Poisson Vector Graphics (PVG)

Fei Hou, Qian Sun, Zheng Fang, Yong-Jin Liu, *Senior Member, IEEE*, Shi-Min Hu, Hong Qin, Aimin Hao, and Ying He

Abstract—This paper presents Poisson vector graphics (PVG), an extension of the popular diffusion curves (DC), for generating smooth-shaded images. Armed with two new types of primitives, called Poisson curves and Poisson regions, PVG can easily produce photorealistic effects such as specular highlights, core shadows, translucency and halos. Within the PVG framework, the users specify color as the Dirichlet boundary condition of diffusion curves and control tone by offsetting the Laplacian of colors, where both controls are simply done by mouse click and slider dragging. PVG distinguishes itself from other diffusion based vector graphics for 3 unique features: 1) explicit separation of colors and tones, which follows the basic drawing principle and eases editing; 2) native support of seamless cloning in the sense that PCs and PRs can automatically fit into the target background; and 3) allowed intersecting primitives (except for DC-DC intersection) so that users can create layers. Through extensive experiments and a preliminary user study, we demonstrate that PVG is a simple yet powerful authoring tool that can produce photo-realistic vector graphics from scratch.

Index Terms—Poisson vector graphics, diffusion curves, Poisson equations, hue and tone editing, seamless cloning

1 Introduction

Vector graphics provides several practical benefits over traditional raster graphics, including sparse representation, compact storage, geometric editablity, and resolution-independence. Early vector graphics supports only linear or radial color gradients, diminishing their applications for photo-realistic images. Orzan et al. [1] pioneered diffusion curve images (DCIs), which are curves with colors defined on either side. By diffusing these colors over the image, the final result includes sharp boundaries along the curves with smoothly shaded regions between them. Thanks to its compact nature and the ability of producing smoothly shaded images, diffusion curves quickly gain popularity in the graphics field and inspired many follow-up works, such as improving runtime performance and numerical stability [2], [3], [4], and generalization to 3D and non-euclidean domains [3], [5], [6].

Recent research has been focused on extending the expressiveness with more user control. Since diffusion curves do not allow direct manipulation of color gradients, higher-order interpolation is a possible way for gradient control. Using thin-plate splines (TPS), Finch et al. [7] extended diffusion curves to provide smooth interpolation through color constraints while omitting the diffusion curve's blur operation. Although TPS allows more user control and is able to mimic smooth shading, it often produces unwanted local extremals (hereby unpredicted colors) due to the violation of the maximal principle of a harmonic function. Moreover, solving a bi-Laplace's equation is more computationally expensive than solving Laplace's equation, and it may suffer from serious numerical issues since the system is less well-conditioned. To remove the undesired extremals, Jacobson et al. [8] proposed a non-linear optimization guided by a harmonic function. Their method allows the user to specify the exact locations and values of the local extrema, which is a highly desired feature to authoring and editing. Lieng et al. [9] proposed shading curves, which associate shading profiles to each side of the curve. These shading profiles, which can be manually manipulated, represent the color gradient out from their associated curves. Recently, Jeschke [10] proposed generalized diffusion curve images (GDCIs), which spatially blend multiple DCIs. Thanks to more degrees of freedom, GDCI is able to provide a similar expressive power as the TPS model and its solver is highly efficient and numerically stable. However, GDCI has limited support of local shading control. For example, it is difficult to move specular highlights. GDCI is often applied to image vectorization, so that the source DCIs are automatically computed from a given image.

This paper aims at overcoming the above-mentioned limitations of DCI and developing a simple yet powerful authoring tool. Towards this goal, we present a new type of vector graphics, called Poisson vector graphics (PVG), which extends DCI with non-zero Laplacians. To make a PVG, the users first sketch a set of sparse geometric

- *F. Hou is with the State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences and University of Chinese Academy of Sciences, Beijing 100190, China. E-mail: houfei@ios.ac.cn.*
- *Q. Sun is with the School of Software, Tianjin University, Tianjin 300350, China. E-mail: qian.sun@tju.edu.cn.*
- *Z. Fang and Y. He are with the School of Computer Science and Engineering, Nanyang Technological University, Singapore 639798. E-mail: zfang004@e.ntu.edu.sg, yhe@ntu.edu.sg.*
- *Y.-J. Liu and S.-M. Hu are with the BNRist, Department of Computer Science and Technology, Tsinghua Unviersity, Beijing 100084, China. E-mail: {liuyongjin, shimin}@tsinghua.edu.cn.*
- *H. Qin is with Department of Computer Science, Stony Brook University, Stony Brook, NY 11794. E-mail: qin@cs.sunysb.edu.*
- *A. Hao is with the State Key Laboratory of Virtual Reality and Technology and Systems, Beihang University, Beijng 100083, China. E-mail: ham_buaa@163.com.*

Manuscript received 4 Apr. 2018; revised 7 Aug. 2018; accepted 7 Aug. 2018. Date of publication 28 Aug. 2018; date of current version 3 Jan. 2020. (Corresponding author: Ying He.) Recommended for acceptance by P. Wonka. Digital Object Identifier no. 10.1109/TVCG.2018.2867478

Fig. 1. A Poisson vector graphics (PVG) consists of the popular diffusion curves (DCs), specifying the boundary colors, and two new types of primitives, called Poisson curves (PCs) and Poisson regions (PRs), which are associated with Laplacian of colors. The key idea of PVG is to explicitly separate colors and tones so that editing hue and tone is easy and intuitive. For example, we can make a green apple by simply changing only the DCs' color from red to green. Note that all the PCs and PRs remain unchanged. PVG natively supports seamless cloning: The water droplet can be directly copied to the target and then it automatically fits into the new background. DCs, PCs and PRs are depicted by solid lines, dashed lines and loops with hatches, respectively.

primitives (e.g., curves and/or regions) $\{\gamma_i\}_{i=1}^N$. Then, for each primitive γ_i, specify its Laplacian of color $\triangle\gamma_i = f_i$, where f_i is a *piecewise constant* function defined on γ_i. The final image $u(\mathbf{x})$ is obtained by solving the following Poisson's equation

$$\begin{cases} \Delta u(\mathbf{x}) = f, & \mathbf{x} \in \Omega \setminus \partial\Omega \\ u|_{\partial\Omega} = g, & \mathbf{x} \in \partial\Omega, \end{cases} \tag{1}$$

where Ω is the 2D domain and the constraint f partitions $\Omega = \bigcup_{i=1}^N \Omega_i$ into disjoint sub-regions, so that $f|_{\Omega_i} = f_i$ is a constant, $i = 1, \ldots, N$.

PVG is a natural extension of DCI, which are rasterized via Laplacian diffusion (i.e., solving a Laplace's equation $\Delta u = 0$). The seemingly minor change of replacing the zero Laplacian by a piecewise constant function f is indeed crucial for globally and locally controlling shading profiles, which is difficult or tedious to achieve within the diffusion curve framework. Intuitively speaking, diffusion curve images are the result of diffusing the colors defined along control curves until the color field reaches an equilibrium, which is a harmonic function. Since a harmonic function is completely determined by the Dirichlet boundary condition, diffusion curves do not allow manipulating of color gradients. Although bi-Laplacian based diffusion curves [7] support explicit gradient control, there are too many types of primitives, making it difficult to learn.

PVG is the solution of Poisson's equation, whose solution space is much larger than that of Laplace's equation, hereby providing users more control of the image. Armed with two new types of primitives, called Poisson curves (PC) and Poisson regions (PR), PVG can easily produce photorealistic effects such as specular highlights, core shadows, translucency and halos (see Fig. 1). Within the PVG framework, the users specify colors as the Dirichlet boundary condition of diffusion curves and control tone by offsetting the Laplacian of colors, where both controls are simply done by mouse click and slider dragging. The separation of color and tone not only follows the basic drawing principle that is widely adopted by professional artists (e.g., see page 58 [11]), but also allows the users to edit hue and tone in a direct and easy manner. In fact, many popular image processing software (e.g., Adobe Photoshop) provides dodging and burning tools for tone editing, following the same principle. Yet none of the existing vector graphics allows tone editing in a way as easy and intuitive as what the dodging and burning tools of Photoshop could offer. PVG provides a simple yet effective solution to bridge such gap.

