. Coons, "Surfaces for computer-aided design of space forms," DTIC Document, Tech. Rep., 1967.

[37] J. Bonet and R. D. Wood, *Nonlinear Continuum Mechanics for Finite Element Analysis*. Cambridge University Press, 1997.

Shuang Zhao is an Assistant Professor in the Department of Computer Science at University of California, Irvine. Before joining UCI, he was a postdoctoral associate at MIT. Shuang received his M.S. and Ph.D. in computer science from Cornell University. His research focuses on material appearance modeling and physically-based rendering. He has been serving as the Information Director for ACM Transactions on Graphics.

Frédo Durand is a professor of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology, and a member of the Computer Science and Artificial Intelligence Laboratory (CSAIL). He received his PhD from Grenoble University, France. His research interests span most aspects of picture generation and creation. He received an Eurographics Young Researcher Award in 2004, an NSF CAREER award in 2005, a Microsoft Research New Faculty Fellowship in 2005, a Sloan fellowship in 2006, and the ACM SIGGRAPH Computer Graphics Achievement Award in 2016.

Changxi Zheng is an Assistant Professor in the Computer Science Department at Columbia University. Prior to joining Columbia, he received his M.S. and Ph.D. from Cornell University and his B.S. from Shanghai Jiaotong University. His research spans computer graphics, physically-based simulation, computational design, computational acoustics, scientific computing and robotics. He has been serving as an Associated Editor of ACM Transactions on Graphics, and won the NSF CAREER Award and the Cornell CS Best Dissertation award in 2012.

