A Vectorial Solver for Free-form Vector Gradient

Simon Boyé, Pascal Barla, Gael Guennebaud

A Vectorial Solver for Free-form Vector Gradients

Simon Boyé * Pascal Barla Gaël Guennebaud
Inria - U. Bordeaux - IOGS - CNRS

Figure 1: ***Free-form vector gradients:*** *Our vectorial solver permits to create complex vector gradients (a). Our method is based on a triangular representation (b) that is output-insensitive and thus works with arbitrary high image resolutions. Our solver does not need to be updated for a variety of operations: (c) instancing, layering and deformation; (d) texture mapping; and even (e) environment mapping.*

Abstract

The creation of free-form vector drawings as been greatly improved in recent years with techniques based on harmonic or bi-harmonic interpolation. Such methods offer the best trade-off between sparsity (keeping the number of control points small) and expressivity (achieving complex shapes and gradients). Unfortunately, the lack of a robust and versatile method to compute such images still limits their use in real-world applications. In this paper, we introduce a *vectorial* solver for the computation of free-form vector gradients. Based on Finite Element Methods (FEM), its key feature is to output a low-level vector representation suitable for very fast GPU accelerated rasterization and close-form evaluation. This intermediate representation is hidden from the user: it is dynamically updated using FEM during drawing when control points are edited. Since it is output-insensitive, our approach enables novel possibilities for (bi)-harmonic vector drawings such as instancing, layering, deformation, texture and environment mapping. Finally, in this paper we also generalize and extend the set of drawing possibilities. In particular, we show how to locally control vector gradients.

Keywords: Vector graphics, diffusion curves, finite elements

Links: DL PDF

1 Introduction

Vector graphics techniques have been tremendously improved in recent years, both through research and software products. Two different types of applications have motivated this burst of innovation: image vectorization and free-form drawing. Vectorization methods usually require a dense set of control points to accurately represent all the details of an input image. With drawing tools, this is quite the contrary: a sparse set of vector primitives with rich expressivity is preferred, since it provides for more direct editing and creation. In this paper, we are primarily interested in free-form vector drawing.

*e-mail: boye@labri.fr

One of the first vector primitives that made tractable the creation of complex color gradients is the Gradient Mesh [Sun et al. 2007] (GM). It takes inspiration from geometric modeling, and lets artists specify color gradients by assigning color values and derivatives to the vertices of a 2D control mesh. With increasing mesh resolution, this technique permits to represent very complex color gradients. On the down side, the denser the mesh, the more difficult it becomes to edit it (Section 2.1).

Recent work has shown that complex color gradients can advantageously be produced through the harmonic or bi-harmonic interpolation of a few spline curves and sample points defining colors [Orzan et al. 2008; Finch et al. 2011]. Such Diffusion Curves (DC) based representations have also been successfully used for the design of 3D images, height fields or normal maps (Section 2.2).

Although DC-based representations provide an appealing alternative to GMs, computing such images involves a global differential equation which is rather challenging when seeking for real-time performance, high-quality, and robustness. Current methods strive to *rasterize* DCs directly in the regular pixel grid of the destination image using finite differences. As explained in more details in Section 2.3, such an approach is subject to aliasing when multiple curves pass through a single pixel, and zooming-in can only be approximated via expensive multiple evaluation passes thus breaking the infinite resolution property of the vector representation. Moreover, the expensive differential equation has to be fully solved not only when the curves are edited, but also for simple operations such as translation, rotation and scaling of the entire drawing. This is a major practical limitation, since it makes DCs not tractable for applications that require instancing, deformation or 3D mapping.

In this paper we address all these limitations by introducing a novel *vectorial* solver for DC images, i.e., a solver that outputs a vectorial representation. The key idea is to dynamically convert a DC image into a high-order mesh-based representation that is automatically adapted to the complexity of the input. To this end, we propose a robust solution that builds on constrained Delaunay triangulations and Finite Element Methods (FEM). Contrary to mesh-based approaches, this intermediate representation is hidden from the user, and updated only when DCs are edited; hence the approach takes the best of both worlds. A direct benefit is that a variety of common operations do not require any update of the solution, as illustrated in Figure 1a-c. Moreover, since the solution is vectorial, it may be accessed randomly, permitting novel uses of free-form vector graphics such as texture or environment mapping (Figure 1d-e).

In addition, we introduce the notion of asymmetric transmission that permits to progressively and precisely limit the influence of some primitives, thus allowing for the first time the mix of *global*

and *local* diffusion. We present this extension in Section 3, together with a unified taxonomy of curve constraints that identifies novel types of artistic controls. Our vectorial solver and intermediate representation are detailed in Section 4, and we show in Section 5 a variety of applications leveraging their benefits. Finally, we discuss limitations as well as further extensions in Section 6.

2 Previous works

The focus of this paper is on free-form vector drawing, which among other requirements demands sparse controls, interactive feedback and fast rendering. We review methods that target these goals, and may be used either for creating vector graphics from scratch, or for editing images converted to vector representations.

2.1 Mesh-based techniques

One of the earliest vectorization methods is the ArDeco system [Lecot and Lévy 2006]: it uses a mesh-based representation to convert bitmap images into linear, radial or quadratic gradients. In the same vein, GMs consist of cubic color patches arranged into a quad mesh structure [Sun et al. 2007]. More recently, Liao et al. [2012] have proposed to use triangular subdivisions surfaces to relax the constraints imposed by the regular structure of GMs. However, all these representations are rather difficult to create from scratch or even to manipulate since they expose to the user a lot of control points, and require to explicitly deal with the underlying mesh structure. This is why such methods have been often confined to automatic conversion.

Our approach also makes use of a triangular mesh, but it serves as an intermediate representation that is automatically generated and hidden from users.

2.2 Diffusion Curves: primitives

DCs have been introduced to provide a sparse alternative to GMs. Among other benefits, their number may be adapted to the amount of details one desires to produce, they do not have to be closed, and position and color control points do not have to be co-located. They have been defined as Bézier curves with color controls on each side of the curve [Orzan et al. 2008]. Colors are interpolated along each curve, and then propagated to the rest of the image by solving a Laplace equation. Blurring may be applied in post-process to smooth transitions across curves. Curves and color/blur control points may be set by hand, or captured from an image using for instance the optimized solution of Jeschke et al. [2011].

The artistic control of DCs has been improved by several extensions such as blockers and directional diffusion [Bezerra et al. 2010; Bowers et al. 2011]. Finch et al. [2011] proposed to solve the higher order Bi-Laplace equation to avoid "tent-like" gradients and produce a smoother color interpolation.

DCs have been used for the creation of other types of gradients: Winnemoeller et al. [2009] propagate normal coordinates from silhouettes and creases, normalizing them in post-process; Hnaidi et al. [2010] create height fields by diffusing height from curves that represent ridges and cliffs; Jeschke et al. [2011] use diffusion to control the parameters of procedural textures. DCs have also been extended to 3D by Jeschke et al. [2009b] to render high quality surface details on 3D objects and by Takayama et al. [2010] to create volumetric models with 3D color gradients.

In our approach, we solve the Bi-Laplace equation, expanding artistic controls with novel constraints; in particular, we provide local control over vector gradients thanks to novel transmission curves, and handle complex deformations like environment mapping.

