Fast Multipole Representation of Diffusion Curves and Points

Timothy Sun* Papoj Thamjaroenporn* Changxi Zheng

Columbia University

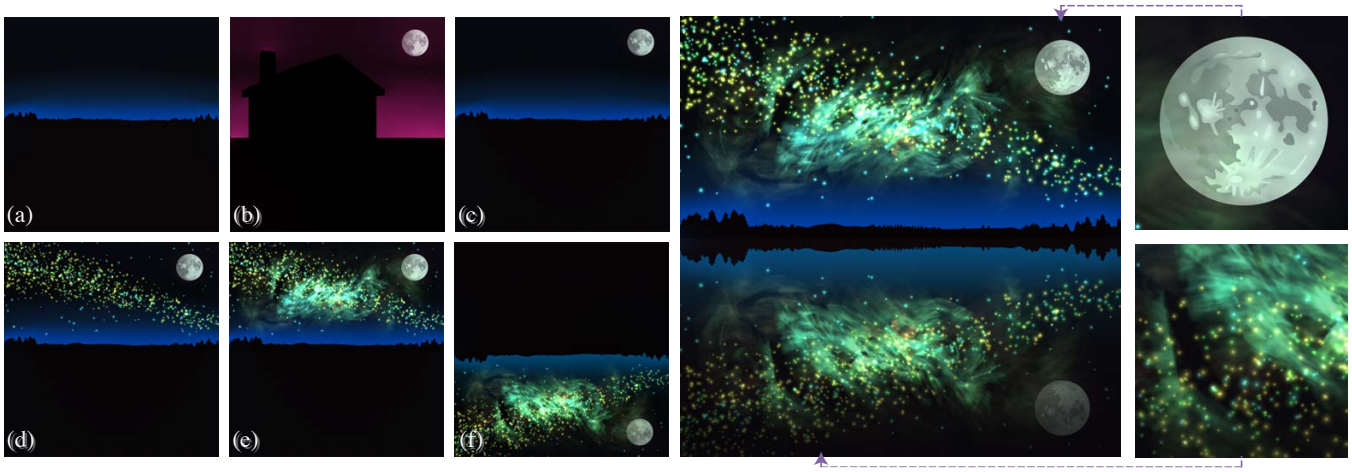


Figure 1: Starry Night: *Starting with two diffusion curve images (a) and (b), the moon in (b) is cloned into (a), producing (c). Diffusion points are added to simulate stars (d) and an aurora (e). For the reflection on the lake (f), the hue is shifted towards green. Finally, (e) and (f) are composited with a mask to produce the final image on the right. Note that the reflection is darker away from the horizon. All these editing operations were performed on the fast multipole representation.*

Abstract

We propose a new algorithm for random-access evaluation of *diffusion curve images* (DCIs) using the *fast multipole method*. Unlike all previous methods, our algorithm achieves *real-time* performance for rasterization and texture-mapping DCIs of up to millions of curves. After precomputation, computing the color at a single pixel takes nearly constant time. We also incorporate Gaussian radial basis functions into our *fast multipole representation* using the fast Gauss transform. The fast multipole representation is not only a data structure for fast color evaluation, but also a framework for vector graphics analogues of bitmap editing operations. We exhibit this capability by devising new tools for fast diffusion curve Poisson cloning and composition with masks.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation, Graphics Utilities—Display Algorithms

Keywords: diffusion curves, vector graphics, fast multipole method, fast Gauss transform, image editing

Links: DL PDF WEB VIDEO

*joint first authors

1 Introduction

Diffusion curves [Orzan et al. 2008] are powerful vector graphics primitives for creating smoothly shaded images. Roughly, user-defined colors along a set of control curves are diffused across the entire image plane. This model is compact and resolution-independent like traditional vector graphics models, and its increasing popularity has motivated recent work on improving its runtime evaluation performance [Pang et al. 2012; Jeschke et al. 2009; Ilbery et al. 2013], extending its expressiveness [Bezerra et al. 2010; Finch et al. 2011], shading volumetric objects [Takayama et al. 2010], and texturing [Sun et al. 2012].

One trend for rasterizing diffusion curve images makes use of boundary element methods (BEMs), which start by precomputing color derivatives along the control curves. Current BEM-based algorithms [Sun et al. 2012; Ilbery et al. 2013] accelerate color evaluation by introducing adaptive schemes or evaluating several points at once, but none are able to render images in real time.

In this paper, we propose a different BEM method, the *fast multipole representation* (FMR), for rendering DCIs. The FMR stores a lattice on the image plane, and for any point located in a cell, its color is described by a precomputed asymptotic expansion at the cell's center and a boundary integral of a few nearby curves. This representation produces a continuous color distribution nearly identical to the original DCI. In contrast to previous methods, our color evaluation per pixel has nearly *constant* complexity after precomputation except in rare cases where many curves meet at a point.

Furthermore, using the fast Gauss transform [Greengard and Strain 1991], the FMR can also represent Gaussian radial basis functions, which we call *diffusion points* in this context. We use diffusion points for adding simple effects (Figure 1) to DCIs that are inefficient to achieve with just diffusion curves. Unlike DCIs, evaluating an image composed of diffusion points requires no boundary integral, and thus each pixel can be rendered in constant time.

In addition, via computations on the FMR, we build new editing operations for DCIs. We introduce DCI analogues of Poisson cloning and composition with masks that run at interactive rates by directly manipulating fast multipole expansion coefficients and avoiding any recomputation of the boundary element solve.

2 Related Work

Diffusion curves Diffusion curves were introduced by Orzan et al. [2008] as a new vector graphics format that allowed for intuitive design of complex color gradients. Diffusion curve images are the result of diffusing the colors defined along control curves until the color field reaches an equilibrium, and this resting state can be described by a Laplace equation. Since directly solving a Laplace equation on a large grid is expensive, the standard approach for rasterizing such images is to use a multigrid method (e.g. [Jeschke et al. 2009]). In these methods, capturing sharp features typically requires a high-resolution discretization, motivating methods which do not require discretizing the image plane.

Random access evaluation Especially for texture-mapping, it is useful to quickly evaluate the color value at a random point in the image plane rather than an entire grid of points. One such method [Pang et al. 2012] triangulates the image plane and carefully interpolates the color values to the rest of the domain. However, they cannot guarantee higher-order color continuity across adjacent triangles. Boye et al. [2012] improve on that approach by interpolating higher-order terms across triangles, but it is unclear whether or not they can render images in real time. Bowers et al. [2011] reinterpreted color evaluation as a global illumination problem and applied a stochastic ray-tracing method to compute the color, but one would need to incorporate reflections to accurately evaluate the color value in a complicated region.

BEM-based methods One family of methods starts with the boundary element method. By rephrasing the Laplace equation as a boundary integral along the curves, color gradients along each control curve are computed by solving a dense linear system. After this point, Sun et al. [2012] evaluate color values by computing a boundary integral while Ilbery et al. [2013] rasterize an image using a so-called "line-by-line" approach. We also start with a BEM solve, but unlike all previous methods, our color evaluation algorithm is *real-time*, even without any adaptive discretization or sampling.

Fast multipole method The fast multipole method of Greengard and Rokhlin [1987] was originally developed for quickly stepping n-body simulations but has since been applied to other partial differential equations such as the Laplace equation. Applications of the fast multipole method in computer graphics include radiosity computation [Hanrahan et al. 1991], smoke simulation [Brochu et al. 2012] and physics-based sound rendering [Zheng and James 2010; Zheng and James 2011].

Image Editing There is a wealth of tools for editing pixel images such as Poisson cloning [Pérez et al. 2003] and composition with masks [Porter and Duff 1984], but in contrast, there are few such tools for vector graphics, especially DCIs. Bezerra et al. [2010] introduced a set of editing operations for DCIs. They argued that controlling the diffusion process (as opposed to editing control curves) not only simplifies designing complex images, but also allows for more expressive and varied visual effects. However, their algorithm only operates on a grid of pixel values. In contrast, our proposed editing tools directly operate on the resolution-independent FMR of DCIs.

3 Fast Multipole Representation

Notation Throughout this paper, a scalar field is always denoted by a lowercase letter. Since the RGB channels of a color value are solved independently, we use the letter u to denote one of the three channels when there is no confusion. A 2D coordinate is denoted by a bold letter such as $\boldsymbol{x}$ with two components x_1 and x_2. A complex number is always denoted by z, possibly with subscripts.

$\alpha = (\alpha_1, \alpha_2), \alpha_i \in \mathbb{Z}^+$ denotes a 2D *multi-index* which is used to describe an asymptotic series in two variables. Its length $|\alpha|$ is the sum of its components $\alpha_1 + \alpha_2$, and its factorial is $\alpha! = \alpha_1!\alpha_2!$. A 2D monomial $\boldsymbol{x}^\alpha$ is defined as $\boldsymbol{x}^\alpha = x_1^{\alpha_1} x_2^{\alpha_2}$.

