Computational Visual Media
https://doi.org/10.1007/s41095-018-0109-9 Vol. 4, No. 2, June 2018, 149–160
Research Article

Diffusion curves with diffusion coefficients

Hongwei Lin[1,2](✉), **Jingning Zhang**[2], **and Chenkai Xu**[1]

Abstract Diffusion curves can be used to generate vector graphics images with smooth variation by solving Poisson equations. However, using the classical diffusion curve model, it is difficult to ensure that the generated diffusion image satisfies desired constraints. In this paper, we develop a model for producing a diffusion image by solving a diffusion equation with diffusion coefficients, in which color layers and coefficient layers are introduced to facilitate the generation of the diffusion image. Doing so allows us to impose various constraints on the diffusion image, such as diffusion strength, diffusion direction, diffusion points, etc., in a unified computational framework. Various examples are presented in this paper to illustrate the capabilities of our model.

Keywords diffusion curves; diffusion coefficients; color layers; coefficient layers; vector graphics

1 Introduction

The diffusion curve model [1] is a powerful tool for generating vector graphics images with smooth variation. Unlike traditional representations of vector images [2–4], which usually depend on a mesh, diffusion curves are simply defined in terms of curves with colors along either side, and the image is generated by a diffusion procedure represented by a Poisson equation. Thus, diffusion curves present a simple way to create and edit vector images.

As noted, the generation of diffusion curves relies on the solution of a Poisson equation. However, little can be done to control the solution of this Poisson equation. Consequently, it is not possible to ensure that the vector images generated by diffusion curves satisfy various diffusion constraints, such as diffusion strength, diffusion direction [5], and diffusion points [6]. Currently, such constraints require either dedicated constraint systems [5] or different representations [6]. Overall, the generation of diffusion curves with constraints is not only very complicated, but also incompatible with the methods of generating classical diffusion curves [1]. There is thus a need to develop a convenient unified computational framework to generate diffusion curves that satisfy constraints.

This paper develops a model that employs a diffusion equation with diffusion coefficients to produce a vector image; we call it the *diffusion equation with coefficients* (DCC) model. Compared to the traditional diffusion curve model based on a Poisson equation [1] (referred to as the *Poisson model*), where diffusion properties of the diffusive medium are not considered, in the DCC model, diffusion coefficients of the media are utilized to control the diffusion procedure, producing a wide range of results. Using a concept similar to that of image layers used in some image processing software such as *Photoshop*, the DCC model uses two types of layers, i.e., *color layers* and *coefficient layers*. Coefficient layers include *strength coefficient layers* and *direction coefficient layers*. (Color layers and coefficient layers are explained in Section 3 in detail.) Using color layers as initial values, and coefficient layers as diffusion coefficients, the vector image produced by the DCC model can be generated by solving a diffusion equation. With suitably designed coefficient layers, it is possible to ensure that DCC vector images satisfy various constraints, such as diffusion strength, diffusion direction, diffusion points,

1 School of Mathematical Science, Zhejiang University, Hangzhou 310027, China. E-mail: hwlin@zju.edu.cn (✉).
2 State Key Laboratory of CAD&CG, Zhejiang University, Hangzhou 310058, China.
Manuscript received: 2017-12-28; accepted: 2017-12-31

etc., all in a unified computational framework.

The structure of the rest of this paper is as follows. In Section 2, related work on diffusion curves is briefly reviewed. In Section 3, we introduce the DCC model, including its representation and storage, and some default settings and terminology. Techniques for generating DCC vector images are elucidated in Section 4. In Section 5, implementation details are discussed together with experimental results. Finally, Section 6 discusses limitations and future work, then concludes the paper.

2 Related work

The diffusion curve model was developed by Orzan et al. [1] for the generation of vector images, by solving a Poisson equation taking user designed color curves as boundary conditions. Since the shapes of curves and colors along curves are easy to generate manually, diffusion curves present a simple way to create or edit vector images. While a Poisson equation is employed in the diffusion curve model developed by Orzan et al. [1], a bi-harmonic equation is utilized in Ref. [7] to enable more natural interpolation and greater expressive control. Moreover, diffusion curves have been extended to diffusion surfaces to model many different kinds of objects with internal structure [8].

In general, the computation of diffusion curves is very complicated, so some researchers have striven for greater speed. Jeschke et al. [9] utilized a multigrid method on the GPU to accelerate rasterization of diffusion curves; the image quality depends on the resolution at which the Poisson equation is solved. To keep sharp features in close-up views, a high resolution is required, with large time and storage costs. This problem may be alleviated by warping the texture space according to the current view, and employing a dynamic feature embedding algorithm to keep sharp features [10].

To avoid the need for a computational grid for diffusion curves, Bowers et al. [11] recast the generation of diffusion curves as a global illumination problem, and employed a stochastic ray-tracing method to calculate colors. Pang et al. [12] triangulated the image plane and interpolated color values on the triangular mesh using mean value coordinates. Boyé et al. [13] improved on that approach by quadratic interpolation across triangles to solve the bi-Laplace equation. Recently, Prévost et al. [14] combined a triangular representation with an extended ray tracing formulation, using cubic interpolation within each triangle, to produce high-quality images.

The representations of the diffusion curves in the aforementioned work are implicit, requiring a grid or triangular mesh. An explicit representation of diffusion curves was presented in Ref. [15], formulating the diffusion process in terms of Green's functions. This allows the vector image formed by the diffusion curves to be solved in closed form, giving the color value at any point directly. Going further, a fast multipole representation was proposed based on Green's functions, for random-access evaluation of diffusion curve images [6]. Similarly, an explicit representation of the bi-harmonic equation was also developed for rasterizing a diffusion curve in a line-by-line approach [16].

However, it is difficult to control the solutions of the corresponding Laplace, Poisson, or bi-harmonic equation so that they satisfy desired constraints. Instead, Bezerra et al. [5] reformulated the generation of diffusion curves as a constrained minimization problem. Extra constraints on diffusion strength and diffusion direction may be imposed by the users. In a further development, Gaussian radial basis functions were incorporated into the fast multipole representation to generate diffusion points [6].

Although these strategies allow diffusion curves to generate various special effects, they are incompatible with the numerical methods for solving Laplace, Poisson, and bi-harmonic equations, thus increasing the difficulty of implementation. However, the DCC model developed in this paper employs a diffusion equation with diffusion coefficients, allowing us to enforce various kinds of constraints, on DCC vector images, including diffusion strength, diffusion direction, diffusion points, etc., all in a unified computational framework.

