Freeform Vector Graphics with Controlled Thin-Plate Splines

Mark Finch John Snyder Hugues Hoppe
Microsoft Research

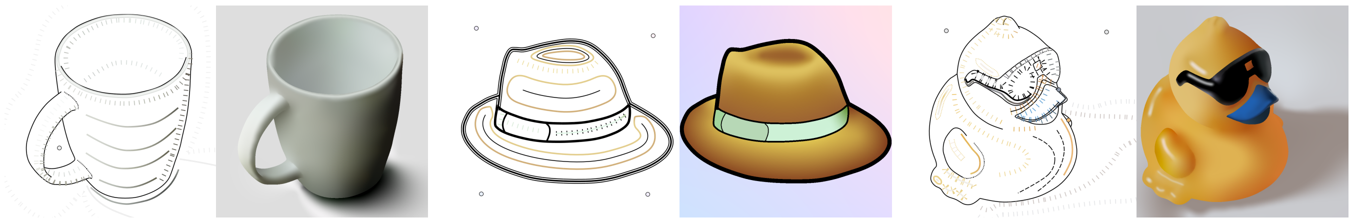


Figure 1: *We build on thin-plate splines to enrich vector graphics with a variety of powerful and intuitive controls.*

Abstract

Recent work defines vector graphics using diffusion between colored curves. We explore higher-order fairing to enable more natural interpolation and greater expressive control. Specifically, we build on thin-plate splines which provide smoothness everywhere except at user-specified tears and creases (discontinuities in value and derivative respectively). Our system lets a user sketch discontinuity curves without fixing their colors, and sprinkle color constraints at sparse interior points to obtain smooth interpolation subject to the outlines. We refine the representation with novel contour and slope curves, which anisotropically constrain interpolation derivatives. Compound curves further increase editing power by expanding a single curve into multiple offsets of various basic types (value, tear, crease, slope, and contour). The vector constraints are discretized over an image grid, and satisfied using a hierarchical solver. We demonstrate interactive authoring on a desktop CPU.

Keywords: bilaplacian/biharmonic PDE, slope/contour curves

Links: DL PDF WEB VIDEO DATA CODE

1 Introduction

Traditional vector graphics fills each closed shape independently with a simple color function. Recent work applies more global and powerful Laplacian interpolation between diffusion curves with colors on each side [Orzan et al. 2008; Jeschke et al. 2009].

A Laplacian solution yields a *membrane* function which is "as-constant-as-possible". Its low-order smoothness objective has drawbacks as illustrated in Figure 2. The solution is smooth only away from constrained points. Value constraints yield tent-like responses at isolated points and form creases along curves. The Laplacian objective is also incompatible with derivative constraints, because it already seeks zero first-derivatives everywhere in all directions. Only a higher-order notion of smoothness supports sparse constraints on directional derivatives.

Our approach builds on *thin-plate splines* (TPS) [Courant and Hilbert 1953], which define a higher-order interpolating function that is "as-harmonic-as-possible". This smoothness objective overcomes previous limitations (Figure 2). Thin-plate splines have been applied in several areas including geometric modeling [e.g. Welch and Witkin 1992; Botsch and Kobbelt 2004; Sorkine and Cohen-Or 2004; Botsch and Sorkine 2008], computer vision [Terzopoulos 1983], and machine learning [Bookstein 1989]. They have also been adapted to allow discontinuity control with explicit tears and creases [Terzopoulos 1988]. We extend these controls and demonstrate their usefulness in vector graphics authoring.

In the simplest case, an artist sketches some outlines (tears) without fixing their colors, and specifies color constraints at a few interior points or curves to obtain a smooth color wash within the outlines. This ink-and-paint ordering of tasks is similar to hand drawing. The result is then refined by adding creases, contour curves, slope curves, and critical points. These features increase editing power by anisotropically constraining interpolation derivatives (e.g. along or across the curves, or in both directions).

In addition to the basic curves, we introduce *compound* curves, with user-assigned widths, for more complex effects. These internally yield several offset curves, of possibly different types. For instance, a value-slope curve juxtaposes the two basic types to create a smooth ridge-like feature. A wide contour uses two offset contours to form a constant-colored strip without fixing its color. Other combinations produce a variety of interesting and useful results.

We demonstrate a prototype system based on these ideas. Like other variational approaches such as diffusion curves, our system is easy to use and supports "freeform" input based on a general network of curves, which we augment with points. Smoother interpolation and more flexible constraints enhance naturalness and editing power and produce rich results from a compact input (Figure 1).

Our contributions include:

- Extension of the diffusion curves framework to benefit from higher-order interpolation and general discontinuity control.
- A discretized least-squares kernel for accurate modeling of crease curves.
- Contour and slope curves that constrain derivatives anisotropically for intuitive control.
- A variety of compound curve types for added expressiveness.
- Discontinuity-aware upsampling for improved accuracy in a multiresolution setting.

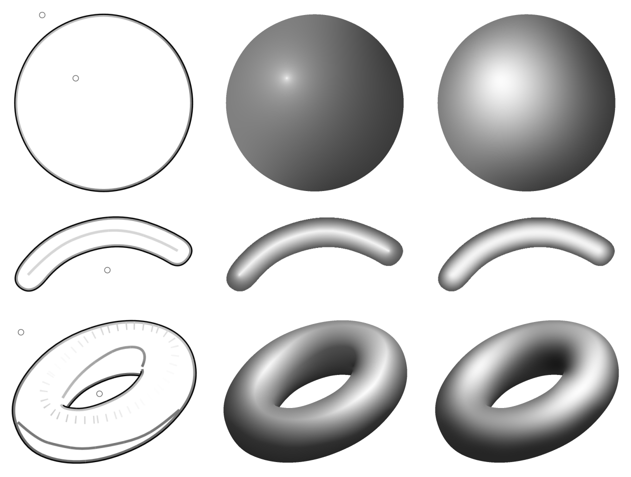


Figure 2: *When expressing 2D vector graphics using PDEs, Laplacian solutions yield derivative discontinuities at constraints, while thin-plate splines interpolate smoothly except where creases are explicitly specified. The inputs here include tear-value (solid gray with superimposed black), value (solid gray), and slope-value (hatched gray) curves, as well as point-value constraints (circles).*

2 Related Work

Vector graphics and diffusion Sun et al. [2007] optimize *gradient meshes*, regular quadrilateral meshes comprised of parametric patches, to semi-automatically fit a vector description to an image. In *diffusion curves* (DCs), Orzan et al. [2008] constrain colors on either side of specified curves and solve Laplace's equation $\Delta u = 0$ to interpolate between them. This offers several benefits. A general curve network is more flexible and easier to manage than a regular quad mesh. DCs also represent color discontinuities explicitly, and specify interpolation more compactly than a mesh of parametric patches, at a greater but still affordable computational cost.

Bezerra et al. [2010] describe extensions to let the user guide interpolation: definition of colors in homogeneous (projective) space, and anisotropic control over the diffusion process using an orientation field obtained in a separate diffusion step. Jeschke et al. [2009] and Hnaidi et al. [2010] apply diffusion curves to the geometric modeling of terrains and displaced surfaces. The Laplacian operator makes it difficult to specify features like mountains and valleys without introducing creases. Takayama et al. [2010] generalize the Laplacian framework to diffusion surfaces for volumetric modeling.

DCs developed from earlier work that applied Laplacian diffusion to image synthesis [e.g. Bertalmio et al. 2000; Johnston 2002; Pérez et al. 2003]. A function is sought that is harmonic ($\Delta u = 0$) except at constraint points. Because the solution's Laplacian Δu is nonzero at the points themselves, value constraints yield undesirable, tent-like artifacts. The fact that smoothness can be realized by solving a similar, but higher-order, quadratic minimization problem is well-known in geometric modeling but has received little application so far for 2D vector graphics.