PVG natively supports seamless cloning in the sense that PCs and PRs, carrying on relative colors, can automatically fit into the target background. PVG also allows users to create layers by supporting intersecting primitives (except for DC-DC intersection). Moreover, the solution of Poisson's equation is not sensitive to small change of boundary condition, therefore, PVG can apply to simple animation with small changes in-between frames. Through extensive experiments and a preliminary user study, we demonstrate that PVG is a simple yet powerful authoring tool that can produce photo-realistic vector graphics from scratches.

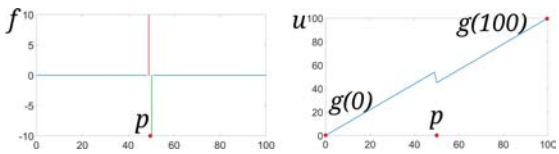
2 Related Work

This section briefly reviews the related work on diffusion curve images and gradient domain image editing.

2.1 Diffusion Curve Images

To rasterize a diffusion curve image, one needs to solve a Laplace's equation defined on the entire image domain. Since a direct solver is expensive, Orzan et al. [1] adopted a multigrid solver, which uses a coarse version of the domain to efficiently solve for the low frequency components of the solution, and a fine version of the domain to refine the high frequency components. Although being fast, this solver suffers from flickering artifacts due to the rasterization of the curves over a discrete multi-scale pixel grid. Jeschke et al. [2] used finite differences with variable step size to accelerate the convergence rate of Jacobi iterations, which guarantees the convergence to the right solution. Their method also supports zoom-in of arbitrary resolution by solving Laplace equation only for the region of interest. However, it ignores the primitives that are outside of the region but still have impact to the color diffusion inside, the solver is sensitive to boundary conditions and may produce unpleasing visual artifacts, such as color jumps. Boyé et al. [12] developed a finite element method (FEM) based bi-Laplacian solver that can directly deal with gradient constraints. However, it is only C^0 continuous on the triangle edges.

Motivated by the parallel between diffusion curve rendering and final gathering in global illumination, Bowers et al. [13] developed a stochastic ray tracing method that



(a) $f(x)$ (b) $u(x)$ and $g(x)$

Fig. 2. The 1D analogy of Poisson curve. (a) We define a Poisson curve (which becomes a point in 1D) at $p = 50$ in the domain $\Omega = [0, 100]$ and set the Laplacian constraints $f(p^+) = 10$ (red) and $f(p^-) = -10$ (green) so that the zero-sum condition is met. (b) Given the Dirichlet boundary condition $g(0) = 0$ and $g(100) = 100$, we solve $\triangle u(x) = f$ and $u|_{\partial\Omega} = g$ to compute $u(x)$.

allows trivial parallelism by using shaders and provides a unified treatment of diffusion curves with classic vector and raster graphics. However, it densely computes values even in smooth regions and sacrifices support for instancing and layering. Later, Prevost et al. [14] improved the ray tracing method by using an intermediate triangular representation with cubic patches to synthesize smooth images faithful to the per-pixel solution.

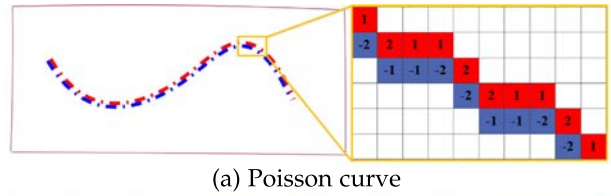
Another family of methods for rasterizing DCIs is to use boundary element method (BEM), which is based on Green's third identity and rephrases Laplace's equation as a boundary integral along control curves. Sun et al. [3] formulated the solution as a sum of Green's functions of the Laplacian. Thanks to the random-access formula, this approach is efficient and enables integrating the solution over any rectangular region and supports anti-aliasing. However, it requires pre-calculating the weights of the Green's function kernels, which depends on normal derivatives along the control curves. As a result, it can only take a DCI with fixed geometry and color constraints as input, and is not suitable for interactive authoring. Ilbery et al. [15] proposed a BEM based solver for rendering TPS vector graphics in a line-by-line manner. Sun et al. [4] presented a fast multipole representation for random-access evaluation of DCIs. Their method is able to achieve real-time performance for rasterization and texture-mapping DCIs of up to millions of curves. Note that these Green's identity based solvers are only applicable to Laplace/bi-Laplace equations.

Most DCI solvers require GPU acceleration to achieve real-time performance. Aiming at efficiently rendering DCIs on devices with only a CPU, Pang et al. [16] developed a mesh-based approach, which sets the diffusion curves as constraints in the triangulation and employs mean value coordinate-based interpolant to estimate vertex colors. Their algorithm supports random access evaluation, but the produced DCI is only an approximation.

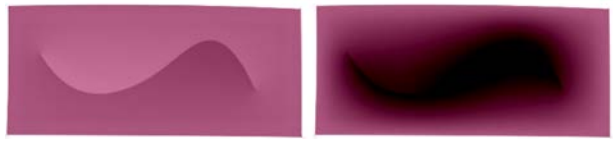
2.2 Gradient Domain Image Editing

Solving 2D Poisson's equations plays an important role in various image processing tasks, such as composition [17], [18], alpha matting [19], colorization [20], filtering and relighting [21]. Fast Fourier transformation is a popular method to solve Poisson's equation in rectangular domains due to its high performance. For irregular domains, one often adopts the finite element method (FEM), which subdivides the domains into smaller parts and solves a large sparse linear system.

Poisson image editing [17] enables seamless cloning by matching the gradients of the source and the target images,



(a) Poisson curve



(b) $f_+ = 41 = -f_-$ (c) $f_+ = 41, f_- = -45$

Fig. 3. A Poisson curve is a two-sided curve created by cubic B-splines, where each side is associated with a Laplacian constraint. The numbers in the close-up view (a) are the weights to discretize the Laplacian operator and also ensure the zero sum condition. Notice that the zero sum is a necessary condition for local shading control. If it is not satisfied, unwanted artifacts occur (c). The rectangular boundary in (a) is a diffusion curve, which specifies the boundary condition of the Poisson's equation.

which is formulated as a Poisson's equation. However, directly solving such an equation is computationally expensive. Agarwala [18] improved the scalability by using quadtrees to substantially reduce both memory and computational requirements. McCann and Pollard [22] presented a multigrid Poisson solver on GPUs, with which they can achieve real-time performance. Their method allows the users to edit the color gradient along curves, whereas our method supports gradient domain editing for both curves and regions. Using mean value coordinates [23], Farbman et al. [24] developed a closed-form Laplacian solver for seamless cloning of opaque images.

3 Poisson Vector Graphics

Poisson vector graphics extends the DC framework by adding two new geometric primitives, namely Poisson curves and Poisson regions. The former is to model color discontinuity across curves, while the latter is to design smooth shading within the user specified regions. Mathematically speaking, PVG solves a Poisson's equation with piecewise constant Laplacians f. As the Laplacian diffusion, PVG takes DCI as a special case with $f \equiv 0$. Extending the zero Laplacian to a piecewise constant function f brings 4 unique advantages. First, the users can explicitly control the local and/or global shading profiling via manipulating f (which is a scalar for each color channel). Second, the users can easily control the color extrema, which are either on the curves (for PC and DC) or inside a region (for PR). Third, PVG allows intersection among the geometric primitives (except DC and DC). Fourth, PVG natively supports seamless cloning.

Although a PVG can have an arbitrary number of PCs and PRs, it must contain at least one diffusion curve, serving as the boundary condition g. In the following, we detail Poisson curves and Poisson regions.