3 Diffusion model with diffusion coefficients

3.1 Preliminaries

Here, the following diffusion equation is employed to generate the vector image:

$$\begin{cases} \mathrm{d}I(x,y,t)/\mathrm{d}t = \mathrm{div}(c(x,y)\nabla I) + f(x,y) \\ I(x,y,0) = f(x,y) \end{cases} \tag{1}$$

where $I(x,y,t)$ is the image at time t, $I(x,y,0)$ is the initial image, div and ∇ are the divergence and gradient operators respectively, $c(x,y)$ is the diffusion coefficient, and $f(x,y)$ is the diffusion source.

In the diffusion equation (1), there are two functions which must be designed by the user:

1) the diffusion source $f(x,y)$,

2) the diffusion coefficient $c(x,y)$.

On one hand, the diffusion source function $f(x,y)$ is taken as one component of the RGB color value along either side of some geometric curves; in other places, $f(x,y)$ is set to 0. Therefore, to generate a color image, Eq. (1) must be used three times, once for each RGB component. On the other hand, the diffusion coefficient function $c(x,y)$ measures the diffusion strength at point (x,y) of the diffusion medium. The larger the coefficient $c(x,y)$, the stronger the diffusion at (x,y). In our implementation, we let $c(x,y) \in [0,1]$. If $c(x,y)=0$, this means that point (x,y) is a *sink*, and diffuses nothing to its neighbours. By selecting appropriate values, the coefficient function $c(x,y)$ can be used to control both diffusion direction and diffusion strength (see below).

Consider the following differential equation (see Eq. (1)):

$$\frac{\mathrm{d}I(x,y,t)}{\mathrm{d}t} = \mathrm{div}(c(x,y)\nabla I) = c(x,y)\Delta I + \nabla c \cdot \nabla I \tag{2}$$

where

$$\Delta I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2} \quad \text{and} \quad \nabla I = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right)$$

As in Ref. [17], we use a 4-nearest-neighbor discretization of differential equation (2) on a square lattice (see Fig. 1):

$$\begin{aligned} I_{ij}^{t+1} = I_{ij}^t + \lambda[&c_{\mathrm{N}_{ij}} \cdot \partial_{\mathrm{N}} I_{ij}^t + c_{\mathrm{S}_{ij}} \cdot \partial_{\mathrm{S}} I_{ij}^t \\ &+ c_{\mathrm{E}_{ij}} \cdot \partial_{\mathrm{E}} I_{ij}^t + c_{\mathrm{W}_{ij}} \cdot \partial_{\mathrm{W}} I_{ij}^t] \end{aligned} \tag{3}$$

where $0 \leqslant \lambda \leqslant 1/4$ for the numerical scheme to be stable [17], $\mathrm{N}, \mathrm{S}, \mathrm{E}, \mathrm{W}$ denote North, South, East, West respectively, $c_{\mathrm{N}ij}, c_{\mathrm{S}ij}, c_{\mathrm{E}ij}$, and $c_{\mathrm{W}ij}$ are the values of the coefficient function $c(x,y)$ at the four neighbors of the lattice vertex (i,j) (see Fig. 1): $c_{\mathrm{N}_{ij}} = c_{i,j+1}$, $c_{\mathrm{S}_{ij}} = c_{i,j-1}$, $c_{\mathrm{E}_{ij}} = c_{i+1,j}$, $c_{\mathrm{W}_{ij}} = c_{i-1,j}$, and

$$\begin{aligned} \partial_{\mathrm{N}} I_{ij}^t &= I_{i,j+1}^t - I_{ij}^t, \quad \partial_{\mathrm{S}} I_{ij}^t = I_{i,j-1}^t - I_{ij}^t, \\ \partial_{\mathrm{E}} I_{ij}^t &= I_{i+1,j}^t - I_{ij}^t, \quad \partial_{\mathrm{W}} I_{ij}^t = I_{i-1,j}^t - I_{ij}^t \end{aligned}$$

3.2 The diffusion model

As noted, the DCC model developed in this paper uses two kinds of layers, color layers and coefficient layers. It is well known that the classical Poisson model [1] consists of a geometric curve, a *color source curve* along either side of the geometric curve, and a linearly interpolated blur curve along the geometric curve. A *color source curve* is a geometric curve with color defined along it; the colors are used as diffusion sources. In our DCC model, the discretized color source curves constitute the color layers.

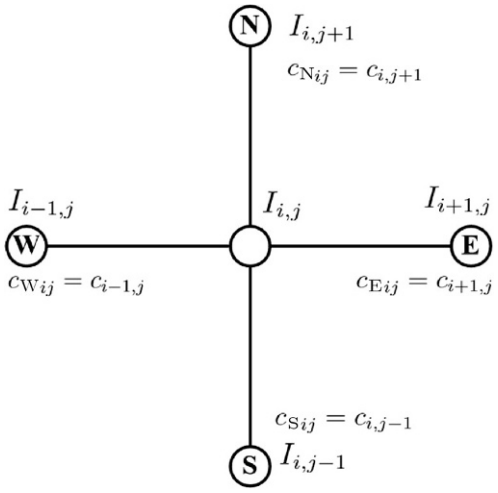


Fig. 1 The discretization scheme of the divergence operator (2).

The coefficient layers are further divided into *strength coefficient layers* and *direction coefficient layers*. A strength coefficient layer represents a scale function defined at each lattice vertex (i,j), specifying the diffusion strength at (i,j). The value of the kth strength layer at vertex (i,j) is denoted by c_{ij}^k. A direction coefficient layer is a vector-valued function defined at (i,j), giving the diffusion direction. Specifically, the vector for the lth direction coefficient layer at (i,j) is denoted by

$$(d_{\mathrm{N}ij}^l, d_{\mathrm{E}ij}^l, d_{\mathrm{S}ij}^l, d_{\mathrm{W}ij}^l) \tag{4}$$

and its components act on the North, South, East, and West neighbors of vertex (i,j), respectively (see Fig. 1). While a strength coefficient layer represents isotropic behaviour of the diffusion medium, a direction coefficient layer represents anisotropic behaviour.