2.3 Diffusion Curves: solvers

Representations based on DCs require specific methods to solve their associated partial differential equation. Interactive performance is obtained by adopting multi-scale approaches inspired by the multigrid algorithm [Briggs et al. 2000] that applies Jacobi iterations in a coarse-to-fine scheme. Despite GPU implementations of the multigrid, the convergence may still be slow to generate images in high resolution. Moreover, aliasing and flickering artifacts may occur because of the rasterization of the curves over a discrete multi-scale pixel grid. Jeschke et al. [2009a] present a faster solution, which initializes the color at each pixel with the color of the closest curve, and uses finite differences with variable step size to accelerate the convergence rate of Jacobi iterations. However, extensions that have been further proposed are not trivial to render with such an optimized approach.

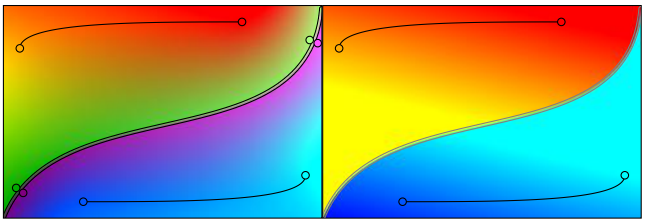


Figure 2: ***Simple curves****. The top and bottom curves diffuse color gradients on both images. The middle curve either acts as a diffuser with different colors on each side (left), or as a barrier (right).*

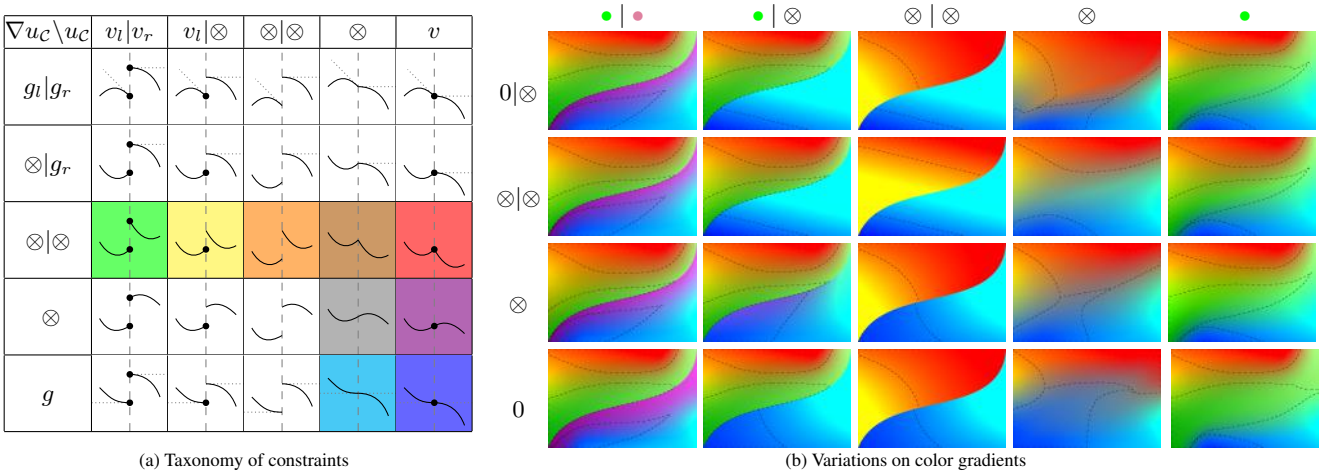
These solvers are output sensitive, and thus possess important limitations: they become slow for high-resolution images, and they must be updated for most image operations. Moreover, a simple operation such as zooming-in implies many difficulties to take into account the constraints which are not visible. Jeschke et al. [2009b] adapt their 2D solver to deal with deformations due to texture mapping on 3D surfaces, using axis-aligned warping techniques. Their method is still limited regarding instancing (it must be recomputed for each instance) and complex deformations (it does not permit environment mapping).

Pang et al. [2011] take a different approach whereby DCs are converted into a triangle-mesh representation and colors are linearly interpolated over each triangle. Positive mean-value coordinates give the color at a vertex as a weighted sum of the colors of visible curves. To evaluate visibility, they employ an exhaustive sorting algorithm. An alternative consists in performing ray tracing on the GPU via a pixel shader [Bowers et al. 2011]. The approach of Sun et al. [2012] avoids the explicit computation of visibility thanks to Green functions, which provide direct evaluation at individual points. Although the method is exact for closed curves, it only provides an approximation for open curves. Moreover, these three techniques are limited to harmonic interpolation, which lacks smoothness and control abilities.

Like [Pang et al. 2011], our solver makes use of an intermediate triangulation. However, it is based on higher-order patches and elegantly solves the Bi-Laplace equation using FEM while allowing for the largest set of artistic controls.

2.4 FEM for the Bi-Laplace equation

Even though recalling the huge literature on solving the biharmonic equation using FEM is out of the scope of this paper, it is worth noting that in the context of geometric modeling, similar fourth order PDEs often arise in the construction or preservation of high smoothness surfaces that are subject to zero and first order constraints. To achieve interactive performance, various discrete approximations that lack convergence have been employed (see for instance [Botsch and Sorkine 2008] for a comprehensive survey). To address the artifacts that may be produced by these approximations, Grinspun et al. [2006] described a discrete shape operator that builds on non conforming elements. More recently, Jacobson et al. [2010] presented an appealing mixed FEM discretization which exhibits the same order of complexity than former approximations, but math-



(a) Taxonomy of constraints

(b) Variations on color gradients

Figure 3: ***Types of curves.*** *(a) By combining value constraints (columns) and gradient constraints (rows), we define a rich set of curve types. Their specificities are best understood by inspecting them row by row, observing how different types of value constraints yield new configurations (points & dotted lines correspond to value & gradient constraints). Our taxonomy generalizes previous work, as indicated by cell colors that correspond to existing types of DCs: diffusion, barrier, tear, crease, crease-value, value, value-slope, slope. The nil curve has no effect since it sets no constraints. (b) The four last rows of the table are illustrated with color gradients. The top and bottom curves of Figure 2 are left unchanged, but the type of middle curve is modified. We overlay iso-values of primary colors for visualisation purposes.*

ematically more sound while permitting to seamlessly hanlde first order constraints. However, all these techniques have been developed specifically for linear meshes, while, as we show in this paper, color gradients require higher order elements.

3 Free-form vector gradients

The main goal of this paper is to produce smooth free-form gradients controlled by a sparse set of curve and point vector primitives. Finch et al. [2011] have shown that the solution to the Bi-Laplace equation $\Delta^2 u = 0$ provides for gradients that closely mimic smooth shading. It also offers advanced artistic controls in the form of user-specified value and gradient constraints, as illustrated in Figure 2. We thus target the same solution, considering u as an arbitrary scalar function (color channel, height, etc). In this section, we provide a taxonomy of constraints that unifies previous methods and further extends artistic controls. The way these constraints are handled in our solver is presented in Section 4.

Taxonomy of constraints. For the sake of simplicity, in this paper we will assume curves are provided as Bézier splines, though any other representation could be used. At any point along a curve $\mathcal{C}$, the value and/or the gradient magnitude in the normal direction may be constrained independently on each side of the curve. Denoting by $u_\mathcal{C}$ the restriction of u to $\mathcal{C}$, we summarize the space of possible constraints by the following diagrams:

where l and r denote the 'left' and 'right' sides respectivelly. For instance, the right side of $\mathcal{C}$ may be assigned a specific scalar function v_r or be left unspecified (we use the symbol $\otimes$ in such cases). Moreover, left and right values may be constrained to be equal (we then write $u_\mathcal{C} = v$). Similar choices are available for gradient constraints. The equality constraint is required whenever values or gradient magnitudes are left unspecified and one wants to enforce either C^0 or C^1 continuity. It also facilitates user interactions when the same values have to be set on both sides of a curve. As with previous work on DCs, scalar and gradient magnitude functions along curves are obtained through arc-length interpolation of an arbitrary number of constraint control points attached to curves at arbitrary locations. The case of points is slightly different: we treat them as isotropic constraints, and let users control not only their value and gradient magnitude, but also the gradient direction.