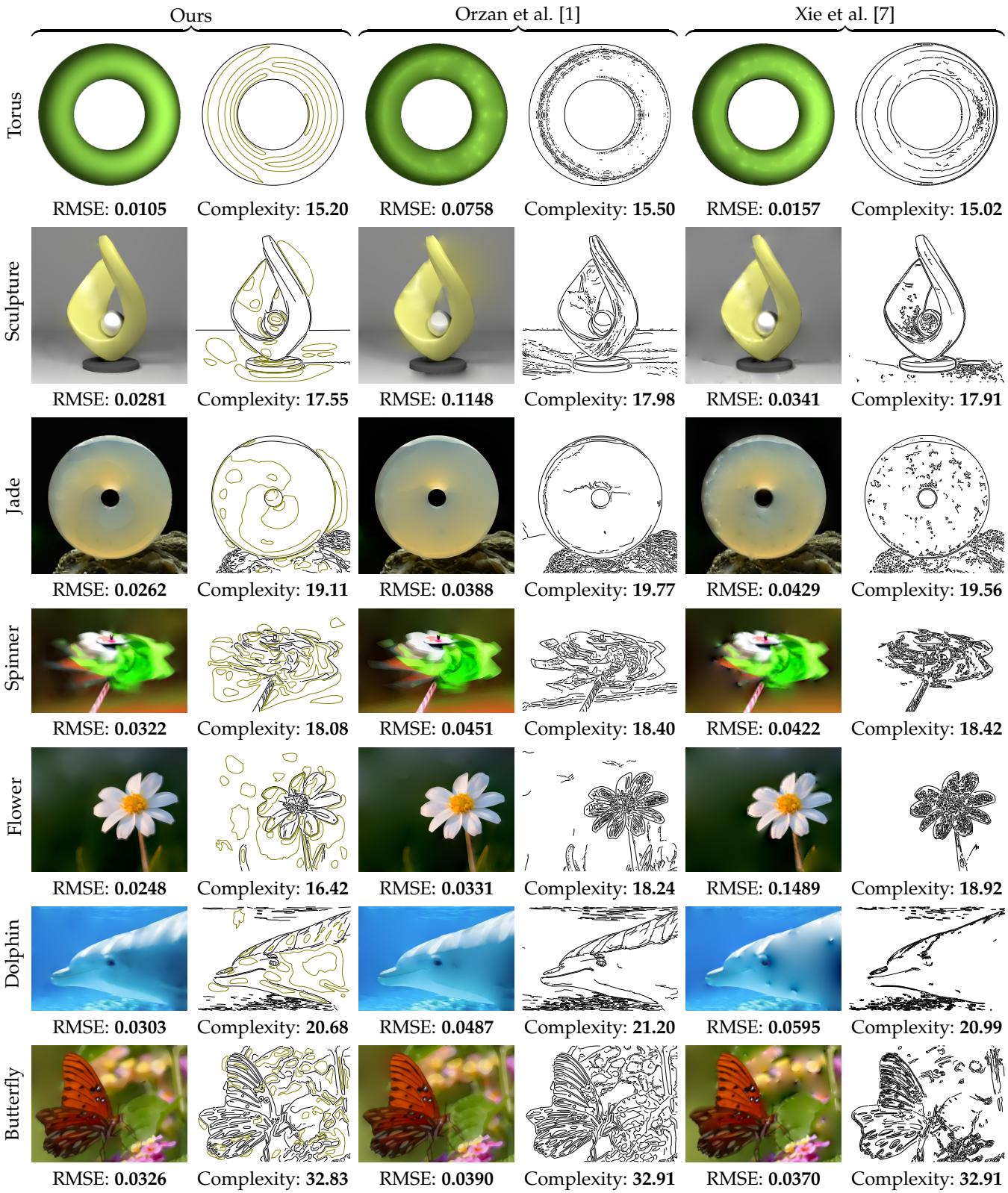


Fig. 14: **Comparisons** between diffusion curve images generated by our approach and previous methods using **pixel images** (top row) as input. The parameters are adjusted so that the resulting curves generated by each method have roughly *identical complexities*. Our approach not only yields lower approximation error (measured in RMSE), but also generates better shaped and relatively simple curves.