3.1 Poisson Curves

Similar to diffusion curves, a Poisson curve γ is two-sided, denoted by γ^+ and γ^-, and can be either open or closed. We associate each side a Laplacian value, denoted by f_+ and f_- respectively, such that $f_+ + f_- = 0$ (see Fig. 3). To discretize

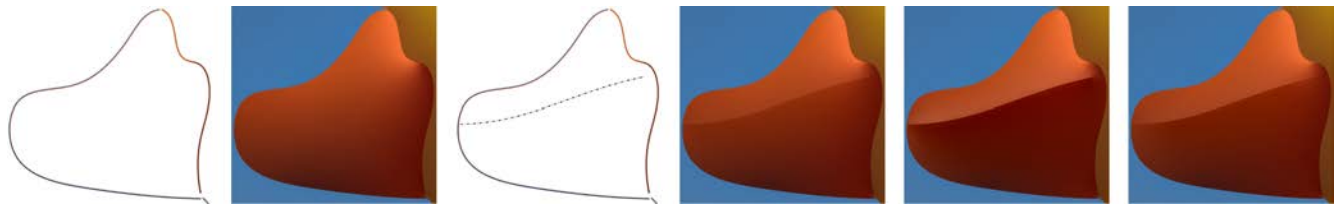


Fig. 4. Poisson curves are used to model color discontinuity in a bounded region. (a)-(b): A diffusion curve is used to create the beak of the Rubber Duck. (c)-(e): We can create a sharp edge using a Poisson curve γ (dashed line). Increasing the Laplacian constraint $|f|$ makes a stronger edge. (f) We can also use a spatially varying constraint to define sharp edges with varying strength.

γ, we assign each pixel of γ^+ or γ^- a weight: a pixel receives a weight $\pm n$ if has exactly n neighboring pixels on the other side of γ. These non-constant weights are used to define the discrete Laplacian and also ensure that they add up to zero. The requirement for zero sum is for *local* shading control. To explain this, consider a region $D \supset \gamma$ that bounds the diffusion of γ. The divergence theorem $\iint_D \Delta u dA = \oint_{\partial D} \nabla u \cdot \mathbf{n} dl$ relates the double integral of Δu to the line integral of ∇u. Since the color function u remains unchanged on ∂D, the line integral of ∇u is a constant, implying that the double integral of Laplacian is also a constant. Based on this observation, we set $(\Delta u)|_{\gamma^+} + (\Delta u)|_{\gamma^-} = 0$ (see Figure 3). Although the zero-sum is only a *necessary* condition to ensure local shading control, it works pretty well in our experiments. To better understand the concept, we show the 1D analogy of Poisson curve in Fig. 2.

The zero-sum condition is hidden to users, who simply drag over a slider to specify f_+. As long as $f_+ \neq 0$, the Poisson curve corresponds to a sharp edge. The larger the value $|f_+|$, the stronger the sharp feature. Since we model Poisson curves as cubic splines, the users can also specify spatially varying constraints $(\Delta u)|_\gamma$ by setting weights on the control points. See Fig. 4.

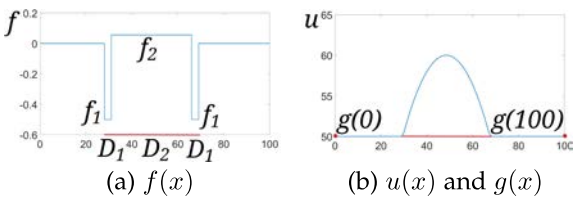


Fig. 5. The 1D analogy of Poisson curve is an interval $[30, 70] \subset \Omega$. (a) Define the Laplacian constraints with zero-sum $f_1 A(D_1) + f_2 A(D_2) = 0$. (b) The function $u(x)$ with the Dirichlet boundary condition $g(0) = g(100) = 50$.

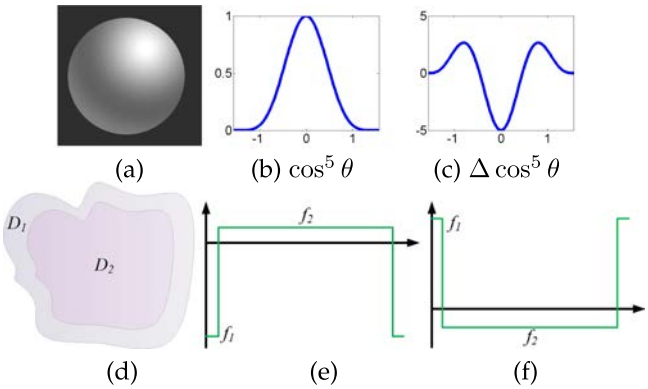


Fig. 6. (a) A shiny sphere rendered using the Phong illumination model. (b) The specular highlight is $\cos^k\theta$, where θ is the angle between the viewing vector and the reflection vector, and the exponent k is the shininess constant. (c) The Laplacian of $\cos^k\theta$ is U-shaped. (d) A Poisson region R consists of two disjoint sub-regions $D = D_1 \cup D_2$, each of which is associated with a Laplacian constraint f_i. We set $f_1 \propto \frac{\lambda_1}{A(D_1)}$, where λ_1 is the user-specified value and $A(D_1)$ is the area of D_1. We then compute $f_2 = -f_1 \frac{A(D_1)}{A(D_2)}$. (e) The f_is are to approximate the U-curve of $\Delta\cos^k\theta$, simulating specular highlights. (f) Similarly, the f_is are to model the Laplacian of core shadows.

3.2 Poisson Regions

We develop Poisson regions to produce photorealistic effects, such as specular highlights, core shadows, translucency and halos. Observe that the Laplacian of a specular highlight is a "bell" shaped curve (see Fig. 6). To discretize such a function, we decompose a Poisson region into two disjoint sub-regions D_1 and D_2, where the outer part D_1 is relatively thin so that it preserves the geometry of ∂D well. We assign a constant constraint f_i to each sub-region D_i, so that they have opposite signs $f_1 f_2 < 0$ and satisfy $\sum_{i=1}^{2} \iint_{D_i} f_i dA = 0$ for local shading control. As Fig. 6 shows, f_is with decreasing values form a U-shaped curve, which simulate the Laplacian of specular reflection, while f_is with increasing values are bell-shaped, which simulate the Laplacian of core shadows. To simulate halo, we also provide additional control that offsets f_is. Fig. 5 shows the 1D counterpart of Poisson region (which is an interval).

To determine the sub-regions D_1 and D_2, we take the boundary ∂D as the source and compute the euclidean distance transform. Let d_{max} be the maximal distance to the boundary. We then define $D_1 = \{\mathbf{x} | d(\mathbf{x}) \leq 0.05 d_{max},\ \mathbf{x} \in D\}$ and $D_2 = D \setminus D_1$ (see Fig. 6d). To produce halos with small change of gradients in a Poisson region, we also allow the user to add an increment δ to f_i (see Fig. 7).

There are 2 differences between PC and PR: First, a PC is a double-sided curve, which can be either open or closed, and a PR is a region whose boundary is closed. Second, PC

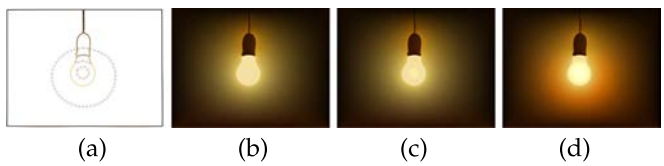


Fig. 7. We can produce halos in a Poisson region by adding each Laplacian constraint f_i an increment δ, where $f_i + \delta$, $i = 1, 2$, have opposite signs. (a) There are two Poisson regions (loops with hatches) in this PVG. (b) Rendering without PR, i.e., $f_i = 0$, $\delta = 0$. (c) Rendering without increments, i.e., $f_i \neq 0$, $\delta = 0$. (d) Rendering with increments i.e., $f_i \neq 0$, $\delta \neq 0$.

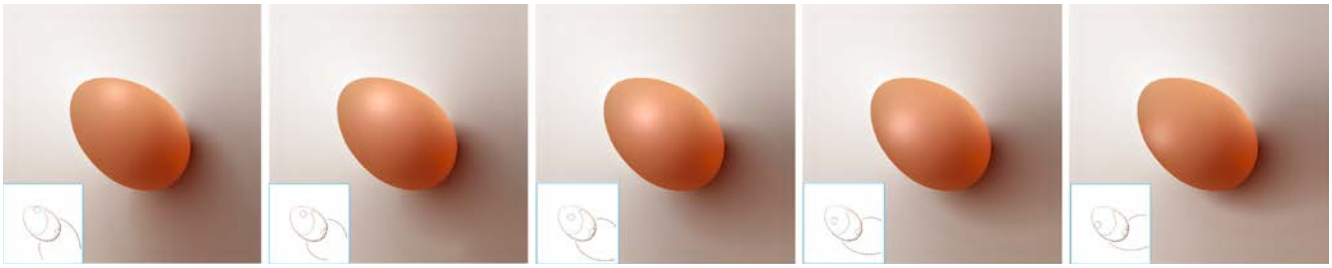


Fig. 8. PVG simulates a specular highlight and core shadow using Poisson regions. See the accompanying video for the animation.

is mainly designed for modeling color discontinuity, whereas PR is to produce smooth shadings, such as specular highlights and core shadows (see Fig. 8).

