Diffusion Curve Textures for Resolution Independent Texture Mapping

Xin Sun* Guofu Xie†‡ Yue Dong* Stephen Lin* Weiwei Xu* Wencheng Wang† Xin Tong* Baining Guo*

*Microsoft Research Asia †State Key Laboratory of Computer Science, ISCAS ‡GUCAS

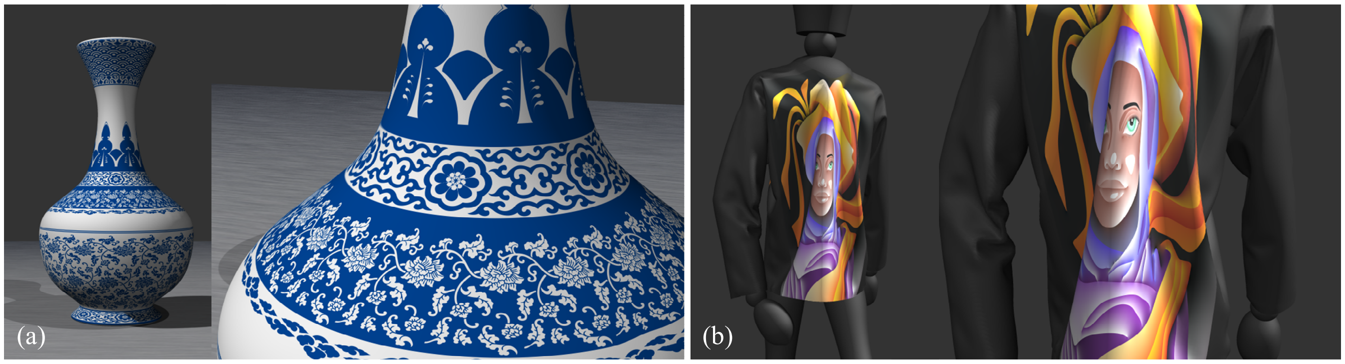

Figure 1: *Our method provides a compact and explicit representation of diffusion curve images for texture mapping onto a surface. The sharp features and detailed color variations of textures are well preserved in the rendering results.*

Abstract

We introduce a vector representation called *diffusion curve textures* for mapping diffusion curve images (DCI) onto arbitrary surfaces. In contrast to the original *implicit* representation of DCIs [Orzan et al. 2008], where determining a single texture value requires iterative computation of the entire DCI via the Poisson equation, diffusion curve textures provide an *explicit* representation from which the texture value at any point can be solved directly, while preserving the compactness and resolution independence of diffusion curves. This is achieved through a formulation of the DCI diffusion process in terms of Green's functions. This formulation furthermore allows the texture value of any rectangular region (e.g. pixel area) to be solved in closed form, which facilitates anti-aliasing. We develop a GPU algorithm that renders anti-aliased diffusion curve textures in real time, and demonstrate the effectiveness of this method through high quality renderings with detailed control curves and color variations.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture;

Keywords: vector images; diffusion curves; texture mapping and rendering

Links: DL PDF

1 Introduction

Diffusion curves [Orzan et al. 2008] are powerful primitives for creating and editing smooth-shaded vector graphics images. They consist of control curves with different colors defined along each side. Images are produced from them by diffusing the colors over the image space in a manner that resembles radiative heat transport. As with other vector graphics models, diffusion curves have a concise representation that is resolution independent and easy to manipulate. But what makes diffusion curves especially appealing among such techniques is its ability to model a broad array of images with subtle shading variations. Because of its simplicity and representation power, diffusion curves has become a popular tool for graphic artists and is considered a possible addition to the Scalable Vector Graphics (SVG) specification.

Diffusion curve images (DCIs) and other forms of vector graphics have drawn considerable attention as a representation for texture maps [Qin et al. 2006; Nehab and Hoppe 2008; Jeschke et al. 2009b]. In contrast to conventional bitmap textures that suffer from limited resolution, a DCI is resolution independent in addition to being compact. However, there exist two major challenges in using them for texture mapping. First, diffusion curves provide an *implicit* representation that does not support random access. In order to obtain the DCI value at a particular point, the full image must be synthesized by solving a Poisson equation. Second, pixels on the control curves need to be specially processed to generate anti-aliased results, which complicates the rendering process and decreases performance. Jeschke et al. [2009b] designed a real-time rasterization method for converting a DCI into bitmaps for anti-aliased texture mapping. However, very high resolution pixel grids are required by this approach to accurately model fine-scale texture details, with a consequent reduction of speed. Recently, Bowers et al. [2011] and Pang et al. [2011] proposed schemes that support random access but generate only an approximation to the DCI. Moreover, it is unclear how to perform anti-aliased rendering with these techniques.

In this paper, we propose an *explicit* representation of diffusion curve images called *diffusion curve textures* that efficiently supports random access and anti-aliased rendering, while maintaining the favorable characteristics of diffusion curves. The key idea of our work is to formulate the diffusion process of DCIs in terms of Green's functions. We define weighted kernels based on Green's functions along each of the control curves. These Green's function kernels are fixed-distance functions that describe the contribution of a control point to each point in the 2D image domain, while the

ACM Reference Format
Sun, X., Xie, G., Dong, Y., Lin, S., Xu, W., Wang, W., Tong, X., Guo, B. 2012. Diffusion Curve Textures for Resolution Independent Texture Mapping. *ACM Trans. Graph. 31* 4, Article 74 (July 2012), 9 pages. DOI = 10.1145/2185520.2185570 http://doi.acm.org/10.1145/2185520.2185570.

kernel weights are determined from the colors defined along the diffusion curves. With this Green's function based representation, we show that the image value at any given point can be directly evaluated by aggregating the contributions of weighted Green's function kernels along all of the control curves.

A major advantage of the Green's function based representation is that the integration of Green's function kernels for a rectangular image region has a closed-form solution. Diffusion curve texture values for arbitrary sized rectangles (e.g. pixel areas) can thus be computed at a constant rendering cost, an important feature for fast anti-aliased rendering. We develop an efficient algorithm for rendering anti-aliased diffusion curve textures that incorporates a curve culling scheme and an adaptive sampling strategy to expedite processing. Since our algorithm renders all pixels in a uniform manner, it can be easily implemented on the GPU. The rendering performance depends only on image resolution and is independent of texture scales over surfaces and viewing directions. With this technique, we demonstrate high quality renderings from diffusion curve textures for various levels of diffusion curve complexity.

In summary, the contributions of this paper are

- *Diffusion curve textures*, an explicit representation of diffusion curve images that supports random access and fast integration.
- An efficient algorithm for rendering anti-aliased diffusion curve textures mapped over 3D surfaces.
- A GPU implementation of the rendering method that provides real-time performance.

2 Related Work

Vector Images and Textures A vector image can be directly modeled by a set of simple vector primitives such as points, lines, curves, and polygons associated with colors. Qin et al. [2006; 2008] developed techniques for rendering anti-aliased vector textures based on the distance between pixel samples and primitives. Nehab and Hoppe [2008] used a lattice structure for speeding up the random access of vector primitives and generated anti-aliased results by prefiltering and supersampling the vector primitives in a pixel. In our work, we aim for similar goals but for smoothly shaded images defined by diffusion curves.

Different vector representations have been introduced for modeling smoothly shaded images. The gradient mesh is widely used in industry for vectorizing images that consist of smooth color regions [Lecot and Levy 2006; Sun et al. 2007; Lai et al. 2009]. Xia et al. [2009] segmented images into triangular patches and modeled the color variation inside a patch with thin plate splines. Diffusion curve images [Elder and Goldberg 2001; Orzan et al. 2008] and its extensions [Bezerra et al. 2010; Takayama et al. 2010] construct images or volumes by diffusing colors specified on curves and surfaces. Most recently, Finch et al. [2011] introduced controlled thin plate splines for authoring vector images with greater expressive control. These works all focus on authoring or vectorizing raster images, rather than on how to use them for texture mapping. Recently, Wang et al. [2010; 2011] presented a vector representation for solid textures in which the region boundary is defined by implicit distance functions and the volumetric color variations are modeled by radial basis functions (RBFs). This approach, however, is difficult to extend to DCIs with non-closed control curves.

Several hybrid representations combine raster textures with sharp vector features embedded in texels for anti-aliased texture mapping [Ramanarayanan et al. 2004; Sen et al. 2003; Sen 2004; Tumblin and Choudhury 2004; Tarini and Cignoni 2005]. Most of these methods store a fixed number of primitives in each texel and need high resolution images for representing textures with high local complexity. Although storing various numbers of primitives in texels [Ramanarayanan et al. 2004] could avoid this problem, this leads to high computational costs for texture sampling and interpolation. We use the compact representation of diffusion curve textures to model both sharp features and smooth color variations in the image and render them in the same way.

Diffusion Curve Images Diffusion curve images were introduced by Orzan et al. [2008] for modeling vector images with smooth color variations. A significant difference of DCI from other vector graphics representations is that the image has to be computed by solving a Poisson equation defined over the entire 2D domain. Generally, the solution at an $n \times n$ resolution can be solved by multigrid methods [Bolz et al. 2003; Kazhdan and Hoppe 2008; Jeschke et al. 2009a] with $O(n^2)$ time complexity.