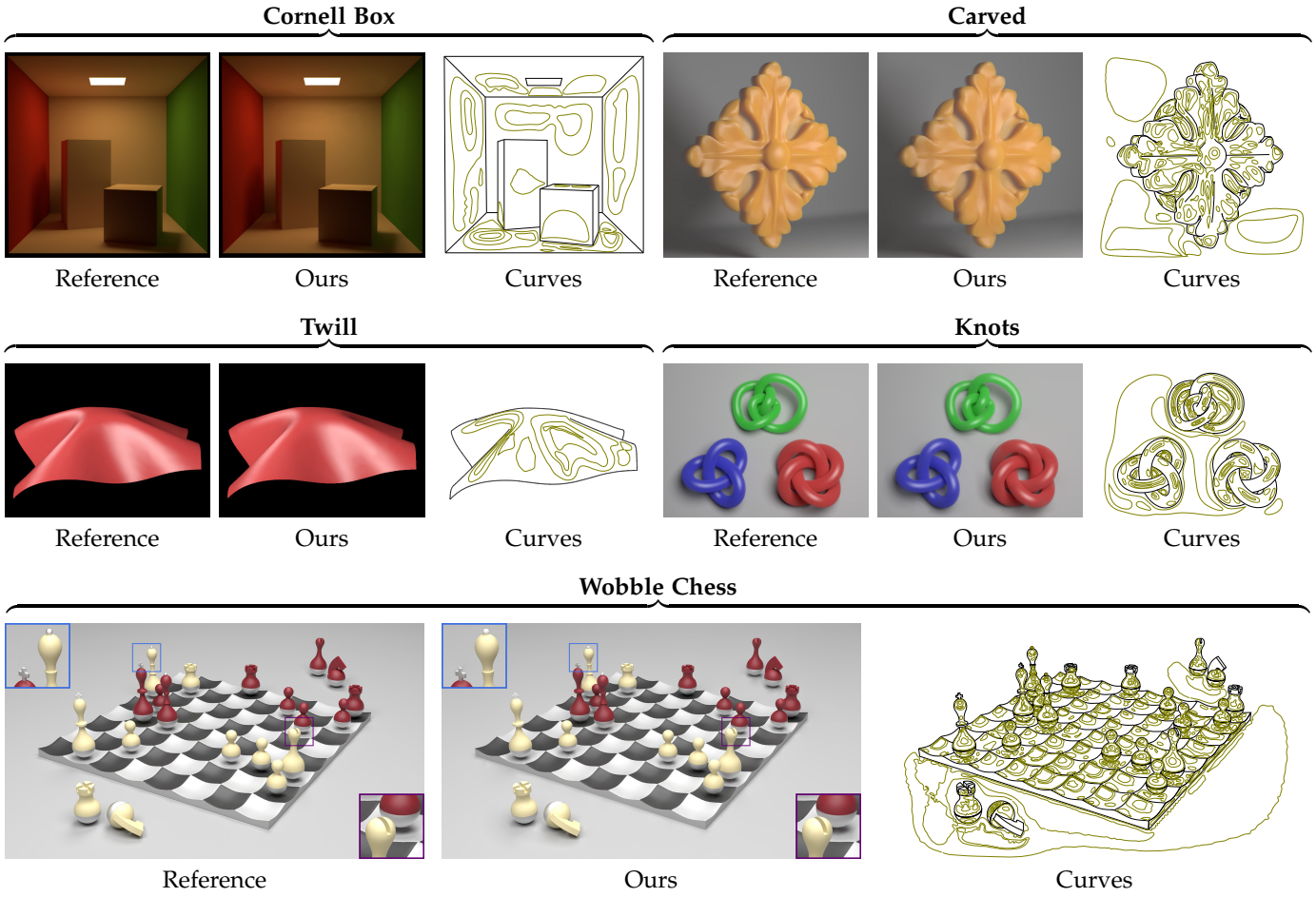


Fig. 15: Diffusion curve images generated from **renderings of 3D scenes** using our approach. The boundary curves are obtained using mesh contours extracted from the scene geometries.