4 Features

PVG has 3 salient features: explicit separation of hues and tones, supporting layers by allowing intersecting primitives (except for DC-DC intersection), and native support for seamless cloning, that are favorable for authoring. It is worth noting that none of the existing methods (e.g., DCI, TPS, GDCI) has all of the above-mentioned features.

4.1 Separation of Hues and Tones

Although both PCs and DCs are able to produce color discontinuity across the curves, they are fundamentally different. Using DCs, the users explicitly specify the boundary condition (i.e., colors) on both sides of a curve. To change hue either locally or globally, the users have to re-assign the colors for all the curves involved, which is tedious. In contrast, PVG separates colors and tones by specifying the Laplacian constraint of PC, which is a *relative* value. Since the color of a PC is determined by the surrounding DC. Changing hue does not require any modification of PCs. See Fig. 9 for an example of local hue editing. Poisson regions also enable local tone control, since the area integral of the Laplacian on a PR is zero. See Fig. 10 for an example of changing a specular highlight.

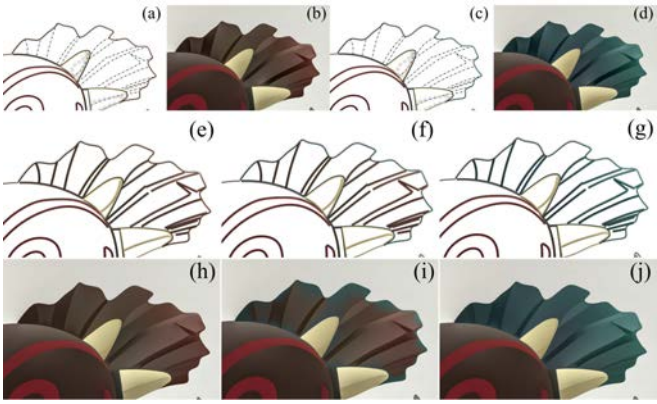


Fig. 9. Changing hue in the Warrior's cloak, The region of interest R is bounded by a closed diffusion curve, whose inner side color is dark brown. Row 1 (PVG): We simply changed the boundary color of ∂R to blue, then the Poisson curves inside R (dashed lines in (a) and (c)), still with the same Laplacian constraints, can automatically adjust to the new boundary condition so that the hues in R are coherent with the boundary color. Rows 2 and 3 (DCI): Replacing each Poisson curve to a diffusion curve with proper color conditions (see colored lines in (e)), one can generate a DCI image (see (h)) similar to the PVG image in (b). However, simply changing the boundary color of ∂R (see (f)) in the DCI produces strange hue, due to the significant difference between the colors of the interior DCs and the boundary DC. To fix this, one has to manually adjust the boundary condition for each interior DC (see (g)), which is tedious and error prone.

4.2 Layers

When two diffusion curves are intersecting, their attached colors compete with each other, often leading to undesired effects (see Fig. 11a). Hence, within the diffusion curve framework, the users have to split those curves into disjoint segments with different colors and re-adjust the boundary colors for each segment to obtain smooth colors. This extra operation becomes a serious drawback to designers, who have to deal with a large amount of short segments and their constraints. In contrast, PCs and PRs can intersect any type of primitives, including themselves. Note that Laplacian is a linear operator, we have $\Delta(f(x)+g(x)) = \Delta f(x) + \Delta g(x)$ for any differentiable functions f and g. When PC and PR intersect each other, we simply add their Laplacian constraints for the intersection point. In fact, the DC-DC intersecting issue can also be partially solved if one DC is converted to a PC. Such a strategy is adopted in seamless cloning (see Fig. 13). This feature not only simplifies the drawing process, but also allows the users to use layers, which is a powerful technique for making complex drawings.

Fig. 10. The user can easily change tone by manipulating the Laplacian constraints of PRs. See the accompanying video.

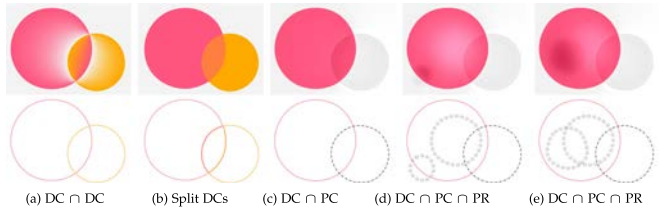


Fig. 11. Intersecting primitives. (a) The colors attached to two intersecting DCs compete with each other, leading to undesired artifacts. (b) Within the diffusion curve framework, the users have to split the curves into disjoint segments and re-adjust the boundary colors for each segment in order to produce smooth colors. (c)-(e) In contrast, PVG allows all types of intersection except for DC-DC intersection.

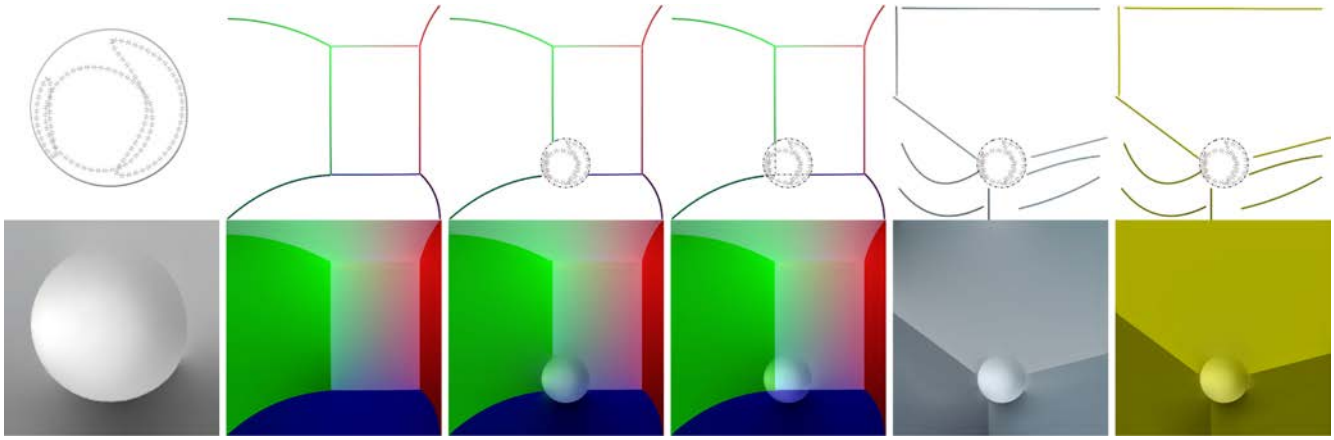


Fig. 12. Seamless cloning with PVG. We convert the DC (the boundary of the sphere) of the source PVG to a Poisson curve and copy it together with the PRs inside to the destination region. Unlike diffusion curve based cloning, our method does not need to re-compute the boundary color of the DC. Since the source PVG can *automatically* adapt to the new environment, making the cloning task easy. We can also edit the hue by changing the boundary color of DCs.

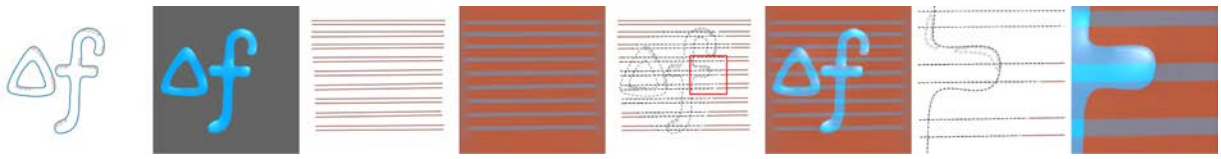


Fig. 13. Seamless cloning with transparent PVG, where the source and target gradients are mixed. The DCs in the source PVG (colored solid lines in (a)) are converted into PCs (see the dashed lines in (c) and (d)). The DCs that are outside the region of interest in the target PVG remain unchanged.