To render a DCI mapped onto a surface, a naïve solution is to rasterize the DCI as bitmaps at specific resolutions and then render the result via the traditional rendering pipeline. However, a high resolution image is needed to preserve sharp features at close-up views, which leads to large computational and storage costs. Jeschke et al. [2009b] alleviated this issue by warping the texture space according to the current view and using a dynamic feature embedding algorithm to retain sharp features in the generated texture image. However, for diffusion curve images with rich features or complex curves, a high resolution solution is still needed. By contrast, our method directly integrates texture values in each pixel region from the compact Green's function based representation and preserves sharp features at any resolution. With the two acceleration schemes presented in the paper, the time complexity of our rendering algorithm is proportional to the screen resolution and independent of model and image complexity.

Bowers et al. [2011] designed a ray tracing based approach to simulate DCIs. By reformulating the problem to be equivalent to final gathering in global illumination, the image values can be evaluated by GPU based stochastic raytracing. They also extended the curve attributes of DCI to support additional authoring tasks. Most recently, Pang et al. [2011] used mean value coordinates interpolation on a triangle mesh to generate images similar in appearance to DCIs. Both methods provide only approximate solutions to the DCI. Also, it is unclear how to efficiently perform integration over regions with these methods, which is a requirement for texture anti-aliasing. Unlike these approximation techniques, our method provides an accurate solution for diffusion curve images and supports random access as well as fast integration.

Green's Functions Green's functions have been applied in various graphics applications. D'Eon and Irving [2011] used a quantized Green's function for rendering light diffusion in translucent materials. Lipman et al. [2008] presented Green Coordinates for mesh deformation based on the Green's function for the Laplacian. An analytic integration of the Green's function over a closed polyhedral cage is used for Green Coordinates computation. Farbman et al. [2011] modeled the solution of the Poisson equation in the image domain as the convolution of a Green's function and the image gradients. They then approximated the convolution with specially designed convolution pyramids. Our method adapts the Green's function for the Laplacian in 2D space to model and render diffusion curve images, for which the underlying Poisson equation can be accurately modeled by Green's function kernels defined on control curves. Moreover, we derive an analytic solution for integrating the 2D Green's function kernel contributions over a rectangular region and present a method to compute this in real time.

3 Diffusion Curve Textures

In this section, we briefly review Green's functions, and then derive a representation of diffusion curves based on Green's functions from which diffusion curve images can be directly solved.

3.1 Preliminaries on Green's Functions

Green's functions are a useful tool for solving inhomogeneous differential equations with boundary conditions [Bayin 2006]. For the Laplacian operator Δ, a Green's function $G(\mathbf{x}, \mathbf{x}')$ at a point $\mathbf{x}'$ satisfies the following equation:

$$\Delta G(\mathbf{x}, \mathbf{x}') = \delta(\|\mathbf{x} - \mathbf{x}'\|) \tag{1}$$

where δ is the Dirac delta function.

For a scalar function $u(\mathbf{x})$ that is twice continuously differentiable on region $\mathbf{D}$ in $\mathbb{R}^2$, Green's second identity relates a double integral over $\mathbf{D}$ to a line integral over the region boundary:

$$\begin{aligned}&\iint_{\mathbf{D}} \left(u(\mathbf{x}') \Delta G(\mathbf{x}, \mathbf{x}') - G(\mathbf{x}, \mathbf{x}') \Delta u(\mathbf{x}')\right) \mathrm{d}\mathbf{x}' \\ &= \oint_{\partial\mathbf{D}} \left(u(\mathbf{x}') \frac{\partial G(\mathbf{x}, \mathbf{x}')}{\partial \mathbf{n}(\mathbf{x}')} - G(\mathbf{x}, \mathbf{x}') \frac{\partial u(\mathbf{x}')}{\partial \mathbf{n}(\mathbf{x}')}\right) \mathrm{d}\mathbf{x}' \end{aligned}\tag{2}$$

where $\mathbf{n}(\mathbf{x})$ is the normal direction on the boundary $\partial\mathbf{D}$.

The following expression for $u(\mathbf{x})$ can be formulated from Eq. (1):

$$\iint_{\mathbf{D}} u(\mathbf{x}') \Delta G(\mathbf{x}, \mathbf{x}') \,\mathrm{d}\mathbf{x}' = u(\mathbf{x}). \tag{3}$$

From Eq. (2) and Eq. (3), we can derive Green's third identity to evaluate $u(\mathbf{x})$:

$$\begin{aligned} u(\mathbf{x}) = &\iint_{\mathbf{D}} G(\mathbf{x}, \mathbf{x}') \Delta u(\mathbf{x}') \,\mathrm{d}\mathbf{x}' \\ &+ \oint_{\partial\mathbf{D}} \left(u(\mathbf{x}') \frac{\partial G(\mathbf{x}, \mathbf{x}')}{\partial \mathbf{n}(\mathbf{x}')} - G(\mathbf{x}, \mathbf{x}') \frac{\partial u(\mathbf{x}')}{\partial \mathbf{n}(\mathbf{x}')}\right) \mathrm{d}\mathbf{x}'. \end{aligned}\tag{4}$$

3.2 Green's Function Kernels

We utilize Green's functions to solve for values in a diffusion curve image. Diffusion curves [Orzan et al. 2008; Jeschke et al. 2009a] represent an image as a harmonic function $u(\mathbf{x})$ that satisfies the Laplace equation:

$$\Delta u(\mathbf{x}) = 0 \tag{5}$$

with boundary conditions C_l, C_r specified on the two sides of each control curve $\mathbf{B}$:

$$u(\mathbf{x}')\,|_{\mathbf{x}' \in \partial\mathbf{D}} = \{C_l(\mathbf{x}'), C_r(\mathbf{x}')\}|_{\mathbf{x}' \in \mathbf{B}}. \tag{6}$$

As illustrated in Fig. 2(a), we define boundary normals to be oriented outwards from their domain (i.e. toward the other side of $\mathbf{B}$) and the normal of the control curve $\mathbf{B}$ as $\mathbf{n}(\mathbf{x}') = \mathbf{n_r}(\mathbf{x}') = -\mathbf{n_l}(\mathbf{x}')$.

With Eq. (5) and Eq. (6), we can simplify Eq. (4) to

$$\begin{aligned} u(\mathbf{x}) &= \oint_{\partial\mathbf{D}} \left(u(\mathbf{x}') \frac{\partial G(\mathbf{x}, \mathbf{x}')}{\partial \mathbf{n}(\mathbf{x}')} - \frac{\partial u(\mathbf{x}')}{\partial \mathbf{n}(\mathbf{x}')} G(\mathbf{x}, \mathbf{x}')\right) \mathrm{d}\mathbf{x}' \\ &= \int_{\mathbf{B}} \left(C_l(\mathbf{x}') \frac{\partial G(\mathbf{x}, \mathbf{x}')}{\partial \mathbf{n_l}(\mathbf{x}')} - \frac{\partial u(\mathbf{x}')}{\partial \mathbf{n_l}(\mathbf{x}')} G(\mathbf{x}, \mathbf{x}')\right) \mathrm{d}\mathbf{x}' \\ &+ \int_{\mathbf{B}} \left(C_r(\mathbf{x}') \frac{\partial G(\mathbf{x}, \mathbf{x}')}{\partial \mathbf{n_r}(\mathbf{x}')} - \frac{\partial u(\mathbf{x}')}{\partial \mathbf{n_r}(\mathbf{x}')} G(\mathbf{x}, \mathbf{x}')\right) \mathrm{d}\mathbf{x}' \\ &= \int_{\mathbf{B}} \left(C(\mathbf{x}') \frac{\partial G(\mathbf{x}, \mathbf{x}')}{\partial \mathbf{n_r}(\mathbf{x}')} - E(\mathbf{x}') G(\mathbf{x}, \mathbf{x}')\right) \mathrm{d}\mathbf{x}', \end{aligned}\tag{7}$$

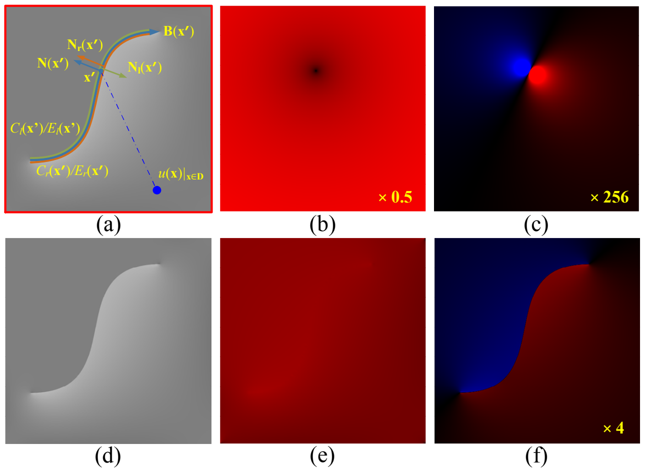

Figure 2: *A diffusion curve texture defined by two control curves: an S-shaped curve, and a curve along the image boundary. (a) Control curves with boundary conditions (0.75 for the right S-shaped boundary, 0.5 for the left S-shaped boundary and the image boundaries). (b) Profile of Green's function kernel $G(\mathbf{x}, \mathbf{x}')$ from one point on a control curve. (c) Profile of Green's function kernel $G_n(\mathbf{x}, \mathbf{x}')$ from one point on a control curve. (d) Diffusion curve image computed from all of the Green's function kernels. (e) Image generated from only the Green's function kernels $G(\mathbf{x}, \mathbf{x}')$. (f) Image generated from only the Green's function kernels $G_n(\mathbf{x}, \mathbf{x}')$. In the rightmost two columns, red indicates positive values, while blue indicates negative values.*

where $C(\mathbf{x}') = -C_l(\mathbf{x}') + C_r(\mathbf{x}')$, $E(\mathbf{x}') = \frac{\partial u(\mathbf{x}')}{\partial \mathbf{n_l}(\mathbf{x}')} + \frac{\partial u(\mathbf{x}')}{\partial \mathbf{n_r}(\mathbf{x}')}$.