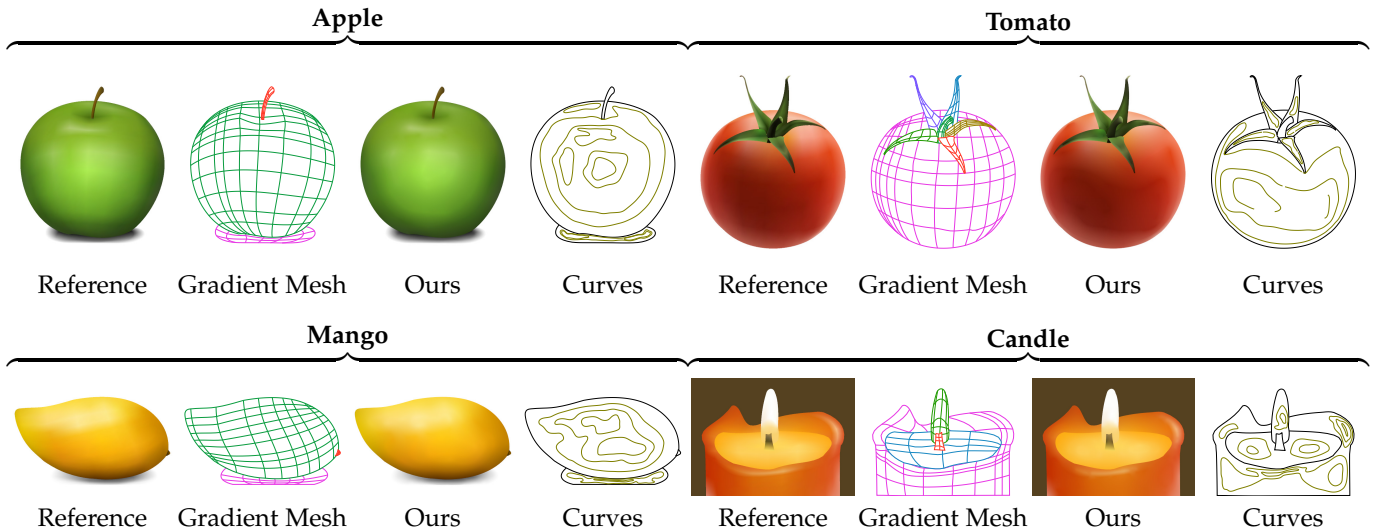


Fig. 16: Diffusion curve images generated from **gradient meshes** directly (i.e., without rasterizing into pixel images) using our method. The boundary curves are given by the mesh boundaries.

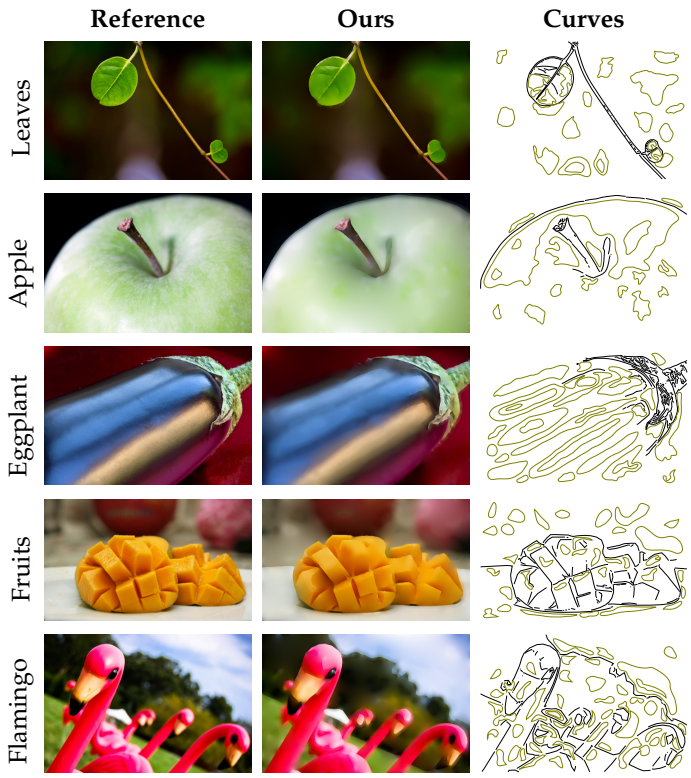


Fig. 17: Additional results generated by our approach from **pixel input**. Please see the supplementary materials for more results.