Motivation for TPS Fundamentally, approaches based on second-order PDEs, including DCs and Laplacian diffusion, (1) cannot obtain *smooth* local minima/maxima, (2) lose derivative continuity wherever a new DC is inserted even if its color values are identical on both sides, and (3) cannot directly control interpolation derivatives. The higher-order approach of TPS addresses these limitations elegantly and seamlessly.

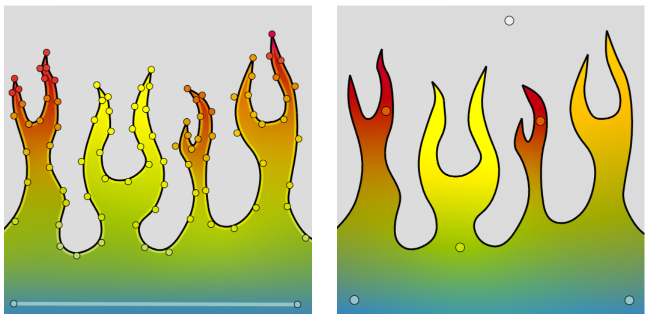


Figure 3: *Using thin-plate interpolation, a complex shape can be smoothly filled using just a few color constraints at interior points.*

Unintended discontinuities that arise in DCs can be smoothed through post-process image blurring [Orzan et al. 2008]. This requires its own set of user handles, and does not allow explicit control over the value or position of color extrema. In the sphere example (top row) of Figure 2 it is unclear how blurring can alter the single highlight point from its tent-like response with diffusion to the smooth, diffusely-shaded appearance obtained with TPS. Our method allows direct specification of the location of shading highlights with slope curves or critical points.

Bezerra et al. [2010] observe that specifying colors along entire diffusion curves can lead to "tedious and nonintuitive interactions". One example is when filling an irregular region with a smooth function. The user must evaluate the function along the boundary and store the resulting colors at all curve control points. If the boundary is edited, these colors must also be updated to reproduce the same fill function. With TPS, one need only insert a few interior constraints to smoothly extrapolate up to the boundary, regardless of its complexity. Two or three interior constraint points yield a linear gradient within any shape.[1] Additional points produce fill functions that are more complex but still smooth (Figure 3).

Biharmonic interpolation A biharmonic formulation allows smoothly interpolated constraints. Such solutions are also described as least-squares harmonic or *thin-plate splines* (see Section 3). An analytic solution exists for the special case of soft point-value constraints; in 2D it involves a global affine transformation plus a weighted sum of *radial basis functions* $\phi(r) = r^2 \log r$ in terms of the distance r around each constraint point [Duchon 1977; Wahba 1990].

Georgiev [2004] designs the Healing Brush in Photoshop using biharmonic interpolation ($\Delta^2 u = 0$) to smoothly fill a selected image region based on Cauchy (fixed value and derivative) boundary conditions. This avoids boundary derivative discontinuities but supports only this one type of boundary condition.

In geometric modeling work, Welch and Witkin [1992] introduce an objective based on thin-plate splines for variational surface modeling. Sorkine and Cohen-Or [2004] reconstruct triangle meshes from their connectivity and a set of sparse point constraints by minimizing the least-squares Laplacian. Botsch and Kobbelt [2004] use the thin-plate spline functional as well as higher-order curvature constraints on the domain boundary. Constraints similar to our value and crease-value curves can be specified in the interior. Joshi and Carr [2008] adapt this approach to automatically inflate 2D vector art with 3D geometry. Fisher et al. [2007] create smooth vector fields on meshes by minimizing a weighted least-squares harmonic subject to constraints. Jacobson et al. [2010]

[1]The TPS objective term, $T[u]$ from Section 3, yields 0 (minimum) energy for any global affine function $u(x, y) = ax + by + c$.

explore general boundary and derivative constraints in biharmonic functions on FEM meshes.

Our approach differs from applications of thin-plate splines in geometric modeling in that we discretize over a regular pixel grid rather than an irregular mesh. Like Terzopoulos [1988], we remove discretized first- and second-derivative penalties near specified vector curves to allow for tears and creases. We add new constraints on first derivatives along or across curves (contours and slopes).

3 Overview

We let the user draw a variety of features, geometrically represented by isolated points or quadratic B-spline curves. Depending on its attributes, each feature (1) reduces continuity of the solution, and/or (2) constrains its values and derivatives. In the absence of features, our objective is based on a thin-plate spline (TPS).

We first define this objective in the continuous domain. Given a continuous and differentiable bivariate scalar function $u(x, y)$, and its gradient $\nabla u = (u_x, u_y)$, the Laplacian and bilaplacian differential operators are defined via

$$\begin{aligned} \Delta u &= \nabla \cdot \nabla u \;=\; u_{xx} + u_{yy}, \\ \Delta^2 u &= \Delta(\Delta u) \;=\; u_{xxxx} + 2\,u_{xxyy} + u_{yyyy}. \end{aligned}$$

Solutions that satisfy the PDEs $\Delta u = 0$ and $\Delta^2 u = 0$ are called harmonic and biharmonic respectively. Using the calculus of variations, these are obtained as minimizers of two respective objectives:

$$u^* = \arg\min_u \iint (\nabla u \cdot \nabla u)\, dx\, dy \quad \Rightarrow \quad \Delta u^* = 0, \;\; \text{and}$$

$$u^* = \arg\min_u \iint \; T[u]\, dx\, dy \quad \Rightarrow \quad \Delta^2 u^* = 0, \;\; \text{where}$$

$$T[u] = (u_{xx})^2 + 2\,(u_{xy})^2 + (u_{yy})^2,$$

within a 2D domain with suitable boundary conditions (e.g., Dirichlet for the first, Cauchy for the second). The *membrane* (Laplacian) functional satisfies the first equation while the *thin-plate spline* satisfies the second. The same TPS solution also results from a different objective formulation [Botsch and Sorkine 2008]:

$$\widetilde{T}[u] = (\Delta u)^2 = (u_{xx} + u_{yy})^2,$$

which justifies referring to TPS as "least-squares harmonic".

The continuous TPS objective can be discretized directly via

$$\min_u \sum_{i,j} (D^2_{xx}[u])_{i,j} + 2(D^2_{xy}[u])_{i,j} + (D^2_{yy}[u])_{i,j}$$

where i indexes pixel columns (x position) and j indexes rows (y position). The necessary discrete kernels are shown in Figure 4. For example, $D_{xx}[u]$ corresponds to an (unscaled) central-difference approximation of the second derivative u_{xx}:

$$(D_{xx}[u])_{i,j} = -u_{i-1,j} + 2\,u_{i,j} - u_{i+1,j}.$$

The TPS minimum is reached by setting the partial derivative with respect to each pixel $u_{i,j}$ (shown in red) to zero, resulting in the discretized biharmonic condition $\Delta^2 u = 0$.

The membrane function is similarly discretized in terms of first-order kernels:

$$\min_u \sum_{i,j} (D^2_x[u])_{i,j} + (D^2_y[u])_{i,j}.$$

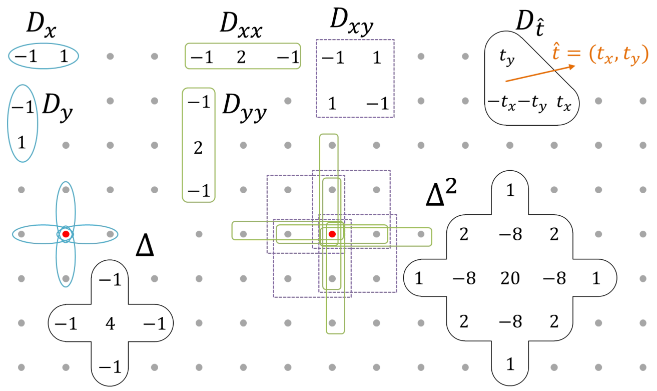


Figure 4: *Discrete kernels for least-squares smoothness penalties. Away from constraints, first-order differences (left) result in a Laplacian kernel at each pixel, and second-order differences (middle) result in a bilaplacian kernel. For anisotropic control, we introduce a generalized first-order difference (upper-right).*

The minimum of this objective by itself yields the discretized harmonic condition, $\Delta u = 0$.