4.3 Seamless Cloning

Pérez et al. [17] proposed a set of image editing tools that permits seamless importation of opaque and transparent source image regions into a destination region. The key idea is to compute a function that agrees with the target image on the boundary of the target region, whose gradient field is as close as possible to that of the source image. Pérez et al. [17] showed that such a function is the solution of a Poisson's equation with Dirichlet boundary condition which specifies the Laplacian over the domain of interest. Since PVG allows the users to directly specify the Laplacian for all its primitives, it natively supports gradient domain editing.

We use subscripts s and t to distinguish the source and target PVGs for cloning. The user clones a region $P_s \subset D_s$ in the source to a region $P_t \subset D_t$ in the target, where P_s and P_t have identical boundary. Notice that the source region P_s may contain an arbitrary number of DCs, PCs and PRs.

If the source region P_s is opaque, we clear all the primitives in the target region P_t. Since PCs and PRs carry the Laplacian of colors (i.e., offset), they can be directly copied to the target region P_t. Therefore, we only need to convert each DC γ inside P_s to a PC whose Laplacian constraint is readily available with $f|_\gamma = (\triangle u_s)|_\gamma$.

If P_s is transparent, we keep the original PCs and PRs inside the target region P_t and convert the remaining DCs of P_t to PCs. Then we take the sum of Laplacians of P_s and P_t as the new Laplacian constraint for P_t. The pseudo code of seamless cloning is shown in Algorithm 2.

It is worth noting that PVG based seamless cloning is fundamentally different from the DC counterpart [2], [4] for 2 reasons. First, our method can *directly* combine the primitives of the source and target PVGs, since the source DCs are converted to PCs, which can *automatically* adapt to the target PVG. In contrast, the DC based seamless cloning needs to re-compute the boundary color of the source DCs using the information of the target domain. Figs. 12c and 12e shows that we can easily clone a *opaque* source PVG to different targets without using the target domain information. Second, the users can easily edit tone and hue in the composite PVG (Figs. 12d and 12f), taking advantage of the converted PCs. In contrast, to change the tone in the DC based framework, the users have to re-compute the boundary colors of *all* affected diffusion curves, which is tedious.

Algorithm 1. Seamless Cloning with PVG

Require: Source PVG $u_s : D_s \to \mathbb{R}$, $\triangle u_s = f_s$ and $u_s|_{\partial D_s} = g_s$; target PVG $u_t : D_t \to \mathbb{R}$, $\triangle u_t = f_t$ and $u_t|_{\partial D_t} = g_t$; a patch $P_s \subset D_s$ and a patch $P_t \subset D_t$ with identical boundary.
Ensure: The blended PVG u_t where P_s is seamless cloned to P_t.
1: **if** the source region P_s is opaque **then**
2: Clear the destination region P_t, $\forall \mathbf{p} \in P_t$, $f(\mathbf{p}) = 0$
3: **else**
4: **for** each DC $\gamma \subset P_t$ **do**
5: Convert γ to a PC with Laplacian $f|_\gamma = (\triangle u_t)|_\gamma$;
6: **end for**
7: **end if**
8: Copy the PCs and PRs in P_s to P_t;
9: **for** each DC $\gamma' \subset P_s$ **do**
10: Convert γ' to a PC with Laplacian $f|_{\gamma'} = (\triangle u_s)|'_\gamma + (\triangle u_t)|'_{\gamma'}$;
11: Copy the PC to P_t;
12: **end for**

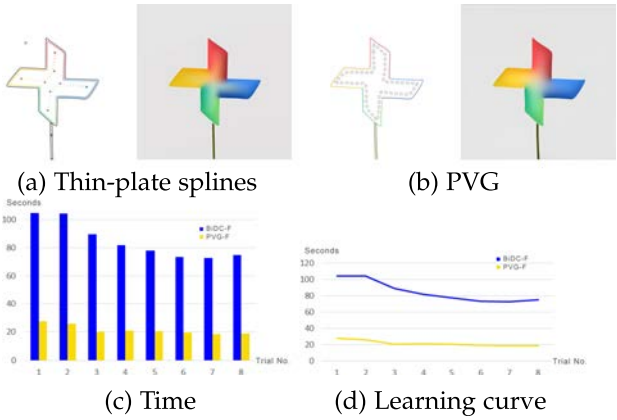


Fig. 14. Experiment #1: Local tone control.

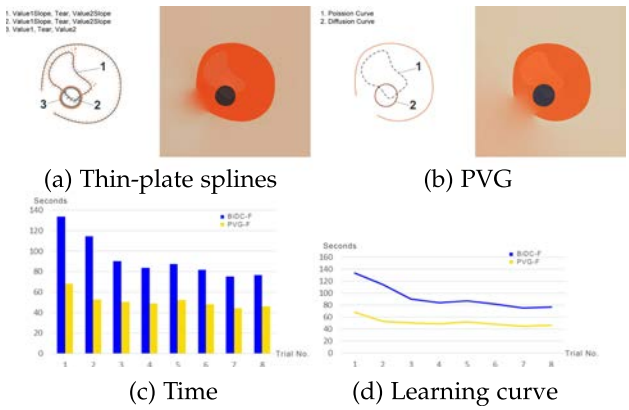


Fig. 15. Experiment #2: Handling intersecting primitives. DC and TPS do not allow intersecting primitives. Therefore, one has to partition the closed curve into two disjoint segments (labeled 1 and 2 in (a)). With PVG, one can draw Poisson curves/regions anywhere.

Fig. 16. PVG supports simple animation, where the change of frame-to-frame geometry is small. See the accompanying video.

5 User Study

We conducted a preliminary user study to explore benefits (efficiency and usability) of PVG and compared it with TPS based vector graphics developed by Finch et al. [7]. They developed 5 types of basic curves, namely, tear, crease, slope, contour and value curves, for controlling color values and their directional derivatives. Since the basic curves, in general, cannot be used alone, users need to combine a few types of curves to produce the desired effects. Their system consists of more than 50 combinations of the basic curves. On one hand, it is highly flexible and provides users advanced controls to produce photo-realistic images. On the other hand, the learning curve is usually steep, due to its complexity. In contrast, PVG provides only three types of primitives with simple purposes—DC is used for hue and PC and PR are for tone.

Fig. 17. Poisson regions produce translucency on the balloons.

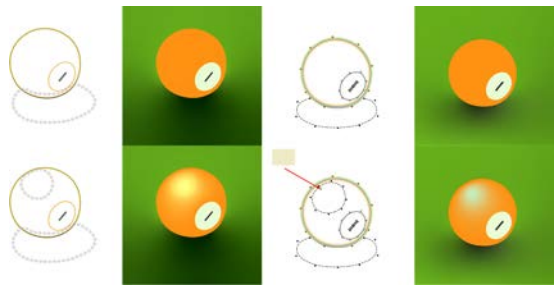
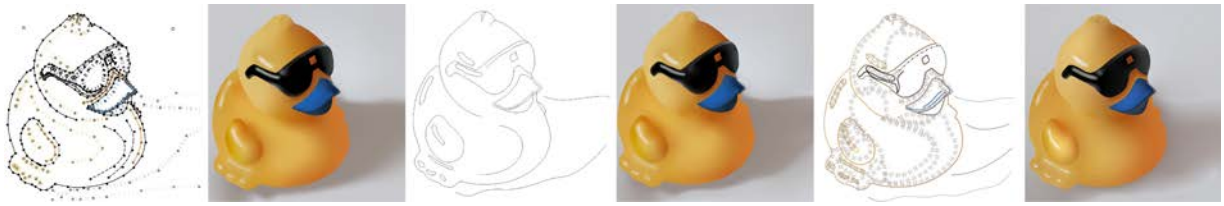


Fig. 18. Comparing with TPS vector graphics [7]. To add a specular highlight to the billiard ball, the PVG user sketches a PR and then tunes its Laplacian to increase/decrease the highlight. No color manipulation is required in this process. The TPS user specifies a closed contour curve to bound the highlight region and then adds a value point inside to control the color (highlighted by the red arrow). He assigns light yellow to the value point to match the PVG result. However, since the value point is close to the contour, color oscillation occurs, leading to unexpected color (green) in the specular highlight.

Fig. 19. The colors produced by shading curves are not as realistic as those of PVG and diffusion curves. Image courtesy of Lieng et al.