Background on the Boundary Integral Formulation As formulated in [Jeschke et al. 2009; Sun et al. 2012], the color u at $\boldsymbol{x}$ is a *harmonic* function; that is, it satisfies the Laplace equation

$$\begin{aligned} u(\boldsymbol{x}) &= \{C_l(\boldsymbol{x}), C_r(\boldsymbol{x})\} && \forall \boldsymbol{x} \in \mathbb{B} \\ \Delta u(\boldsymbol{x}) &= 0 && \text{otherwise} \end{aligned} \tag{1}$$

where the boundary $\mathbb{B}$ is the set of control curves, and C_l and C_r are the colors along the "left" and "right" sides of the control curves. We follow Sun et al. [2012] and Ilbery et al. [2013] by expressing (1) as a boundary integral using its Green's function [Liu and Nishimura 2006]:

$$u(\boldsymbol{x}) = -\int_{\mathbb{B}} \left[E(\boldsymbol{y}) G(\boldsymbol{x};\boldsymbol{y}) - C(\boldsymbol{y}) \frac{\partial G(\boldsymbol{x};\boldsymbol{y})}{\partial \boldsymbol{n}(\boldsymbol{y})} \right] ds(\boldsymbol{y}), \tag{2}$$

where $\boldsymbol{n}(\boldsymbol{y})$ is the normal vector at a point $\boldsymbol{y} \in \mathbb{B}$, and

$$G(\boldsymbol{x};\boldsymbol{y}) = \frac{1}{2\pi} \ln \|\boldsymbol{x} - \boldsymbol{y}\| \;\text{ and }\; \frac{\partial G(\boldsymbol{x};\boldsymbol{y})}{\partial \boldsymbol{n}(\boldsymbol{y})} = \frac{(\boldsymbol{x}-\boldsymbol{y})^T \boldsymbol{n}}{2\pi \|\boldsymbol{x}-\boldsymbol{y}\|^2} \tag{3}$$

are the Laplace Green's function and its normal derivative, respectively. The functions $C(\boldsymbol{y})$ and $E(\boldsymbol{y})$ are the *differences* between the boundary values on each side of the control curves. That is,

$$C(\boldsymbol{y}) = C_l(\boldsymbol{y}) - C_r(\boldsymbol{y}) \;\text{ and }\; E(\boldsymbol{y}) = \frac{\partial u}{\partial \boldsymbol{n}_l}(\boldsymbol{y}) + \frac{\partial u}{\partial \boldsymbol{n}_r}(\boldsymbol{y}), \tag{4}$$

where $\boldsymbol{n}_l$ and $\boldsymbol{n}_r$ are the normals on the left and right side of a curve (i.e., $\boldsymbol{n}_l = -\boldsymbol{n}_r$). *Boundary element methods* (BEM) obtain gradient values E by solving a discretized version of (2). For a description of a BEM for diffusion curves, see Appendix A.

Remark. The BEM solve only considers color and gradient *differences* along control curves, as indicated by (4). For this reason, Sun et al. [2012] need a user-specified control curve enclosing the entire image which incidentally encodes *absolute* values, even though the BEM solve treats that curve no differently. Since the boundary integral evaluated at a point outside of a closed boundary is 0 (Appendix B), the color evaluated on the outer boundary is equal to the relative difference defined on that curve, so we can consider that outer curve as storing absolute instead of relative values (see Figure 2).

Our solution for setting the outer boundary values was to automatically assign color and gradient values along a rectangular outer curve (see Figure 2) using a low-resolution finite element solver [Jeschke et al. 2009]. By specifying an outer boundary this way, the BEM solve will generate images comparable to those of the finite element-based approaches without any additional user-specified artificial boundary values.

To avoid having to integrate over all the control curves in (2), Sun et al. [2012] first check if the evaluation point is within some closed region, and only integrate over curves inside the closed region. Furthermore, instead of integrating over an entire curve, they adaptively

(a) (b) (c)

Figure 2: *The outer boundary (the green curve in (a)) controls absolute colors, while the interior curves (the white curves in(a)) specifies color differences across the curves. By shifting the color only on the boundary, the color in the interior shifts correspondingly (b)-(c). (DCI from [Orzan et al. 2008])*

sample the integral. Our algorithm also avoids having to consider all control curves by only integrating nearby curves and capturing all remaining curves in an asymptotic expansion, as detailed in the following sections.

3.1 Fast Evaluation Algorithm in Brief

Our reformulation of (2) starts by defining a lattice on the image plane (Figure 4). The color value at any position $\boldsymbol{x}$ located in a cell $\mathcal{C}$ is decomposed into far-field and near-field terms:

$$u(\boldsymbol{x}) = \sum_{|\alpha|<N} c_\alpha (\boldsymbol{x}-\boldsymbol{x}_c)^\alpha - \int_{\mathbb{D}} \left[E(\boldsymbol{y})G(\boldsymbol{x};\boldsymbol{y}) - C(\boldsymbol{y})\frac{\partial G(\boldsymbol{x};\boldsymbol{y})}{\partial \boldsymbol{n}(\boldsymbol{y})} \right] ds(\boldsymbol{y}), \tag{5}$$

where $\boldsymbol{x}_c$ is the center of cell $\mathcal{C}$, c_α are precomputed expansion coefficients, and $\mathbb{D}$ is the set of curve segments contained in the 3×3 neighborhood centered at $\mathcal{C}$ (e.g., the purple and white cells in Figure 4). If our order of expansion is N, then the number of coefficients c_α is $N(N+1)/2$.

Remark. Sun et al. [2012] added antialiasing by integrating (2) over a rectangular region. Since our method is based on the same boundary integral, we can also achieve this effect by integrating (5). The first term can be integrated because it is a polynomial, and the second term can be integrated using their rectangle integral. However, since a pixel can span several grid cells, each corresponding to a different integration of (5), the performance will drop when we rasterize a zoomed-out image. We also note that it is unclear how Sun et al. combine their antialiasing technique with curve culling when the rectangular region intersects multiple closed regions. Instead, because we can quickly evaluate the color value at a single point, our current example figures use supersampling for antialiasing (we use 9 samples per pixel in practice), and we leave a more efficient antialiasing scheme as future exploration.

3.2 Preliminary Formulas

We start by preparing a few computational building blocks for the fast multipole method, leaving the mathematical derivations in Appendix C. In many of our formulas, we use several other points as centers for series expansions, which are summarized in Figure 3.

The derivation of our formulas is carried out in complex notation for mathematical convenience. We will first derive a power series expansion that is equivalent to the first term of (5). Namely,

$$\mathrm{Re}\left(-\sum_{t=0}^{N} L_t(z_L)\frac{(z-z_L)^t}{t!} \right), \tag{6}$$

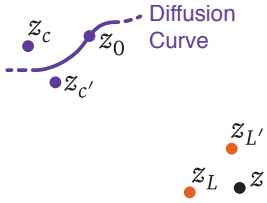


Figure 3: *We need expansion centers near the boundary curve (z_c and $z_{c'}$) for the multipole expansion (used in* (17)*), and expansion centers near the evaluation point (z_L and $z_{L'}$) for the local expansion (used in* (20)*).*

where the L_t terms are called *local coefficients* for the cell center z_L of $\mathcal{C}$. We refer to this series as the *local expansion* centered at z_L. Once we have established this local expansion, the c_α coefficients in (5) can be computed using the formula

$$c_\alpha = -\frac{\mathrm{Re}(i^{\alpha_2} L_{|\alpha|}(z_L))}{\alpha!}\binom{|\alpha|}{\alpha_1}. \tag{7}$$

The Laurent series of the Green's function Now we rewrite the boundary integral (2) as an expansion. First, the Green's function in (3) is expressed as

$$G(z;z_0) = \frac{1}{2\pi}\ln(z-z_0) \quad z \in \mathbb{C}, \tag{8}$$

where $z = x_1 + ix_2$ and $z_0 = y_1 + iy_2$ are just complex representations of the 2D coordinates $\boldsymbol{x}$ and $\boldsymbol{y}$, respectively. This is an analytic function with real part equal to the standard Laplace Green's function. The major advantage of the complex notation is that given any point z_c that satisfies $|z_c - z_0| \ll |z - z_c|$ (see Figure 3), we can rewrite the new Green's function as

$$G(z;z_0) = \frac{1}{2\pi}\left[\ln(z-z_c) + \ln(1 - \frac{z_0 - z_c}{z - z_c}) \right], \tag{9}$$

and expand the second term using its Laurent series. This yields a power series of $G(z;z_0)$ expanded at z_c:

$$G(z;z_0) = \frac{1}{2\pi}\sum_{k=0}^{\infty} S_k(z-z_c)R_k(z_0-z_c), \tag{10}$$

where S_k and R_k are the *singular* and *regular* functions respectively:

$$S_k(z) = \begin{cases} \dfrac{(k-1)!}{z^k} & \text{for } k \geq 1 \\ -\ln(z) & \text{for } k = 0, \end{cases} \qquad R_k(z) = -\frac{z^k}{k!}, \quad \text{for } k \geq 0. \tag{11}$$

We write the first term of (2) using complex notation as

$$g(z) = \int_{\mathbb{B}} E(z_0)G(z;z_0)ds(z_0), \tag{12}$$

where $E : \mathbb{C} \to \mathbb{R}$ now takes in a complex number instead of a 2D coordinate. Substituting the Laurent series in this integral yields an asymptotic expansion

$$g(z) = \frac{1}{2\pi}\sum_{k=0}^{\infty} S_k(z-z_c)M_k(z_c), \tag{13}$$

where the coefficients $M_k(z_c)$ are defined as

$$M_k(z_c) = \int_{\mathbb{B}} E(z_0)R_k(z_0-z_c)ds(z_0). \tag{14}$$

Following the same derivation for the second integral of (2), we get another expansion of the form

$$f(z) = \frac{1}{2\pi} \sum_{k=1}^{\infty} S_k(z - z_c) N_k(z_c), \tag{15}$$

where the coefficients $N_k(z_c)$ are defined as

$$N_k(z_c) = \int_{\mathbb{B}} C(z_0) n(z_0) R_{k-1}(z_0 - z_c) ds(z_0). \tag{16}$$

Here $n(z_0)$ is the normal at z_0 represented as a complex number.