One of our contributions is to introduce a generalized first-difference kernel, $D_{\hat{t}}$, to control the interpolation derivative in a given, non-axis-aligned, unit-length direction $\hat{t} = (\hat{t}_x, \hat{t}_y)$. The continuous operator represents directional derivative $\nabla u \cdot \hat{t}$. The corresponding discrete operator $D_{\hat{t}}[u]$, shown in Figures 4 and 9b, involves a triplet of pixel values:

$$(D_{\hat{t}}[u])_{i,j} = \hat{t}_x\,(u_{i+1,j} - u_{i,j}) + \hat{t}_y\,(u_{i,j+1} - u_{i,j}).$$

The squared objective, $D^2_{\hat{t}}$, thus penalizes a nonzero first-derivative in the direction $\hat{t}$, and leaves *unconstrained* the derivative in the orthogonal direction $\hat{t}^\perp = (-\hat{t}_y, \hat{t}_x)$.

Finally let $P = \{(x_k, y_k, v_k)\}$ denote the set of value constraints, where we desire $u(x_k, y_k)$ to attain the value v_k.

Combining these objectives as well as a diagonal regularization based on the sum of squares of all pixel values $u_{i,j}$, we minimize the following linear least-squares function:

$$\begin{aligned} E(u) = \;& w_2 \left(\sum (D_{xx})^2 + 2 \sum (D_{xy})^2 + \sum (D_{yy})^2 \right) + \\ & w_1 \left(\sum (D_x)^2 + \sum (D_y)^2 \right) + w_0 \|u\|^2 + \\ & w_{\hat{t}} \left(\sum (D_{\hat{t}})^2 \right) + w_p \left(\sum (u(x_k, y_k) - v_k)^2 \right) \\ = \;& \|Lu - c\|^2 \end{aligned} \tag{1}$$

where L is a sparse rectangular matrix. Its coefficients are spatially uniform and based on the thin-plate kernel, except where constraint or discontinuity features add, omit, or reweight individual terms as will be described in Section 4. The two terms with weights w_1, w_0 are regularizing functions as discussed in Section 5. Minimizing the resulting quadratic energy is equivalent to solving the symmetric linear system $L^T L u = L^T c$, denoted $Au = b$.

We use traditional raster-scan pixel ordering. The matrix A is sparse with at most 13 nonzero coefficients per row, compared to 5 for a Laplacian system. It is also block pentadiagonal with a span of $\pm 2n_x$ nonzero elements away from the diagonal, where n_x is the image resolution in x. This compares to block tridiagonal and a diagonal span of $\pm n_x$ for the Laplacian case.

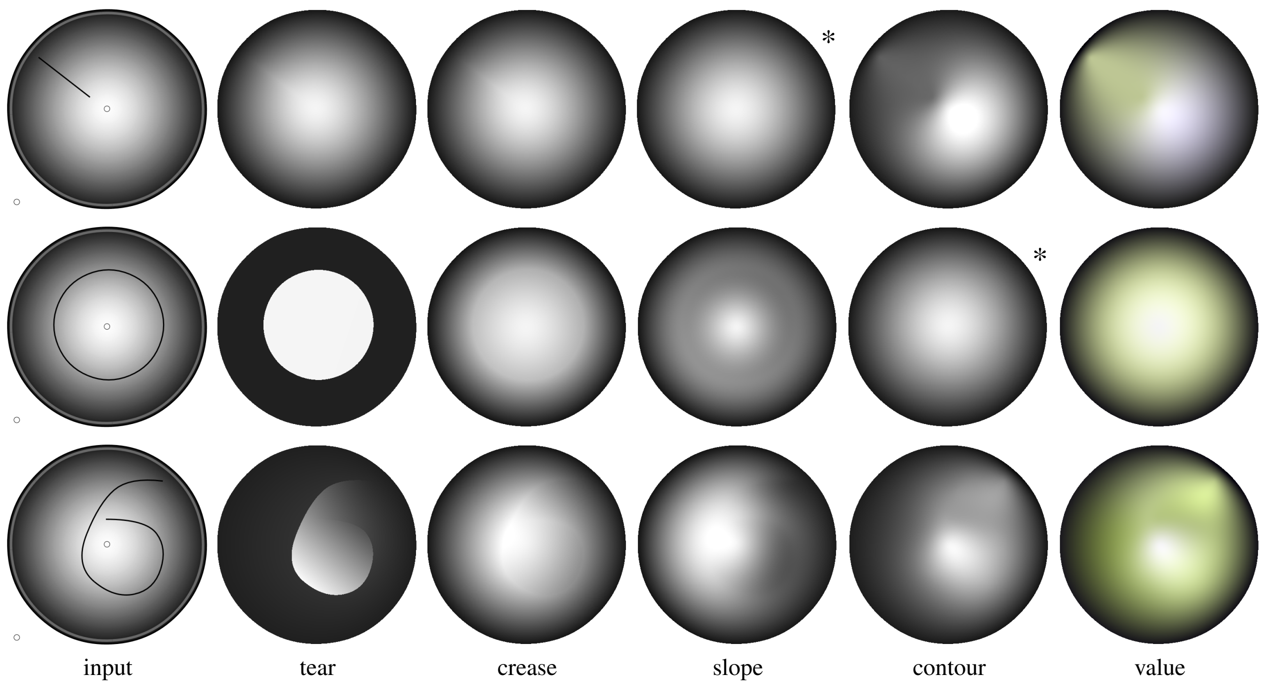


Figure 5: *Basic feature examples. Each row introduces a differently-shaped curve: a radial segment (top), a concentric circle (middle), and a "6"-shape (bottom). Columns show the effect of changing this curve's basic type. Without any curve, the background is a circularly symmetric interpolation between a bright central point and a dark outer rim. The two results marked by an asterisk (radial slope and circular contour) leave the solution unchanged, as expected. The value curve (rightmost column) is colored green to isolate its effect.*

Point type	Implementation
point value	add a bilinear value constraint on the 4 nearest pixels
critical point	add D_x, D_y penalties on edges between 4 nearest pixels
Curve type	**Implementation**
tear (T)	remove D_{xx},D_{xy},D_{yy} penalties that straddle the curve
crease (C)	same, plus add $D_{\hat{t}^\perp}$ penalties like (S) below
slope (S)	add $D_{\hat{t}^\perp}$ penalties in the curve normal direction $\hat{t}^\perp$
contour (N)	add $D_{\hat{t}}$ penalties in the curve tangent direction $\hat{t}$
value (V)	add linear value constraints on all grid edge crossings

Figure 6: *Basic features and their effects in the discretization.*

4 Features

We next describe how various features modify the objective function. As will be discussed later, minimization is computed with a coarse-to-fine hierarchical solver, so features are discretized separately into the grid at each resolution level.