We recruited 12 professional 2D artists who are from 3 different studios/institutions and with 9.3 years painting experience on average. Before going to the tasks in the experiments, we introduced the key concepts of DC, PR, PC and TPS to the participants, and gave them 20 minutes to familiarize themselves with both pieces of software. After that, they were asked to complete two painting tasks. To help the participants complete the tasks, we showed them the expected results.

We designed two tasks to evaluate local shading control and permission of intersection among geometric primitives in the user study. In experiment #1, the participants were given a colorful windmill and they were asked to add a highlight on it (see Fig. 14). With TPS, they sketched two VS curves—the compound value and slope curve—each curve has at least 4 control points for specifying the boundary colors. Note that, four colors are the least number of colors to simulate the rich colors of the area we provided. With PVG, participants sketched one Poisson region and then specified



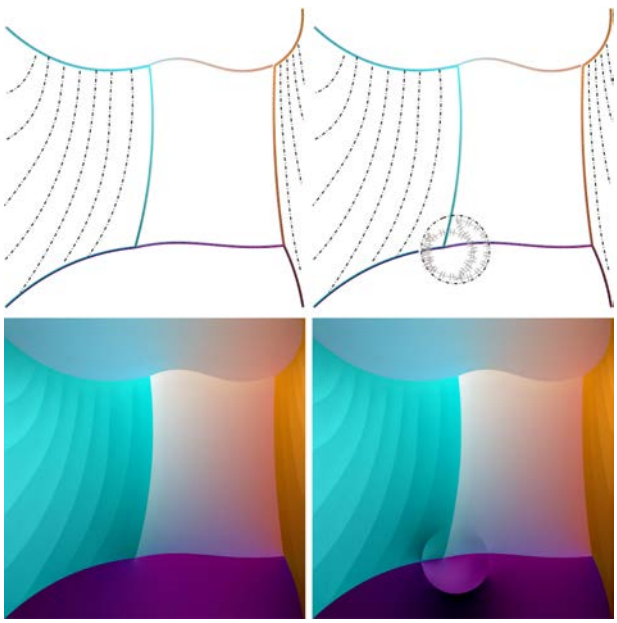
(a) TPS vector graphics (b) GDCI (c) PVG

Fig. 20. The TPS vector graphics [Finch et al. 2011] consists of 53 primitives in 13 categories. Our PVG, producing a similar image, contains 51 primitives in only 3 categories: 26 DCs, 1 PCs and 24 PRs. GDCI consists of 28 diffusion curves and each DC is associated with 2 different boundary colors.

its Laplacian constraint using slider, which is much easier than specifying *discrete* colors. Also, it is non-intuitive to *explicitly* specifying colors for *shaded* objects with spatially-varying colors but the same tone. As Fig. 14c shows, PVG reduces roughly 75 percent of drawing time as compared to DC and TPS.

In experiment #2, they were given a "ladybug" and required to add a highlight on its back (see Fig. 15). To simplify the task, the ladybug has only one spot. Since the highlight crosses the black spot, the participants had to partition DCs and TPS into disjoint segments and then specify their colors separately. With PVG, they simply sketched one PC in a single stroke and specified only one Laplacian constraint. As a result, PVG saves 40 percent of the drawing time as compared to DC and TPS (see Fig. 15e).

To evaluate the ease of learning, we asked the participants to repeat each experiment 8 times. As the learning curves in Figs. 14d and 15f show, the PVG curves tend to stabilize in only 3 or 4 trials, whereas the TPS curves take longer, from trial 6 onwards, to become stable implying that PVG is easier to learn compared to TPS. See the accompanying video for details.



Base layer Base & top layers

Fig. 21. PVG allows users to create layers. In a typical design, diffusion curves are placed in the base layer. Then the users can create additional layers with an arbitrary number of Poisson curves and Poisson regions, which can intersect each other and/or the diffusion curves.

The participants commented that PVG faithfully follows the basic painting principle by separating color and tone. In painting theory [25], tones are comprised of highlight, halftone, core shadow, reflected light and cast shadow, which are the key factors to produce photorealistic rendering. With PC and PR, they can simulate various types of tones and control them in an easy and intuitive manner (e.g., experiment #1). Although raster graphics naturally supports this painting style (e.g., the dodging and burning in Adobe Photoshop), this is the first time that they saw it in vector graphics. They also commented that PVG is more flexible than DC and TPS, since it allows geometric primitives intersecting each other. This feature enables them to use layers in complex drawings, such as experiment #2.

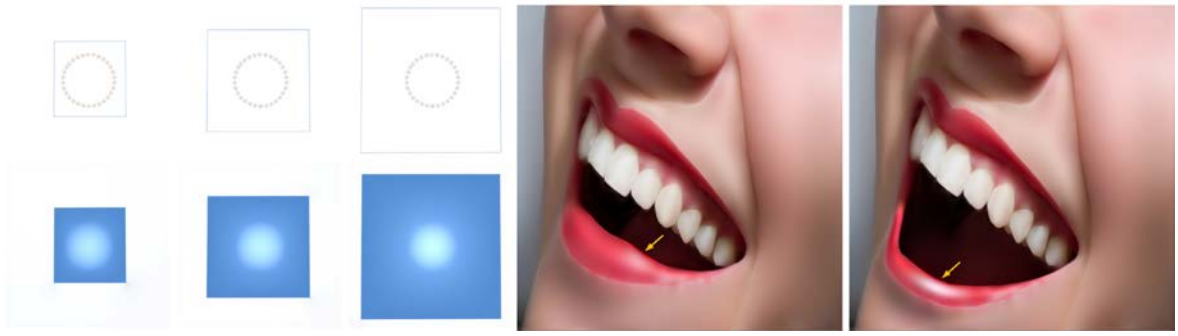
6 Experimental Results

PVG is able to express both cartoon-like images (Fig. 25) and photorealistic images (Fig. 24). Poisson curves, as a double-sided curve with opposite Laplacian constraint, create high contrast (i.e., color discontinuity) across the curve, which is desired to model object boundaries and sharp features. Poisson regions, on the other hand, have controllable "soft" boundaries, are effective to produce highlights, core shadows, halos and transparency. See Figs. 8, 7 and 17. In addition, we allow PRs to intersect each other and other types of primitives, providing more flexibility and expressive power to mimic complex shadings in photorealistic images.

It is known that the solution of Poisson/Laplace's equation is less sensitive to the change of boundary condition than that of bi-Laplace equation. Therefore, PVG can be applied to animation with small inter-frame changes of colors and/or geometries. See Figs. 8 and 16 and the accompanying video for the animation examples. Bi-Laplacian based vector graphics often suffers from visual artifacts, such as color jumps.

Comparing with diffusion curves, PVG is superior in tone and hue control, since PCs and PRs, constraining Laplacians of colors, are *loosely* coupled with the neighboring diffusion curves. Fig. 9 shows an example where the user wants to change the color of a region of interest R. For DCI, the user has to manually adjust the colors for all DCs inside R. With PVG, the user only needs to change the color of the boundary DC ∂R, then the Poisson curves inside R, still with the same Laplacian constraint, automatically adapt to the new boundary color.

It is known that unintended derivative discontinuities may arise in DCI [7]. Although a post-process image blurring [1] can reduce the artifacts, it does not allow explicit control over the value or position of color extrema. As



(a) PVG (b) Original GDCI (c) GDCI with modified curve geometry

Fig. 22. Color dependency on the curve geometry. We scale the boundary diffusion curve and fix the Poisson region inside. Since PR specifies only the *relative* value, the resulting colors scale with the curve geometry. If the change of curve geometry is significant, over saturation artifact occurs. We observe that GDCI's color extrema also depends on the curve geometry. The GDC image is courtesy of S. Jeschke.

discussed above, PVG is C^1 continuous everywhere except where creases (diffusion curves) are explicitly specified.

As mentioned in Section 4, diffusion curves are not allowed to intersect each other, since their associated colors are competing. In contrast, PCs and PRs can intersect any type of primitives, including themselves. Moreover, by converting DC to PC, PVG can also solve the DC-DC intersection issue (see Figs. 12 and 13). This feature enables the PVG users to create layers and provides them more flexibility in the design process. Take the Apple (Fig. 1) as an example. Note that there are a large number of intersections among the primitives.