Equations (13) and (15) are the *multipole expansions* of the Laplace Green's function and its normal derivative. If the points z_0 on the integrated curves all satisfy the condition $|z_c - z_0| \ll |z - z_c|$ for any expansion center z_c, the boundary integral (2) has the converging expansion

$$u(z) = \frac{1}{2\pi} \sum_{k=1}^{\infty} A_k(z_c) S_k(z - z_c), \tag{17}$$

where $A_k(z_c) = N_k(z_c) + M_k(z_c)$, the *moments* of the integral, depend only on the expansion center z_c.

Moment-to-Moment (M2M) Translation The expansion center z_c in (17) is arbitrary as long as the condition $|z_c - z_0| \ll |z - z_c|$ is satisfied. When we select a different expansion center $z_{c'}$ (also satisfying $|z_{c'} - z_0| \ll |z - z_{c'}|$), we want to find moments $A_k(z_{c'})$ for the expansion (17), i.e.,

$$u(z) = \frac{1}{2\pi} \sum_{k=1}^{\infty} A_k(z_{c'}) S_k(z - z_{c'}). \tag{18}$$

These moments can be efficiently computed from the known moments $A_k(z_c)$ using the so-called *M2M formula*,

$$A_k(z_{c'}) = -\sum_{j=0}^{k} A_j(z_c) R_{k-j}(z_c - z_{c'}). \tag{19}$$

Moment-to-Local (M2L) Expansion Armed with the M2M formula, it would seem like the multipole expansion in (17) can be directly used for computing the color distribution, but because the Laurent series only converges when the expansion center z_c is close to the control curves and far from the evaluation point, we would have to evaluate many multipole expansions to cover all the curves. Instead, we use a single local expansion (6) to capture all of the far-field contribution because it converges when expanded far away from control curves.

Fortunately, we can convert moments into local coefficients efficiently. Given moments $A_k(z_c)$ of an expansion centered at z_c, we can also rewrite it as an asymptotic expansion with respect to another point z_L that is farther from z_c than the evaluation point z, i.e., $|z - z_L| \ll |z_L - z_c|$ (see Figure 3). The series expansion can be expressed as:

$$u(z) = \sum_{t=0}^{\infty} L_t(z_L; z_c) R_t(z - z_L), \tag{20}$$

where the coefficients $L_t(z_L; z_c)$ are computed using the *M2L formula*

$$L_t(z_L; z_c) = \frac{(-1)^{t+1}}{2\pi} \sum_{k=0}^{\infty} S_{k+t}(z_L - z_c) A_k(z_c). \tag{21}$$

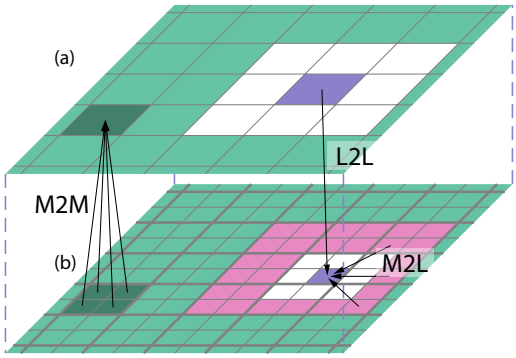


Figure 4: *Part of a hierarchical lattice with depictions of moment and local translation formulas. The color evaluation is performed on the finest level, while the hierarchical structure is used to accelerate the precomputation of expansion coefficients on the finest level of grids.*

Local-to-Local (L2L) Translation Our final subroutine is for translating the local expansion (20) from an expansion center z_L to another one $z_{L'}$. Given a known set of coefficients $L_t(z_L; z_c)$ and a new point $z_{L'}$ also satisfying $|z - z_c| \ll |z_{L'} - z_c|$, we can compute the local coefficients for $z_{L'}$ using the *L2L formula*

$$L_t(z_{L'}; z_c) = -\sum_{k=0}^{\infty} L_{k+t}(z_L; z_c) R_k(z_{L'} - z_L). \tag{22}$$

3.3 Hierarchical Precomputation of Local Coefficients

We now describe an efficient algorithm based on the *fast multipole method* [Greengard and Rokhlin 1987] for precomputing local coefficients L_t for all the cells hierarchically. When we associate moments and local coefficients with a cell, we are always referring to those coefficients from expansions centered at the cell's center. For a local expansion, we refer to the cell by $\mathcal{C}_L$ and its center by z_L, and for a multipole expansion, $\mathcal{C}_c$ and z_c, respectively.

The convergence conditions have a simple interpretation when we expand at cell centers. For the multipole expansion of a cell $\mathcal{C}_c$, we integrate over only the curves inside of that cell. Then, the expansion converges when the evaluation point is *outside* of the 3×3 neighborhood of cells around $\mathcal{C}_L$. On the other hand, the local expansion with coefficients $L_t(z_L; z_c)$ converges when the evaluation point is *inside* $\mathcal{C}_L$ and z_c is outside of the 3×3 neighborhood around $\mathcal{C}_L$. The local coefficients $L_t(z_L)$ associated with a cell $\mathcal{C}_L$ should capture the contributions of all curves outside of the 3×3 neighborhood around $\mathcal{C}_L$. Thus we can compute them by summing over all coefficients:

$$L_t(z_L) = \sum_{\mathcal{C}_c \in \mathcal{L}_{\text{far}}(\mathcal{C}_L)} L_t(z_L; z_c), \tag{23}$$

where $\mathcal{L}_{\text{far}}(\mathcal{C}_L)$, the *far cells* of $\mathcal{C}_L$, are the cells outside of its 3×3 neighborhood.

We first create a flat lattice with square cells on the image plane. The lattice cells cut the diffusion curves into small segments so that every segment is wholly contained in a cell. The most straightforward way of computing $L_t(z_L)$ is to compute the moments for each cell $\mathcal{C}_c$ using (14) and (16). For each cell $\mathcal{C}_L$, one can apply M2L translation (21) to every other cell $\mathcal{C}_c$ outside of its 3×3 neighborhood to obtain its local coefficients. Computing the moments requires integrating each curve segment once and the M2L translation iterates

Algorithm 1 Fast multipole method for DCIs

Require: Color and gradient values C, E from BEM, depth L.
  procedure FMMDCI(C, E, L)
    Initialize a hierarchy with L levels.
    for all cells $\mathcal{C}$ in level L **do**
      Compute moments expanded at $\mathcal{C}$'s center (14), (16).
    end for
    for $l = L-1, \ldots, 0$ **do**
      for all cells in level l **do**
        Translate children's moments with M2M (19).
      end for
    end for
    for $l = 2, \ldots, L$ **do**
      for all cells in level l **do**
        Translate moments with M2L (19).
        Translate parent's local coefficients with L2L (22).
      end for
    end for
  end procedure

over almost all the cells when computing the coefficients of a single cell. The complexity of this naive approach is $O(n^2 + s)$ where n is the number of cells in the lattice and s is the number of curve segments.

Hierarchical Lattice A much faster algorithm uses a hierarchical lattice structure (see Figure 4) and is summarized in Algorithm 1. For simplicity, we assume the dimension of the flat lattice is a power of 2. We build a hierarchy of lattices $\mathcal{L}_0, \mathcal{L}_1, \ldots, \mathcal{L}_h$, where the top lattice $\mathcal{L}_0$ has one cell and $\mathcal{L}_h$ is the initial flat lattice. Each cell in lattice $\mathcal{L}_{h'}$ (where $h' < h$) has four child cells in $\mathcal{L}_{h'+1}$. In general, like in the case of a rectangular image, the lattice $\mathcal{L}_0$ is not always one cell. The hierarchical lattice facilitates a "divide-and-conquer" approach for computing the local expansion by a two-pass propagation of the expansion coefficients.

Upward Propagation As in the naive algorithm, we first compute the moments of each cell in the finest level (i.e., the flat lattice $\mathcal{L}_h$). For each cell $\mathcal{C}_c$ in $\mathcal{L}_{h-1}$ (the dark green cell in Figure 4(a)), we apply M2M translation (19) to shift the moments of the child cells (the dark green cells in Figure 4(b)) in $\mathcal{L}_h$ from their centers to the center of $\mathcal{C}_c$. Summing the translated moments yields the moments of $\mathcal{C}_c$, and we continue propagating upwards until we reach lattice $\mathcal{L}_2$. We can compute moments for $\mathcal{L}_1$ and $\mathcal{L}_0$, but they will not be used.

Downward Propagation The local expansion at each cell is calculated by propagating downwards. The downward propagation phase starts from $\mathcal{L}_2$ of our hierarchy, since only from that level downwards does a cell have a set of far cells (we only computed moments up to $\mathcal{L}_2$ for the same reason). At each level h', we separate the far cells of each cell $\mathcal{C}_L$ into two sets:

1. the cells not adjacent to $\mathcal{C}_L$ but their parents at level $h'-1$ are adjacent to the parent of $\mathcal{C}_L$ (the pink cells in Figure 4(b)), and
2. the rest of far cells (the green cells in Figure 4(b)).