Eq. (7) indicates that $u(\mathbf{x})$ at any point within the entire domain $\mathbf{D}$ can be evaluated through computations only on the control curves. Since the Green's function is symmetric with respect to $\mathbf{x}$ and $\mathbf{x}'$ (i.e. $G(\mathbf{x}', \mathbf{x}) = G(\mathbf{x}, \mathbf{x}')$), we can view Eq. (7) in terms of two Green's function based kernels defined along the control curves, instead of over the domain $\mathbf{D}$. Therefore, we explicitly represent a diffusion curve image with the two Green's function based kernels $G(\mathbf{x}, \mathbf{x}')$ and $G_n(\mathbf{x}, \mathbf{x}') = \frac{\partial G(\mathbf{x}, \mathbf{x}')}{\partial \mathbf{n}(\mathbf{x}')}$, and their weights $C(\mathbf{x}')$ and $E(\mathbf{x}')$ defined along the control curves:

$$u(\mathbf{x}) = \sum_i \int_{\mathbf{B_i}} \left(C_i(\mathbf{x}') G_n(\mathbf{x}, \mathbf{x}') - E_i(\mathbf{x}') G(\mathbf{x}, \mathbf{x}')\right) \mathrm{d}\mathbf{x}'. \tag{8}$$

In our work, we employ the Green's function $G(\mathbf{x}, \mathbf{x}')$ in $\mathbb{R}^2$:

$$G(\mathbf{x}, \mathbf{x}') = \frac{1}{2\pi} \ln(\|\mathbf{x} - \mathbf{x}'\|) \tag{9}$$

with the corresponding normal derivative $G_n(\mathbf{x}, \mathbf{x}')$:

$$G_n(\mathbf{x}, \mathbf{x}') = \frac{\partial G(\mathbf{x}, \mathbf{x}')}{\partial \mathbf{n}(\mathbf{x}')} = \frac{\mathrm{d}\,G(\mathbf{x}, \mathbf{x}')}{\mathrm{d}\,\mathbf{x}} \cdot \mathbf{n}(\mathbf{x}') = \frac{\hat{x}\, n_x + \hat{y}\, n_y}{2\pi(\hat{x}^2 + \hat{y}^2)} \tag{10}$$

where $\mathbf{n}(\mathbf{x}') = (n_x, n_y)$ and $(\hat{x}, \hat{y}) = \mathbf{x} - \mathbf{x}'$.

Note that this explicit representation is different from the original implicit diffusion curves representation that is based on the Poisson equation. First, our method allows evaluation at only the point itself without considering other areas, while a linear system for the Poisson equation must be solved over the entire domain to compute the value at a single point. Second, unlike the explicit solution provided by Green's function kernels, the original implicit representation requires discretization of the domain to solve the Poisson equation, which intrinsically enforces a tradeoff between rendering quality and computational performance.

Fig. 2(b)(c) illustrate the two Green's function kernels defined at one point on the control curve, where G is isotropic while G_n is anisotropic. As shown in Fig. 2 (e)(f), the contributions of the two kernels defined along the curve result in the global low frequency color variation and the high frequency color change at boundaries, respectively.

To calculate $E(\mathbf{x}')$, we note that for a closed domain $\mathbf{D}$ with Hölder continuous boundary values $u(\mathbf{x}')\,|_{\mathbf{x}'\in\partial\mathbf{D}}$, the solution for $u(\mathbf{x})$ is unique. So we solve

$$u(\mathbf{x}) = \oint_{\partial\mathbf{D}} \left(u(\mathbf{x}')\, G_n(\mathbf{x},\mathbf{x}') - \frac{\partial u(\mathbf{x}')}{\partial \mathbf{n}(\mathbf{x}')} G(\mathbf{x},\mathbf{x}') \right) \mathrm{d}\mathbf{x}', \mathbf{x} \in \partial\mathbf{D} \quad (11)$$

and compute the values of $E(\mathbf{x}')$ from $\frac{\partial u(\mathbf{x}')}{\partial \mathbf{n_l}(\mathbf{x}')}$ and $\frac{\partial u(\mathbf{x}')}{\partial \mathbf{n_r}(\mathbf{x}')}$. We note that to solve the diffusion equation over an image, boundary conditions at the image borders (or beyond the image borders) must be specified. So in practice we always place a control curve at or beyond the image borders for a given diffusion curve image.

4 Rendering with Diffusion Curve Textures

Given the Green's function kernels on the control curves, the harmonic function $u(\mathbf{x})$ can be constructed according to Eq. (8). For rendering purposes such as anti-aliasing, we wish to evaluate the integral of $u(\mathbf{x})$ over a rectangular region $\mathbf{R}$ instead of at just an individual point $\mathbf{x}$:

$$\phi(\mathbf{R}) = \iint_{\mathbf{R}} u(\mathbf{x})\, \mathrm{d}\mathbf{x}. \quad (12)$$

Rendering with this function involves three components: integrating over the region $\mathbf{R}$, integrating along each control curve $\mathbf{B}$, and accumulating contributions from all the control curves. We address each of these components in this section.

4.1 Integration over a Rectangle

The integral of $u(\mathbf{x})$ in Eq. (8) over a region $\mathbf{R}$ as in Eq. (12) can be expressed in terms of integrations of the Green's function kernels $G(\mathbf{x},\mathbf{x}')$ and $G_n(\mathbf{x},\mathbf{x}')$ over the rectangular region $\mathbf{R} = \{x \in (x_0, x_1), y \in (y_0, y_1)\}$:

$$\phi(\mathbf{R}) = \iint_{\mathbf{R}} \left(\sum_i \int_{\mathbf{B_i}} C_i(\mathbf{x}')\, G_n(\mathbf{x},\mathbf{x}')\, \mathrm{d}\mathbf{x}' \right) \mathrm{d}\mathbf{x} - \iint_{\mathbf{R}} \left(\sum_i \int_{\mathbf{B_i}} E_i(\mathbf{x}')\, G(\mathbf{x},\mathbf{x}')\, \mathrm{d}\mathbf{x}' \right) \mathrm{d}\mathbf{x} \quad (13)$$

A closed-form integral $F_G(\mathbf{R},\mathbf{x}') = \iint_{\mathbf{R}} G(\mathbf{x},\mathbf{x}')\, \mathrm{d}\mathbf{x}$ exists for this Green's function over a rectangular region $\mathbf{R}$:

$$F_G(\mathbf{R},\mathbf{x}') = \sum_{i,j\in\{0,1\}} \frac{(-1)^{i+j}}{4\pi} H_G(\hat{x},\hat{y}) \quad (14)$$

$$H_G(\hat{x},\hat{y}) = -3\,\hat{x}\hat{y} + \hat{x}\hat{y}\ln\left(\hat{x}^2+\hat{y}^2\right) + \hat{x}^2 \operatorname{arctg}\frac{\hat{y}}{\hat{x}} + \hat{y}^2 \operatorname{arctg}\frac{\hat{x}}{\hat{y}}. \quad (15)$$

A closed-form integral $F_{G_n}(\mathbf{R},\mathbf{x}') = \iint_{\mathbf{R}} G_n(\mathbf{x},\mathbf{x}')\, \mathrm{d}\mathbf{x}$ likewise exists for G_n:

$$F_{G_n}(\mathbf{R},\mathbf{x}') = \sum_{i,j\in\{0,1\}} \frac{(-1)^{i+j}}{2\pi} H_{G_n}(\hat{x},\hat{y},n_x,n_y) \quad (16)$$

$$H_{G_n}(\hat{x},\hat{y},n_x,n_y) = n_y\hat{y}\operatorname{arctg}\left(\frac{\hat{x}}{\hat{y}}\right) + n_x\hat{x}\operatorname{arctg}\left(\frac{\hat{y}}{\hat{x}}\right) + \frac{1}{2}(n_y\hat{x}+n_x\hat{y})\ln\left(\hat{x}^2+\hat{y}^2\right) \quad (17)$$

where $\mathbf{x} = (x,y)$, $\mathbf{x}' = (x',y')$, $\mathbf{n}(\mathbf{x}') = (n_x,n_y)$, and $(\hat{x},\hat{y}) = \mathbf{x}-\mathbf{x}'$. Detailed derivations of Eq. (14) and Eq. (16) are provided in Appendix A.