In general, in the DCC model, there is at least one color layer, in which color sources are defined. Each color layer is accompanied by at least one strength layer and one direction layer. Diffusion is performed on each color layer using the accompanying strength and direction layers, generating a diffusion image. All of the generated diffusion images are composited with

masks to produce the final image. In the diffusion on each color layer, the effects of strength and direction layers are combined by multiplication. Consequently, diffusion in the DCC model is performed as follows for each color layer:

$$\begin{aligned} I_{ij}^{t+1} = I_{ij}^{t} + \lambda[& \prod_k c_{\mathrm{N}_{ij}}^k \prod_l d_{\mathrm{N}_{ij}}^l \cdot \partial_{\mathrm{N}} I_{ij}^t \\ & + \prod_k c_{\mathrm{S}_{ij}}^k \prod_l d_{\mathrm{S}_{ij}}^l \cdot \partial_{\mathrm{S}} I_{ij}^t \\ & + \prod_k c_{\mathrm{E}_{ij}}^k \prod_l d_{\mathrm{E}_{ij}}^l \cdot \partial_{\mathrm{E}} I_{ij}^t \\ & + \prod_k c_{\mathrm{W}_{ij}}^k \prod_l d_{\mathrm{W}_{ij}}^l \cdot \partial_{\mathrm{W}} I_{ij}^t] + f_{ij} \end{aligned} \quad (5)$$

where $f_{i,j}$ is the value of $f(x, y)$ at vertex (i, j), $c_{\mathrm{N}_{ij}}^k = c_{i,j+1}^k, c_{\mathrm{S}_{ij}}^k = c_{i,j-1}^k, c_{\mathrm{E}_{ij}}^k = c_{i+1,j}^k, c_{\mathrm{W}_{ij}}^k = c_{i-1,j}^k$ and $d_{\mathrm{N}_{ij}}^l = d_{i,j+1}^l, d_{\mathrm{S}_{ij}}^l = d_{i,j-1}^l, d_{\mathrm{E}_{ij}}^l = d_{i+1,j}^l, d_{\mathrm{W}_{ij}}^l = d_{i-1,j}^l$

In the example shown in Fig. 2(a), each vector in the direction layer is set to ($d_{\mathrm{N}ij} = 0.8$, $d_{\mathrm{E}ij} = 0.6$, $d_{\mathrm{S}ij} = 1.0$, $d_{\mathrm{W}ij} = 1.0$), while each value in the strength layer is set as $c_{ij} = 1.0$, except for those pixels on the curve adjacent on the right (explained later) to the mountain-shaped color source curve, which are set to 0.0. Using these settings for the direction and strength layers, diffusion in the color layer (illustrated in Fig. 13(a)) spreads northward and eastward, generating the *aurora* image in Fig. 2(a). As a comparison, Fig. 2(b) demonstrates the image produced by the Poisson model [1] using the same color layer as in Fig. 13(a), in which the color sources are uniformly spread. Diffusion is iterated for 1000 steps to generate the two images in Figs. 2(a) and 2(b), but the anisotropic diffusion in Fig. 2(a) is faster than the isotropic diffusion in Fig. 2(b).

(a)

(b)

Fig. 2 With suitably designed values, the coefficient layer $c(x, y)$ can be used to control the diffusion direction and strength. (a) *Aurora* image generated by the DCC model with direction coefficient vector ($d_{\mathrm{N}\,ij} = 0.8, d_{\mathrm{E}\,ij} = 0.6, d_{\mathrm{S}\,ij} = 1.0, d_{\mathrm{W}\,ij} = 1.0$) at each pixel. (b) *Aurora* image produced by the classical Poisson model [1].

3.3 Representation and storage

As noted, the classical Poisson model uses a geometric curve, and color source curves along either side of it [1]. In the DCC model, the color source curves are discretized into pixels, giving the *color layer*. Each color layer is stored as an RGB color image.

To facilitate creation of the coefficient layer, an *intermediate* representation is introduced between the vector image representation and pixel representation. The intermediate representation is generated while discretizing the color source curves. It stores labels for the color source curves, and coordinates of pixels belonging to each color source curve. This intermediate representation allows easy selection of color source curves in the pixel representation. When users wish to select a piece of a color source curve, they only need to choose a pixel on it, and the whole curve can readily be determined from the intermediate representation.

In our implementation, we use gray images to store the coefficient layers. As coefficient values are in $[0, 1]$, they must be converted to gray values in $[0, 255]$.

As a direction coefficient is a vector with four components, each direction coefficient layer is stored as a four-component image, where the values of each component are scaled and rounded to integers between 0 and 255 as above.

3.4 Default settings and terminology

By default, there are one color layer, one strength layer, and one direction layer in a DCC model. The default values of the strength coefficients are 1, while the default direction coefficient vectors are $(1, 1, 1, 1)$.

A color layer is composed of several color source curves; each piece of color source curve contains a set of coloured pixels. As noted above, color source curves always appear in pairs (see Fig. 5(a)). Colors on one source curve diffuse in one direction locally, while those on the other curve spread in the opposite direction. For example, in Fig. 5(a), there is a pair of color source curves along either side of the outermost circle. Colors on the outer source curve spread outwards, and those on the inner curve diffuse inwards. In general, the color source curves in a pair are separated by a set of pixels forming the *separating curve*.

To facilitate the design of coefficient layers, we need some notations: see Fig. 3(a). When walking along a closed color source curve anticlockwise, the pixels adjacent to the closed color source curve on the left constitute the *left adjacent curve*, while those adjacent on its right make up the *right adjacent curve*. Similarly, when walking along an open color source curve from the lower right corner to the upper left corner, the left and right pixels adjacent to the open color source curve form the *left adjacent curve* and *right adjacent curve*, respectively (see Fig. 3(b)). If both color source curves in a pair are close, the separating curve is both the left adjacent curve to the outer color source curve, and the right adjacent curve to the inner color source curve. In order to ensure colors on a pair of color source curves spread in correct directions, strength coefficients at pixels on the separating curve are taken to be 0 by default.

4 Generation of diffusion curves with coefficients

The DCC model has color layers and coefficient layers, the former containing color sources for diffusion. The latter are further classified into strength layers and direction layers. In this section, we will show that, by suitably designed coefficient layers, diffusion images generated by the DCC model can satisfy a variety of constraints in a unified computational framework.

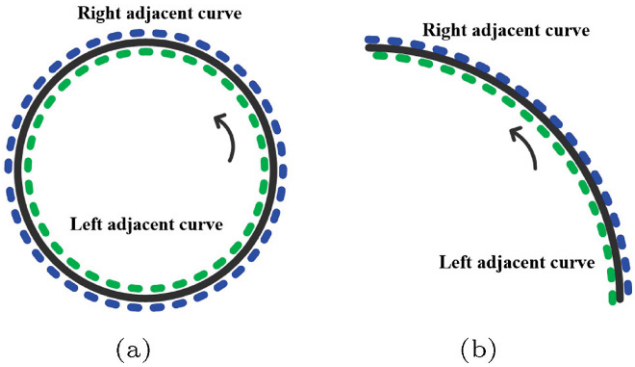


Fig. 3 Left adjacent curve (green) and right adjacent curve (blue) to (a) a closed color source curve (black) and (b) an open color source curve (black), respectively.