We iterate over each cell in the first group of cells (the pink cells) and apply M2L translation (21) to translate each cell's moments to a local expansion for $\mathcal{C}_L$. For the remaining far cells of $\mathcal{C}_L$, their contribution has already been approximated by the local expansion coefficients of the parent of $\mathcal{C}_L$ (the purple cell in Figure 4(b)) during downward propagation on the previous level. That is, the parents of the remaining cells were far cells in level $h'-1$ (compare the green cells of Figure 4(a) and (b)). Therefore, we can translate the local coefficients of $\mathcal{C}_L$'s parent to the center of $\mathcal{C}_L$ using L2L translation (22). By linearity of integration, we add the expansion coefficients from both parts to form the final local coefficients L_t of $\mathcal{C}_L$. This propagation proceeds until we reach the finest level L_h. Both propagation steps require $O(n \log n)$ time, so the entire computation has $O(n \log n + s)$ complexity. Table 1 shows the timing of the propagation steps for different hierarchy depths.

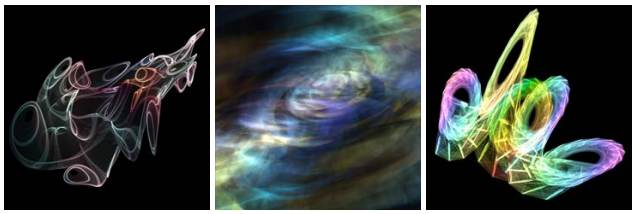


Figure 5: *We simulate a variety of* strange attractors *[Kemp 1998], which are 3D chaotic systems. We then assign more than 1 million diffusion points to track the system's geometric structure and procedurally generate these graphics. This is also the way we generate the aurora in Figure 1. Equations and parameters of the attractors are from [Abraham 2010].*

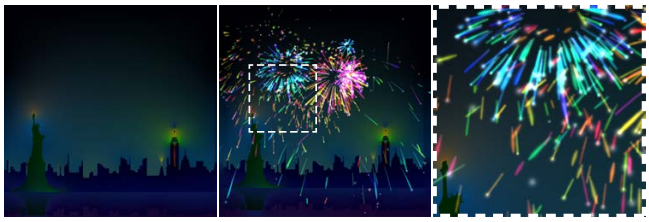


Figure 6: Fireworks*: Using the FMR, our algorithm renders vector graphics consisting of both complex diffusion curves (the background) and numerous diffusion points (the firework particles) fast enough to make real-time animations where the user can zoom in and out freely (see the supplemental video).*

Numerical Integral and Series Truncation Throughout the hierarchical computation, we perform various integrals and series evaluations. These integrals are computed using a piecewise-constant discretization, as we did for the BEM solve (see Appendix A). We evaluate each series by truncating them with a finite number of orders. Since each color channel is quantized in the range $[0, 255]$, a small number of orders are sufficient for producing results nearly identical to directly evaluating (2). Unless otherwise stated, each image was rendered using an expansion order of $N = 4$.

4 Diffusion Points

We now extend the FMR to represent any smooth color field using *Gaussian radial basis functions* (RBFs). Let $u(\boldsymbol{x})$ be some 2D scalar field represented by Gaussian RBFs, i.e.,

$$u(\boldsymbol{x}) = \sum_{i=0}^{M} A_i e^{-\|\boldsymbol{x}-\boldsymbol{x}_i\|^2/r_i^2}, \tag{24}$$

where the number of basis functions M can be large. We can also represent Gaussian RBFs in the FMR, in which the color value $u(\boldsymbol{x})$ in a cell $\mathcal{C}$ is similarly approximated by a power series expansion

$$u(\boldsymbol{x}) = \sum_{|\alpha|<N} b_\alpha (\boldsymbol{x} - \boldsymbol{x}_c)^\alpha, \tag{25}$$

where N is small fixed value, and b_α is a set of expansion coefficients for cell $\mathcal{C}$. For our examples, we used $N = 5$. In contrast

Algorithm 2 Fast Gauss transform for diffusion points

Require: Gaussian functions $\phi_1, \ldots, \phi_M$, error tolerance ϵ.
  procedure FASTGAUSS($\{\phi_i\}, \epsilon$)
    Initialize a hierarchy with cell size $\min_i h_i$.
    for $i = 1, \ldots, M$ **do**
      Set $d_i \leftarrow 2\left\lceil (r_i/l)\sqrt{\ln(A_i/\epsilon)}\right\rceil + 1$.
      for all cells $\mathcal{C}$ in $d_i \times d_i$ neighborhood around $\boldsymbol{x}_i$ **do**
        Add ϕ_i to b_α coefficients of $\mathcal{C}$ using FGT (26).
      end for
    end for
  end procedure

to (5) for the Laplace Green's function, there is no boundary integral because Gaussians have no singularities. Consequently, evaluating a color value using (25) always has constant complexity. To distinguish them from diffusion curves, we call a Gaussian RBF a *diffusion point* described by its position $\boldsymbol{x}_i$, kernel size r_i, and color value A_i. In general, Gaussians are impractical to simulate using diffusion curves because they are smooth everywhere, while diffusion curves typically have color or derivative discontinuities across control curves.

Diffusion points allow us to represent smooth nonharmonic color fields which can be rasterized in time *strictly* linear in the number of pixels after precomputation, regardless of the number of points. Since diffusion curves and points share a similar color evaluation formula, we can incoporate both primitives by merging the coefficients in (25) with the expansion coefficients in (5) without any storage and performance overhead (see Figure 6).

4.1 Fast Gauss Transform

We achieve (25) by expanding a Gaussian function using the *fast Gauss transform* [Greengard and Strain 1991],

$$e^{-\|\boldsymbol{x}-\boldsymbol{x}_i\|^2/r_i^2} = \sum_{|\alpha|<\infty} \frac{1}{\alpha! r_i^{|\alpha|}} h_\alpha\left(\frac{\boldsymbol{x}_i - \boldsymbol{x}_c}{r_i}\right)(\boldsymbol{x} - \boldsymbol{x}_c)^\alpha. \quad (26)$$

The derivation of this formula and the associated error bound we use can be found in Appendix D. Note that this summation has the same form as (25), except that $|\alpha|$ runs to infinity, so we truncate the series up to $|\alpha| = N$. The coefficient $h_\alpha(\boldsymbol{y})$ is a 2D Hermite expansion coefficient defined as

$$h_\alpha(\boldsymbol{y}) = h_{\alpha_1}(\boldsymbol{y}_1) h_{\alpha_2}(\boldsymbol{y}_2), \quad (27)$$

where $h_n(a)$ is an "Hermite function" computable by the recurrence relation

$$\begin{aligned} h_0(a) &= e^{-a^2}, \quad h_1(a) = 2ae^{-a^2}, \\ h_{n+1}(a) &= 2ah_n(a) - 2nh_{n-1}(a). \end{aligned} \quad (28)$$

4.2 Precomputation of Expansion Coefficients

We can derive analogous multipole and local expansions similar to the procedure in §3 for a Gaussian basis function, but fortunately, there is a simpler algorithm (Algorithm 2) for computing the expansion coefficients using the observation that Gaussians only affect a local region, i.e., they decay exponentially as we get farther from its center.

Assume that the image plane is the unit square $[0,1] \times [0,1]$. In our precomputation process, we first create a lattice with cells of side length $l = \min_i r_i$. For each cell, we wish to compute, within an error tolerance of ϵ, the local expansion coefficients b_α in (25), where the expansion center $\boldsymbol{x}_c$ is the center of the cell. In all our examples, we used $\epsilon = 10^{-5}$. For each Gaussian basis function ϕ_i contained in cell C_i, we consider the $(2n_i + 1) \times (2n_i + 1)$ neighborhood around C_i, where

$$n_i = \left\lceil (r_i/l)\sqrt{\ln(A_i/\epsilon)}\right\rceil. \quad (29)$$

For each cell C' in the neighborhood, we add the contribution of ϕ_i to the local expansion of C' using the fast Gauss transform (26). Since the contribution of a diffusion point is additive and only affects a small region, a diffusion point image can be updated in real time, allowing the user to interactively edit such an image.

#Curves	FMR Upward/Downward Pass (ms)				
	L 5	L 6	L 7	L 8	L 9
2048	2.6/3.2	3.6/10	7.6/21	22/86	92/318
8192		4.5/7.6	8.5/23	27/87	96/310
32768			12/24	35/85	106/315
131072				59/81	126/318
524288					187/324

Table 1: *The cost of propagating moments and local coefficients depends on the number of cells and the number of curves.*

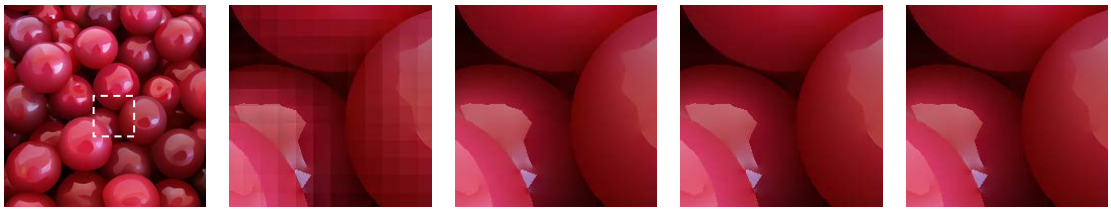
#Curves	1024×1024 Rasterization (ms)					
	L 5	L 6	L 7	L 8	L 9	L 10
2048	176	68.3	39.4	31.1	23.9	19.8
8192		170	68.1	38.2	29.8	23.1
32768			171	68.8	39.8	32.2
131072				176	72.0	39.2
524288					180	72.5

Table 2: *The total rasterization time depends on the ratio between curves and cells, and along diagonals (i.e. where the ratio is the same), the performance is roughly constant.*

5 Performance Evaluation

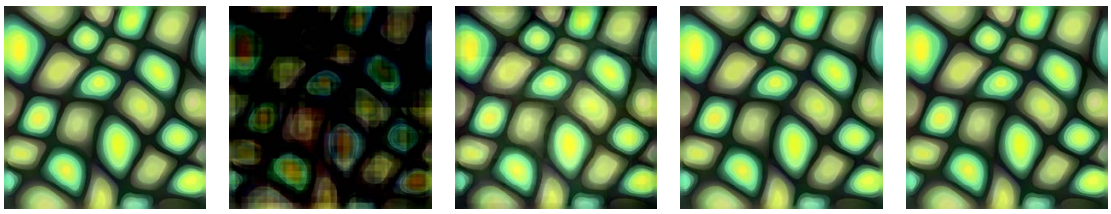
Complexity We consider the cost of rasterizing a single DCI from our fast multipole representation compared to the costs of other methods. Because we truncate the local expansion, its evaluation has constant cost per pixel. Let n and s be the number of lattice cells and curve segments, respectively. The average number of segments in a 3×3 neighborhood of cells is $9s/n$. Assuming that the curve segments and pixels are uniformly distributed in the image plane, the average complexity of computing the boundary integral contribution is $O(s/n)$. Thus the total cost per pixel is $O(s/n + 1)$, which is nearly constant, as the number of segments s is often much smaller than the number of lattice cells n. Refining the lattice only improves the performance—n increases quadratically while s only grows linearly.