Eq. (14) and Eq. (16) are derived for an axially aligned rectangular region, but are also valid for rotated rectangles since the Green's function $G(\mathbf{x},\mathbf{x}')$ is rotationally symmetric about $\mathbf{x}'$. We note that though $G(\mathbf{x},\mathbf{x}')$ has a singular point at $\mathbf{x}=\mathbf{x}'$, it does not affect the evaluation of $H_G(\hat{x},\hat{y})$ and $H_{G_n}(\hat{x},\hat{y},n_x,n_y)$. An important feature of the closed-form integrals $F_G(\mathbf{R},\mathbf{x}')$ and $F_{G_n}(\mathbf{R},\mathbf{x}')$ is that the computational load is constant for different sizes of $\mathbf{R}$, in contrast to sampling schemes which generally require greater computation for larger regions.

With the control curve representation, we can compute Eq. (12) as

$$\phi(\mathbf{R}) = \sum_i \int_{\mathbf{B_i}} \left(C_i(\mathbf{x}')\, F_G(\mathbf{R},\mathbf{x}') + E_i(\mathbf{x}')\, F_{G_n}(\mathbf{R},\mathbf{x}') \right) \mathrm{d}\mathbf{x}'. \quad (18)$$

To rapidly solve Eq. (18), integration along control curves $\mathbf{B}$ and accumulation of control curve contributions need to be performed efficiently. These two problems are addressed in Section 4.2 and Section 4.3.

4.2 Adaptive Sampling on Control Curves

To efficiently evaluate $\phi(\mathbf{R})$ in Eq. (18), we compute a piecewise constant approximation with respect to a set of sample points $\{\mathbf{x}'_{\mathbf{i,j}}\}$ along each control curve $\mathbf{B_i}$:

$$\phi(\mathbf{R}) \approx \sum_i \sum_j \left(\bar{\phi}_C\left(\mathbf{x}'_{\mathbf{i,j}}\right) - \bar{\phi}_E\left(\mathbf{x}'_{\mathbf{i,j}}\right) \right) l\left(\mathbf{x}'_{\mathbf{i,j}}\right) \quad (19)$$

where $l\left(\mathbf{x}'_{\mathbf{i,j}}\right)$ is the arc length between $\mathbf{x}'_{\mathbf{i,j}}$ and $\mathbf{x}'_{\mathbf{i,j+1}}$, and $\bar{\phi}_C\left(\mathbf{x}'_{\mathbf{i,j}}\right)$ and $\bar{\phi}_E\left(\mathbf{x}'_{\mathbf{i,j}}\right)$ are weighted kernels of $C_i\left(\mathbf{x}'_{\mathbf{i,j}}\right) F_{G_n}\left(\mathbf{R},\mathbf{x}'_{\mathbf{i,j}}\right)$ and $E_i\left(\mathbf{x}'_{\mathbf{i,j}}\right) F_G\left(\mathbf{R},\mathbf{x}'_{\mathbf{i,j}}\right)$ respectively. Sampling with a small, uniform value of $l\left(\mathbf{x}'_{\mathbf{i,j}}\right)$ would be inefficient over different levels of detail, so we propose an adaptive sampling scheme.

Our adaptive scheme aims to sample points such that the error in the piecewise constant approximation of $\mathbf{B_i}$ between neighbors $\mathbf{x}'_{\mathbf{i,j}}$ and $\mathbf{x}'_{\mathbf{i,j+1}}$ falls below a given bound. We approximate this by selecting sample points according to

$$\left| \left(\bar{\phi}_S\left(\mathbf{x}'_{\mathbf{i,j+1}}\right) - \bar{\phi}_S\left(\mathbf{x}'_{\mathbf{i,j}}\right) \right) / \bar{\phi}_S\left(\mathbf{x}'_{\mathbf{i,j}}\right) \right| < \alpha,\ S \in \{C,E\} \quad (20)$$

where α denotes a user-specified bound.

For a given sample point $\mathbf{x}'_{\mathbf{i,j}}$, the next sample $\mathbf{x}'_{\mathbf{i,j+1}}$ cannot be determined from Eq. (20) without evaluating different candidate points. To avoid this inefficiency, we use the first order differential at $\mathbf{x}'_{\mathbf{i,j}}$ to approximate the value at $\mathbf{x}'_{\mathbf{i,j+1}}$:

$$\bar{\phi}_S\left(\mathbf{x}'_{\mathbf{i,j+1}}\right) \approx \bar{\phi}_S\left(\mathbf{x}'_{\mathbf{i,j}}\right) + \bar{\phi}'_S\left(\mathbf{x}'_{\mathbf{i,j}}\right) l\left(\mathbf{x}'_{\mathbf{i,j}}\right),\ S \in \{C,E\} \quad (21)$$

where $\bar{\phi}'_S\left(\mathbf{x}'_{\mathbf{i,j}}\right) = \partial\bar{\phi}_S\left(\mathbf{x}'_{\mathbf{i,j}}\right)/\partial\mathbf{t}\left(\mathbf{x}'_{\mathbf{i,j}}\right)$ and $\mathbf{t}\left(\mathbf{x}'_{\mathbf{i,j}}\right)$ is the tangent direction along the curve. We then can determine the arc length $l\left(\mathbf{x}'_{\mathbf{i,j}}\right)$ from α:

$$l\left(\mathbf{x}'_{\mathbf{i,j}}\right) = \alpha \min\left\{ \left| \bar{\phi}_S\left(\mathbf{x}'_{\mathbf{i,j}}\right) / \bar{\phi}'_S\left(\mathbf{x}'_{\mathbf{i,j}}\right) \right|, S \in \{C,E\} \right\}. \quad (22)$$

Since $\bar{\phi}'_C$ and $\bar{\phi}'_E$ can be solved analytically (see Appendix B), we can determine $l\left(\mathbf{x}'_{\mathbf{i,j}}\right)$ through calculations only at $x'_{i,j}$. To sample a control curve $\mathbf{B_i}$, we start by placing the first sample $\mathbf{x}'_{\mathbf{i,0}}$ at an arbitrary endpoint, then iteratively place subsequent samples $\mathbf{x}'_{\mathbf{i,j+1}}$ based on computed values of $l\left(\mathbf{x}'_{\mathbf{i,j}}\right)$ until reaching the end.

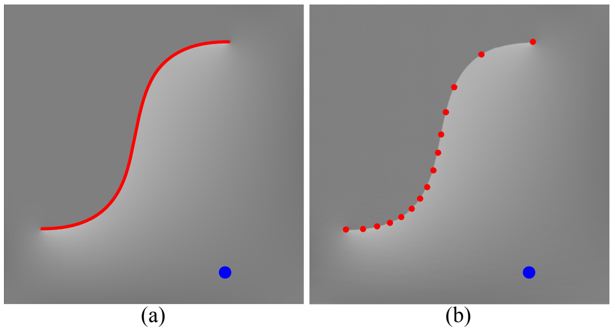

Figure 3: *Control curve sampling. (a) Uniform sampling with more than* 500 *samples along the control curve. (b) Adaptive sampling requires only about* 10 *points to produce similar results at the blue point (*$\alpha = 0.02$*).*

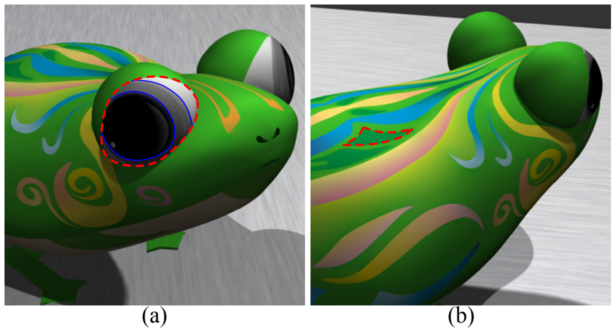

Figure 4: *Culling of control curves. (a) The control curves in blue are enclosed by the control curve in red, so integration regions outside the red boundary can be computed independently of the blue curves and the inner boundary values of the red curve. (b) Areas within the red control curve are unaffected by exterior control curves and the outer boundary values of the red curve.*

In this adaptive sampling scheme, we evaluate the sampling intervals with respect to $\bar{\phi}_C\left(\mathbf{x}'_{\mathbf{i},\mathbf{j}}\right)$ and $\bar{\phi}_E\left(\mathbf{x}'_{\mathbf{i},\mathbf{j}}\right)$ individually instead of together as the single term shown in Eq. (19). Since the two Green's function kernels produce different effects as shown in Fig. 2, we wish to bound each of their piecewise constant approximation errors, so we set the arc length in Eq. (22) to the minimum value computed from among the two. As shown in Fig. 3, this adaptive sampling scheme greatly reduces the sampling rate while maintaining the rendering quality similar to that of dense uniform sampling.