4.1 Feature preservation

Features in an image are usually regions with sharp boundaries. Using the DCC model, features in a diffusion image can be preserved by setting appropriate strength coefficients for pixels on the boundary of a region. These coefficients c_{ij} have values in $[0, 1]$; the lower the values for c_{ij}, the sharper the boundary of the feature becomes. The strength coefficients for the pixels inside a region can be used to control the diffusion strength (see Section 4.4).

In the example in Fig. 4, to preserve the features forming the stamens and veins, the strength coefficients on the left and right adjacent curves to

Fig. 4 Feature preservation capability of the DCC model. Main figure: *lotus* image generated by the DCC model, in which the yellow stamens of the lotus and the veins of the lotus leaf are clearly preserved. Insets, upper right: image from the Poisson model [1], in which the features of stamens and veins are not well kept. Inset, lower left: color layer.

their color source curves are all set to 0.0. After diffusion, the yellow stamens of the lotus and the veins of the lotus leaf are clearly preserved by the DCC model (see Fig. 4). In comparison, insets show local regions of the lotus and lotus leaf generated by the classical Poisson model [1]. In this case the stamens and veins in the image are not kept well. A further inset at the lower left shows the color layer, in which color sources are defined.

4.2 Emissivity

The emissivity of a color source curve can be controlled by the strength coefficients for the pixels on the left and right adjacent curves to the color source curve. Figure 5(a) shows the color layer for the *moon* image. A halo can be simulated for the moon by adjusting the strength coefficients for the pixels on the right adjacent curve to the outermost circle (see the color source). In Fig. 5(b), the strength coefficients on the right adjacent curve are set to 0.0, so the color sources on the outermost circle emit nothing outside: there is no halo around the moon. In Fig. 5(c), the strength coefficients on the right adjacent curve are changed to 0.1, resulting in a faint halo around the moon. Finally, in Fig. 5(d), we set the strength coefficients on the right adjacent curve to 1.0, making the emissivity of the color sources on the outermost circle at their strongest, generating the brightest halo around the moon.

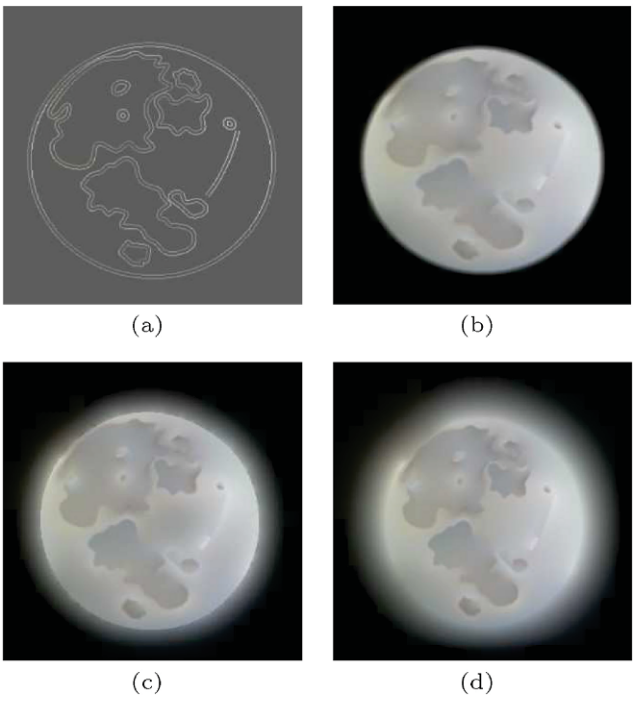


Fig. 5 *Moon* image: the emissivity of the outermost color source curve is controlled by setting different strength for the coefficients of pixels on its right adjacent curve, to generate halos of different intensity. (a) Color layer. (b)–(d) Strength coefficients of 0.0, 0.1, and 1.0 respectively.

4.3 Blurring

As noted, the classical Poisson model consists of a geometric curve, color sources along either side of the geometric curve, and a linearly interpolated blurring curve along the geometric curve [1]. In images generated by this model, a blurring effect is achieved by a postprocessing procedure based on the linearly interpolated blurring curve, after generation of the diffusion image [1]. However, in the DCC model, the blurring effect can be produced at the same time as the diffusion image is generated, without postprocessing: the blurring procedure of the DCC model is integrated into the diffusion procedure.

In the DCC model, to control blurring, we just need to set appropriate coefficients for the pixels on the separating curve. For instance, see Fig. 6, in which there is a pair of color source curves. The separating curve between them is divided into two sub-curves at the midpoint. While the strength coefficients at the pixels on the left sub-curve are interpolated between 1.0 at the left corner to 0.0 at the right corner, those on the right sub-curve are interpolated between 0.0 at the left corner to 1.0 at the right corner. Therefore, the strength coefficient for the middle pixel of the separating curve is 0, and increases from the middle to the two ends, where the strength coefficients are both 1.0. With the strength coefficients so set, a blurring effect can be generated by the DCC model without

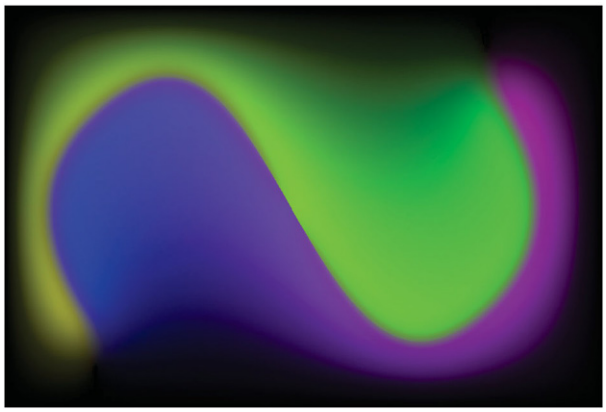


Fig. 6 Generate a blurring effect by setting appropriate strength coefficients for pixels on the separating curve.

postprocessing. The result in Fig. 6 is that a sharp boundary is preserved in the middle region, because of the small strength coefficients there, while blurring increases from the middle to the two ends, as the strength coefficients increase. It should be noted that to produce the blurring effect in Fig. 6, the color sources stop emitting colors in the last 1000 time diffusion steps by setting $f_{ij} = 0$ in Eq. (5).