4.1 Basic Features

Basic features consist of point values, critical points, and five curve types summarized in Figure 6 and shown in Figure 5. The value curve constrains color along its entire length, leaving thin-plate smoothness undisturbed. Tear and crease curves introduce discontinuities by removing smoothness penalties straddling the curve. The crease differs from a tear in that it preserves C^0 continuity. Contour and slope curves zero the first derivatives in the curve tangent or normal direction respectively. Intuitively, a contour is a curve of constant value, while a slope is a curve of steepest ascent (i.e. an integral curve of the gradient) and remains perpendicular to contours.

Figure 7 shows an example where a slope curve is used to precisely locate the solution's "ridge line" maxima, corresponding to a shading highlight. Similarly, Figure 8 shows how a contour curve reshapes the interpolating function, by controlling its gradient direction without specifying additional values.

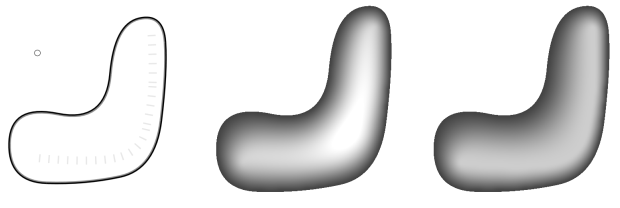


Figure 7: *A slope-value curve forces the solution to interpolate with a zero cross-curve derivative. Here it explicitly specifies the position and value of intensity maxima in a shading highlight.*

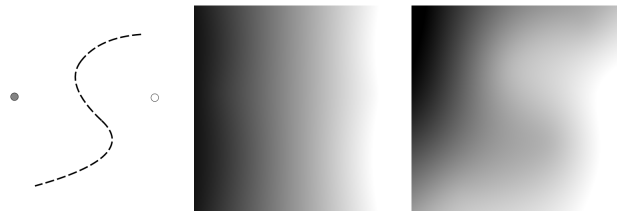
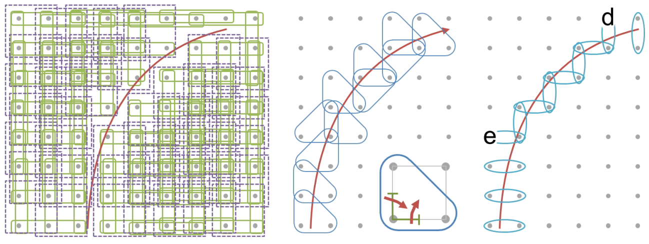


Figure 8: *A contour curve shapes the interpolated solution by constraining it to be constant along its extent.*

Rasterization Point and curve features are "rasterized" into the pixel grid as follows. Point value constraints use bilinear weights on the four nearest pixels, yielding an objective term of the form

$$(w_{0,0}\, u_{i,j} + w_{1,0}\, u_{i+1,j} + w_{0,1}\, u_{i,j+1} + w_{1,1}\, u_{i+1,j+1} - v)^2.$$



(a) tear curve (b) crease curve (c) [Terzopoulos]

Figure 9: *Tear and crease curves omit straddling C^1 smoothness terms. Creases reintroduce C^0 continuity across the curve. Our method uses generalized derivative kernels (b) rather than axis-aligned membrane kernels (c) [Terzopoulos 1988]. Red path vectors in the inset in (b) indicate two possible curves for which this particular orientation of the triangular $D_{\hat{t}^\perp}$ kernel is instanced: the curve must enter the triangle closest to its right-angle vertex.*

(a) input (b) axis-aligned kernels (c) generalized kernel

Figure 10: *Crease comparison. Axis-aligned kernels (b) cause unwanted smoothing along this diagonal crease. We overcome this problem by using a generalized kernel (c).*

Critical points constrain the values of the four nearest pixels to all be equal using four least-squares edge constraints.

Curves are rasterized by computing their intersections with pixel grid edges and inserting new constraint penalties or adjusting smoothness penalties there. Such intersections are easily computed for quadratic B-splines by solving a quadratic equation. We now describe how each curve type is implemented.

Value curves introduce an affine constraint on the two nearest pixels at each grid intersection. Tear curves remove D_{xx}, D_{yy}, D_{xy} smoothness kernels as shown in Figure 9a. When a tear curve is adjacent to a point or curve value constraint, it also omits from the value penalty term any pixels on the opposite side of the tear.

Crease curves remove second-derivative smoothness kernels like tears do, but recover C^0 continuity by adding a set of $D_{\hat{t}^\perp}$ generalized first-difference kernels across the curve, as shown in Figure 9b. The kernel direction $\hat{t}^\perp$ is given by the normal vector to the curve obtained at each grid intersection. The inset in (b) shows how the $D_{\hat{t}^\perp}$ kernel is formed in a pixel triplet when the curve passes through either of the two half-edges nearest its corner pixel.

Generalized $D_{\hat{t}^\perp}$ kernels constrain crease derivatives more precisely than axis-aligned membrane kernels D_x, D_y as in [Terzopoulos 1988]. In a diagonal crease curve for example, axis-aligned kernels transitively constrain distant pixels (e.g., d and e in Figure 9c). This needlessly removes intended variation *along* the crease as shown in Figure 10b.

A slope curve introduces the same set of $D_{\hat{t}^\perp}$ kernels as the crease curve, but does not remove the thin-plate smoothness penalties. A contour curve also involves the same kernel triplets, but assigns kernel weights based on the tangent rather than normal direction. As with value curves, tears remove straddling kernels from adjacent slope or contour curves.

4.2 Compound Features

Compound curves provide more complex behavior from a single user-specified curve. They define two or more intermediate curves, identical to or offset from the original one specified and of possibly different basic types. The offset distance τ can be infinitesimally small to juxtapose the resulting curves, or some desired thickness that widens the feature's effect. Figure 11 shows examples of some of the types we have found useful. Our notation uses a sequence of basic-type letters (VTCNS), with a comma between two successive curves if they are offset by a non-infinitesimal distance. Even for juxtaposed curves, the ordering may matter; e.g., TV is different from VT and puts the tear on the left or right side of the feature curve respectively, as determined by the curve's direction. Section 6 discusses how various curve types are used in examples.

A crease-value (CV) compound introduces a value curve and a superimposed crease, both implemented as described previously. Recall that a crease involves an implicit tear which removes straddling D_{xx}, D_{xy}, D_{yy} smoothness terms. This implicit tear also interrupts any straddling kernels from nearby value curves. The one exception is that it does not affect the CV compound's own value curve itself. This yields a symmetric definition; there is no difference between CV and VC curves.

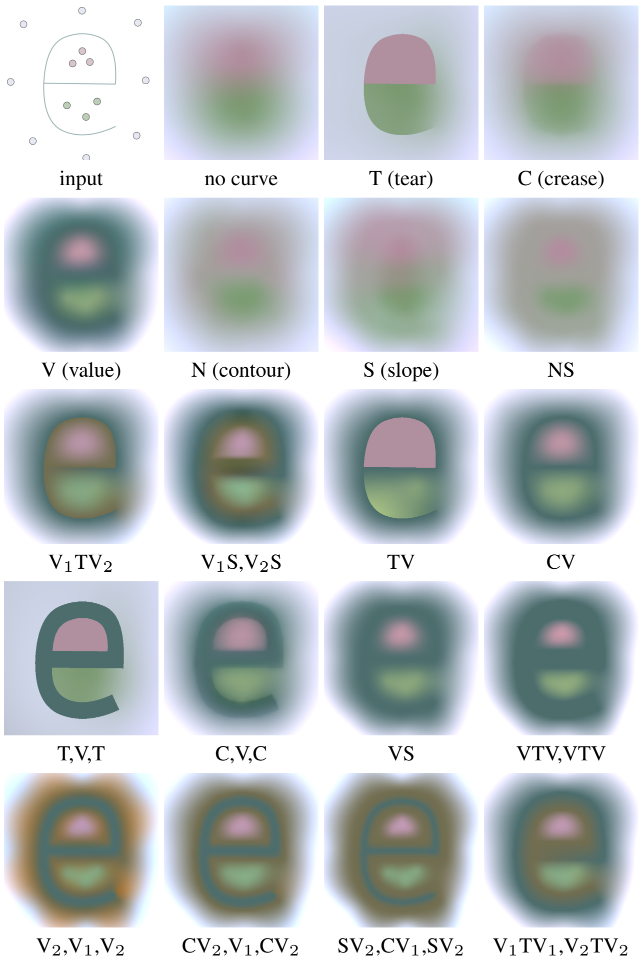


Figure 11: *Examples of compound features, in which a single vector primitive defines multiple offset curves of various basic types to realize more complex effects. See also Figure 13.*

We note that a two-sided value curve (V_1TV_2) has similar local behavior to a diffusion curve [Orzan et al. 2008]. A single-sided value curve (TV) is similar to that in [Bezerra et al. 2010].