4.3 Culling of control curves

In Eq. (18), ϕ is computed from all of the control curves, which can be expensive when there is a large number of curves, even with the adaptive sampling scheme. However, it is often possible to exclude many control curves without compromising rendering quality.

Shown in Fig. 4(a), control curves that are enclosed by another control curve do not contribute to image values beyond the outer curve. Only the outside boundary values of the outer curve need to be considered; the inside boundary values and the inner curves can be disregarded. Likewise, regions enclosed by a control curve, as shown in Fig. 4(b), are unaffected by exterior curves.

To facilitate culling, we identify areas enclosed by control curves, and organize them hierarchically according to containment relationships. With this preprocessing, unnecessary control curves can be efficiently removed for a given integration region prior to adaptive sampling.

Texture	Figure	Number of curves	Storage (KBytes)	Perf. (ms)	Fitting (s)
Porcelain	Fig. 1 (a)	37,020	2962	98	0
Zephyr	Fig. 1 (b)	327	26	36	0.46
Frog	Fig. 4	820	66	39	0.96
Coffee cup	Fig. 9 (a)	3,321	266	49	0.87
Wood	Fig. 9 (b)	2,698	216	60	660
Dragon	Fig. 9 (c)	160	13	43	0.05

Table 1: *Statistics and rendering performance for the texture data shown in this paper. All images shown in this paper are rendered with a resolution of* 800×600 *and resized to fit the pages.*

5 Experimental Results and Discussions

Texture generation Given a diffusion curve image, the texture is generated by solving Eq. (11) with the normal derivatives $\frac{\partial u(\mathbf{x}')}{\partial \mathbf{n_l}(\mathbf{x}')}$ and $\frac{\partial u(\mathbf{x}')}{\partial \mathbf{n_r}(\mathbf{x}')}$. We uniformly sample a number of points $\{\mathbf{x_i}\}$ along the control curves, with a distance of l between each pair of adjacent points. We denote the left and right sides of the curve at point $\mathbf{x_i}$ as $\mathbf{x_i^0}$ and $\mathbf{x_i^1}$ respectively. Then Eq. (11) is evaluated as

$$u\left(\mathbf{x_i^s}\right) \approx \sum_k \sum_{j \neq i} l \left(u\left(\mathbf{x_j^k}\right) G_n\left(\mathbf{x_i^s}, \mathbf{x_j^k}\right) - \frac{\partial u\left(\mathbf{x_j^k}\right)}{\partial \mathbf{n_k}\left(\mathbf{x_j}\right)} G\left(\mathbf{x_i^s}, \mathbf{x_j^k}\right) \right) + \sum_k \left(u\left(\mathbf{x_i^k}\right) \bar{G}_n(s,k) - \frac{\partial u\left(\mathbf{x_i^k}\right)}{\partial \mathbf{n_k}\left(\mathbf{x_i}\right)} \bar{G}(l) \right). \tag{23}$$

$\bar{G}$ and $\bar{G}_n$ are the analytic integrals of $G(\mathbf{x}, \mathbf{x}')$ and $G_n(\mathbf{x}, \mathbf{x}')$ given in Eq. (9) and Eq. (10) respectively, with $\mathbf{x} \to \mathbf{x}'$ to avoid the singularity:

$$\bar{G}(l) = \frac{1}{2\pi} \int_{l/2}^{l/2} \ln|x| \, \mathrm{d}x = \frac{1}{2\pi}\left(-l + l \ln \frac{l}{2}\right) \tag{24}$$

$$\bar{G}_n(s,k) = (-1)^{s+k} \lim_{x \to 0+} \frac{1}{2\pi} \int_{l/2}^{l/2} \frac{x}{x^2+y^2} \, \mathrm{d}y = \frac{(-1)^{s+k}}{2}. \tag{25}$$

For regions enclosed by a curve with constant color values, such as in the blue and white porcelain of Fig. 1 (a), this evaluation procedure can be skipped. If the texture includes regions enclosed by a curve with color variations, as exhibited in the coffee cup of Fig. 9 (a), then each of these regions can be evaluated independently.

To derive a diffusion curve representation from a natural image, such as Fig. 9 (b), we trace the edges in the image to obtain the control curves and then solve for their colors and normal derivatives from colors sampled throughout the image:

$$u\left(\tilde{\mathbf{x}}_\mathbf{i}\right) \approx \sum_{j,k} l \left(u\left(\mathbf{x_j^k}\right) G_n\left(\tilde{\mathbf{x}}_\mathbf{i}, \mathbf{x_j^k}\right) - \frac{\partial u\left(\mathbf{x_j^k}\right)}{\partial \mathbf{n_k}\left(\mathbf{x_j}\right)} G\left(\tilde{\mathbf{x}}_\mathbf{i}, \mathbf{x_j^k}\right) \right) \tag{26}$$

where points $\{\tilde{\mathbf{x}}_\mathbf{i}\}$ are uniformly sampled over the image, excluding points on the control curves. Since colors need to be fit in addition to the normal derivatives and $|\{\tilde{\mathbf{x}}_\mathbf{i}\}| \gg |\{\mathbf{x_i}\}|$, this process requires much time.

Rendering on GPUs For high performance rendering of diffusion curve textures, we developed a GPU algorithm that utilizes the OpenGL pipeline to render each frame, with texture fetching and computation implemented with NVidia CUDA [NVIDIA 2011]. The geometry, lighting, material and viewpoint are loaded into the OpenGL pipeline. The control curves are loaded into the global memory of CUDA. To render a frame, we compute the transformation, culling, shadow testing, and material shading in the OpenGL

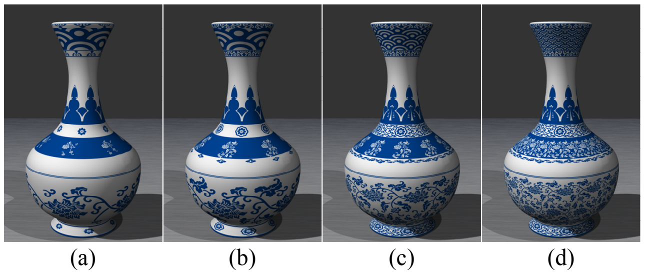

Figure 5: *A vase covered with diffusion curve textures of different complexity. From left to right, the numbers of control curves are 5K, 10K, 20K and 37K respectively.*

Number of control curves	5,309	10,747	20,505	37,020
Adaptive sampling with curve culling	0.020 s	0.043 s	0.076 s	0.098 s
Adaptive sampling w/o curve culling	0.600 s	1.763 s	4.104 s	7.938 s
Uniform sampling w/o curve culling	39.60 s	114.6 s	242.1 s	444.5 s
Resulting image in Fig. 5	(a)	(b)	(c)	(d)

Table 2: *Rendering performance with different numbers of control curves and different rendering algorithms.*

pipeline. At the beginning of the fragment shader, we apply a separate pass to extract the texture coordinates and the corresponding DDX and DDY for each fragment. Then we switch to CUDA processing with this data. The texture coordinates represent the center of the integration region, while the DDX and DDY represent the orientation of the region as well as the length of the two corresponding dimensions. After that, we integrate the texture value for each pixel with the algorithm described above. Finally, the integration values are transported back to the OpenGL pipeline as a texture for subsequent shading in the fragment shader.

Performance We implemented our algorithm on a PC with two Intel Xeon E5620 2.44 GHz CPUs and an NVidia GeForce GTX 480 graphics card with 1 GB of graphics memory. Table 1 lists the statistics and rendering performance for all of the texture data. As shown in the table, our method renders all of these examples in real time. Diffusion curve textures are generated from DCIs at interactive rates. With a natural image such as the wood in Fig. 9 (b) as input, diffusion curve texture generation takes substantially more time as previously discussed.

To test the performance of our rendering method for diffusion curve textures with different levels of complexity, we constructed four diffusion curve textures with different numbers of control curves (5K, 10K, 20K and 37K respectively) and mapped them onto the same vase model as shown in Figure 5. To examine the effects of the two rendering acceleration schemes, we rendered each texture using three implementations: full algorithm, without curve culling, and without both adaptive sampling and curve culling. Table 2 lists the performance results for the four textures. It shows that the adaptive sampling scheme provides a 10× speedup over the implementation without it, while the curve culling scheme further improves the rendering speed by two orders of magnitude. Note that both acceleration schemes are based on the explicit texture representation proposed in this paper. With these two schemes, the rendering performance of our method decreases sub-linearly as the number of control curves increases.