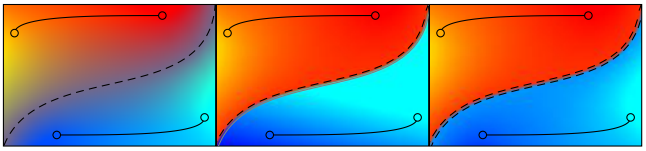


Figure 4: ***Contour curves*** *may be used with different types of value constraints, as shown here for the middle curve. Images from left to right correspond to $\otimes$, $\otimes|v_r$, and $\otimes|\otimes$ columns.*

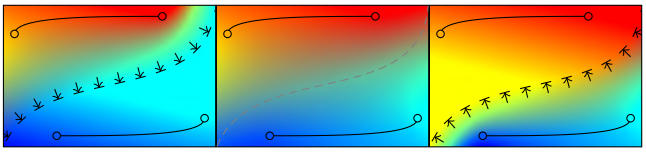


Figure 5: ***Transmission curves*** *permit to locally control color diffusion, here using the nil curve in the middle. From left to right, we have used $\alpha = -1$, $\alpha = 0$, and $\alpha = 1$.*

This small and intuitive set of constraints permits to achieve a large number of curve types. They are summarized in Figure 3(a), where we show all possible combinations of value and gradient constraints, each cell representing one type of curve. Such a taxonomy unifies and generalizes DCs and their extensions presented in earlier work, like barriers or slopes; these are directly identified by color in the table for completeness. In particular, observe that up to now, the use of gradient constraints has been limited to curves that have the same constraints on each of their sides (the 4 colored cells at bottom right). The offset curves of Finch et al. [2011] are not referenced since these are more a matter of user interface.

A benefit of our classification is that it lets emerge novel curve types that make more complex usages of gradient constraints (white cells), even though not all curve types make sense for all applications. For instance, choosing a specific gradient magnitude in color images makes little sense, apart from $\nabla u_\mathcal{C} = 0$ that we use in some instances in the paper. In contrast, when creating height fields (terrains), a gradient constraint $\nabla u_\mathcal{C} = g$ is useful since it constrains the normal of the underlying surface. A selected subset of curve types is illustrated in Figure 3(b) with color gradients.

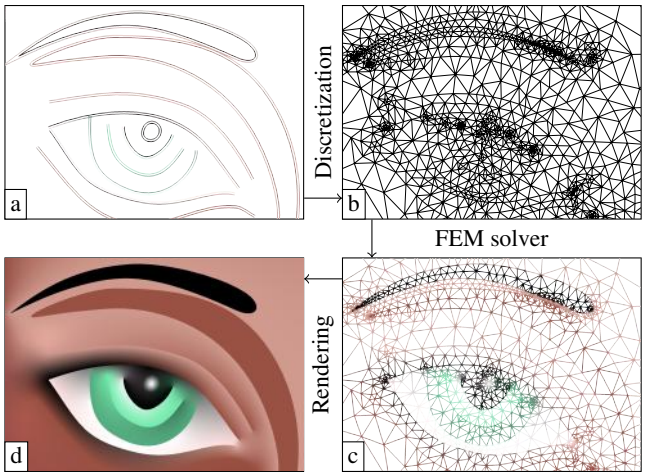


Figure 6: ***Overview of our solver****. Input vector primitives (a) are first discretized to produce our intermediate representation (b). It is used as input to our FEM solver that outputs a set of elements (c), which are easily rendered to obtain the final image (d).*

Contours. When values or gradient magnitudes on one or both sides of a curve are left unspecified, additional constraints at the curve level may be enforced. This is already found in Finch et al. [2011], where they define contour curves as curves that have an unspecified, yet *constant* value along one side. In our taxonomy, such iso-values may be assigned independently to the right and/or left side of a curve, and take on arbitrary gradient constraints. Refering to the table in Figure 3(a), this applies to the three middle columns.Various configurations of iso-values are illustrated in Figure 4. Note that we could similarly enforce iso-gradients, which correspond to the three middle rows of the table.

Transmission. When values on *both* sides of a curve are left unspecified but enforced to be equal ($\otimes$ column), any value in a family of solutions may be chosen to meet the equality constraint. We thus request an additional artistic parameter $\alpha \in [-1, 1]$ for such type of curves that permits to pick one specific solution. Intuitively, α controls the relative amount of diffusion across a curve: with $\alpha \neq 0$, values are partially blocked on one side, but transmitted on the other; $\alpha = 0$ boils down to a symmetric transmission; and $\alpha = \pm 1$ performs a purely asymmetric transmission. As shown in Figure 5, such a transmission control is very useful since it permits to control the influence of other curves. Nil curves (gray cell in Figure 3(a)) are particularly useful when combined with transmission. As opposed to the method of Bezerra et al. [2010] that relies on global homogeneous color coordinates for this purpose, the α parameter provides an accurate *local* control over diffusion.

4 Vectorial solver

The main technical contribution of this paper is on the evaluation of the sparse representation defined in the previous section. This amounts to solving the Bi-Laplace equation $\Delta^2 u = 0$, subject to the constraints of Figure 3, 4 and 5. In contrast to previous approaches, we propose to output an intermediate denser vector representation enabling closed-form evaluation. We first present this representation, which consists in a set of triangular patches (Section 4.1). We then explain how we create it in two steps: the unknown function u is discretized into triangles covering the domain of resolution (Section 4.2), after which unknown patch parameters are solved using an appropriate Finite Element Method (FEM) (Section 4.3). This is depicted in Figure 6, where we also show the final result obtained after rendering (Section 5).

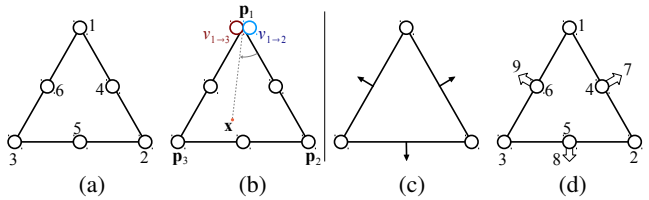


Figure 7: ***Triangular patches and elements.*** *(a) A Lagrange patch with its six nodes. (b) Our special quadratic patch for singularities. (b) Morley's element. (d) Fraiejs de Veubeke's element.*

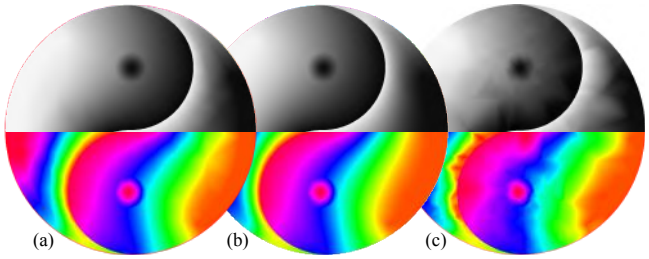


Figure 8: ***Patch order.*** *We provide a visual comparison between 2400 linear (a), and 1200 quadratic (b) patches. Both images requires about 5000 nodes, and have been computed in 60ms each. On the bottom, grey levels are mapped to RGB values to better reveal artifacts. Converting Morley's element to quadratic patches (c) leads to even stronger artifacts.*

4.1 Intermediate representation

Central to our framework is the design of our intermediate vectorial representation. Quadtree-based meshes are fast to generate [Frey and Marechal 1998] but they would essentially lead to the same issues than directly solving in a pixel grid. Meshless representations [Belytschko and Chen 2007] are easy to manipulate and known to produce very smooth results, but they are not well suited to faithfully represent discontinuities and are relatively costly to evaluate. A triangular mesh with curved edges is therefore a natural choice since it is easy to make its vertices and edges match input constraints. More precisely we consider non-overlapping polynomial patches u_i such that the function u is then represented by:

$$u(\mathbf{x}) = u_{\mathrm{idx}(\mathbf{x})}(\mathbf{x}), \tag{1}$$

where $\mathrm{idx}(\mathbf{x})$ returns the index of the unique triangular patch containing the evaluation point $\mathbf{x}$. This choice significantly simplifies and speeds up our FEM solver, rasterization, and random evaluations. Using a higher-order representation such as subdivision surfaces (as in [Liao et al. 2012]) would greatly complexify these operations, hence limiting the benefits of an intermediate representation.