Speed We measured the performance on an Intel Quad-Core Xeon E5 (3.10 GHz) processor with 64GB RAM. In order to test how our algorithm scales on large examples, we generated DCIs representing different iterations of the Hilbert space-filling curve (Figure 9(a)), where the number of curve segments ranges from 2048 to 524288, depending on the space-filling iterations. Every time we increase the iteration number by 1, the number of curves quadruples. To create a suitable FMR, we cut these segments further: the largest example has 1922732 segments. We also note that our Hilbert curves are drawn as one large closed loop, and therefore we cannot effectively cull curves as suggested in [Sun et al. 2012]. Table 1 contains timings of the FMR construction for different curve complexities and grid sizes. The only dependence on the number of



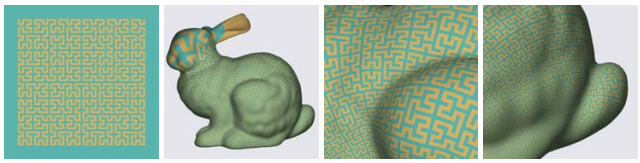
(a) (b) (c) (d) (e)

Figure 7: Rasterization with varying orders of expansion: *We generated the same image using the original boundary integral evaluation (a) and FMR rasterization on a* 64×64 *grid with varying orders of expansion (b)-(e). When we use* $N = 1, 2$ *(b)-(c), cell boundaries are visible. For higher orders, the cell boundaries disappear, and for* $N = 4$ *(e), the result is indistinguishable from the true solution.*



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

Figure 8: Precomputation with varying orders of expansion: *By using too small of an order of expansion (*$N = 1, 2, 3$ *for (b)-(d)) during precomputation, the resulting local coefficients visibly differ from the true solution up to* N*th order. For* $N = 4$ *(e) we get an accurate computation comparable to the true solution (a).*



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

Figure 9: Hilbert's Bunny: *Our method can handle DCIs with a large number of curves. (a) shows the 5th iteration of the Hilbert curve. On the GPU (b)-(d), we can obtain real-time framerates even for DCIs with over a million curves. The textures we used here are not antialiased.*

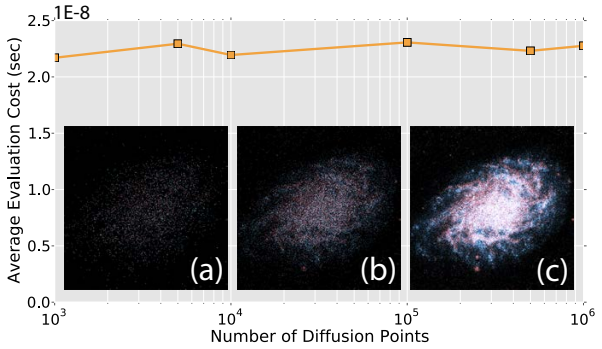


Figure 10: A Galaxy of Points: *The average computation time per pixel was roughly constant, even as we increased the number of diffusion points from* 10^4 *(a) to* 10^6 *(c).*

curves is computing the moments on the finest level. Thus, upward propgation takes longer for more complicated DCIs, but downward propagation has roughly constant performance for the same grid size.

In accordance with our complexity analysis, Table 2 shows that the rasterization time depends on the *density* of the curve segments in the grid as opposed to the absolute number of segments. Here the number of local coefficients in the FMRs is fixed at $N = 4$. All the timings are done for non-antialiased 1024×1024 images. In the case of diffusion points, Figure 10 conveys the constant complexity of evaluating a pixel. As a result, we are able to rasterize these images quickly—even with a CPU implementation, we can achieve *interactive* framerates for DCIs with millions of diffusion curves and points. Sun et al. [2012] rendered DCIs of up to about 38000 curves at interactive framerates using adaptive sampling and curve culling heuristics. We accomplished *real-time* framerates on the GPU (Figure 9) for all of our Hilbert curve DCIs. Our implementation was written in OpenGL GLSL and executed on an Nvidia Quadro K600 graphics card with 1GB memory.

Accuracy If we ignore truncation, our color evaluation formula (5) converges exactly to the original boundary integral (2) over all the curves. That is, changing the rasterization resolution of the grid only affects speed but not accuracy. We found that an order of expansion $N = 4$ is sufficient for all of our examples, as the resulting images are identical to the naive boundary integral evaluation: no color channel in any pixel differs by more than 1 unit. Using too few coefficients, however, causes discontinuities across cell boundaries, as seen in Figure 7.

Because we truncate the number of moments and local coefficients in upward and downward propagation, the M2L and L2L formulas introduce error that accumulates for each level in the hierarchy (note that the M2M formula involves only a finite summation, introducing no truncation error). Once again, we found that $N = 4$ was sufficient for all our examples (see Figure 8). Since the time required for upward and downward propagation is already on the order of tenths of a second, one can conservatively compute many more coefficients during propagation than necessary for rasterization.

Space Consumption For color evaluation purposes, it is enough to store just the control curve segments and the local coefficients of

Figure 11: DCI Cloning: *Given source and target DCIs (1st and 2nd columns), copying the control curves directly and recomputing the BEM solve leaves obvious artifacts (3rd column). Our algorithm makes the cloning seamless (4th and 5th columns). Cloning the top (DCI from [Orzan et al. 2008]) and bottom rows took 301ms and 242ms, respectively. The grid size for each image is* 512×512*, and the total number of curves in the final face and fishtank is* 21301 *and* 18870*, respectively. In each example, the target's colors "bleed" into the copied source patch, a property also found in pixel-based Poisson cloning.*

the finest lattice. The total size is dominated by the local coefficients, since control curves are usually sparse in the image plane. For example, for a 256×256 grid with an order of expansion $N = 4$, storing the local coefficients with single floating-point precision takes about 7.5MB, while a 512×512 grid will require four times as much space. The sparsity of the control curves also suggests adaptive space partitioning, and we leave that as a future optimization of our implementation.

6 Editing FMRs

The fast multipole representation can be constructed whenever we know both the color and gradient values along the control curves, even if they are not harmonic. That is, the computational bottleneck of the dense linear solve in precomputation can be avoided if these values are known. We propose some simple editing operations that involve either known or manually set gradient values. Since the fast multipole representation can be computed quickly (as shown in Table 1), these editing operations run interactively. The examples presented here were generated using DCIs created by Orzan et al. [2008] and those we created ourselves, the latter of which will be released to the public domain.

6.1 Diffusion Curve Cloning

One of the most useful local editing tools for bitmap images is *Poisson cloning* [Pérez et al. 2003; Agarwala et al. 2004; Farbman et al. 2009]. When cloning a region of a source image into a target image, this method automatically adjusts the local color distribution to create a seamless transition across the boundary of the region. For bitmap images, most of these algorithms solve a Poisson equation discretized at image pixels. In our version of Poisson cloning for DCIs, we do not require any linear solve.

When we merge two DCIs together, simply copying or replacing parts of the control curves in a target DCI is insufficient, as seen in the third column of Figure 11. These artifacts are due to the global effects of the colors on the control curves—the curves from the source image will influence the color distribution on the target image. Consequently, one has to carefully adjust the control colors. For a complex source DCI, this process is laborious and time-consuming.

Let Γ_s be the boundary of a selection region in a source image $\mathcal{D}_s$.

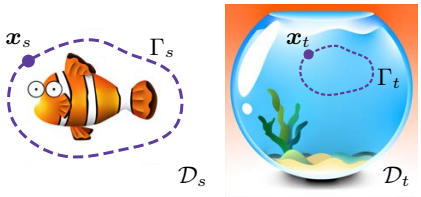


Figure 12: *Cutting out a source DCI (left) and pasting it into the target (right).*

We first copy Γ_s and all the control curves (along with their color and gradient values) inside of Γ_s to the target image $\mathcal{D}_t$ (see Figure 12). The user can optionally rotate or scale the copied curves. We set the color value of $\boldsymbol{x}_t \in \Gamma_t$ to be the difference between the color in $\mathcal{D}_t$ and the color at the corresponding point $\boldsymbol{x}_s$ in $\mathcal{D}_s$, just like in Poisson cloning. The gradient value is set to 0 so the transition across the boundary is smooth. Now that all the control curves have color and gradient values, the fast multipole representation can be constructed using upward and downward propagation.