The rendering cost is linearly dependent on the number of curve

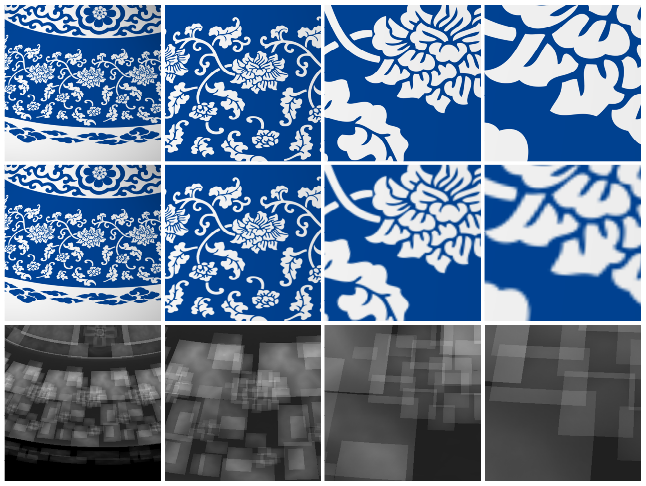

Figure 6: *Rendering for close-up views. The top row rendered with our diffusion curve texture maintains sharp boundaries. The middle row rendered with a rasterized texture of resolution* 4096 × 4096 *exhibits blur when zooming in. The bottom row shows that the sampling rates with our rendering method remain almost unchanged through the zoom sequence. Our rendering times are 76 ms, 76 ms, 70 ms, and 64 ms from left to right.*

sampling points, as the most computationally expensive part in rendering is the evaluation of Green's function kernels. Culling of control curves significantly reduces this cost without requiring much overhead. The number of sampling points is also considerably lessened by our adaptive sampling scheme, which is based on the observation that the final color of a pixel is influenced more by intersecting or nearby control curves than by distant ones. The adaptive scheme densely samples along neighboring control curves and sparsely samples along curves farther away. This leads to artifact-free textures without the prohibitively heavy rendering cost that would result from dense sampling of all the control curves.

Anti-aliasing For a given diffusion curve texture, our method can compute the anti-aliased texture value for each pixel in almost constant time. As a result, the rendering performance of our method is proportional to only the screen resolution and independent of the viewpoint and model complexity. The top row of Figure 6 displays rendering results of our method for the vase under close-up views that magnify the porcelain texture mapped over the surface. With just 3 MB of texture data, our method maintains the resolution independence of diffusion curve images and preserves the sharp edges of the texture pattern in all of the close-up views. By contrast, the results rendered from a 4096 × 4096 raster texture cannot maintain the sharp edges of the texture pattern in the most close-up view (shown in the second row of Figure 6). The bottom row of Figure 6 displays grayscale maps that encode the computational cost for each pixel, which is determined by the number of control curve samples that contribute to it. As the viewpoint moves closer to the object, the computational cost of corresponding pixels is almost unchanged, since the pixel areas generally receive contributions from the same control curve samples.

Figure 7 compares the rendering results generated by our method and the anti-aliased results rendered from a mipmapped raster texture under a series of zoomed-out views. In these zoom-outs, each pixel covers a relatively large region in the texture domain and thus contains many sharp, detailed features. For this texture minification case, our solution generates results similar to the conventional mipmap solution. However, our method is based on a compact vector representation that needs no extra storage and compu-

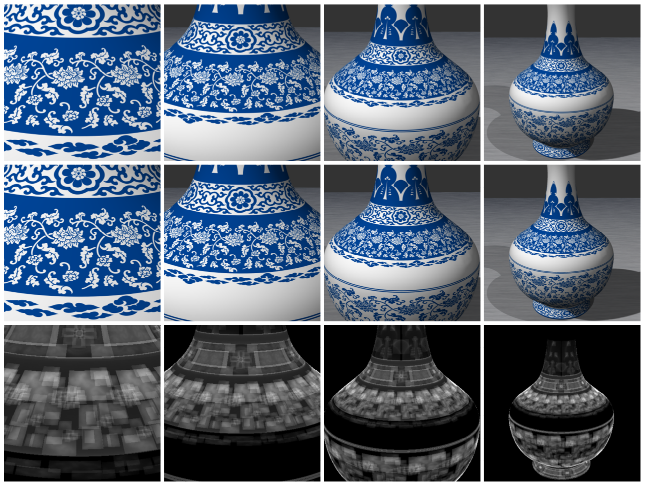

Figure 7: *Rendering for zoomed-out views. The top row is rendered with our diffusion curve texture. The middle row is rendered with a rasterized texture of resolution* 8192×8192*, which generates results similar to our method. The bottom row shows the sampling rates with our method. Our rendering times are 74 ms, 76 ms, 82 ms, and 88 ms from left to right.*

tation for this rendering task, while the raster-based solution needs a 8192×8192 bitmap to represent all the high frequency details and an extra mipmap for anti-aliased rendering. The bottom row of Figure 7 shows a grayscale map of the computational cost. Similar to the texture magnification case, the computational cost of corresponding pixels is fairly stable across the four renderings.

Results Figure 1 shows two objects mapped with diffusion curve textures designed by an artist. The color and texture variations at different scales (e.g. the smooth color variations and the rich texture variations) are effectively modeled and rendered by our method.

Figure 9 (a) displays renderings of a coffee cup using our method. Note that the subtle shading details and sharp features in the diffusion curve texture are well preserved for this and the remaining examples in this section. In a diffusion curve image, boundary colors are taken to be piecewise linear between the color control points, but the corresponding normal derivatives are not. Since we linearly interpolate the normal derivatives between the same set of control points, there may be some color leakage along the curve. This problem can be alleviated by incorporating more control points. Finding a more efficient representation of normal derivatives is a direction for future work.

Figure 9 (b) illustrates the rendering results of a sculpture model. The diffusion curve texture of the wood pattern is obtained by fitting diffusion curves to a rasterized image of the wood texture. Our rendering results accurately convey the natural color variations and sharp boundaries of the wood pattern under the different views.

Figure 9 (d) exhibits the renderings of a dragon model decorated with complicated color patterns designed by an artist using a diffusion curve authoring tool [Orzan et al. 2008]. In this example, both the geometric shapes and texture patterns are complicated, which makes it a challenge for existing diffusion curve rendering methods [Jeschke et al. 2009b; Jeschke et al. 2009a]. Our method well models the complicated color patterns and rich color variations, and generates convincing anti-aliased rendering results. Please see the supplemental video for more rendering results with these models.

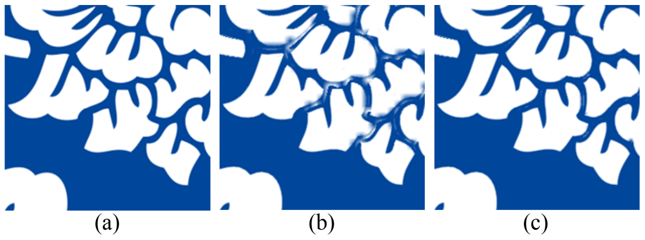

Figure 8: *Close-ups of the blue and white porcelain texture showing detailed features. (a) Texture reconstructed by the Green's function kernels. (b)(c) Texture reconstructed by Jeschke's GPU solver [2009a] at a resolution of* 2048×2048 *and* 4096×4096 *respectively, then rendered with the method of [Jeschke et al. 2009b].*

Discussion Since the Green's function kernels are an explicit representations of DCIs, our method differs substantially from previous methods as it can directly compute image values at arbitrary positions and support random access without a predefined resolution. Sharpness is maintained at close inspection, while smooth anti-aliasing is provided with minified viewing. The storage is both compact and independent of the level of zoom. By contrast, previous implicit solutions obtain results by solving diffusion equations over a grid, which requires prohibitive computation and storage for high magnifications. Jeschke et al. [2009b] solved this problem by proposing an efficient dynamic feature embedding technique that produces crisp, anti-aliased curve details. However, it still needs a reconstructed texture with sufficient resolution which makes sure each pixel only contains at most one control curve. If more than one control curve passes through a pixel, the undersampled details between the control curves cannot be recovered in magnification. A higher grid resolution would be needed to adequately sample these detailed features. This problem is illustrated in Fig. 8 with the method of [Jeschke et al. 2009b]. Because of the numerous control curves in the texture, it takes 0.3s and 170 MBytes of video memory to solve the DCI at a low resolution (2048×2048) that does not fully capture the detailed features in magnification. To generate a result comparable to ours, a higher resolution texture of at least 4096×4096 needs to be solved, at a cost of 0.5s and 552MBytes of video memory. In Jeschke et al. [2009b], the texture is warped according to the view so that it provides sufficient resolution in magnified areas while reducing the resolution in minified areas. While this works well for sparsely distributed control curves, aliasing problems arise in minification for pixels that include multiple curves. Moreover, the warping scheme requires a good surface parameterization for the texture atlas, which is a challenging task for surfaces of complex geometry or high genus (e.g. the dragon and sculpture in Fig. 9). Ours only needs the texture coordinates of each pixel for rendering, and is independent of the texture mapping method.

6 Conclusion

We presented *diffusion curve textures* based on Green's functions for anti-aliased texture mapping of diffusion curve images. Our technique yields exact reconstructions of DCIs, and our GPU implementation provides real-time rendering performance.

The most significant limitation of diffusion curve textures is the overhead for calculating the weights of the Green's function kernels, which depends on normal derivatives along the control curves. While diffusion curve images may be authored using the system in [Orzan et al. 2008], this overhead in converting diffusion curves into our representation causes a delay before the diffusion curve texture can be used. Our technique nevertheless achieves interactive performance with a DCI as input, and we leave real-time texture generation from dynamic DCI input for future work. Faster processing may potentially be obtained by further optimizing the scheme for control curves culling, which can greatly reduce the system scale, and by developing a GPU implementation.