We use quadratic Lagrange polynomials: $u_i(\mathbf{x}) = \sum_{k=1}^{6} v_k B_k(\mathbf{x})$ for our patches, where B_k is the basis function associated to the k^{th} node of the patch with associated value v_k. Let Φ_k, $k = 1, 2, 3$ be the linear basis functions associated with the triangle vertices (i.e., barycentric coordinates); then, using the notation of Figure 7a, we have $B_1 = \Phi_1(2\Phi_1 - 1)$ at a vertex node, and $B_4 = 4\Phi_1\Phi_2$ at an edge node. The other basis functions are obtained by cyclic permutations. As shown in Figure 8, quadratic patches outperform linear ones while remaining compact (6 coefficients) and fast to evaluate. Even though cubic polynomials might constitute an appealing alternative, they have not been considered because they would significantly increase the computational complexity of our FEM solver.

Representing singularities. Curve extremities or intersections might require to specify two different values at the same node. This creates a singularity that cannot be represented with standard triangular patches. Let's consider an extremity point $\mathbf{p}_1$ for instance,

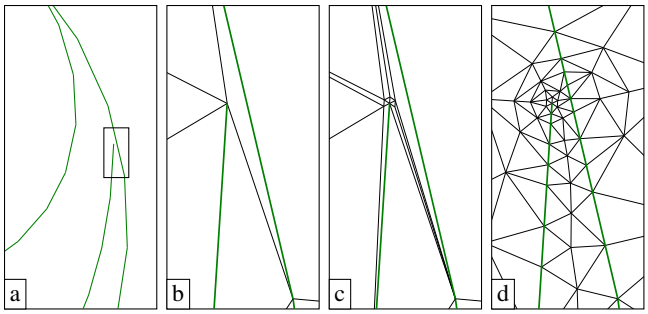


Figure 9: ***Constrained triangulation.*** *(a) Curve distretization yields a PSLG in green. The other figures show a closeup view of the triangulation process: (b) Delaunay triangulation; (c) Explicit refinement around critical points; (d) Geometric refinement.*

with value constraints $v_{1\to2}$ and $v_{1\to3}$, one for each abutting edge in the triangulation (see Figure 7b). Our key observation here is that at such an extremity, we expect the bi-harmonic interpolation to yield a smooth radial gradient from one edge to the other. To capture this behavior, we make the value v_1 a function of the evaluation point $\mathbf{x}$, performing a linear interpolation between $v_{1\to2}$ and $v_{1\to3}$ based on the relative angular position:

$$v_1(\mathbf{x}) = v_{1\to2} + (v_{1\to3} - v_{1\to2})\frac{(\mathbf{x}-\mathbf{p}_2)^T(\mathbf{x}-\mathbf{p}_1)}{(\mathbf{p}_3-\mathbf{p}_2)^T(\mathbf{x}-\mathbf{p}_1)}.$$

4.2 Constrained triangulation

The goal of the triangulation step is to produce a triangular partition of the domain that satisfies the following criteria:

- it must exactly match the input constraints,
- it must be dense enough to faithfully represent u,
- it must be fast enough to permit interactive drawing,
- it must be locally uniform to avoid numerical issues and provide high visual quality.

Note that the last three criteria may be in conflict: a high quality often requires a high density, which also demands high computational and memory costs. To achieve these goals, we implemented a four-step triangulation procedure, illustrated in Figure 9. The first step consists in the construction of a *planar straight lines graph* (PSLG) by discretizing the input curves into a set of polylines. Then a constrained Delaunay algorithm yields an initial triangulation, which is refined by two additional steps: a refinement around singularities, and a geometric refinement that provides an overall high-quality (dense enough) triangulation. Our procedure may be seen as a variant of the work of Pang et al. [2011], and fixes several shortcomings as detailed below.

Construction of the PSLG. The purpose of the first step is mostly to simplify and speedup the Delaunay triangulation and FEM solver by approximating input curves by polylines. In order to satisfy our first criterion, we construct the PSLG such that it includes all input constraint points as well as curve intersections. The remaining curve segments are then subdivided until the maximal angle between the segment and tangent extremities is lower than a given threshold (15° in our examples). It is important to discretize input curves in such an order, otherwise two vertices might be generated arbitrarily close to each other, thus leading to over-refinement in the final Delaunay refinement step.

Singularity refinement. The strongest variations often occur around curve singularities (tips and intersections) and isolated point constraints. To guarantee high quality around such a point $\mathbf{p}$, we insert up to six vertices $\mathbf{q}_i$ uniformly spread around $\mathbf{p}$ at a distance l such that the radial angle $\widehat{\mathbf{q}_i\mathbf{p}\mathbf{q}_{i+1}}$ between two successive vertices is smaller or equal to $\pi/3$. For tips or intersection points, the $\mathbf{q}_i$ are inserted along the constrained edges in priority. The insertion distance l is adaptively determined from the Delaunay triangulation which provides meaningful information about the relative distances between neighboring constraints. In particular, we chose l as the third of the shortest triangle altitude passing through $\mathbf{p}$ (Figure 9c).

Geometry refinement. Since quadratic polynomials cannot represent inflections, we must ensure that two constraints are not directly connected by an edge. In order to limit the number of generated triangles, we satisfy this requirement through Delaunay refinement [Shewchuk 2000] while ensuring a minimal triangle quality. Given a triangle metric, Delaunay refinement iteratively inserts a new vertex into the constrained Delaunay triangulation for the triangle having the worst score. To achieve our goals, triangles connecting two constraints are prioritized and then sorted according to the ratio between the circumradius and the shortest edge length, which is a standard quality metric implicitly bounding the minimal and maximal angles [Shewchuk 2000]. For our results, we stopped the refinement once the minimum triangle angle is greater than 20°.

We have found these *a priori* refinement heuristics preferable to more complex mechanisms that would violate our performance criterion. For instance, interleaving re-meshing and solving operations would likely provide a better control over density, but would be prohibitively expensive to compute. Regarding performance, it is important to remark that our triangulation procedure only depends on input *geometry*; hence it does not have to be recomputed when constraint *values* are modified while their positions are kept unchanged.

4.3 FEM solver

At this stage, we have discretized the function u into adjacent quadratic triangular patches u_i. This section explains how to make use of Finite Element Methods (FEM) to accurately and efficiently compute nodal values v_j such that u satisfies the Bi-Laplace equation.