6.2 Composition with Masks

For multi-layer DCI composition, we address two problems: how to represent a vector graphic mask, and how to render the composite image efficiently. In this subsection, we assume that the two DCIs to be composited have FMRs where the cells align. If not, one of the FMRs can be quickly recomputed using upward and downward propagation so that both lattices have the same resolution.

Mask Representation If we desire a gradual transition between the two images, we represent the mask using uniformly-spaced diffusion points. Given a user-specified mask region, we place a diffusion point on each cell center contained in the region, as in Figure 14. The user can vary the smoothness by adjusting the kernel size of the diffusion points.

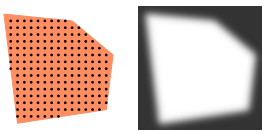


Figure 14: *For each cell center in the user specified region, we place a diffusion point (shown on the left), yielding the resulting mask on the right.*

Figure 13: DCI Composition: *Given two source DCIs and a diffusion point mask (1st-3rd columns), we can smoothly interpolate between the two images according to the mask (4th and 5th columns). Our method also handles multi-layer composition (bottom row).*

Composition Now we have three FMRs with the same lattice resolution, two for the original DCIs and one for the mask. The composition combines the three FMRs together into one single representation. Let $V_1(\boldsymbol{x})$ and $V_2(\boldsymbol{x})$ denote the color values of the two source DCIs at the position $\boldsymbol{x}$, and $M(\boldsymbol{x})$ denote the mask value clamped to $[0, 1]$. The color value at $\boldsymbol{x}$ in the resulting FMR is

$$V_o(\boldsymbol{x}) = M(\boldsymbol{x})V_1(\boldsymbol{x}) + (1 - M(\boldsymbol{x}))V_2(\boldsymbol{x}). \tag{30}$$

We create the resulting FMR by merging the expansion coefficients and copying the local curve segments in each cell. The color values $V_1(\boldsymbol{x})$ and $V_2(\boldsymbol{x})$ can be written as

$$V_i(\boldsymbol{x}) = \sum_{|\alpha|<N} c_{i,\alpha}(\boldsymbol{x} - \boldsymbol{x}_c)^\alpha + A_i(\boldsymbol{y}), \quad i = 1, 2, \tag{31}$$

where A_i is the local integral term in (5) while the mask value $M(\boldsymbol{x})$ is expressed using (25). Substituting both expressions into (30) yields

$$\begin{aligned} V_o(\boldsymbol{x}) = &\sum_{|\alpha'|<N} \rho_{\alpha'}(\boldsymbol{x} - \boldsymbol{x}_c)^{\alpha'} \\ &+M(\boldsymbol{x})A_1 + (1 - M(\boldsymbol{x}))A_2(\boldsymbol{y}), \end{aligned} \tag{32}$$

where $\rho_{\alpha'}$ are the merged expansion coefficients (see details in Appendix E). It turns out that $\rho_{\alpha'}$ is determined by coefficients whose order is smaller than that of α', so by merging, we do not lose any accuracy. These coefficients and curve segments form the FMR of the composited image. For masks with sharp transitions, we can use diffusion curves, where the only difference is that formulas involving the mask $M(x)$ will have an integration term.

At first, it seems that (32) achieves little speedup over the naive approach of evaluating color individually because the mask value $M(\boldsymbol{x})$ and the local line integrals A_1 and A_2 still have to be evaluated. As noted in §3.1, most cells' neighborhoods do not have any curve segments, so the last two terms of (32) vanish. Therefore, for most of the color evaluation, we only need to evaluate the constant-complexity first term of (32).

7 Conclusion

We have proposed the fast multipole representation, a new vector graphics representation for diffusion curves and points. This representation allows random-access color evaluation in nearly constant time as well as new tools for interactive DCI editing operations.

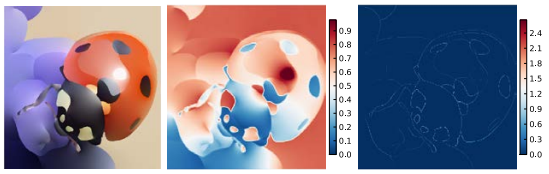


Figure 15: Spatial Coherence: *For an FMR rasterization (left), the multipole and local expansions are spatially coherent (middle and right). (DCI from [Orzan et al. 2008])*

We believe that the FMR approach leads to many interesting future possibilities for the improvement of color evaluation and antialiasing efficiency as well as the enrichment of vector graphics editing tools. For instance, fast multipole methods have also been developed for biharmonic functions [Gumerov and Duraiswami 2006], and it is possible to extend our work to biharmonic DCIs. One limitation of the current algorithm is its storage size: a single FMR on a 1024×1024 grid can be as large as 100MB. One future direction is to develop a compression scheme for FMRs (perhaps on top of adaptive space partitioning). Because the expansion coefficients are distributed in a spatially coherent manner (Figure 15), one possibility is to adapt high dynamic range image compression methods [Li et al. 2005; Munkberg et al. 2006] to FMRs.

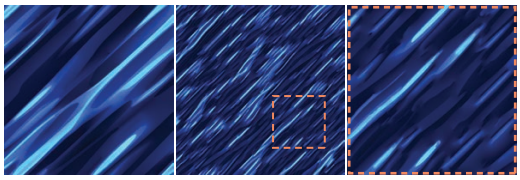


Figure 16: *Preliminary texture synthesis results on DCIs: We use a small DCI (left) as input to synthesize a large DC texture (middle and right).*

Lastly, one of the main objectives of Sun et al. [2012] is representing textures as DCIs. Given a small texture, a natural problem is synthesizing a larger DCI texture. The lattice structure of the FMR essentially divides the image plane in a way similar to pixels, and the hierarchy of lattices might be adapted to hierarchical texture synthesis methods, such as those based on image pyramids. Figure 16 shows some results we obtained from a naive adaptation of [Lefebvre and Hoppe 2005].

A A BEM Solver for DCIs

Recall the boundary integral representation of a DCI (2)

$$u(\boldsymbol{x}) = -\int_{\mathbb{B}} \left[E(\boldsymbol{y})G(\boldsymbol{x};\boldsymbol{y}) - C(\boldsymbol{y})\frac{\partial G(\boldsymbol{x};\boldsymbol{y})}{\partial \boldsymbol{n}(\boldsymbol{y})} \right] ds(\boldsymbol{y}),$$

where the Green's function is

$$G(\boldsymbol{x};\boldsymbol{y}) = \frac{1}{2\pi}\ln\|\boldsymbol{x}-\boldsymbol{y}\|, \quad \frac{\partial G(\boldsymbol{x};\boldsymbol{y})}{\partial \boldsymbol{n}(\boldsymbol{y})} = \frac{1}{2\pi\|\boldsymbol{x}-\boldsymbol{y}\|^2}(\boldsymbol{x}-\boldsymbol{y})^T\boldsymbol{n}$$

Since the input DCI has colors defined on each control curve, we have *Dirichlet boundary conditions*. That is, we know the color value $C(\boldsymbol{y})$ for all points $\boldsymbol{y} \in \mathbb{B}$, so we need to solve for the normal gradients $E(\boldsymbol{y})$. More explicitly, after an application of the chain rule, we have the constraints of the form

$$2\pi C(\boldsymbol{x}) = \int_{\mathbb{B}} \left[\frac{C(\boldsymbol{y})}{r_{\boldsymbol{xy}}}\frac{\partial r_{\boldsymbol{xy}}}{\partial \boldsymbol{n}} - \ln(r_{\boldsymbol{xy}})E(\boldsymbol{y}) \right] ds(\boldsymbol{y}),$$

where $\boldsymbol{x} \in \mathbb{B}$ and $r_{\boldsymbol{xy}} = \|\boldsymbol{x}-\boldsymbol{y}\|$. We discretize $\mathbb{B}$ into linear segments $b_1, b_2, \ldots b_M$ with constant shape function $C(\boldsymbol{y}_i)$ where $\boldsymbol{y}_i$ denotes the midpoint of b_i. For each pair $\boldsymbol{y}_i$ and b_j, consider the orthogonal projection $\boldsymbol{p}_{ij}$ of $\boldsymbol{y}_i$ onto b_j. The perpendicular distance $||\boldsymbol{y}_i - \boldsymbol{p}_{ij}||$ is denoted n_{ij}. Let s_{ij} and t_{ij} be the signed distance of the beginning and end of b_j, respectively, from $\boldsymbol{p}_{ij}$, where the positive direction is along b_j. For any point $\boldsymbol{y}$ on b_j, let y_s denote its signed distance from p_{ij}. Then we have the identities

$$r_{y_i y}^2 = y_s^2 + n_{ij}^2, \text{ and } \frac{1}{r_{y_i y}}\frac{\partial r_{y_i y}}{\partial \boldsymbol{n}} = \frac{n_{ij}}{r_{y_i y}^2} = \frac{n_{ij}}{y_s^2 + n_{ij}^2}.$$

These relationships allow us to express the integrals of the Green's function and its normal derivative in terms of an integral over signed distance. Our constraints thus become

$$\begin{aligned}\pi C(\boldsymbol{y}_i) &= \sum_{j=1}^{M}\int_{s_{ij}}^{t_{ij}} \left(\frac{C(\boldsymbol{y}_j)n_{ij}}{q^2+n_{ij}^2} - \frac{1}{2}\ln(q^2+n_{ij}^2)E(y_j) \right) \mathrm{d}q \\ &= \sum_{j=1}^{M} C(\boldsymbol{y}_j)a_{ij} - E(y_j)b_{ij},\end{aligned}$$

where

$$a_{ij} = \int_{s_{ij}}^{t_{ij}} \frac{n_{ij}}{q^2+n_{ij}^2}\mathrm{d}q, \quad b_{ij} = \frac{1}{2}\int_{s_{ij}}^{t_{ij}} \ln(q^2+n_{ij}^2)\mathrm{d}q.$$

We can now obtain the normal gradients $E(\boldsymbol{y}_i)$ by solving the linear system $\mathbf{Be} = (\mathbf{A} - \pi\mathbf{I})\mathbf{c}$, where $\mathbf{A}$ and $\mathbf{B}$ have entries a_{ij} and b_{ij}, and $\mathbf{c}$ and $\mathbf{e}$ are vectors with entries $C(\boldsymbol{y}_i)$ and $E(\boldsymbol{y}_i)$.