We also will seek to speed up other components of the rendering algorithm. For example, our current method for culling control curves considers only closed boundaries. Additionally removing open curve segments that are distant from the rendering point could lead to further computational savings. We also would like to extend our framework to other differential operators that may be used in vector image creation, such as the bilaplacian. Diffusion surfaces are another interesting extension, as Green's function kernels can be derived similarly for them.

Acknowledgements

We would like to thank the reviewers for their valuable comments. The authors also want to thank Tian Yuan and Shuitian Yan for 3D modeling and texture design. Guofu Xie and Wencheng Wang were partially supported by NSF of China (60833007).

References

BAYIN, Ş. 2006. *Mathematical methods in science and engineering*. Wiley-Interscience.

BEZERRA, H., EISEMANN, E., DECARLO, D., AND THOLLOT, J. 2010. Diffusion constraints for vector graphics. In *Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering*, ACM, New York, NY, USA, NPAR '10, 35–42.

BOLZ, J., FARMER, I., GRINSPUN, E., AND SCHRÖODER, P. 2003. Sparse matrix solvers on the gpu: conjugate gradients and multigrid. *ACM Trans. Graph. 22* (July), 917–924.

BOWERS, J. C., LEAHEY, J., AND WANG, R. 2011. A Ray Tracing Approach to Diffusion Curves. *Computer Graphics Forum 30*, 4, 1345–1352.

D'EON, E., AND IRVING, G. 2011. A quantized-diffusion model for rendering translucent materials. *ACM Trans. Graph. 30* (Aug.), 56:1–56:14.

ELDER, J., AND GOLDBERG, R. 2001. Image editing in the contour domain. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 23*, 3 (mar), 291 –296.

FARBMAN, Z., FATTAL, R., AND LISCHINSKI, D. 2011. Convolution pyramids. *ACM Trans. Graph. 30* (Dec.), 175:1–175:8.

FINCH, M., SNYDER, J., AND HOPPE, H. 2011. Freeform vector graphics with controlled thin-plate splines. *ACM Trans. Graph. 30* (Dec.), 166:1–166:10.

JESCHKE, S., CLINE, D., AND WONKA, P. 2009. A gpu laplacian solver for diffusion curves and poisson image editing. In *ACM SIGGRAPH Asia 2009 papers*, ACM, New York, NY, USA, SIGGRAPH Asia '09, 116:1–116:8.

JESCHKE, S., CLINE, D., AND WONKA, P. 2009. Rendering surface details with diffusion curves. *ACM Trans. Graph. 28* (December), 117:1–117:8.

KAZHDAN, M., AND HOPPE, H. 2008. Streaming multigrid for gradient-domain operations on large images. *ACM Trans. Graph. 27* (August), 21:1–21:10.

LAI, Y.-K., HU, S.-M., AND MARTIN, R. R. 2009. Automatic and topology-preserving gradient mesh generation for image vectorization. *ACM Trans. Graph. 28* (July), 85:1–85:8.

LECOT, G., AND LEVY, B. 2006. Ardeco: Automatic region detection and conversion. In *Eurographics Symposium on Rendering*.

LIPMAN, Y., LEVIN, D., AND COHEN-OR, D. 2008. Green coordinates. In *ACM SIGGRAPH 2008 papers*, ACM, New York, NY, USA, SIGGRAPH '08, 78:1–78:10.

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

NVIDIA, 2011. CUDA programming guide 4.0, May. http://developer.nvidia.com/object/cuda.html.

ORZAN, A., BOUSSEAU, A., WINNEMÖLLER, H., BARLA, P., THOLLOT, J., AND SALESIN, D. 2008. Diffusion curves: a vector representation for smooth-shaded images. In *ACM SIGGRAPH 2008 papers*, ACM, New York, NY, USA, SIGGRAPH '08, 92:1–92:8.

PANG, W., QIN, J., COHEN, M., HENG, P., AND CHOI, K. 2011. Fast rendering of diffusion curves with triangles. *Computer Graphics and Applications, IEEE PP*, 99, 1.

QIN, Z., MCCOOL, M. D., AND KAPLAN, C. S. 2006. Real-time texture-mapped vector glyphs. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA, I3D '06, 125–132.

QIN, Z., MCCOOL, M. D., AND KAPLAN, C. 2008. Precise vector textures for real-time 3d rendering. In *Proceedings of the 2008 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA, I3D '08, 199–206.

RAMANARAYANAN, G., BALA, K., AND WALTER, B. 2004. Feature-Based Textures. *Eurographics Symposium on Rendering*.

SEN, P., CAMMARANO, M., AND HANRAHAN, P. 2003. Shadow Silhouette Maps. *ACM Transactions on Graphics (TOG) (Proceedings of SIGGRAPH 2003) 22*, 3, 521–526.

SEN, P. 2004. Silhouette maps for improved texture magnification. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, ACM, New York, NY, USA, HWWS '04, 65–73.

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

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

TARINI, M., AND CIGNONI, P. 2005. Pinchmaps: textures with customizable discontinuities. *Computer Graphics Forum 24*, 3, 557–568.

TUMBLIN, J., AND CHOUDHURY, P. 2004. Bixels: Picture samples with sharp embedded boundaries. In *Rendering Techniques*, 255–264.

WANG, L., ZHOU, K., YU, Y., AND GUO, B. 2010. Vector solid textures. *ACM Trans. Graph. 29* (July), 86:1–86:8.

WANG, L., YU, Y., ZHOU, K., AND GUO, B. 2011. Multiscale vector volumes. *ACM Trans. Graph. 30* (Dec.), 167:1–167:8.

XIA, T., LIAO, B., AND YU, Y. 2009. Patch-based image vectorization with automatic curvilinear feature alignment. *ACM Trans. Graph. 28* (December), 115:1–115:10.

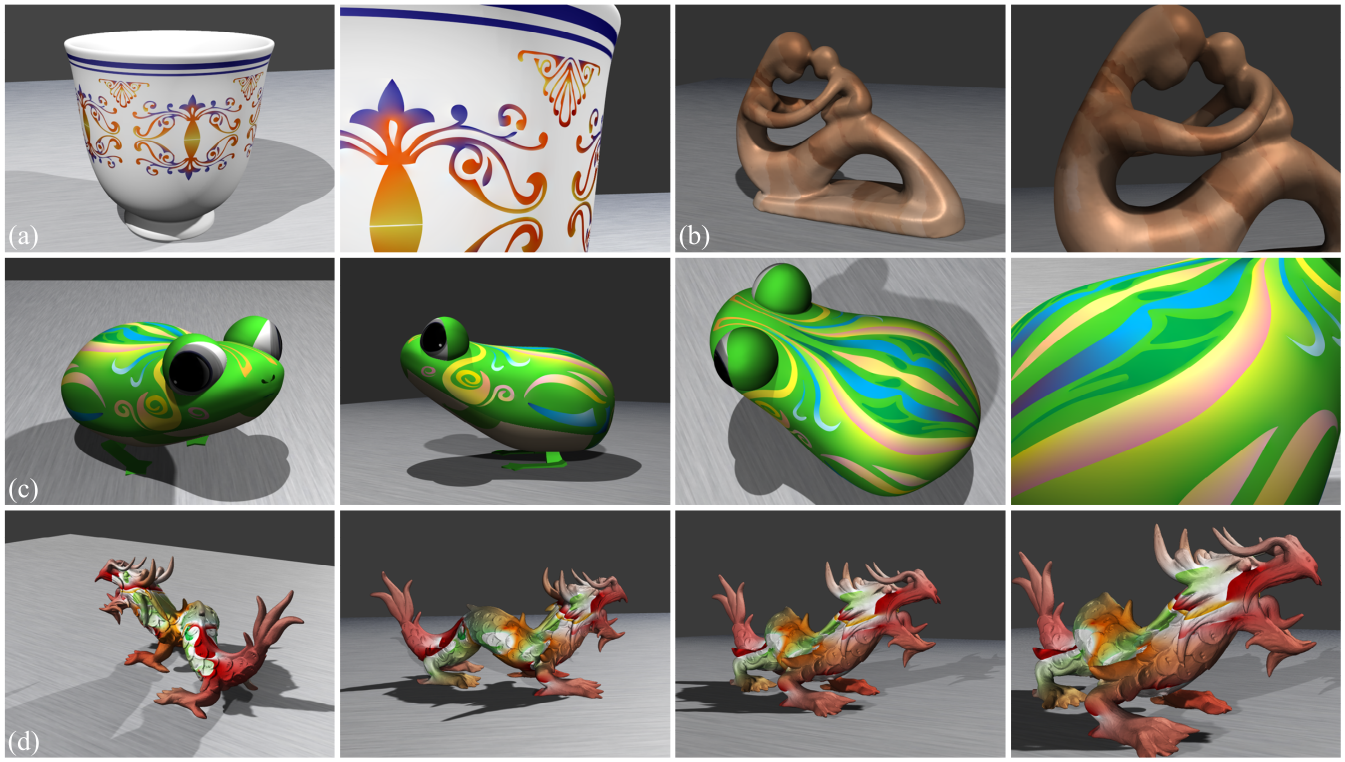

Figure 9: *Some results rendered with diffusion curve textures. (a) A coffee cup that exhibits both subtle shading details and sharp features. (b) A sculpture mapped with wood texture obtained by fitting diffusion curves to a rasterized texture image. (c) A frog model with a diffusion curve texture designed using a diffusion curve authoring tool. (d) A dragon model with abundant geometry details.*

Appendix A

In the following, we provide a detailed derivation of Eq. (14) and Eq. (16).