As discussed in Section 2.4, similar problems have already been studied in the graphics community. However, all these techniques have been developed specifically for linear meshes, while, for our application, quadratic patches appear to be the best trade-off to convey visually pleasant results while keeping a coarse triangle mesh. Indeed, a general result in the FEM literature is that the convergence rate with respect to element order is higher than for increasing element resolution [Zienkiewicz et al. 2005]. As discussed in Section 4.1, we have confirmed that in our experiments: for color gradients quadratic patches offer an obvious higher visual quality than using four times more linear patches. The difference is amplified when editing or animating curves.

Moreover, our application exhibits special needs such as singularities and punctual gradient constraints, which are not straightforward to handle using the aforementioned discretizations. To address these difficulties, we propose to use a *weak* formulation of our differential equation as well as non-conforming elements as detailed below.

Weak formulations. FEM never directly solves for the *strong* formulation of a differential equation. The equation is rather projected onto an appropriate set of *test functions* t_j in order to reduce requirements on continuity and degree of the elements, as well as to improve the stability of the numerical scheme [Zienkiewicz et al. 2005]. In our case, this leads to the following problem [Lascaux and Lessaint 1975]:

$$\forall j,\ \int_\Omega \Delta u \cdot \Delta t_j - \sigma\left(\frac{\partial^2 u}{\partial x^2}\frac{\partial^2 t_j}{\partial y^2} - 2\frac{\partial^2 u}{\partial x \partial y}\frac{\partial^2 t_j}{\partial x \partial y} + \frac{\partial^2 u}{\partial y^2}\frac{\partial^2 t_j}{\partial x^2}\right) = 0\,, \tag{2}$$

where σ is Poisson's coefficient, which controls the influence of the regularization terms. In practice, σ has to be chosen in the range $[\frac{1}{2}, 1]$, in which case its value does not influence the result.

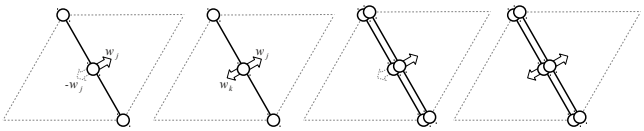


Figure 10: ***Constraints*** *along a curve are obtained by sharing or duplicating value or gradient nodes.*

Choice of elements. Ideally, we would like to directly use our current discretization of u in quadratic triangular patches to solve Equation 2. Unfortunately, the numerical scheme requires C^1 continuity across such *conforming* elements to reach convergence. This typically implies the use of quintic polynomial elements, which are prohibitively expensive to compute.

Non-conforming elements permits to reach convergence without requiring C^1 continuity across triangle edges. For instance, the Morley element [Morley 1971] shown in Figure 7c is the lowest-degree element that can solve Equation 2. It is a quadratic polynomial basis function with six degrees of freedom: the three nodal values at the vertex positions, and the three normal derivatives at the edge middles. Note that a variant of Morley's elements have been successfully used to obtain high quality and stable solutions for geometric modeling purposes [Grinspun et al. 2006]. In practice, Morley's elements are not even C^0 continuous across their boundaries. C^0 continuity could be enforced by computing the nodal edge values of our Lagrange patches from the average of the edge middle values of the two adjacent Morley elements. Unfortunately, as shown Figure 8c this naive strategy leads to very unpleasant results.

In order to be able to output quadratic patches with at least C^0 continuity, we thus propose to use the third-order *non-conforming* Fraeijs de Veubeke's element [1974] (FV) which is illustrated in Figure 7d. The FV element can be seen as a higher degree variant of Morley's element. It is controlled by six nodal values at vertices and mid-points of the edges, and the mean value of the normal derivative along each edge (which we call mean gradient nodes). This element is perfectly suited to our problem. Indeed, since its nodal parameters form a super-set of parameters for our quadratic patches, the conversion from FV elements to our Lagrange polynomials is straightforward: we simply dismiss the mean gradient nodes. Moreover, it provides a control over the gradient magnitude in the direction normal to an edge, which is necessary for handling gradient constraints.

Under these settings, we are now seeking for an approximation of u of the form $\sum_j w_j F_j$ where the F_j are the basis functions of the FV elements given in appendix, and the w_j are the sought-for nodal values (with $w_j = v_j$ for $j \leq m$, m being the total number of nodes in our intermediate representation). As it is often done in FEM, we take for the test functions t_j the same set of basis functions F_j. Then Equation 2 becomes a sparse linear problem of the form $A\mathbf{w} = 0$ where the matrix A is called the *stiffness* matrix whose coefficients $A_{i,j}$ are given in the appendix.

In Section 4.4, we give some implementation details; in particular we explain how simple edits of constraint values only require partial updates of the system. In the rest of this section we show how to handle our specific constraints.

Handling constraints. Thanks to the use of FV elements, both value and gradient constraints are easily satisfied (because all constraint points and curves are vertices and edges of the triangulation). As illustrated in Figure 10, the non-equality of values (resp. gradient) across a curve is handled by doubling the value (resp. mean gradient) nodes along the curves. In table 3 this concerns curves of the form $\cdot|\cdot$. A node attached to a constraint providing a specific value directly takes its value from the constraints, and it is removed from the list of unknowns (this corresponds to columns $v_l|v_r$, $v_l|\otimes$, and v). Likewise, gradient values specified on a curve are integrated over the respective edge of the element to fix the respective mean gradient node (this corresponds to rows $g_l|g_r$, $\otimes|g_r$, and g). Isocontours are trivially handled by having a single nodal value, i.e., a single unknown, for all the nodes lying on the same iso-contour.

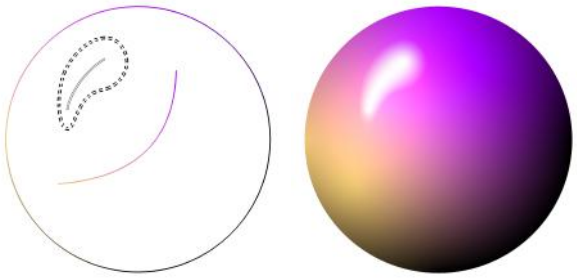


Figure 11: ***Transmission curve*** *(dashed) is used to create a complex highlight and blended with global gradients (plain curves).*

Taking into account the gradient specified at an isolated constrained point is slightly more difficult because the FV element only offers a control over the gradient magnitude along an edge. Let us consider the FV element $\mu(\mathbf{x}) = \sum_{k=1}^{9} F_k(\mathbf{x}) w_k$ as in Figure 7d, assuming the vertex $\mathbf{p}_1$ is an isolated constraint point with gradient $\mathbf{g}$. Let $\mathbf{e}_2 = \mathbf{p}_2 - \mathbf{p}_1$ and $\mathbf{e}_3 = \mathbf{p}_3 - \mathbf{p}_1$ be the two incident edges, and $\mu_2(t)$, $\mu_3(t)$ the value of the element over these two edges. Constraining $\nabla\mu(\mathbf{p}_1) = \mathbf{g}$, is equivalent to constraining $[\mu_2'(0)\ \mu_3'(0)]^T = \bar{\mathbf{g}}$, where $\bar{\mathbf{g}} = [\mathbf{e}_2\ \mathbf{e}_3]^{-1}\mathbf{g}$ is the projection of $\mathbf{g}$ into the frame spanned by $\mathbf{e}_2$,$\mathbf{e}_3$. Looking at the FV's basis functions (Equation 3), one can remark that the mean gradient node of a given edge does not influence the value along its edge. Therefore the derivative constraint in each direction can be enforced independently by constraining the mean gradient nodes w_7 and w_9 respectively. For instance, this yields for w_7:

$$w_7 = \frac{\bar{g}_0 - \sum_{k \neq 7,9} (F_k^2)'(0) w_k}{(F_7^2)'(0)} ,$$

where $F_k^2 = F_k(\mathbf{p}_1 + t\mathbf{e}_2)$ are the FV basis functions restricted to the edge $\mathbf{e}_2$. These constraints are introduced in equation 4 when assembling the stiffness matrix, thus effectively removing two degrees of freedom.