4.4 Diffusion strength

Strength coefficients in a region can be employed to control the diffusion strength in that region. We provide a user-friendly method to set strength coefficients within a region, using the interpolatory basis functions proposed in Ref. [18]. Firstly, the values of strength coefficients at various isolated pixels, or pixels on some curves in the region Ω are assigned by the user, denoted by

$$c_{i_k,j_k}, \quad (i_k, j_k) \in \Omega, \quad k = 1, \ldots, K$$

Secondly, the assigned strength coefficient values are interpolated using the interpolatory basis functions [18] to give the strength coefficient $c(i, j)$ for pixel (i, j) in Ω:

$$c(i,j) = \sum_{k=1}^{K} c_{i_k,j_k} \phi(i - i_k, a)\phi(j - j_k, a) \qquad (6)$$

where

$$\phi(x,a) = \begin{cases} (\sin(\pi x)/\pi x)\, \mathrm{e}^{-ax^2}, & x \neq 0 \\ 1, & x = 0 \end{cases}$$

and we take $a = 1/3$, as suggested in Ref. [18].

An example of controlling the diffusion strength using the DCC model is presented in Fig. 7. Figure 7(a) shows the color layer for a *girl* image. There are two red dots (i.e., color sources) on the girl's left and right cheeks. To prevent these red color sources diffusing too much into the face, we set relatively small strength coefficients for pixels around the two dots, allowing the strength coefficients in a region around the two dots to be calculated using the method stated above, as illustrated in Fig. 7(b). As the strength coefficients in the region are small, diffusion is effectively weakened in the face region (see Fig. 7(c)). In comparison, in the image produced by the Poisson model [1] (see Fig. 7(d)), too much red color is spread out in the face region.

4.5 Diffusion direction

In the DCC model, with the help of the coefficient layer, not only the diffusion strength, but also the diffusion direction can be controlled, using the direction coefficient layers. Specifically, to cause diffusion to occur in some directions, the diffusion route should first be designated. See Fig. 8(a). The lower left pixel in green is the color source, while the red curve specifies the diffusion route, with black arrows indicating diffusion directions. Similarly, in Fig. 8(b), the red curve is the diffusion route, the green dot represents the center of pixel (i, j) on the route, and $\boldsymbol{T}$ is the unit tangent vector to the route at pixel (i, j). To make the color source spread along the diffusion route according to the diffusion directions, the direction coefficients at the pixels on the route should be assigned as follows. Firstly, the unit tangent vector $\boldsymbol{T}$ is reversed to $-\boldsymbol{T}$ (blue arrow in Fig. 8(b)). Secondly, from the four directions, North, East, South, and West, select one or two directions, making an angle with $-\boldsymbol{T}$ of less than 90°. In Fig. 8(b), the two directions selected are West and South. Let the angle between the West direction and $-\boldsymbol{T}$ be α. Project $-\boldsymbol{T}$ onto the selected directions, and take the lengths of the projections as the corresponding components of the direction coefficient

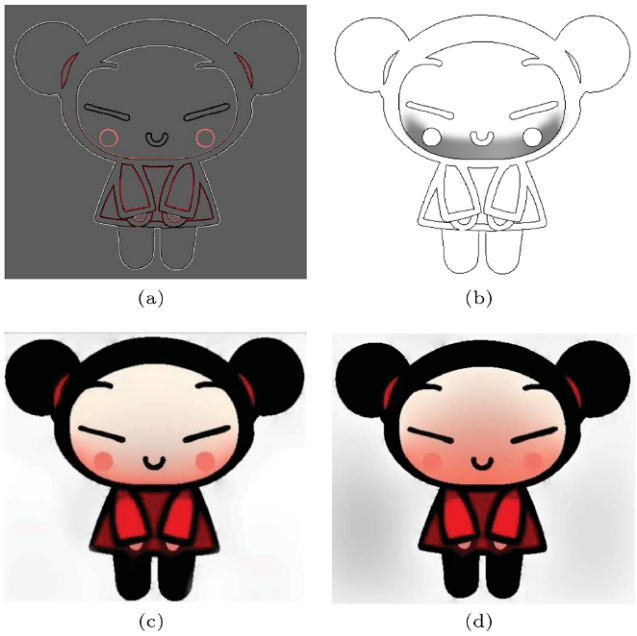


Fig. 7 *Girl* image generated by the DCC model, using strength coefficients to control diffusion strength within a region. (a) Color layer. (b) Strength coefficient layer, with small strength coefficients in the regions around the two red dots. (c) Image generated by the DCC model with the color layer in (a) and the coefficient layer in (b); diffusion of the red dots is effectively hindered in the face region. (d) Image produced by the Poisson model with the color sources defined using the color layer in (a); too much red color spreads into the face region.

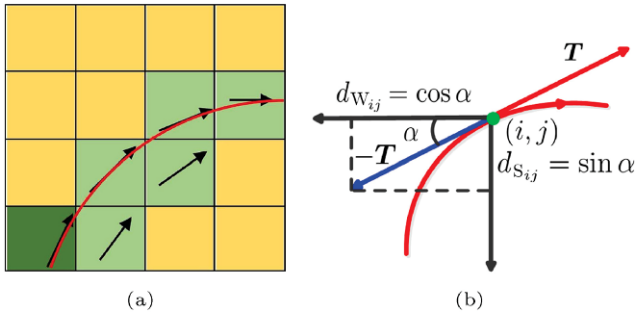


Fig. 8 Design of the direction coefficients. (a) Diffusion route (red curve) is specified by users with diffusion directions (indicated by black arrows). (b) The direction coefficient is calculated by reversing the unit tangent vector $\boldsymbol{T}$, and projecting the reversed tangent vector $-\boldsymbol{T}$ to corresponding directions.

vector (see Eq. (4)). The remaining components are set to 0.0. In the example in Fig. 8(b), the direction coefficients at (i, j) are

$$(d_{N_{ij}}, d_{E_{ij}}, d_{S_{ij}}, d_{W_{ij}}) = (0.0, 0.0, \sin\alpha, \cos\alpha)$$

An example illustrating the effect of the direction coefficients is presented in Fig. 9. In this example, we wish to generate a partial rainbow based on the color layer in Fig. 9(a). The rainbow is emitted from the color sources in the lower left corner to the lower right corner, along a specified diffusion route. To design the direction coefficients, we first draw two concentric circles, with point $O = (o_i, o_j)$ at their center (see Fig. 9(b)). Then, direction coefficients for pixels on the two circles are calculated following the method given. Specifically, suppose each circle is given by

$$(i - o_i)^2 + (j - o_j)^2 = r^2$$

The unit tangent vector at pixel (i, j) on the circle is

$$((j - o_j)/r, -(i - o_i)/r)$$

Therefore, the direction coefficients are taken as

$$\begin{cases} (0, 0, -(i - o_i)/r, (j - o_j)/r), & i \leqslant o_i \\ ((i - o_i)/r, 0, 0, (j - o_j)/r), & i > o_i \end{cases}$$