We compute offsets of B-spline curves by offsetting their control points along the curve normal direction. Our system also supports mitering and beveling at curve joints and a variety of end caps (flat, rounded, pointed, and open).

5 Solution Method

The linear system $Au = b$ is symmetric and banded. Many solution approaches are possible. For small images, we use a direct solver that computes the banded-diagonal Cholesky decomposition (CD) of matrix A. Specifically, we use the `dpbtrf()` LAPACK function in the Intel MKL library, which has been optimized to exploit OpenMP multithreading and SSE vectorization.

For larger images, the direct solver becomes too slow for interactive use. We invoke it on a coarser grid, and use a coarse-to-fine hierarchical strategy [Botsch et al. 2005] to upsample and relax the solution. This same strategy is also needed to compute a solution on a zoomed-in viewport.

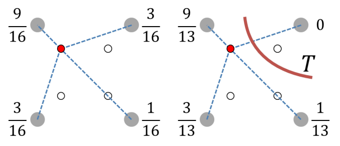
Our solutions are based on 4-channel color vectors u and b. Displayable values are in the range from 0 to 1. Elements of the system matrix A are scalars.

Regularization If the set of user-provided features does not include a sufficient number of value constraints (e.g. at least 3 value constraints for each interior region bounded by a tear curve), the solution to the thin-plate objective is not unique and matrix A is only positive semi-definite. Various regularizing functions can be introduced to obtain a positive-definite system. Following Equation (1), we add a small weight w_1 times the solution Laplacian $\|\Delta u\|^2$, plus an even smaller weight w_0 times the solution magnitude $\|u\|^2$. Consequently, in each isolated region, we obtain the exact solution $(0, 0, 0, 0)$ (i.e. transparent color) given zero constraints, and almost exactly the constant solution given a single constraint and the linear interpolant given two constraints. The small regularization weights cause the solution to deviate from this behavior far away from the constraints.

Coarse-to-fine relaxation Normally we invoke the CD direct solver on a grid at 128×128 resolution, and use three progressively finer steps to obtain a 1024×1024 image. To improve response when the user is actively editing curves, we instead compute the CD solution on a 64×64 level and upsample in four steps.

Each step upsamples the solution to the next finer level and performs Gauss-Seidel (GS) iterations on the linear system obtained from rasterizing at that level. GS iteration is terminated when the RMS change in the solution falls below a threshold of 1e-4, or at most 30 iterations.

Discontinuity-aware upsampling We biquadratically upsample in smooth areas using weights 1/16, 3/16, 3/16, 9/16 on the four nearest parent pixels. For accurate upsampling near tears, we determine if the segment between the finer-level pixel and each of its four parent pixels intersects any tear curve. If so, we omit that parent's contribution in the upsampling.



We also invoke a discontinuity-aware flood fill after upsampling which propagates defined pixels ($\alpha \neq 0$) to replace undefined neighbors ($\alpha = 0$) if they are not cut off by a tear. This improves the initialization of regions that were pinched off at coarser levels of the solution pyramid.

Viewport solution We allow continuous pan and zoom of a limited viewport and so must account for features lying outside it. In Orzan et al. [2008], the viewport boundary is assigned Dirichlet conditions taken from a coarse solution spanning a fixed canvas. Our goal is to support a virtual canvas of infinite extent. We achieve this by forming windows of progressively coarser sampling and larger extent (similar to clipmaps [Tanner et al. 1998]), until all user features are contained. In addition to initializing it by upsampling, each window's solution fixes a two-pixel band around the next-finer window's boundary. This essentially defines Cauchy boundary conditions for the bilaplacian. The coarsest window, and any finer window still containing all features, applies "natural" boundary conditions, equivalent to a tear circumscribing the entire boundary.

Scale invariance and objective weighting To obtain consistent solutions across different grid resolutions, the discrete kernel coefficients for smoothness and constraint terms must scale properly with resolution. Let h_l be the sample spacing at level l, i.e. $h_{l+1} = h_l/2$. Then the weight, w_2, of thin-plate objective components, D_{xx}, D_{xy}, and D_{yy}, are scaled by $1/h_l^2$ to account for scaling effects due to squared second derivatives and 2D integration. The Laplacian kernel (based on D_x and D_y with weight w_1) is scale-invariant since the scaling effects of squared first-derivatives and 2D integration cancel. The weight for diagonal terms, w_0, are scaled by h_l^2. Value constraints along curves, introduced at grid intersections, are scaled by h_l; derivative constraints along curves are scale-invariant. Isolated point constraints are also scale-invariant.

We use raw (before scaling) weights of w_2 = 1e-5, w_1 = 1.6e-6, and w_0 = 1.4e-7 in all our examples. For features, we use raw weights $w_{\hat{t}}$ = 5e-4 for the generalized first-derivative objective at creases, $w_{\hat{t}}$ = 2 for derivative constraints at slopes and contours, and w_p = 5e+2 for value constraints (both along curves and at isolated points). For Laplacian-only solutions in Figures 2(middle) and 3(left), we use weights w_2 = 0 and w_1 = 1e-3.

Setup and data structures At each resolution level, we first traverse through feature curves, "rasterizing" each to create two data structures. One accumulates a list of constraints from all value-curve grid intersections and isolated value points. The second stores a *discontinuity map* of 2 bits per pixel, indicating whether a tear is present in the x and y edges emanating from that pixel. We also store the min/max location of all tear curve intersections per edge.

We then traverse the list of value constraints, using discontinuity information to determine which pixels should be included in each value kernel. For value curves, we check the discontinuity map to see whether each intersected edge also intersects some tear. If so, the min/max tear location for that edge indicates which of its two pixels to exclude from the affine value constraint. Both pixels may be cut off, in which case we omit the entire kernel. For value points, we check whether each segment joining the point to its four surrounding pixels is intersected by a tear curve. If so, we eliminate that pixel from the bilinear objective term and reweight the remaining pixels, as in discontinuity-aware upsampling.

We accumulate values (for vector b) and kernel weights (for matrix A) in a sparse pixel array called the *constraint value map*. From the discontinuity map and constraint value map, we can then compute a sparse representation of the matrix A. We exploit the problem's regular, 2D structure by avoiding indices and simply storing an $n \times 13$ matrix, since at most 13 pixels at known locations have nonzero coefficients in any row of A.