Handling transmission. Transmission permits values coming from one side of a curve to influence the other side, and may be made asymmetric by partly blocking the diffusion in the opposite direction. This is particularly useful to control highlights for instance, as seen in Figure 11. Such a curve has no value constraint except that values must be equal on each side ($\otimes$ column). Observe that the unknown w_j of a given node belonging to such a curve has an influence on the nodal values of the incident elements. Given two weights β_l and β_r, the key idea is to make the node j emit $\beta_l w_j$ to the left nodes, and $\beta_r w_j$ to the right nodes. When building the stiffness matrix, this amounts to inserting $\beta_l A_{i,j}$ instead of $A_{i,j}$ if the node i is on the left, $\beta_r A_{i,j}$ if the node i is on the right, and directly $A_{i,j}$ otherwise (i is on the curve). Note that this leads to an asymmetric stiffness matrix. The weights β_l and β_r are computed from the transmission factor α. When $\alpha = 0$, we set both $\beta_l = \beta_r = 1$, leading to the default behavior. With $\alpha > 0$, $\beta_r = 1$ and $\beta_l = 1 - \alpha$, thus elements on the right will behave normally, but the ones on the left will be less influenced by w_j and therefore less influenced by the right side. Ultimately, when $\alpha = 1$, $\beta_l = 0$, elements on the left do not depend on values on the right side anymore, which effectively amounts to block diffusion from right to left. Conversely, when $\alpha < 0$ we take $\beta_l = 1$ and $\beta_r = 1 + \alpha$.

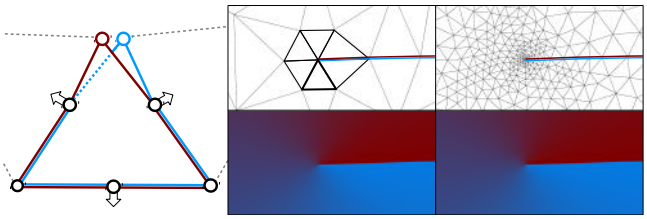


Figure 12: ***Singularities*** *are handled by duplicating elements (left). Results obtained with a coarse or very dense triangulation are indistinguishable (right). The highlighted triangles correspond to the singular ones with in bold the one of the left figure.*

	Fig 1a (symm.)	Fig 8 (non-symm.)	Fig 11 (non-symm.)	Fig 13b (symm.)
PSLG	41	2	1	8
Refinement	38	12	3	9
Assembly	65	16	7	14
Solve	58	35	14	13
Total	202	65	25	44

Table 1: ***Performances*** *(in ms) for the different steps of our solver on a few examples. Images with asymmetric transmission yield non-symmetric sparse problems which are more involved to solve.*

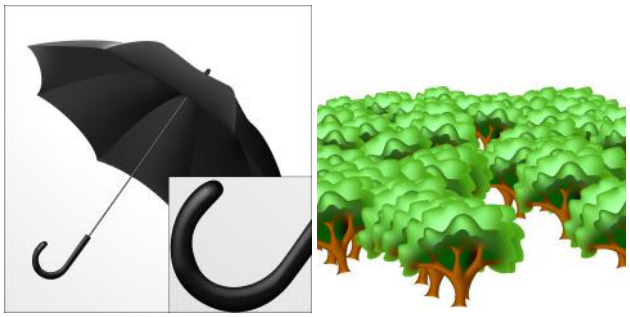
Handling singularities. As shown in section 4.1, curve extremities and intersections require special patches to appropriately represent the singularity that appears when two different values are assigned to a same node. While our novel singular patch works well for evaluation purposes, it cannot be used for solving purpose since it is not differentiable. We workaround this issue by approximating a singular element with a pair of standard FV overlapping elements that are connected to two different nodes at the singularity. As shown in Figure 12, this simple approach works remarkably well in practice.

4.4 Implementation details

Our current implementation entirely runs on the CPU, and does not exploit multi-threading yet. It makes use of CGAL for the Delaunay triangulation and refinement, and Eigen's direct solver for solving our sparse problems. In contrast to iterative methods, a direct solver permits to factorize the left hand side matrix once, to then very efficiently solve for multiple right sides. For color images, this permits to efficiently solve for the red, green, blue, and alpha components at once. Moreover, since the modification of a constraint value only changes the right hand side of the system, the factorization does not have to be recomputed during constraint value editing. This implementation permits the setting of a rich number of constraints (like gradient magnitudes, contour curves or asymmetric transmission) directly in the linear system of equation.

We have tested our method on one single core of a Intel Core i7-3610QM, and we report our performances in Table 1. It is difficult to compare our performances with those of the pixel-based method of Finch et al. [2011]. Indeed, the complexity of their solver grows with image resolution, while ours grows with the number of vector primitives. We believe that output insensitive solvers are preferable since the entire goal of vector drawing is to keep a small number of sparse primitives. Moreover, let us note that for images composed of a number of primitives similar to their results, we obtain similar performances *irrespective of image resolution*. On top of that, we have used a single core implementation, while they explicitly mention using a parallelized implementation on a 2.67GHz dual quad-core Intel Xeon.

Our approach thus proves to be efficient for the interactive editing and drawing of free-form vector gradients. However, its main benefit is to output a vectorial representation that makes subsequent rendering tractable, as explained in the next section.



(a) (b)

Figure 13: ***Color results.*** *Our approach is independent of image resolution (a), and is especially efficient for instancing (b).*

5 Rendering

Our solver needs to be fully updated everytime the position of a control point is modified, and partially when constraints are changed. Such modifications are part of the drawing/editing process. This is to contrast with rendering, which essentially consists in evaluating our intermediate representation at specific locations. Since our solution is given in closed-form, it tremendously simplifies and improves applications that make use of vector gradients, as detailed in the following.

Color images. With our approach, color images are best displayed by directly rasterizing the quadratic patches. Even though we assumed linear edges when solving, at rendering time edges tied to an input curve must be properly refined according to the zoom and deformation levels. In our implementation, such triangles are subdivided in software, and then rasterized using OpenGL and pixel shaders to evaluate the quadradic patches. Hardware tessellation could be used for this purpose too.

Our approach permits to naturally handles a broad range of deformations at rendering time: translations, rotations and scaling, but also perspective deformation as shown in Figure 1. One simply has to deform the intermediate representation itself. This is particularly useful for printing since arbitrary high image resolutions may be output (Figure 13a); for copy-pasting since copies are treated as (possibly deformed) instances with no increase in complexity (Figure 13b); and for storage and dissemination. In contrast, pixel-based solvers must impose additional constraints at image borders when zooming in; they must be fully updated for most deformations; and their complexity quickly increases with higher image resolutions and with additional deformed instances.