To calculate the direction coefficient at pixel R in the region enclosed by the two concentric circles in Fig. 9(b), we make a line connecting the center O and the point R, which intersects the two circles at the points P and Q, respectively. Then, the direction coefficient at R is calculated by interpolating the two direction coefficients at P and Q. The direction coefficients at pixels outside the region enclosed by the concentric circles are set to $(0, 0, 0, 0)$. The resulting rainbow generated using the specified direction coefficients is illustrated in Fig. 9(c). In comparison, Fig. 9(d) shows the result produced by the classical

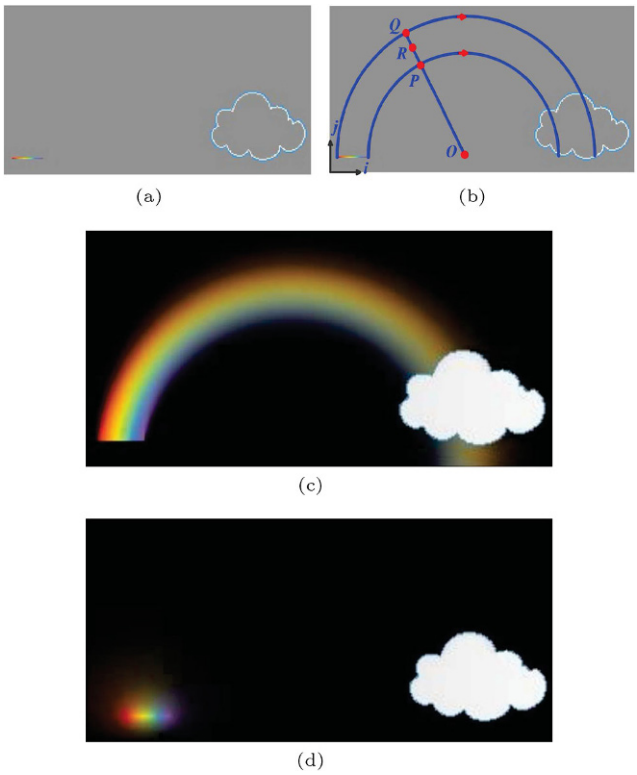


Fig. 9 *Rainbow* image generated by the DCC model, with diffusion directions specified by the direction coefficient layer. (a) Color layer. (b) The diffusion route is enclosed by two concentric circles. (c) *Rainbow* image generated by the DCC model with the direction coefficient layer. (d) Image produced by the classical Poisson model [1] with the same color sources as in (c).

Poisson model [1]: as the color sources are uniformly distributed, no rainbow is produced. Once again, the anisotropic diffusion in Fig. 9(c) is faster than the isotropic diffusion in Fig. 9(d), as the diffusion times of the two images are the same.

The direction coefficient layer can also be used to produce shadows, which is not easy to do using the Poisson model [1]. Figure 10(a) shows a *pepper* image with a clear shadow generated by the DCC model, by means of the direction coefficient layer. However, in Fig. 10(b), produced by the Poisson model from the same color sources as used for Fig. 10(a), there is no clear shadow.

A further example of the effects of direction coefficients is demonstrated in Fig. 2(a). Here, the direction coefficients at each pixel are set to $(0.8, 0.6, 1.0, 1.0)$, and diffusion spreads northwards and eastwards.

4.6 Diffusion points

Using the Poisson model, it is hard to produce the

(a) (b)

Fig. 10 Direction coefficient layers in the DCC model can be employed to generate shadows, which is not easy to do using the classical Poisson model [1]. (a) *Pepper* image with shadows generated by the DCC model. (b) Image produced by the classical Poisson model, using the same color sources as in (a). Note lack of a clear shadow in (b).

effect of diffusion points. To add diffusion points to a diffusion image, Sun et al. [6] proposed using a boundary element method, the fast multipole representation, for rendering diffusion curve images, together with the fast Gauss transform. However, using the DCC model, it is very easy to add diffusion points to a diffusion image. Figure 11 demonstrates a *starry sky* image with diffusion points produced by the DCC model. Color sources in the image are just points with different sizes and colors; there are one thousand diffusion points in the image. It should be pointed out that in the DCC model, computation speed does not depend on the number of diffusion points.

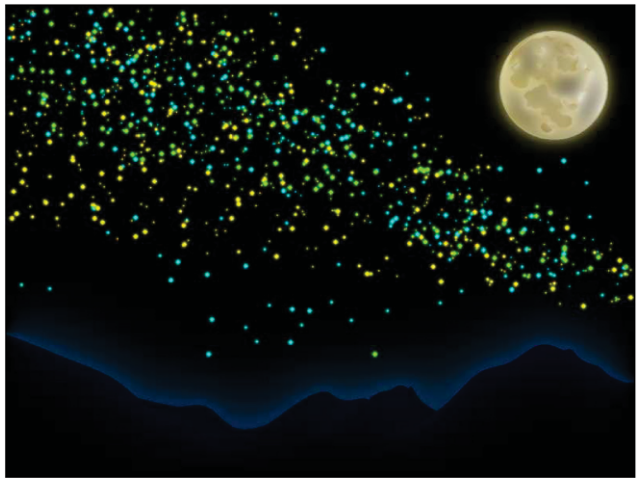
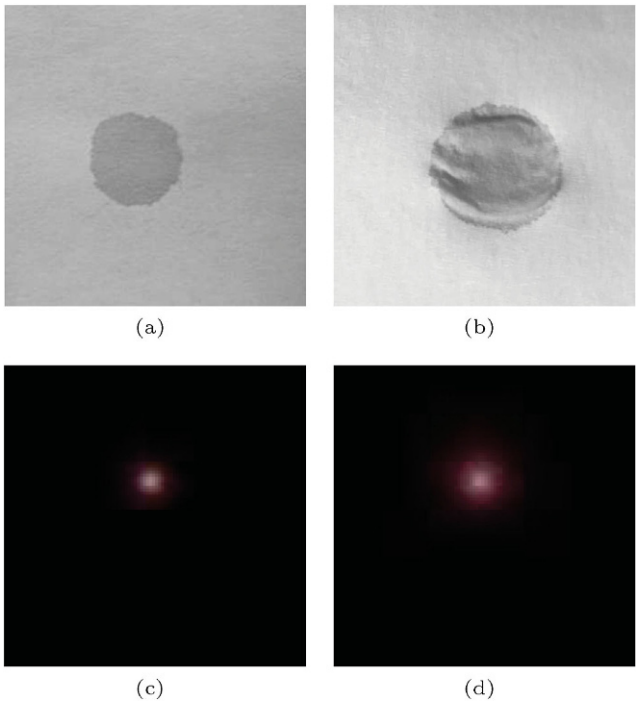


Fig. 11 *Starry sky* image generated by the DCC model. Using the DCC model, diffusion points can be produced in a unified computational framework.