Based on Eq. (9), the indefinite integral of the Green's function $G\left(\mathbf{R},\mathbf{x}'\right)$ can evaluated analytically with the substitution of $\hat{\mathbf{x}} = (\hat{x},\hat{y})$ for $\mathbf{x}-\mathbf{x}'$:

$$\begin{aligned}
&\iint G\left(\mathbf{x},\mathbf{x}'\right)\,\mathrm{d}\mathbf{x} = \frac{1}{2\pi}\int\left(\int \ln\sqrt{\hat{x}^2+\hat{y}^2}\,\mathrm{d}\hat{x}\right)\mathrm{d}\hat{y}\\
&= \frac{1}{4\pi}\int\left(-2\hat{x}+\hat{x}\ln\left(\hat{x}^2+\hat{y}^2\right)+2\hat{y}\,\mathrm{arctg}\left(\frac{\hat{x}}{\hat{y}}\right)+A_0\left(\hat{y}\right)\right)\mathrm{d}\hat{y}\\
&= \frac{1}{4\pi}\left(-3\hat{x}\hat{y}+\hat{x}\hat{y}\ln\left(\hat{x}^2+\hat{y}^2\right)+\hat{x}^2\,\mathrm{arctg}\left(\frac{\hat{y}}{\hat{x}}\right)+\hat{y}^2\,\mathrm{arctg}\left(\frac{\hat{x}}{\hat{y}}\right)\right)\\
&\quad + A_1\left(\hat{x}\right)+A_2\left(\hat{y}\right) = \frac{1}{4\pi}H_G\left(\hat{x},\hat{y}\right)+A_1\left(\hat{x}\right)+A_2\left(\hat{y}\right). \qquad (27)
\end{aligned}$$

$\{A_i\}$ are arbitrary functions because of the indefinite integrals. H_G is given in Eq. (15), and Eq. (14) is obtained from it accordingly.

Similarly, the indefinite integral of the normal derivative $G_n\left(\mathbf{x},\mathbf{x}'\right)$ from Eq. (10) is solved as

$$\begin{aligned}
&\iint G_n\left(\mathbf{x},\mathbf{x}'\right)\,\mathrm{d}\mathbf{x} = \frac{1}{2\pi}\int\left(\int \frac{\hat{x}\,n_x+\hat{y}\,n_y}{\hat{x}^2+\hat{y}^2}\,\mathrm{d}\hat{x}\right)\mathrm{d}\hat{y}\\
&= \frac{1}{2\pi}\int\left(n_y\,\mathrm{arctg}\left(\frac{\hat{x}}{\hat{y}}\right)+\frac{n_x}{2}\ln\left(\hat{x}^2+\hat{y}^2\right)+A_0\left(\hat{y}\right)\right)\mathrm{d}\hat{y}\\
&= \frac{1}{2\pi}\left(n_y\hat{y}\,\mathrm{arctg}\left(\frac{\hat{x}}{\hat{y}}\right)+n_x\hat{x}\,\mathrm{arctg}\left(\frac{\hat{y}}{\hat{x}}\right)+\frac{1}{2}\left(n_y\hat{x}+n_x\hat{y}\right)\ln\left(\hat{x}^2+\hat{y}^2\right)\right)\\
&\quad + A_1\left(\hat{x}\right)+A_2\left(\hat{y}\right) = \frac{1}{2\pi}H_{G_n}\left(\hat{x},\hat{y},n_x,n_y\right)+A_1\left(\hat{x}\right)+A_2\left(\hat{y}\right) \qquad (28)
\end{aligned}$$

where H_{G_n} is given in Eq. (17) and Eq. (16) is determined from it.

Appendix B

Here, we derive the analytical form of $\bar{\phi}'_C\left(\mathbf{x}'_{\mathbf{i},\mathbf{j}}\right)$ and $\bar{\phi}'_E\left(\mathbf{x}'_{\mathbf{i},\mathbf{j}}\right)$ used to compute Eq. (22).

From Eq. (19) and Eq. (21), we can express $\bar{\phi}'_E\left(\mathbf{x}'\right)$ as

$$\bar{\phi}'_E\left(\mathbf{x}'\right) = \frac{\partial\mathbf{E}\left(\mathbf{x}'\right)}{\partial\mathbf{t}\left(\mathbf{x}'\right)}F_G\left(\mathbf{R},\mathbf{x}'\right)+\mathbf{E}\left(\mathbf{x}'\right)\frac{\mathrm{d}}{\mathrm{d}\mathbf{x}'}F_G\left(\mathbf{R},\mathbf{x}'\right)\cdot\mathbf{t}\left(\mathbf{x}'\right) \qquad (29)$$

where the differential of F_G is dependent on the differential of H_G, which we compute from Eq. (15) as

$$\begin{aligned}
\frac{\partial}{\partial x'}H_G\left(\hat{x},\hat{y}\right) &= -2\hat{x}\,\mathrm{arctg}\left(\frac{\hat{y}}{\hat{x}}\right)+\hat{y}\left(2-\ln\left(\hat{x}^2+\hat{y}^2\right)\right);\\
\frac{\partial}{\partial y'}H_G\left(\hat{x},\hat{y}\right) &= -2\hat{y}\,\mathrm{arctg}\left(\frac{\hat{x}}{\hat{y}}\right)+\hat{x}\left(2-\ln\left(\hat{x}^2+\hat{y}^2\right)\right) \qquad (30)
\end{aligned}$$

where $\hat{x} = x_i - x'$ and $\hat{y} = y_i - y'$.

Similarly, we obtain $\bar{\phi}'_C\left(\mathbf{x}'\right)$ based on Eq. (16) and Eq. (17):

$$\bar{\phi}'_C\left(\mathbf{x}'\right) = \frac{\partial\mathbf{C}\left(\mathbf{x}'\right)}{\partial\mathbf{t}\left(\mathbf{x}'\right)}F_{G_n}\left(\mathbf{R},\mathbf{x}'\right)+\mathbf{C}\left(\mathbf{x}'\right)\frac{\mathrm{d}}{\mathrm{d}\mathbf{x}'}F_{G_n}\left(\mathbf{R},\mathbf{x}'\right)\cdot\mathbf{t}\left(\mathbf{x}'\right) \qquad (31)$$

where the differential of F_{G_n} is dependent on the differential on H_{G_n} computed from Eq. (17):

$$\begin{aligned}
&\frac{\partial}{\partial x'}H_{G_n}\left(\hat{x},\hat{y},n_x,n_y\right) = -n_y-\left(n_x+\frac{\partial n_x}{\partial x'}\hat{x}\right)\mathrm{arctg}\left(\frac{\hat{y}}{\hat{x}}\right)\\
&-\frac{\partial n_y}{\partial x'}\hat{y}\,\mathrm{arctg}\left(\frac{\hat{x}}{\hat{y}}\right)-\frac{1}{2}\left(n_y+\frac{\partial n_y}{\partial x'}\hat{x}+\frac{\partial n_x}{\partial x'}\hat{y}\right)\ln\left(\hat{x}^2+\hat{y}^2\right);\\
&\frac{\partial}{\partial y'}H_{G_n}\left(\hat{x},\hat{y},n_x,n_y\right) = -n_x-\left(n_y+\frac{\partial n_y}{\partial y'}\hat{y}\right)\mathrm{arctg}\left(\frac{\hat{x}}{\hat{y}}\right)\\
&-\frac{\partial n_x}{\partial y'}\hat{x}\,\mathrm{arctg}\left(\frac{\hat{y}}{\hat{x}}\right)-\frac{1}{2}\left(n_x+\frac{\partial n_x}{\partial y'}\hat{y}+\frac{\partial n_y}{\partial y'}\hat{x}\right)\ln\left(\hat{x}^2+\hat{y}^2\right). \qquad (32)
\end{aligned}$$

As $\frac{\partial\mathbf{C}(\mathbf{x}')}{\partial\mathbf{t}(\mathbf{x}')}$, $\frac{\partial\mathbf{E}(\mathbf{x}')}{\partial\mathbf{t}(\mathbf{x}')}$, $\frac{\partial\mathbf{n}(\mathbf{x}')}{\partial\mathbf{t}(\mathbf{x}')}$ and $\frac{\mathrm{d}\mathbf{n}(\mathbf{x}')}{\mathrm{d}\mathbf{x}'}$ can be obtained from the control curves, $\bar{\phi}'_C\left(\mathbf{x}'\right)$ and $\bar{\phi}'_E\left(\mathbf{x}'\right)$ can be calculated analytically as above.