2.5D images. Although not all of the gradient constraints from our taxonomy make sense for color images, they reveal their usefulness when working with 2.5D data such as height fields or normal maps. This is shown in Figure 14 with the construction of a height field. Recall our patches are only C^0 continuous across them. Even though this has been proven to be enough for the display of color gradients, the shading applied to a height-field might reveal the lack of C^1 continuity. Higher quality could be obtained at rendering time by, e.g., applying quadratic normal interpolation [Vlachos et al. 2001]. As shown in Figure 15a, normal images may also be directly created with our system by propagating surface normal coordinates. In this case, the user does not set coordinate constraints on curves though: we enforce the projection of a 3D normal constraint to be aligned with the 2D normal of the curve as in [Winnemöller et al. 2009]. This leaves a useful degree of freedom, which corresponds to the normal coordinate in the direction perpendicular to the image plane. Moreover, assuming quadratic surfaces, the normal field is expected to vary linearly in the image. We thus use a simplified version of our solver that performs harmonic interpolation (see appendix).

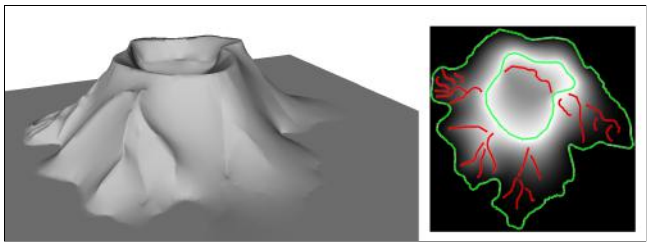


Figure 14: ***Height field construction.*** *A few curves (top right) suffice to create a terrain. Here, green curves impose value constraints, while red curves set gradient constraints only.*

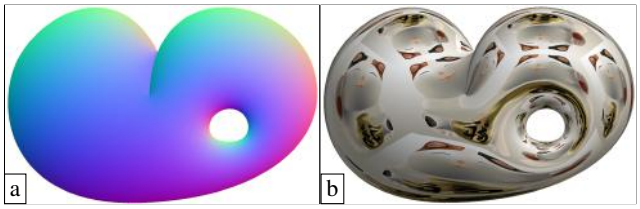


Figure 15: ***Normal and environment maps.*** *A vector normal map is created from scratch (a). It is used to look up the left image of Figure 1 at locations given by the mirror direction at each point (b).*

Our approach has a number of benefits compared to previous methods for the creation of height fields [Winnemöller et al. 2009; Hnaidi et al. 2010]. First, it provides a richer set of artistic controls thanks to our novel taxonomy. Second, it only requires a single optimization, while others require integration in post-process [Winnemöller et al. 2009] or multiple diffusion steps [Hnaidi et al. 2010]. Third, it outputs results directly in a structure amenable to rendering, while others need either a specific renderer or a mesh conversion in post-process. Indeed, our intermediate mesh only has to be lifted based on computed height values, and then be loaded in a rendering engine.

3D mapping. We have also adapted our method to the mapping of vector gradients onto parametrized surface meshes in 3D. One solution consists in intersecting the triangles of our mesh, with the triangles that define the surface in texture space, hence refining the 3D mesh to precisely meet gradient requirements. This is illustrated for color textures in Figure 16, and could be easily adapted to work with displacement maps. In this later case, the output mesh would naturally preserve the discontinuities (cliffs) present in the vector displacement map. Another solution consists in accessing randomly our intermediate representation at required texture coordinates. For instance, the realtime GPU based techniques of Parilov et al. [2008], or Nehab et al. [2008] can already handle quadratic curves, and would require little effort to be adapted to our purpose.

Again, our method provides a lot more flexibility compared to previous work. Instancing is a very common practice in texture mapping, and our solution does not require any specialized solver like [Jeschke et al. 2009b]. Thanks to random access evaluation, it works for arbitrary parameterizations, even environment maps as shown in Figure 15. To the best of our knowledge, no previous method has shown such uses of free-form vector gradients.

6 Discussion

Summary. The vectorial solver presented in this paper combines for the first time the expressivity of sparse curve and point vector drawing primitives, with the rendering simplicity of vector representations. This is permitted by an intermediate triangular structure that is updated interactively during drawing, but most importantly that is trivially evaluated during rendering, even at random locations. Our method unifies and generalizes the types of controls achievable by previous techniques, and introduces a novel transmission parameter that provides a much-needed local control over propagation. The use of FEM with high-order elements permits an accurate and fast solution to the Bi-Laplace equation, and directly deals with gradient constraints. In contrast, other solutions are either limited to the Laplace equation [Orzan et al. 2008; Bezerra et al. 2010; Jeschke et al. 2009a], simple gradient constraints [Finch et al. 2011], or require multiple passes with indirect gradient control [Bezerra et al. 2010; Hnaidi et al. 2010].

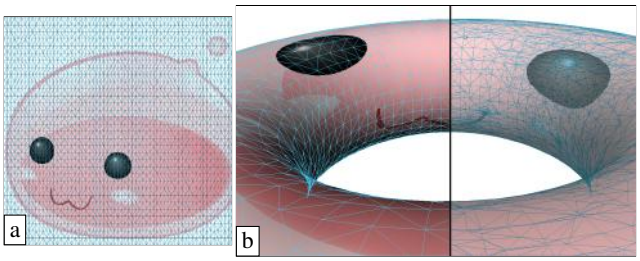


Figure 16: ***Texture mapping.*** *The texture mapping shown in Figure 1 is obtained by: (a) positioning a vector color image in texture space; and (b) intersecting surface triangles with triangles coming from our representation (shown before and after intersection).*

Limitations. Triangulation updates might lead to flickering artifacts when drawing primitives are edited; although this is a major problem with linear elements, our quadratic elements reduce this issue to a point where they are hardly noticeable, as is best observed in the accompanying video. Even though quadratic elements are more costly for optimization, they demand significantly simpler triangulation to reach a reasonable quality and are thus comparable in terms of performance to linear elements. Our optimization procedure could be improved in a number of ways: employing a parallelized implementation, identifying closed curves to reduce dependencies, or using a cheaper triangulation than Delaunay (care should be taken with degenerated triangles though).

Some limitations of our system are not due to the solver itself, but rather to the choice of the Bi-Laplace equation. First, as identified by Finch et al. [2011], when only colinear constraints are used, the solution is not determined and we must revert in this case to the Laplace equation. But more importantly, the Bi-Laplace equation may create new value extrema outside from curve or point constraints, whereas the Laplace equation is guaranteed to avoid such issues. This is especially noticeable when using gradient constraints at singularities. This isssue has been recently addressed through a non linear optimization guided from an harmonic solution [Jacobson et al. 2012]. An alternative solution might be to blend between Laplace and Bi-Laplace equations in problematic regions. We believe we could adapt our solver to this purpose in future work since it is already able to solve Laplace equation.

Future work. In this paper we have focused on free-form vector drawing, but our approach may also be used for image vectorization. The goal would then be three-fold: identify vector primitives and their type in an image; generate a triangulation with a density adapted to remaining variations; find color and gradient constraints that best approximate the input image inside each triangle. Contrary to previous techniques (e.g., [Liao et al. 2012]) this approach would attach constraints to a few vertices, making subsequent manipulations a lot more direct since the interior triangulation could be safely modified during editing. We have also restricted our solver to work in a flat 2D domain. It could be adapted to more complex domains in future work, for the drawing of free-form gradients directly on surfaces, or for creating volumetric gradients in 3D.

7 Acknowledgment

This work has been supported by the ANR SeARCH (ANR-09-CORD-019) and ANR DRAO projects. We would like to thank Noémie-Fleur Sandillon-Rezer for designing Figures 1 and 13a, and Laurence Boissieux for the model of Figure 6.

References

BELYTSCHKO, T., AND CHEN, J. 2007. *Meshfree and Particle Methods*. John Wiley and Sons Ltd.