Comparison with Constrained Diffusion Curves. Bezerra et al. [26] proposed several techniques, such as diffusion barriers, diffusion anisotropy, and spatially varying color strength to control the diffusion process. Their approach is able to diffuse both colors and normal maps, hereby producing interesting non-photorealistic effects. However, the constrained diffusion has two limitations. First, it is non-intuitive to specify the boundary condition for normals. Second, since normals are diffused within a closed diffusion curve, some artifacts (such as color discontinuity) may occur on the region boundary. With PVG, the users can directly produce specular reflection using Poisson regions.

Comparison with Thin-Plate Splines. In contrast, PVG provides only three types of primitives, each of which has a clear definition and purpose. A preliminary user study shows that PVG is more intuitive and easy to use than TPS. Moreover, as pointed out in [8], [9], the biharmonic functions can be negative and have prevalent local extrema, leading to unstable colors (see Fig. 18). Therefore, TPS is not applicable to animation. As Fig. 16 shows, PVG can be applied to animation with small frame-to-frame change of geometry and colors. PVG naturally supports seamless cloning, whereas it would be difficult for TPS, since the bi-Laplacian model does not fit for gradient domain editing.

Comparison with Shading Curves. Lieng et al. [9] proposed shading curves to simulate chiaroscuro drawing, providing strong contrasts between light and dark. Users first draw areas of constant tone with curves, fill in each individual area with constant color and specify the influence of that color to adjacent areas. Colors are then smoothed out with shading profiles, which are associated with each side of the curve. Finally, shading curves are converted to 3D control meshes and rendered as Catmull-Clark subdivision surfaces. Shading curves allow explicit control over color gradients, however, the generated image is not as realistic as PVG (see Fig. 19). Moreover, due to the limitation of their region growing algorithm, shading curves are not able to handle curves with high curvatures and/or intersecting curves.

Comparison with Generalized Diffusion Curves. Generalized diffusion curves [10] takes multiple conventional DCIs as input and then spatially blend them to produce the final result. The source DCIs often share the same geometric primitives, but they are associated with different boundary colors. Technically speaking, one DCI determines the colors for pixels which are close to the diffusion curves, whereas the other controls the region away from the curves. Thanks to the more degrees of freedom provided in the source DCIs, GDCI can provide a similar expressive power of TPS [7]. However, GDCI has limited support of local shading control. For example, it is difficult for the users to move a specular highlight, whereas doing so within PVG is straightforward (see Fig. 8).

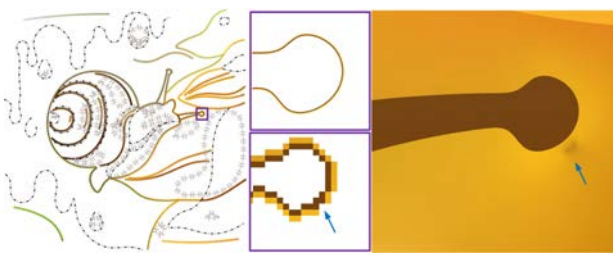


Fig. 23. Color artifacts due to inaccurate curve discretization. The snail's feeler is a two-sided diffusion curve with inner boundary color dark brown and outer boundary color yellow. However, the region is not completely bordered by the *discretized* yellow curve, resulting in "color leaking" artifacts.

Photo PVG PVG image

Fig. 24. PVG is able to produce photo-realistic images with sparse geometric primitives. This PVG consists of 85 diffusion curves (solid lines), 71 Poisson curves (dashed lines) and 24 Poisson regions (loops with hatches).

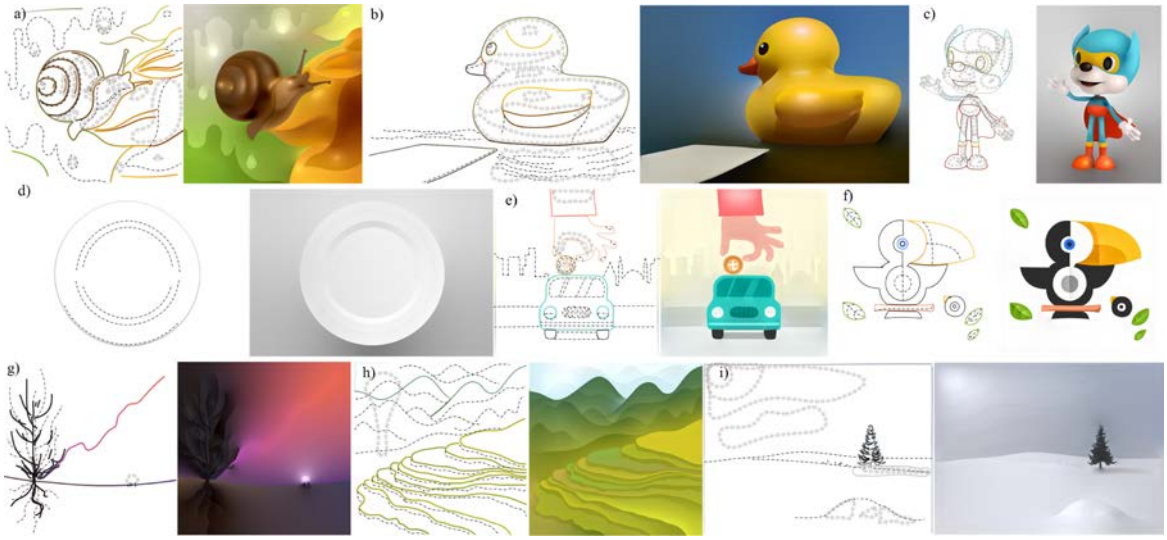


Fig. 25. A gallery of poisson vector graphics.

Since GDCI blends two harmonic functions using affine combination, it guarantees that the color range is invariant under the uniform scaling of curve geometry. However its color extrema are geometry dependent (see Figs. 22b and 22c). PVG's color extrema also depend on the geometry of curves. As Fig. 22a shows over saturation artifacts occur when the curves are changed significantly. Similar to DCI, GDCI does not allow intersecting primitives. In contrast, PCs and PRs can intersect any type of primitives, including themselves. As a result, the users can create layers in PVG, as shown in Fig. 21.

7 Conclusion

We presented Poisson vector graphics, an extension of the popular diffusion curves, for generating smooth-shaded images. Armed with two new types of primitives, Poisson curves and Poisson regions, PVG can easily produce photorealistic effects such as specular highlights, core shadows, translucency and halos. PVG distinguishes itself from the existing drawing tools by separating color and tone, which brings three unique features, i.e., local tone and hue editing, permission of intersecting primitives (except for DC-DC intersection) to enable layers, and native support for seamless cloning. Our preliminary user study confirms that PVG is more intuitive and easy to use than diffusion curves and its variants.

Limitations. To render PVG images, we need to discretize all geometric primitives using a quad-tree data structure, even though they are smooth B-spline curves. The computed color function relies on the user-specified resolution and the error increases when the resolution decreases. Discretization errors often occur near diffusion curves with high curvature, resulting in "color leaking" artifacts (see Fig. 23). A possible solution is to adopt an accurate and robust curve discretization algorithm (e.g., [2]). PVG is able to handle animation with small frame-to-frame change of geometry. If the change of curve geometry is significant, over saturation artifacts occur (see Fig. 22). Since color dependency on curve geometry is a common problem to other *non-harmonic* color functions, it deserves further investigation and improvement. The zero-sum condition is only a necessary condition to ensure the *local* hue and tone control for PCs and PRs. It is desired to derive a sufficient condition.

Acknowledgments

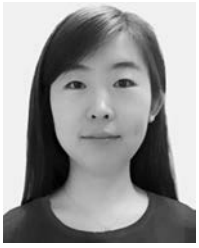
This project was partially supported by the Alibaba-NTU Singapore Joint Research Institute, Singapore Ministry of Education Grant RG26/17, Special Plan for the Development of Distinguished Young Scientists of ISCAS (Y8RC535018), NSFC Grants(61872347, 61725204, 61772016, 61532002, 61661146002, 61672149, 61672077, 61702363), USA NSF IIS-1715985, and CAS Key Research Program of Frontier Sciences (QYZDY-SSW-JSC041). F. Hou and Q. Sun contributed equally to the paper.