B Integrating Over A Closed Curve

Let $\Omega \subseteq \mathbb{R}^2$ be a simply-connected region in the plane. Since the color u is harmonic, i.e. $\Delta u = 0$, by Green's second identity, the boundary integral (2) can be expressed as

$$\begin{aligned}u(\boldsymbol{x}) &= \int_{\partial\Omega} \left[u(\boldsymbol{y})\frac{\partial G(\boldsymbol{x};\boldsymbol{y})}{\partial \boldsymbol{n}(\boldsymbol{y})} - \frac{\partial u}{\partial \boldsymbol{n}}(\boldsymbol{y})G(\boldsymbol{x};\boldsymbol{y}) \right] ds(\boldsymbol{y}) \\ &= \int_{\Omega} u(\boldsymbol{y})\Delta G(\boldsymbol{x};\boldsymbol{y})ds(\boldsymbol{y}) \\ &= \int_{\Omega} u(\boldsymbol{y})\delta(\boldsymbol{x}-\boldsymbol{y})ds(\boldsymbol{y}),\end{aligned}$$

where δ is the Dirac delta function. If $\boldsymbol{x}$ is outside of Ω, then $\delta(\boldsymbol{x}-\boldsymbol{y})$ is always 0, so the color vanishes.

C FMM Translation Formulas

In this appendix, we derive the translation formulas stated in Section 3.2. The following derivations are sketched out in [Liu and Nishimura 2006]—here we merely flesh out the details.

C.1 Moment-to-Moment

We consider the translation of the moment M_k at an expansion center z_c to a new center $z_{c'}$. Recall that this moment is defined as

$$M_k(z_c) = \int_{\mathbb{B}} E(z_0)R_k(z_0 - z_c)\mathrm{d}s$$

where z_0 are points along $\mathbb{B}$ and $R_k(z) = -\frac{z^k}{k!}$. To compute this moment at the new center $z_{c'}$, we rewrite the R_k term as

$$\begin{aligned}M_k(z_{c'}) &= \int_{\mathbb{B}} E(z_0)R_k(z_0 - z_{c'})\mathrm{d}s \\ &= \int_{\mathbb{B}} E(z_0)R_k(z_0 - z_c + z_c - z_{c'})\mathrm{d}s \\ &= -\int_{\mathbb{B}} E(z_0)\frac{(z_0 - z_c + z_c - z_{c'})^k}{k!}\mathrm{d}s.\end{aligned}$$

Applying the binomial theorem to the numerator $((z_0 - z_c) + (z_c - z_{c'}))^k$ yields

$$\begin{aligned}M_k(z_{c'}) &= -\int_{\mathbb{B}} E(z_0)\left(\sum_{m=0}^{k} \frac{(z_0 - z_c)^m}{m!}\frac{(z_c - z_{c'})^{k-m}}{(k-m)!} \right)\mathrm{d}s \\ &= \sum_{m=0}^{k}\left(-\int_{\mathbb{B}} E(z_0)\frac{(z_0 - z_c)^m}{m!}\mathrm{d}s \right)\frac{(z_c - z_{c'})^{k-m}}{(k-m)!} \\ &= -\sum_{m=0}^{k} M_m(z_c)R_{k-m}(z_c - z_{c'}),\end{aligned}$$

which is the formula in (19). Since replacing $E(z_0)$ with $n(z_0)C(z_0)$ and R_k with R_{k-1} in the definition of M_k yields the other moment N_k, the same proof (replacing m with $n-1$) yields the translation formula

$$N_k(z_{c'}) = -\sum_{n=1}^{k} N_n(z_c)R_{k-n}(z_c - z_{c'})$$

for N_k. Using the fact that N_0 vanishes by definition, this translation formula is of the same form as (19) by letting the summation run from $n = 0 \ldots k$, so we can combine these formulas into a unified expression for A_k:

$$A_k(z_{c'}) = -\sum_{m=0}^{k} A_m(z_c)R_{k-m}(z_c - z_{c'}).$$

C.2 Moment-to-Local

The Taylor expansion of $u(z)$ at some point z_L is

$$f(z) = \sum_{t=0}^{\infty} u^{(t)}(z_L)\frac{(z_L - z)^t}{t!} = \sum_{t=0}^{\infty} u^{(t)}(z_L)R_t(z - z_L)$$

where $u^{(t)}$ denotes the t-th derivative of u. In fact, we *define* the local coefficient L_t to be $u^{(t)}$. From (17), the color value at a point can be expressed as

$$u(z) = \frac{1}{2\pi}\sum_{k=0}^{\infty} A_k(z_c)S_k(z - z_c),$$

so differentiating this expression t times yields

$$L_t(z_L) \equiv u^{(t)}(z_L) = \frac{(-1)^t}{2\pi} \sum_{k=0}^{\infty} S_{k+t}(z_L - z_c) A_k(z_c),$$

using the observation that $S_k' = -S_{k+1}$.

C.3 Local-to-Local

In the previous section, we defined the local coefficients at z_L to be coefficients in a Taylor expansion:

$$u(z) = \sum_{t=0}^{\infty} L_t(z_L) R_t(z - z_L).$$

As we did in the M2M formula, we wish to calculate the coefficients at another point $z_{L'}$ by rewriting the R_t term with the binomial theorem. We require the identity $\sum_{b=0}^{\infty} \sum_{a=0}^{b} F(a,b) = \sum_{a=0}^{\infty} \sum_{b=a}^{\infty} F(a,b)$, both sides of which can be interpreted as a summation of $F(a,b)$ for all $a \leq b$. Starting with the local expansion at z_L, we obtain

$$\begin{aligned}
u(z) &= \sum_{l=0}^{\infty} L_l(z_L) R_l(z - z_{L'} + z_{L'} - z_L) \\
&= \sum_{l=0}^{\infty} L_l(z_L) \sum_{t=0}^{l} - \left(\frac{(z - z_{L'})^t}{t!} \frac{(z_{L'} - z_L)^{l-t}}{(l-t)!} \right) \\
&= \sum_{t=0}^{\infty} \left(\sum_{l=t}^{\infty} L_l(z_L) \frac{(z_{L'} - z_L)^{l-t}}{(l-t)!} \right) \left(- \frac{(z - z_{L'})^t}{t!} \right) \\
&= \sum_{t=0}^{\infty} \left(\sum_{l=t}^{\infty} -L_l(z_L) R_{l-t}(z_{L'} - z_L) \right) R_t(z - z_{L'}).
\end{aligned}$$

Since the color value $u(z)$ is also equal to the local expansion at $z_{L'}$, the term inside the parentheses corresponds to the local coefficients expanded at $z_{L'}$. That is,

$$\begin{aligned}
L_t(z_{L'}) &= \sum_{l=t}^{\infty} -L_l(z_L) R_{l-t}(z_{L'} - z_L) \\
&= -\sum_{k=0}^{\infty} L_{k+t}(z_L) R_k(z_{L'} - z_L),
\end{aligned}$$

which completes the derivation of (22).

D Fast Gauss Transform

The *Hermite polynomial* $H_n(y)$ is defined as

$$H_n(y) = (-1)^n e^{y^2} \frac{d^n}{dy^n} e^{-y^2}. \tag{33}$$

The basis of the fast Gauss transform is that a Gaussian function is the generating function for the Hermite polynomials

$$e^{2yx - x^2} = \sum_{n=0}^{\infty} \frac{x^n}{n!} H_n(y). \tag{34}$$

We need a scaled version of this formula that corresponds to a Gaussian function with standard deviation h. First, we multiply both sides by e^{-y^2}, yielding

$$e^{-(y-x)^2} = \sum_{n=0}^{\infty} \frac{x^n}{n!} h_n(y), \tag{35}$$

where $h_n(y) = e^{-y^2} H_n(y)$ are the *Hermite functions*. Let x_0 be a point near x. Applying a similar trick as in Appendix C, we get the following expansion at x_0:

$$\begin{aligned}
e^{-(y-x)^2/h^2} &= e^{-[(y-x_0)-(x-x_0)]^2/h^2} \\
&= e^{-[(y-x_0)/h - (x-x_0)/h]^2} \\
&= \sum_{n=0}^{\infty} \frac{1}{n!} \left(\frac{x - x_0}{h} \right)^n h_n \left(\frac{y - x_0}{h} \right).
\end{aligned}$$

Conveniently, multi-index notation and the fact that the 2D Gaussian can be expressed as the product of two 1D Gaussians allow us to write the 2D expansion in a similar form:

$$e^{-\|\boldsymbol{x} - \boldsymbol{x}_i\|^2/h^2} = \sum_{|\alpha| < \infty} \frac{1}{\alpha! h_i^{|\alpha|}} h_\alpha \left(\frac{\boldsymbol{x}_i - \boldsymbol{x}_c}{h_i} \right) (\boldsymbol{x} - \boldsymbol{x}_c)^\alpha,$$

which is (26).