BEZERRA, H., EISEMANN, E., DECARLO, D., AND THOLLOT, J. 2010. Diffusion constraints for vector graphics. In *NPAR 2010: Proceedings of the 8th International Symposium on Non-photorealistic Animation and Rendering*.

BOTSCH, M., AND SORKINE, O. 2008. On linear variational surface deformation methods. *IEEE Transactions on Visualization and Computer Graphics 14*, 1, 213–230.

BOWERS, J. C., LEAHEY, J., AND WANG, R. 2011. A ray tracing approach to diffusion curves. *Computer Graphics Forum (Proc. of EGSR) 30*, 4, 1345–1352.

BRIGGS, W. L., HENSON, V. E., AND MCCORMICK, S. F. 2000. *A multigrid tutorial*. Society for Industrial and Applied Mathematics.

FINCH, M., SNYDER, J., AND HOPPE, H. 2011. Freeform vector graphics with controlled thin-plate splines. *ACM Trans. Graph. (Proc. of Siggraph Asia) 30*, 6.

FREY, P. J., AND MARECHAL, L. 1998. Fast adaptive quadtree mesh generation. In *Proc. of the 7th International Meshing Roundtable*, 211–224.

GRINSPUN, E., GINGOLD, Y., REISMAN, J., AND ZORIN, D. 2006. Computing discrete shape operators on general meshes. *Computer Graphics Forum (Proc. of Eurographics) 25*, 3.

HNAIDI, H., GUIN, E., AKKOUCHE, S., PEYTAVIE, A., AND GALIN, E. 2010. Feature based terrain generation using diffusion equation. *Computer Graphics Forum (Proc. of Pacific Graphics) 29*, 7, 2179–2186.

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

JACOBSON, A., WEINKAUF, T., AND SORKINE, O. 2012. Smooth shape-aware functions with controlled extrema. *Computer Graphics Forum (Proc. of SGP) 31*, 5, 1577–1586.

JESCHKE, S., CLINE, D., AND WONKA, P. 2009. A GPU laplacian solver for diffusion curves and poisson image editing. *ACM Trans. Graph. (Proc. of Siggraph Asia) 28*, 5, 1–8.

JESCHKE, S., CLINE, D., AND WONKA, P. 2009. Rendering surface details with diffusion curves. *ACM Trans. Graph. (Proc. of Siggraph Asia) 28*, 5, 1–8.

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

LASCAUX, P., AND LESSAINT, P. 1975. Some nonconforming finite elements for the plate bending problem. *ESAIM: Mathematical Modelling and Numerical Analysis 9*, 9–53.

LECOT, G., AND LÉVY, B. 2006. ARDECO: Automatic Region DEtection and Conversion. In *17th Eurographics Symposium on Rendering*, 349–360.

LIAO, Z., HOPPE, H., FORSYTH, D., AND YU, Y. 2012. A subdivision-based representation for vector image editing. *IEEE Transactions on Visualization and Computer Graphics*.

MORLEY. 1971. On the constant moment plate bending element. *Journal of Strain Analysis*.

NEHAB, D., AND HOPPE, H. 2008. Random-access rendering of general vector graphics. *ACM Trans. Graph. (Proc. of Siggraph) 27*, 5, 135:1–135:10.

ORZAN, A., BOUSSEAU, A., WINNEMÖLLER, H., BARLA, P., THOLLOT, J., AND SALESIN, D. 2008. Diffusion curves: A vector representation for smooth-shaded images. In *ACM Trans. Graph. (Proc. of Sigraph)*, vol. 27.

PANG, W.-M., QIN, J., COHEN, M., HENG, P.-A., AND CHOI, K.-S. 2011. Fast rendering of diffusion curves with triangles. *IEEE Computer Graphics and Applications*.

PARILOV, E., AND ZORIN, D. 2008. Real-time rendering of textures with feature curves. *ACM Trans. Graph. 27*, 1, 3:1–3:15.

SHEWCHUK, J. R. 2000. Mesh generation for domains with small angles. In *Proc. of the 16th annual symposium on Computational Geometry*, 1–10.

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

SUN, X., XIE, G., DONG, Y., LIN, S., XU, W., WANG, W., TONG, X., AND GUO, B. 2012. Diffusion curve textures for resolution independent texture mapping. *ACM Trans. Graph. (Proc. of Siggraph) 31*, 4, 74:1–74:9.

TAKAYAMA, K., SORKINE, O., NEALEN, A., AND IGARASHI, T. 2010. Volumetric modeling with diffusion surfaces. *ACM Trans. Graph. (Proc. of Siggraph Asia) 29*, 180:1–180:8.

VEUBEKE, M. F. 1974. Variational principles and the patch test. *Inter. Journal for Numerical Methods in Engineering 8*, 4.

VLACHOS, A., PETERS, J., BOYD, C., AND MITCHELL, J. L. 2001. Curved pn triangles. In *Proc. of the symposium on Interactive 3D graphics*, 159–166.

WINNEMÖLLER, H., ORZAN, A., BOISSIEUX, L., AND THOLLOT, J. 2009. Texture design and draping in 2D images. *Computer Graphics Forum (Proc. of EGSR) 28*, 4, 1091–1099.

ZIENKIEWICZ, TAYLOR, AND ZHU. 2005. *The Finite Elements Method: Its Basis and Fundamentals*.

Appendix

Fraeijs de Veubeke's element. For the sake of reproducibility, the closed form formulas of the basis function for the FV element are given below:

$$\begin{aligned} F_1 &= \Phi_1(\Phi_1 - \frac{1}{2})(\Phi_1 + 1) + 3\Phi_1\Phi_2\Phi_3 \\ &\quad - d_2\Phi_2(2\Phi_2 - 1)(\Phi_2 - 1) + d_3\Phi_3(2\Phi_3 - 1)(\Phi_3 - 1) \\ F_5 &= 4\Phi_1(1 - \Phi_1)(1 - 2\Phi_1) + 4\Phi_2\Phi_3 - 12\Phi_1\Phi_2\Phi_3 \\ F_8 &= -\frac{2\Delta}{\|\mathbf{p}_3 - \mathbf{p}_2\|}\Phi_1(2\Phi_1 - 1)(\Phi_1 - 1) \end{aligned} \tag{3}$$

with $d_1 = \frac{(\mathbf{p}_3 - \mathbf{p}_2)^T(\mathbf{p}_2 - \mathbf{p}_1)}{2|\mathbf{p}_3 - \mathbf{p}_2\|^2}$.

Here, Φ_i corresponds to the linear basis elements (barycentric coordinates), Δ is the area of the triangle, $\mathbf{e}_i$ is the edge opposite to the i^{th} vertex. The quantities d_2, d_3, and the other basis are obtained by circular permutation.

The coefficient $A_{i,j}$ of the stiffness matrix are of the form:

$$A_{i,j} = \int \Delta F_i \cdot \Delta F_j - \sigma \left(\frac{\partial^2 F_i}{\partial x^2}\frac{\partial^2 F_j}{\partial y^2} - 2\frac{\partial^2 F_i}{\partial x \partial y}\frac{\partial^2 F_j}{\partial x \partial y} + \frac{\partial^2 F_i}{\partial y^2}\frac{\partial^2 F_j}{\partial x^2} \right). \tag{4}$$

Harmonic interpolation Solving the Laplace equation $\Delta u = 0$ with our approach is considerably simpler. Its associated weak formulation

$$\forall j,\ \int_\Omega \nabla u \cdot \nabla t_j = 0\,,$$

requires only C^0 continuity across elements. Therefore, it can be directly solved using our Lagrange patches as the elements. On the other hand, gradient constraints cannot be imposed in this case.