5 Discussions and results

Coefficient layers are the key ingredient in our DCC model. Besides the already mentioned methods for designing coefficient layers, we suggest two other methods for generating strength coefficient layers. One produces a strength coefficient layer by solving an inverse problem, and the other method transforms a gray image into a strength coefficient layer.

Strength coefficients for a diffusive medium can be calculated from a real diffusion image by solving an inverse problem. Suppose we are given a diffusion result for a diffusive medium, represented as a diffusion image, with known diffusion sources, and unknown diffusion coefficients for the medium. From it, we wish to calculate the diffusion coefficients, i.e., strength coefficients in the DCC model, for the medium. This is an inverse problem, called *parameter identification* [19]. In each of Figs. 12(a) and 12(b), a drop of water was dripped onto different paper. We employ the regularization method [20] to find the diffusion coefficients for the two papers from these two diffusion images. The calculated diffusion coefficients can be employed to simulate diffusion results for a



(a) (b) (c) (d)

Fig. 12 Calculating strength coefficients in the DCC model by solving an inverse problem. (a, b) Results of dripping a drop of water onto two different papers with different diffusion coefficients. (c, d) Simulated diffusion results using the diffusion coefficients calculated from (a) and (b), respectively.

specific diffusive medium. Figures 12(c) and 12(d) show diffusion results using the diffusion coefficients calculated for Figs. 12(a) and 12(b) respectively. As in the real diffusion results in Figs. 12(a) and 12(b), where the diffusion region on the paper in Fig. 12(b) is larger than that in Fig. 12(a), the simulated diffusion region in Fig. 12(d) is again larger than that in Fig. 12(c).

As a strength coefficient layer is stored as a gray image, any gray image can be used in principle as a strength coefficient layer. However, to get desirable diffusion images, the gray image used should be deliberately selected or designed. For instance, Fig. 13(b) shows a gray image, used as a coefficient layer. As well as the original color layer, strength coefficient layer, and direction coefficient layer used to generate the image in Fig. 2(a), the gray image in Fig. 13(b) is used as an extra strength coefficient layer. The more realistic *aurora* image in Fig. 13(c) is generated by the DCC model, compared to the result in Fig. 2(a).

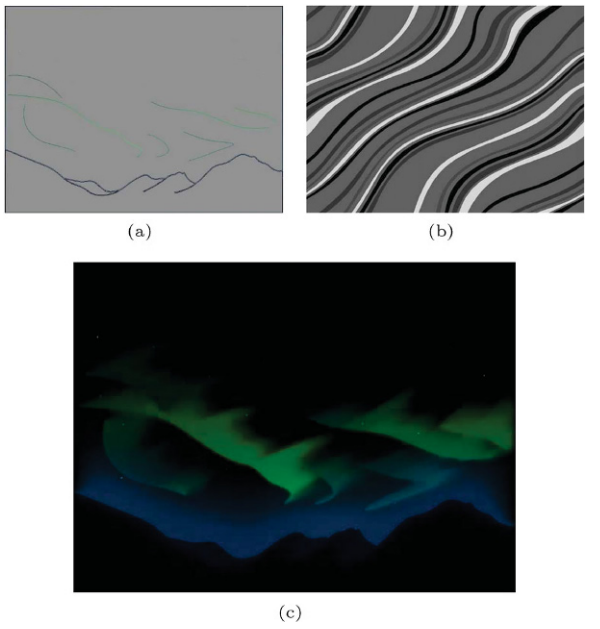


Fig. 13 Gray images used as strength coefficient layers. Using the color layer in (a), and strength and direction coefficient layers as in Fig. 2(a), one further strength coefficient layer, given by the gray image in (b), is added to the DCC model, resulting in a more realistic *aurora* image in (c).

We implemented the DCC model using Visual Studio C++ 2013, and ran it on a PC with a 3.6 GHz Intel Core i7 CPU, 8 GB memory, and an NVIDIA GeForce GTX 960 GPU. Timing and storage requirements for the DCC model are given in Table 1. In this table, the second column gives the resolution of each diffusion image, the third, fourth, and fifth columns are the numbers of color layers, strength layers, and direction layers, respectively, the sixth column is the total storage size for all layers and the final diffusion image, the seventh column is the number of iterations performed to generate each diffusion image, and the last column is the time taken. In our implementation, all color layers, strength layers, direction layers, and the final diffusion images are stored as `.png` files. Storage sizes for the images demonstrated in this paper range from 132 KB to 698 KB, while iteration counts for diffusion vary from 1000 to 50,000. The running time of the DCC model depends on both the image resolution and the number of iterations. Considering the images in this paper, the *starry sky* image (Fig. 11), with resolution 600×800 and $10 + 2000$ iterations (for *star* color layer and *moon* color layer, respectively), took the shortest time of 1.52 s to produce, while the blurring effect in Fig. 6, with resolution 691×1028 and 50,000 iterations, took the longest time of 395.02 s.

As diffusion in the DCC model is performed on a color layer, which is an image, diffusion time does not depend on the complexity of the color sources. For

Table 1 Storage and timing for the DCC model, showing storage used, number of iterations used, and time taken. For the rainbow image, 5000 iterations were used for the rainbow and 1000 for the cloud. For the starry sky image, 10 iterations were used for the stars and 2000 for the moon, and iteration count of cloud is 1000

Image	Size	#Color	#Strength	#Direction	Storage	#Iteration	Time
Fig. 2(a) (aurora)	600×800	1	1	1	191 KB	1000	2.83 s
Fig. 4 (lotus)	720×800	1	1	1	407 KB	15000	45.48 s
Fig. 5(d) (moon)	400×400	1	1	1	139 KB	2000	1.83 s
Fig. 6 (blurring)	691×1028	1	1	1	509 KB	50000	395.02 s
Fig. 7(c) (girl)	600×600	1	1	1	167 KB	10000	19.95 s
Fig. 9(c) (rainbow)	500×1000	2	2	2	132 KB	5000+1000	14.55 s
Fig. 10(a) (pepper)	750×800	1	1	1	319 KB	5000	16.72 s
Fig. 11 (starry sky)	600×800	2	2	2	430 KB	10+2000	1.52 s
Fig. 13(c) (aurora)	600×800	1	2	1	698 KB	1000	2.83 s

example, Table 2 gives the running time for diffusion of the stars in the *starry sky* image (Fig. 11) for varying numbers of stars. Using a fixed resolution of 600×800 and fixed iteration count of 100, although the numbers of diffusion points vary from 10 to 10,000, the diffusion time remains approximately constant.