Name	Fig.	#P	#C	#CP	Fast (ms)		Slow (ms)	
					setup	total	setup	total
cup	1	1	19	283	13.1	59.5	61.8	263.8
hat	1	4	11	128	13.8	35.2	56.8	208.9
duck	1	4	39	449	14.5	61.0	65.7	287.1
flames	3	9	1	118	14.6	61.0	51.6	246.1
gargoyle	12	3	62	562	12.6	34.0	58.8	225.1
terrain	14	2	8	97	11.0	32.3	52.3	197.7
puppy	16	22	26	513	17.4	38.7	64.9	228.8
girl	16	28	47	658	27.4	61.4	100.8	339.6
French	16	5	53	630	10.4	31.8	61.7	241.6
rose	16	0	57	447	16.9	63.2	70.4	268.5
portrait	16	1	60	541	16.5	54.7	70.0	287.5
fire	16	0	70	1299	20.7	58.7	108.8	389.8
Mark	17	6	62	557	18.0	58.7	81.6	325.2
John	17	5	75	698	16.4	54.7	72.5	311.3
Hugues	17	8	73	662	16.3	54.4	73.5	340.6

Table 1: *Statistics for examples. #P = number of isolated point constraints, #C = number of curves (compounds are counted once), #CP = number of curve control points.*

Performance optimization TPS problems have larger condition numbers than ones based on Laplacian diffusion. A robust computation requires double-precision arithmetic to accumulate the system matrix A and compute the linear system solution with CD. The vector b may be accumulated in single precision with no effect on system stability. Computing relatively few iterations of GS in single precision is robust and allows greater SSE acceleration. For pixels away from constraints, the GS update is based on the fixed biharmonic kernel (Δ^2 in Figure 4), and we optimize this case.

Alternative solutions A different approach would be to create an adaptive 2D triangulation around discretized feature curves, and minimize the objective over the resulting irregular mesh. Our choice to discretize over a regular grid simplifies parallelization (OpenMP) and vectorization (SSE). Grid rasterization is extremely fast and avoids having to construct a constrained triangulation at interactive rates. A regular grid also makes implicit all pixel indices and neighbor relations, and yields a banded linear system with a small, fixed number of nonzero off-diagonal entries.

6 Results and Discussion

Examples (Figures 1, 12, 14, 16, and 17) and comparisons (Figures 2, 3, 5, 7, 8, 10, and 11) were recorded directly from our prototype system. Interactive sequences are included in an accompanying video. Example pairs show user input on the left and solution output on the right. Input images indicate point-value constraints as colored circles. Basic value curves are drawn in the solid color of their value, tears in solid black, creases in dotted black, and contours and slopes with dashed black lines, oriented along or perpendicular to the curve respectively. Slope-value, contour-value, and crease-value curves are drawn using dashed/dotted curves as in the corresponding basic type but in the value color rather than black. Other compound curves are drawn using their constituent offset curves. These offsets are all derived from a single user-drawn curve as discussed in Section 4.2.

As the examples demonstrate, our system provides rich but controllable results from a compact user specification. The number of isolated points (#P), curves (#C), and curve control points (#CP) for each example are documented in Table 1 and serve as a rough measure of input complexity.

Our benchmark machine is a dual quad-core Intel Xeon (E5640, 2.67GHz). Images and video were generated using four multiresolution steps using a base-level CD solution of resolution 128×128. Final image resolution was 1024×1024.

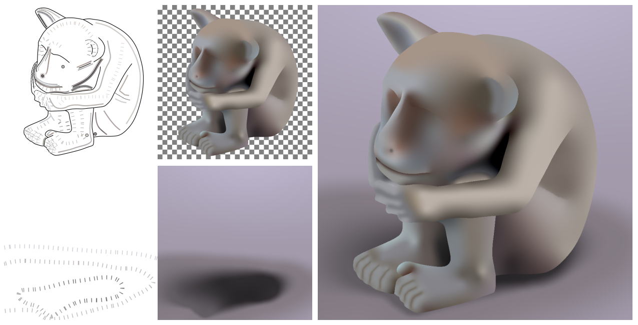
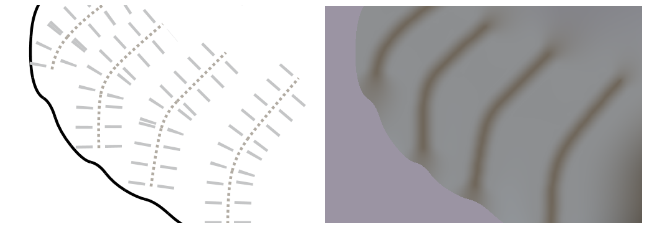
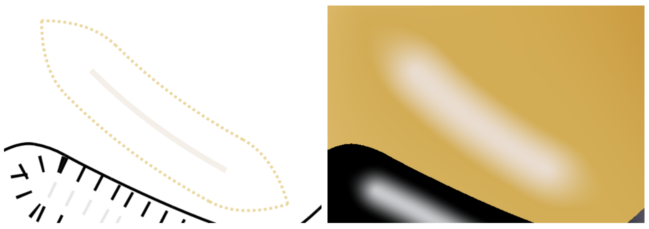


Figure 12: *Example with compositing for the drop shadow. The checkerboard indicates a transparent region, with color (0,0,0,0).*



Set of (SV_2, CV_1, SV_2) compound curves used to create indentations.



Single (CV_2, V_1, CV_2) compound curve used to form a highlight.

Figure 13: *Example close-ups demonstrating some practical uses of compound curves (from the toes of the "gargoyle" in Figure 12 and the head of the "duck" in Figure 1).*

The system achieves interactive performance as shown in Table 1.[2] The table reports total time as well as separately breaking out time for setup (curve rasterization, computation of discontinuity and constraint value maps, and assembly of the system matrix). The time difference is spent on solution relaxation (CD/GS as well as coarse-to-fine upsampling). As discussed in Section 5, our system uses two solution modes: a faster one used for interactions like control point dragging and image panning (based on a 64×64 base-level grid), and a slower one invoked after UI button-release (based on a 128×128 base-level grid). Performance for both modes is reported in the table.

The "gargoyle" example in Figure 12 separates the shadow into its own image layer. Ignoring the shadow caster lets us define the shadow using just a few closed slope-value curves. Because the gargoyle object layer is bounded by a tear and there are no external constraints, the solution automatically becomes transparent outside the tear boundary.

[2]One source of timing variance in the examples is whether its features are completely contained inside the viewport. As explained in Section 5, features that lie outside the viewport require additional coarser-level CD iterations to initialize boundary conditions. Examples needing these additional CD passes were: cup:3, hat:1, duck:3, flames:3, girl:2, and rose:3.

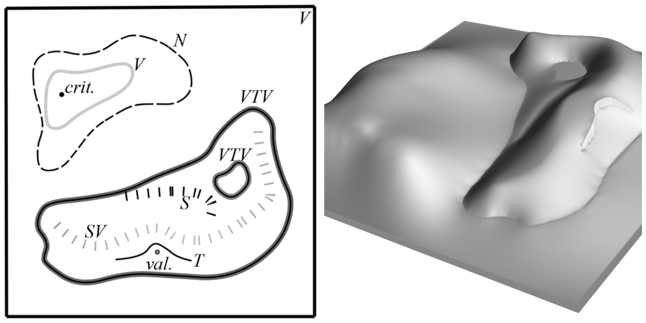


Figure 14: *Terrain authoring example. The height field is smooth except at specified value-tear-value (VTV) and tear (T) curves. The slope-value (SV) and slope (S) curves precisely control the smooth ridge line and valley respectively.*

The “hat” example in Figure 1 uses offset, two-value compound curves (V_1,V_2). Three of these suffice to control shading in the main (orange/brown) region. For the video, we created another version using the same outlines, but based on point and derivative constraints with no value curves. Two contour curves specify shading above and below the green band, and one crease curve forms the crown’s crease.