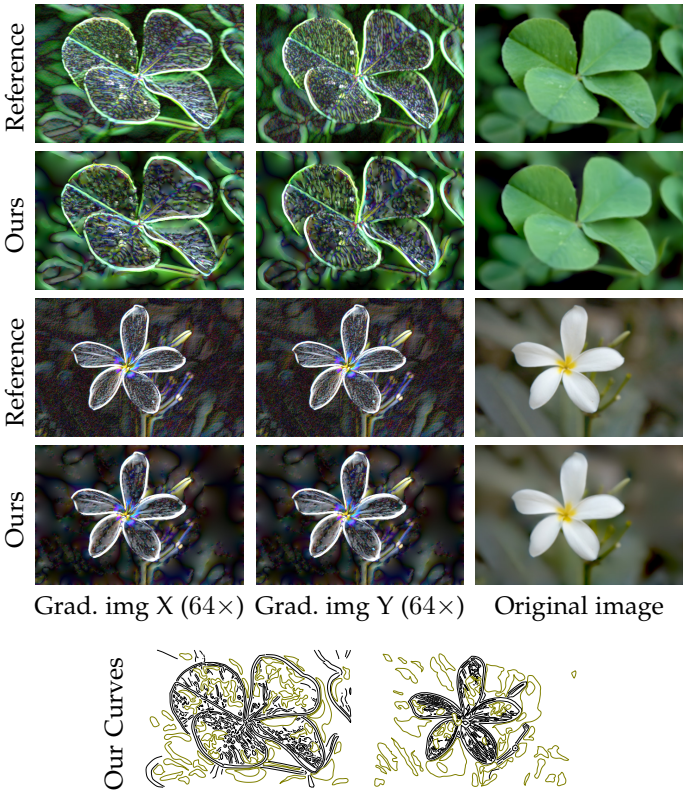


Fig. 19: Application of our method on the **color gradient domain** instead of the original domain. In this case, the input color field to our approach (Algorithm 1) is a six-channel image representing color gradients in horizontal (X) and vertical (Y) directions of the original image.

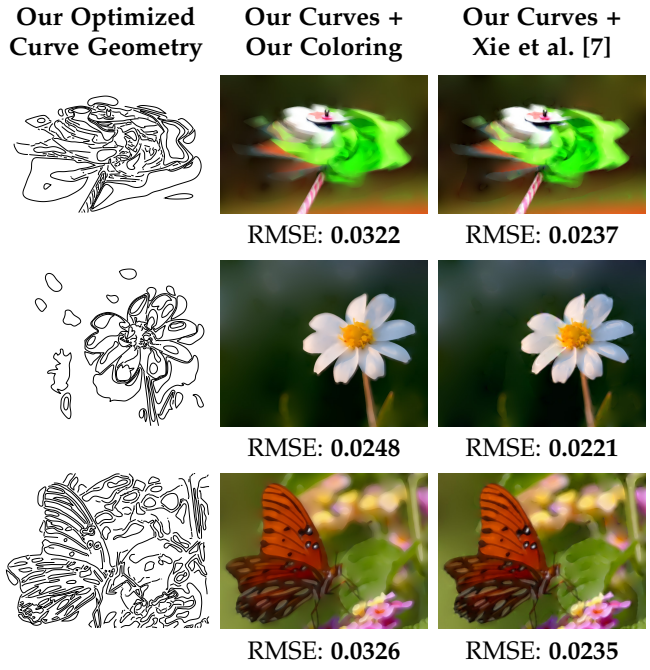


Fig. 18: Our core approach is completely **orthogonal** and **complementary** to coloring optimization techniques. In particular, sophisticated coloring optimization schemes such as [7] can be applied to our optimized curve geometry to further improve reconstruction accuracy.

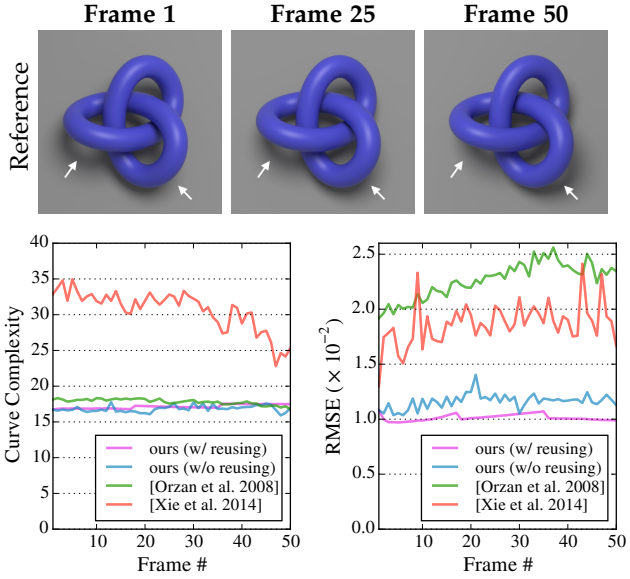


Fig. 20: **Animated results** consisting of 50 frames from relighting a torus knot. Three of these frames are shown on the top (where white arrows indicate regions with moving shadows). Higher curve complexities (left plot) are used for [7] for fewer artifacts. **Reusing** optimized curves from one frame as the starting point (i.e., initial curves) for the optimization of the next frame leads to temporally coherent curve geometry and better approximation accuracy (right plot). See the supplementary video for full animations.