References

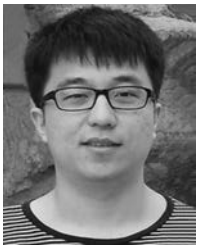
[1] A. Orzan, A. Bousseau, H. Winnemöller, P. Barla, J. Thollot, and D. Salesin, "Diffusion curves: A vector representation for smooth-shaded images," *ACM Trans. Graph.*, vol. 27, no. 3, pp. 92:1–92:8, 2008.
[2] S. Jeschke, D. Cline, and P. Wonka, "A GPU Laplacian solver for diffusion curves and Poisson image editing," *ACM Trans. Graph.*, vol. 28, no. 5, pp. 116:1–116:8, 2009.
[3] X. Sun, G. Xie, Y. Dong, S. Lin, W. Xu, W. Wang, X. Tong, and B. Guo, "Diffusion curve textures for resolution independent texture mapping," *ACM Trans. Graph.*, vol. 31, no. 4, pp. 74:1–74:9, 2012.
[4] T. Sun, P. Thamjaroenporn, and C. Zheng, "Fast multipole representation of diffusion curves and points," *ACM Trans. Graph.*, vol. 33, no. 4, pp. 53:1–53:12, 2014.
[5] S. Jeschke, D. Cline, and P. Wonka, "Rendering surface details with diffusion curves," *ACM Trans. Graph.*, vol. 28, no. 5, pp. 117:1–117:8, 2009.
[6] K. Takayama, O. Sorkine, A. Nealen, and T. Igarashi, "Volumetric modeling with diffusion surfaces," *ACM Trans. Graph.*, vol. 29, no. 6, 2010, Art. no. 180.
[7] M. Finch, J. Snyder, and H. Hoppe, "Freeform vector graphics with controlled thin-plate splines," *ACM Trans. Graph.*, vol. 30, no. 6, pp. 166:1–166:10, 2011.
[8] A. Jacobson, T. Weinkauf, and O. Sorkine, "Smooth shape-aware functions with controlled extrema," *Comput. Graph. Forum*, vol. 31, no. 5, pp. 1577–1586, 2012.
[9] H. Lieng, F. P. Tasse, J. Kosinka, and N. A. Dodgson, "Shading curves: Vector-based drawing with explicit gradient control," *Comput. Graph. Forum*, vol. 34, no. 6, pp. 228–239, 2015.
[10] S. Jeschke, "Generalized diffusion curves: An improved vector representation for smooth-shaded images," *Comput. Graph. Forum*, vol. 35, no. 2, pp. 71–79, 2016.
[11] R. Hale, *Drawing Lessons from Great Masters*. New York, NY, USA: Watson-Guptill Publications, 1964.
[12] S. Boyé, P. Barla, and G. Guennebaud, "A vectorial solver for free-form vector gradients," *ACM Trans. Graph.*, vol. 31, no. 6, pp. 173:1–173:9, 2012.

[13] J. C. Bowers, J. Leahey, and R. Wang, "A ray tracing approach to diffusion curves," in *Proc. 22nd Eurographics Conf. Rendering*, 2011, pp. 1345–1352.
[14] R. Prévost, W. Jarosz, and O. Sorkine-Hornung , "A vectorial framework for ray traced diffusion curves," *Comput. Graph. Forum*, vol. 34, no. 1, pp. 253–264, 2015.
[15] P. Ilbery, L. Kendall, C. Concolato, and M. McCosker, "Biharmonic diffusion curve images from boundary elements," *ACM Trans. Graph.*, vol. 32, no. 6, pp. 219:1–219:12, 2013.
[16] W. Pang, J. Qin, M. Cohen, P. Heng, and K. Choi, "Fast rendering of diffusion curves with triangles," *IEEE Comput. Graph. Appl.*, vol. 32, no. 4, pp. 68–78, Jul./Aug. 2012.
[17] P. Pérez, M. Gangnet, and A. Blake, "Poisson image editing," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 313–318, 2003.
[18] A. Agarwala, "Efficient gradient-domain compositing using quadtrees," *ACM Trans. Graph.*, vol. 26, no. 3, pp. 94:1–94:5, 2007.
[19] J. Sun, J. Jia, C.-K. Tang, and H.-Y. Shum, "Poisson matting," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 315–321, 2004.
[20] A. Levin, D. Lischinski, and Y. Weiss, "Colorization using optimization," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 689–694, 2004.
[21] P. Bhat, C. L. Zitnick, M. Cohen, and B. Curless, "Gradientshop: A gradient-domain optimization framework for image and video filtering," *ACM Trans. Graph.*, vol. 29, no. 2, pp. 10:1–10:14, 2010.
[22] J. McCann and N. S. Pollard, "Real-time gradient-domain painting," *ACM Trans. Graph.*, vol. 27, no. 3, pp. 93:1–93:7, 2008.
[23] M. S. Floater, "Mean value coordinates," *Comput. Aided Geom. Des.*, vol. 20, no. 1, pp. 19–27, 2003.
[24] Z. Farbman, G. Hoffer, Y. Lipman, D. Cohen-Or, and D. Lischinski, "Coordinates for instant image cloning," *ACM Trans. Graph.*, vol. 28, no. 3, pp. 67:1–67:9, 2009.
[25] K. Staiger, *The Oil Painting Course You've Always Wanted: Guided Lessons for Beginners and Experienced Artists*. New York, NY, USA: Watson-Guptill Publications, 2006.
[26] H. Bezerra, E. Eisemann, D. DeCarlo, and J. Thollot, "Diffusion constraints for vector graphics," in *Proc. 8th Int. Symp. Non-Photorealistic Animation Rendering*, 2010, pp. 35–42.

Fei Hou received the PhD degree in computer science from Beihang University, in 2012. He is currently a research associate professor of Institute of Software, Chines Academy of Sciences. He was a postdoctoral researcher with the Beihang University from 2012 to 2014 and a research fellow in School of Computer and Science and Engineering, Nanyang Technological University from 2014 to 2017. His research interests include geometry processing, image based modeling, and data vectorization and medical image processing etc.



Qian Sun received the PhD degree in computer science from Nanyang Technological University, Singapore. She is currently an associate professor with the School of Computer Software, Tianjin University, China. Her current research interests include human-computer interaction and computer graphics.



Zheng Fang received the BS degree in computer science and technology from Tianjin University, China. He is currently working toward the PhD degree with the School of Computer Science and Engineering, Nanyang Technological University, Singapore. His current research interests include computational geometry and computer graphics.

Yong-Jin Liu received the BEng degree from Tianjin University, China, in 1998, and the PhD degree from the Hong Kong University of Science and Technology, Hong Kong, China, in 2004. He is currently a full professor with the BNRist, Department of Computer Science and Technology, Tsinghua University, China. His research interests include computational geometry, computer graphics and computer-aided design. He is a senior member of the IEEE and a member of ACM.

Shi-Min Hu received the PhD degree from Zhejiang University, in 1996. He is currently a full professor with Department of Computer Science and Technology, Tsinghua University, Beijing. His research interests include digital geometry processing, video processing, computer animation and computer-aided geometric design. He is the editor-in-chief of Computational Visual Media, and on the Editorial Board of several journals, including the *IEEE Transactions on Visualization and Computer Graphics*, and Computer Aided Design and Computer and Graphics.

Hong Qin received the BS and MS degrees in computer science from Peking University, China, the PhD degree in computer science from the University of Toronto (UofT), in 1995. He is a full professor of computer science with the Department of Computer Science, State University of New York at Stony Brook (Stony Brook University). His research interests include geometric and solid modeling, graphics, physics-based modeling and simulation, computer aided geometric design, human computer interaction, visualization, and scientific computing.

Aimin Hao received the PhD degree from Beihang University, in 2006. He is a professor in computer school, Beihang University and vice director of the State Key Laboratory of Virtual Reality Technology and Systems. His research interests include virtual reality, database application, and information system development.

Ying He received the BS and MS degrees in electrical engineering from Tsinghua University, China, and the PhD degree in computer science from Stony Brook University. He is currently an associate professor with School of Computer Science and Engineering, Nanyang Technological University, Singapore. His research interests fall into the general areas of visual computing and he is particularly interested in the problems which require geometric analysis and computation.