In Figure 16, the “puppy” uses value points but no value curves. Shading is controlled using contour curves, especially noticeable in the dog’s face and right side of his body. The “girl” uses pairs of VS (value slope) curves to obtain color variation in the hair. The “French” example attempts an ethereal effect using basic creases curves without any hard outlines. Note that such naked creases (without values) are a new contribution of our work. Color is specified sparsely with just four V_1,N,V_2 compound curves (value, contour, value) and five value points. The “rose” uses tear-value, crease-value, and slope-value compound curves. Note that a crease-value is similar to a diffusion curve and gives the user the option of removing smoothness over a value curve if desired. The “portrait” example produces subtle shading in the cheeks and nose primarily using slope-value curves. Finally, the “fire” example attempts to reproduce Figure 11(left) in [Bezerra et al. 2010]. Our result is produced from a single PDE solution rather than two successive diffusion computations.

A more realistic look is targeted in the “cup”, “duck”, and “gargoyle” examples which are also based mainly on value and slope-value curves. The “duck” adds contour curves to control shading. Both the “duck” and “gargoyle” use additional compound curves to introduce highlights or shadows, as shown in Figure 13. Note the smoothness of shading in these examples, especially at value curves such as across the body of the cup in Figure 1.

The terrain example in Figure 14 is rendered as height-field geometry rather than an image. It uses tears, contours, slopes, and a critical-point mountain peak. A flat lake is bounded by a closed VTV compound, which behaves much like a crease-value (CV) curve. A slope-value curve in the bottom-right hill forms a ridge line while a contour guides the upper left hill’s cross-section. A value curve is also used to shape this hill without introducing an unwanted crease.

Numerical accuracy We tested the accuracy of our coarse-to-fine solution approach by varying the resolution level at which we invoked the initial low-resolution CD solver. Results are shown in Figure 15. The reference solution applied CD directly at the final resolution of 512×512 without coarse-to-fine relaxation.

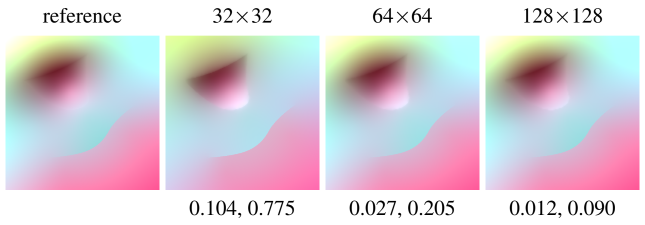


Figure 15: *Numerical accuracy obtained when starting the hierarchical solution at different resolutions. Error numbers are RMS and maximum differences over all pixels, relative to the reference solution.*

We also decreased the termination threshold for GS iteration (used for relaxation at finer resolution levels) from 10^{-4} to 10^{-5} and 10^{-6}, and removed the limit on iterations, holding all other parameters constant. This yielded negligible improvement in the resulting images and quantitative error numbers. Applying a few iterations of GS rapidly relaxes the upsampled estimate whereas further iteration converges too slowly to be worthwhile in an interactive system.

Limitations A single coarse-to-fine pass provides sufficient accuracy in most situations. Topology changes due to rasterization at a low-resolution base level occasionally hamper convergence at finer levels. A particularly bad case is two large regions linked by a narrow neck, e.g. a dumbbell. Overly coarse rasterization that pinches off this neck will incorrectly isolate the two ends. This is a limitation on our current solver, not the overall TPS framework.

Unlike the Laplacian solution, the thin-plate spline does not possess the maximum property and instead extrapolates beyond specified value constraints. This is often natural and convenient, as shown in Figure 3, but can also overshoot to produce out-of-range colors. We simply clamp these back to the displayable range. Such problems are most evident with closely-spaced value point pairs or nearly collinear triples which can create large derivatives. Similar issues arise in surface modeling with interpolated control points; they are easily corrected by adjusting their location. The spatial extent of extrapolation can also be controlled with simple tears, additional value curves, or curves that induce zero derivatives such as creases, slopes, and contours. We did not find extrapolation to be a practical difficulty in our examples.

7 Summary and Future Work

We propose a new image synthesis method that minimizes a least-squares objective on pixel values, combining thin-plate smoothness, explicit discontinuity control, and constraints on values and derivatives. Compared to previous work in variational vector graphics, our higher-order notion of smoothness produces a more natural, fuller-looking result and supports more powerful and intuitive editing controls. Our method inputs a unified network of points and curves which interact in just one PDE solution, without additional relaxation or blurring passes.

Our implementation can easily be generalized to handle other types of curve geometry, offset types, and nonzero derivative constraints. Computing on the GPU may accelerate our system. Antialiasing, especially at tears, is a simple extension that can be computed from our discontinuity map. Solutions can also be computed in spaces other than (r, g, b) color per pixel, with possible application to domains such as the authoring of control fields for texture synthesis, programmable shading, and procedural modeling.