Consider a Gaussian function $\phi_i(\boldsymbol{x}, \boldsymbol{x}_i) = A_i e^{-\|\boldsymbol{x} - \boldsymbol{x}_i\|^2/h_i^2}$ and a flat lattice where each cell has sides of length $\sqrt{2} r h_i$, where $r \leq 1/\sqrt{2}$. If we only add the contribution of ϕ to a $(2n+1) \times (2n+1)$ neighborhood of cells centered at c, where c is the cell containing $\boldsymbol{x}_i$, the error due to ignoring all other cells is bounded by $A_i e^{-2r^2 n^2}$ [Greengard and Strain 1991]. Thus, if we use an error tolerance ϵ, we need

$$n \geq \sqrt{\frac{\ln 1/\epsilon}{2r^2}},$$

and substituting in our chosen side length $l = \sqrt{2} r h_i$ gives the bound in (29). Note that we chose $l = \min_i h_i$, so the requirement that $r \leq 1/\sqrt{2}$ is satisfied.

E Local Coefficients for Composition

Substituting (31) into (30) yields

$$\begin{aligned}
V_o(\boldsymbol{x}) = M(\boldsymbol{x}) &\left(\sum_{|\alpha| < N} c_{1,\alpha} (\boldsymbol{x} - \boldsymbol{x}_c)^\alpha + A_1(\boldsymbol{y}) \right) \\
&+ (1 - M(\boldsymbol{x})) \left(\sum_{|\alpha| < N} c_{2,\alpha} (\boldsymbol{x} - \boldsymbol{x}_c)^\alpha + A_2(\boldsymbol{y}) \right).
\end{aligned}$$

We merge the local expansion terms

$$\sum_{|\alpha| < N} \left(M(\boldsymbol{x}) c_{1,\alpha} + (1 - M(\boldsymbol{x})) c_{2,\alpha} \right) (\boldsymbol{x} - \boldsymbol{x}_c)^\alpha. \tag{36}$$

Because $M(\boldsymbol{x})$ is represented using diffusion points, (25) tells us it has an expansion of the form

$$M(\boldsymbol{x}) = \sum_{|\alpha| < N} b_\alpha (\boldsymbol{x} - \boldsymbol{x}_c)^\alpha. \tag{37}$$

Thus, we have the following expansion of $M(\boldsymbol{x}) c_{1,\alpha} + (1 - M(\boldsymbol{x})) c_{2,\alpha}$ in (36):

$$M(\boldsymbol{x}) c_{1,\alpha} + (1 - M(\boldsymbol{x})) c_{2,\alpha} = \sum_{|\gamma| < N} w_\gamma (\boldsymbol{x} - \boldsymbol{x}_c)^\gamma,$$

where γ is also a 2D multi-index. When $\gamma = (0,0)$,

$$w_\gamma = c_{1,\alpha} b_\gamma + c_{2,\alpha} - c_{2,\alpha} b_\gamma,$$

and when $\gamma \neq (0,0)$,

$$w_\gamma = c_{1,\alpha} b_\gamma - c_{2,\alpha} b_\gamma.$$

Substituting this in (36) yields the expansion

$$\sum_{|\alpha|<N} \sum_{|\gamma|<N} w_\gamma (\boldsymbol{x} - \boldsymbol{x}_c)^{\gamma+\alpha}.$$

Finally, we expand the double summation to obtain the $\rho_{\alpha'}$ coefficients in (32).

Acknowledgments

We thank the anonymous reviewers for their feedback. We also thank Orzan et al. for making their DCIs publicly accessible, Dingzeyu Li and Henrique Maia for help with revising an earlier draft, and Yun Fei and Angela Wei for assistance in video and figure editing. This research was supported in part by Columbia Junior Faculty Startup Fund as well as donations from Intel. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of funding agencies or others.

References

ABRAHAM, R., 2010. Strange attractors, http://www.chaoscope.org.

AGARWALA, A., DONTCHEVA, M., AGRAWALA, M., DRUCKER, S., COLBURN, A., CURLESS, B., SALESIN, D., AND COHEN, M. 2004. Interactive digital photomontage. *ACM Trans. Graph. (SIGGRAPH 2004) 23*, 3 (Aug.), 294–302.

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, 35–42.

BOWERS, J. C., LEAHEY, J., AND WANG, R. 2011. A ray tracing approach to diffusion curves. In *Computer Graphics Forum*, vol. 30, Wiley Online Library, 1345–1352.

BOYÉ, S., BARLA, P., AND GUENNEBAUD, G. 2012. A vectorial solver for free-form vector gradients. *ACM Trans. Graph. 31*, 6 (Nov.), 173:1–173:9.

BROCHU, T., KEELER, T., AND BRIDSON, R. 2012. Linear-time smoke animation with vortex sheet meshes. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, SCA '12, 87–95.

FARBMAN, Z., HOFFER, G., LIPMAN, Y., COHEN-OR, D., AND LISCHINSKI, D. 2009. Coordinates for instant image cloning. *ACM Trans. Graph. (SIGGRAPH 2009) 28*, 3 (July), 67:1–67:9.

FINCH, M., SNYDER, J., AND HOPPE, H. 2011. Freeform vector graphics with controlled thin-plate splines. *ACM Trans. Graph. (SIGGRAPH Asia 2011) 30*, 6 (Dec.), 166:1–166:10.

GREENGARD, L., AND ROKHLIN, V. 1987. A fast algorithm for particle simulations. *J. Comput. Phys. 73*, 2 (Dec.), 325–348.

GREENGARD, L., AND STRAIN, J. 1991. The fast gauss transform. *SIAM J. Sci. Stat. Comput. 12*, 1 (Jan.), 79–94.

GUMEROV, N. A., AND DURAISWAMI, R. 2006. Fast multipole method for the biharmonic equation in three dimensions. *Journal of Computational Physics 215*, 1, 363 – 383.

HANRAHAN, P., SALZMAN, D., AND AUPPERLE, L. 1991. A rapid hierarchical radiosity algorithm. *SIGGRAPH Comput. Graph. 25*, 4 (July), 197–206.

ILBERY, P., KENDALL, L., CONCOLATO, C., AND MCCOSKER, M. 2013. Biharmonic diffusion curve images from boundary elements. *ACM Trans. Graph. (SIGGRAPH Asia 2013) 32*, 6 (Nov.), 219:1–219:12.

JESCHKE, S., CLINE, D., AND WONKA, P. 2009. A gpu laplacian solver for diffusion curves and poisson image editing. *ACM Trans. Graph. (SIGGRAPH Asia 2009) 28*, 5 (Dec.), 116:1–116:8.

KEMP, M. 1998. Attractive attractors. *Nature 394*, 627 (Aug.).

LEFEBVRE, S., AND HOPPE, H. 2005. Parallel controllable texture synthesis. *ACM Trans. Graph. 24*, 3 (July), 777–786.

LI, Y., SHARAN, L., AND ADELSON, E. H. 2005. Compressing and companding high dynamic range images with subband architectures. *ACM Trans. Graph. (SIGGRAPH 2005) 24*, 3 (July), 836–844.

LIU, Y., AND NISHIMURA, N. 2006. The fast multipole boundary element method for potential problems: a tutorial. *Engineering Analysis with Boundary Elements 30*, 5, 371–381.

MUNKBERG, J., CLARBERG, P., HASSELGREN, J., AND AKENINE-MÖLLER, T. 2006. High dynamic range texture compression for graphics hardware. *ACM Trans. Graph. 25*, 3 (July), 698–706.

ORZAN, A., BOUSSEAU, A., WINNEMÖLLER, H., BARLA, P., THOLLOT, J., AND SALESIN, D. 2008. Diffusion curves: A vector representation for smooth-shaded images. *ACM Trans. on Graphics (SIGGRAPH 2008) 27*, 3 (Aug.), 92:1–92:8.

PANG, W.-M., QIN, J., COHEN, M., HENG, P.-A., AND CHOI, K.-S. 2012. Fast rendering of diffusion curves with triangles. *IEEE Computer Graphics and Applications 32*, 4, 68–78.

PÉREZ, P., GANGNET, M., AND BLAKE, A. 2003. Poisson image editing. *ACM Trans. Graph. (SIGGRAPH 2003) 22*, 3 (July), 313–318.

PORTER, T., AND DUFF, T. 1984. Compositing digital images. In *ACM Siggraph Computer Graphics*, vol. 18, ACM, 253–259.

SUN, X., XIE, G., DONG, Y., LIN, S., XU, W., WANG, W., TONG, X., AND GUO, B. 2012. Diffusion curve textures for resolution independent texture mapping. *ACM Trans. Graph. (SIGGRAPH 2012) 31*, 4 (July).

TAKAYAMA, K., SORKINE, O., NEALEN, A., AND IGARASHI, T. 2010. Volumetric modeling with diffusion surfaces. *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH) 29*, 6, 180:1–180:8.

ZHENG, C., AND JAMES, D. L. 2010. Rigid-body fracture sound with precomputed soundbanks. *ACM Transactions on Graphics 29*, 4 (July), 69:1–69:13.

ZHENG, C., AND JAMES, D. L. 2011. Toward high-quality modal contact sound. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2011) 30*, 4 (Aug.).