Table 2 Running time for the *starry sky* image (Fig. 11), for different numbers of diffusion points

Points	10	100	1000	5000	10000
Time	0.620 s	0.634 s	0.641 s	0.610 s	0.641 s

6 Limitations, future work, conclusions

The DCC model contains three kinds of layers: color layers, strength layers, and direction layers. Although an intermediate representation is used to facilitate the creation of strength layers and direction layers, manual creation of these two kinds of layers is still tedious. Therefore, convenient methods for creating strength layers and direction layers should be developed in the future.

The running time of the DCC model heavily depends on the diffusion speed. As demonstrated in Fig. 2, the isotropic diffusion speed is much slower than the anisotropic diffusion speed, so isotropic diffusion usually needs many more iterations and running time. In future, we hope to accelerate diffusion by designing appropriate direction layers for anisotropic diffusion.

In summary, this paper gives the DCC model for generating diffusion images by solving diffusion equations with diffusion coefficients. Underpinning the generation of diffusion images by the DCC model, color layers and coefficient layers are used; these are stored as color images or gray images. Color sources for diffusion are defined in color layers, while diffusion coefficients are specified in coefficient layers. The latter allow constraints such as diffusion strength, diffusion direction, and diffusion points to be easily integrated into the DCC model, permitting diffusion images with constraints to be generated in a unified computational framework.

Acknowledgements

This paper was supported by the National Natural Science Foundation of China (No. 61379072), the National Key R&D Program of China (No. 2016YFB1001501), and the Fundamental Research Funds for the Central Universities (No. 2017XZZX009-03).

References

[1] Orzan, A.; Bousseau, A.; Barla, P.; Winnemöller, H.; Thollot, J.; Salesin, D. Diffusion curves: A vector representation for smooth-shaded images. *Communications of the ACM* Vol. 56, No. 7, 101–108, 2013.

[2] Sun, J.; Liang, L.; Wen, F.; Shum, H.-Y. Image vectorization using optimized gradient meshes. *ACM Transactions on Graphics* Vol. 26, No. 3, Article No. 11, 2007.

[3] Lai, Y.-K.; Hu, S.-M.; Martin, R. R. Automatic and topology-preserving gradient mesh generation for image vectorization. *ACM Transactions on Graphics* Vol. 28, No. 3, Article No. 85, 2009.

[4] Xia, T.; Liao, B.; Yu, Y. Patch-based image vectorization with automatic curvilinear feature alignment. *ACM Transactions on Graphics* Vol. 28, No. 5, Article No. 115, 2009.

[5] Bezerra, H.; Eisemann, E.; DeCarlo, D.; Thollot, J. Diffusion constraints for vector graphics. In: Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering, 35–42, 2010.

[6] Sun, T.; Thamjaroenporn, P.; Zheng, C. Fast multipole representation of diffusion curves and points. *ACM Transactions on Graphics* Vol. 33, No. 4, Article No. 53, 2014.

[7] Finch, M.; Snyder, J.; Hoppe, H. Freeform vector graphics with controlled thin-plate splines. *ACM Transactions on Graphics* Vol. 30, No. 6, Article No. 166, 2011.

[8] Takayama, K.; Sorkine, O.; Nealen, A.; Igarashi, T. Volumetric modeling with diffusion surfaces. *ACM Transactions on Graphics* Vol. 29, No. 6, Article No. 180, 2010.

[9] Jeschke, S.; Cline, D.; Wonka, P. A GPU Laplacian solver for diffusion curves and Poisson image editing. *ACM Transactions on Graphics* Vol. 28, No. 5, Article No. 116, 2009.

[10] Jeschke, S.; Cline, D.; Wonka, P. Rendering surface details with diffusion curves. *ACM Transactions on Graphics* Vol. 28, No. 5, Article No. 117, 2009.

[11] Bowers, J. C.; Leahey, J.; Wang, R. A ray tracing approach to diffusion curves. *Computer Graphics Forum* Vol. 30, No. 4, 1345–1352, 2011.

[12] Pang, W. M.; Qin, J.; Cohen, M.; Heng, P. A.; Choi. K. S. Fast rendering of diffusion curves with triangles. *IEEE Computer Graphics and Applications* Vol. 32, No. 4, 68–78, 2012.

[13] Boyé, S.; Barla, P.; Guennebaud, G. A vectorial solver for free-form vector gradients. *ACM Transactions on Graphics* Vol. 31, No. 6, Article No. 173, 2012.

[14] Prévost, R.; Jarosz, W.; Sorkine-Hornung, O. A vectorial framework for ray traced diffusion curves. *Computer Graphics Forum* Vol. 34, No. 1, 253–264, 2015.

[15] Sun, X.; Xie, G.; Dong, Y.; Lin, S.; Xu, W.; Wang W.; Tong, X.; Guo, B. Diffusion curve textures for resolution independent texture mapping. *ACM Transactions on Graphics* Vol. 31, No. 4, Article No. 74, 2012.

[16] Ilbery, P.; Kendall, L.; Concolato, C.; McCosker, M. Biharmonic diffusion curve images from boundary elements. *ACM Transactions on Graphics* Vol. 32, No. 6, Article No. 219, 2013.

[17] Perona, P.; Malik, J. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence* Vol. 12, No. 7, 629–639, 1990.

[18] Zhang, R.-J.; Ma, W. An efficient scheme for curve and surface construction based on a set of interpolatory basis functions. *ACM Transactions on Graphics* Vol. 30, No. 2, Article No. 10, 2011.

[19] Bambach, M.; Heinkenschloss, M.; Herty, M. A method for model identification and parameter estimation. *Inverse Problems* Vol. 29, No. 2, 025009, 2013.

[20] Vogel, C. R. *Computational Methods for Inverse Problems*. Society for Industrial and Applied Mathematics, 2002.

Hongwei Lin received his B.Sc. degree from the Department of Applied Mathematics, Zhejiang University, in 1996, and Ph.D. degree from the Department of Mathematics, Zhejiang University, in 2004. He is currently a professor in the School of Mathematical Science, State Key Laboratory of CAD&CG, Zhejiang University. His research interests include geometric design, computer graphics, and computer vision.

Jingning Zhang received her B.Sc. degree from Nanjing University of Aeronautics and Astronautics in 2013, and M.Sc. degree from the State Key Laboratory of CAD&CG, Zhejiang University, in 2016. She is currently working in NetEase (Hangzhou) Network Co., Ltd., as a software engineer. Her research interests include computer graphics and computer art.

Chenkai Xu received his B.Sc. degree from the Department of Mathematics, Zhejiang University, in 2016. He is currently a master student in the School of Mathematical Science, Zhejiang University. His research interests include computer vision and 3D feature detection.