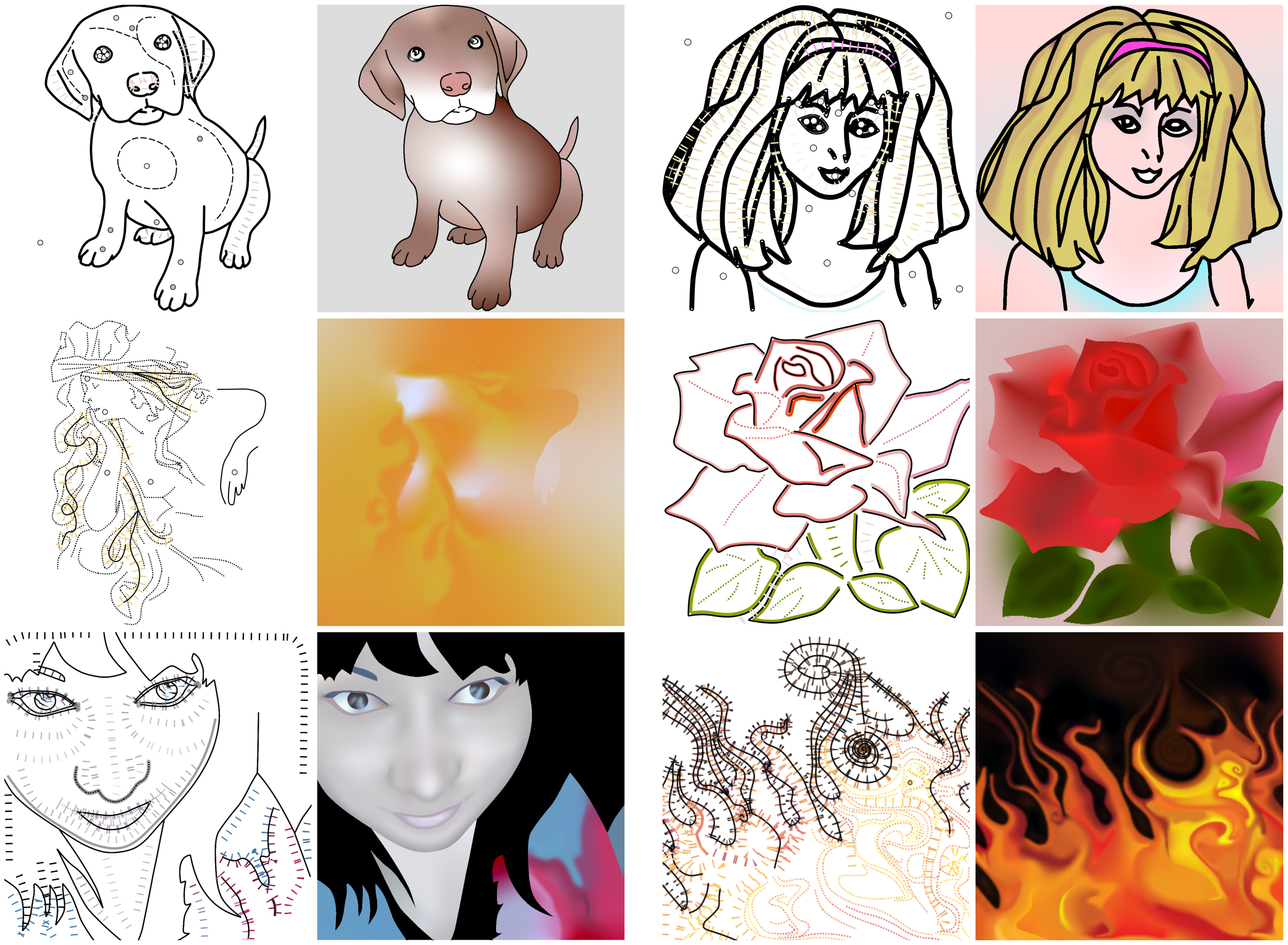


Figure 16: *More examples using a variety of graphic styles.*

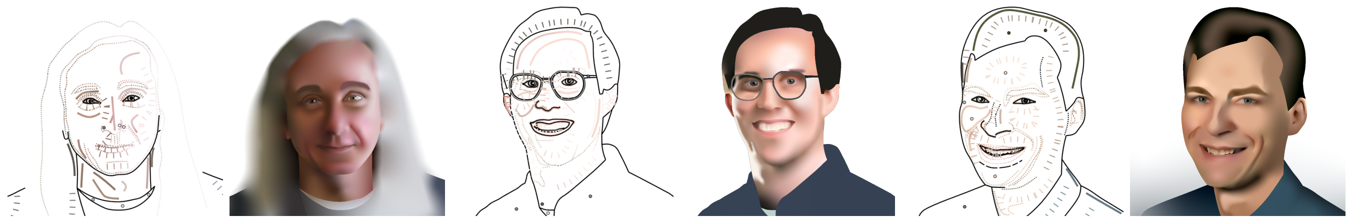


Figure 17: *The authors.*

Several avenues remain for longer-term future work. We have not yet applied our method to automatic vectorization of given images as in [Orzan et al. 2008; Jeschke et al. 2011]. Many techniques accelerate the solution of Laplacian systems and might be beneficially applied to our more general objective formulation, including locally-adapted hierarchical basis preconditioning [Szeliski 2006] and adaptive discretization [Agarwala 2007]. A hard-constraint solver could be substituted for our unconstrained minimization to enforce constraints exactly. It may be possible to extend the smoothness objectives to account for subpixel curve positioning, so that the solution changes continuously as features move over the pixel grid. Our smoothness objective and constraints could be extended to even higher-order derivatives, perhaps based on curvature [Moreton and Sequin 1992; Botsch and Kobbelt 2004]. Finally, our approach could be extended to 3D volumes [Takayama et al. 2010].

Acknowledgments

We manually traced curves over public-domain art in `http://www.openclipart.org/tags/line_art` for the "hat", "puppy", "girl", "French", and "rose" examples. We thank Andrea Finch for capturing source photos that inspired other examples, and Eric Stollnitz for implementation pointers.

References

AGARWALA, A. 2007. Efficient gradient-domain compositing using quadtrees. *ACM Trans. Graphics*, 26(3).

BERTALMIO, M., SAPIRO, G., CASELLES, V., and BALLESTER, C. 2000. Image inpainting. *ACM SIGGRAPH Proceedings*.

BEZERRA, H., EISEMANN, E., DECARLO, D., and THOLLOT, J. 2010. Diffusion constraints for vector graphics. In *Proceedings of NPAR*.

BOOKSTEIN, F. 1989. Principal warps: Thin-plate splines and the decomposition of deformations. *IEEE Trans. on Pattern Anal. Mach. Intell.*, 11(6).

BOTSCH, M., BOMMES, D., and KOBBELT, L. 2005. Efficient linear system solvers for mesh processing. In *Mathematics of Surfaces XI*, volume 3604 of *LNCS*, pages 62–83. Springer Verlag.

BOTSCH, M. and KOBBELT, L. 2004. An intuitive framework for real-time freeform modeling. *ACM Trans. Graphics*, 23(3).

BOTSCH, M. and SORKINE, O. 2008. On linear variational surface deformation methods. *IEEE Trans. on Visualization and Computer Graphics*, 14(1).

COURANT, R. and HILBERT, D. 1953. *Methods of Mathematical Physics, Vol I.* London: Interscience.

DUCHON, J. 1977. Splines minimizing roation-invariant semi-norms in Sobolev spaces. In W. Schempp and K. Zeller, editors, *Constructive Theory of Functions of Several Variables*, pages 85–100. Springer.

FISHER, M., SCHRÖDER, P., DESBRUN, M., and HOPPE, H. 2007. Design of tangent vector fields. *ACM Trans. Graphics*, 26(3).

GEORGIEV, T. 2004. Photoshop Healing Brush: a tool for seamless cloning. In *Proc. of ECCV*.

HNAIDI, H., GUÉRIN, E., AKKOUCHE, S., PEYTAVIE, A., and GALIN, E. 2010. Feature based terrain generation using diffusion equation. *Computer Graphics Forum*, 29(7).

JACOBSON, A., TOSUN, E., SORKINE, O., and ZORIN, D. 2010. Mixed finite elements for variational surface modeling. *Computer Graphics Forum*, 29(5):1565–1574.

JESCHKE, S., CLINE, D., and WONKA, P. 2009. Rendering surface details with diffusion curves. *ACM Trans. Graphics*, 28 (5).

JESCHKE, S., CLINE, D., and WONKA, P. 2011. Estimating color and texture parameters for vector graphics. *Computer Graphics Forum*, 30(2):523–532.

JOHNSTON, S. 2002. Lumo: Illumination for cel animation. In *Proceedings of NPAR*.

JOSHI, P. and CARR, N. 2008. Repoussé: Automatic inflation of 2D art. In *Eurographics Workshop on Sketch-based Modeling*.

MORETON, H. and SEQUIN, C. 1992. Functional optimization for fair surface design. *ACM SIGGRAPH Proceedings*.

ORZAN, A., BOUSSEAU, A., WINNEMÖLLER, H., BARLA, P., THOLLOT, J., and SALESIN, D. 2008. Diffusion curves: a vector representation for smooth-shaded images. *ACM Trans. Graphics*, 27(3).

PÉREZ, P., GANGNET, M., and BLAKE, A. 2003. Poisson image editing. *ACM Trans. on Graphics*, 22(3).

SORKINE, O. and COHEN-OR, D. 2004. Least-squares meshes. In *Proc. of Shape Modeling International*.

SUN, J., LIANG, L., WEN, F., and SHUM, H.-Y. 2007. Image vectorization using optimized gradient meshes. *ACM Trans. Graphics*, 26(3).

SZELISKI, R. 2006. Locally adapted hierarchical basis preconditioning. *ACM Trans. Graphics*, 25(3).

TAKAYAMA, K., SORKINE, O., NEALEN, A., and IGARASHI, T. 2010. Volumetric modeling with diffusion surfaces. *ACM Trans. Graphics*, 29(6).

TANNER, C., MIGDAL, C., and JONES, M. 1998. The clipmap: a virtual mipmap. *ACM SIGGRAPH Proceedings*.

TERZOPOULOS, D. 1983. Multilevel computational processes for visual surface reconstruction. *Computer Vision, Graphics, and Image Processing*, 24(1).

TERZOPOULOS, D. 1988. The computation of visible-surface representations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 10(4).

WAHBA, G. 1990. *Spline Models for Observational Data*, volume 59 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. SIAM, Philadelphia.

WELCH, W. and WITKIN, A. 1992. Variational surface modeling. *ACM SIGGRAPH Proceedings*.