A Survey of Smooth Vector Graphics: Recent Advances in Representation, Creation, Rasterization and Image Vectorization

Xingze Tian and Tobias Günther

Abstract—The field of smooth vector graphics explores the representation, creation, rasterization, and automatic generation of light-weight image representations, frequently used for scalable image content. Over the past decades, several conceptual approaches on the representation of images with smooth gradients have emerged that each led to separate research threads, including the popular gradient meshes and diffusion curves. As the computational models matured, the mathematical descriptions diverged and papers started to focus more narrowly on subproblems, such as on the representation and creation of vector graphics, or the automatic vectorization from raster images. Most of the work concentrated on a specific mathematical model only. With this survey, we describe the established computational models in a consistent notation to spur further knowledge transfer, leveraging the recent advances in each field. We therefore categorize vector graphics papers from the last decades based on their underlying mathematical representations as well as on their contribution to the vector graphics content creation pipeline, comprising representation, creation, rasterization, and automatic image vectorization. This survey is meant as an entry point for both artists and researchers. We conclude this survey with an outlook on promising research directions and challenges to overcome in the future.

Index Terms—Vector graphics, diffusion curves, gradient meshes, survey

1 Introduction

Vector graphics is a branch of computer graphics describing the representation and synthesis of images based on geometric primitives. Compared to *raster graphics*, which require to keep per-pixel information, vector graphics provide a more compact representation. In particular, the specification of the image content in vector-based images is resolution-independent, meaning that they can be scaled and sampled to any desired output resolution. The simplicity of vector graphic representations can be a double-edged sword. While it is easy to create simple color gradients, complex color variations like smooth shadows, caustic effects, and defocus blur are hard to be represented directly through basic gradient functions [101]. When the amount of required detail increases, the editing process of vector graphics demands both solid artistic skills and a large amount of working time to obtain complex results. Therefore, vector graphics are more often used to represent stylized images, such as line drawings [33], [37], [100], fonts [92], [107], clipart [30], [38], [59], [114], [139], image triangulations [76], and sketches [12], [117]. With the goal of increasing the expressiveness of smooth vector graphics, current research on gradient-based vector graphics can be divided into two directions. One direction is to enhance the *artistic control* of vector graphics, such that more complicated effects can be constructed and manipulated with less effort. Thereby, the rendering process that converts vector representations into raster images is a crucial component, as high-quality results are expected to be generated at interactive or even real-time speed during editing. The other direction seeks to find the most suitable vector representation for a given image in a (semi-) automatic way with as few primitives as possible, which is called *vectorization*. The vector representations generated from the two directions are often on opposite ends of the expressiveness spectrum. Usually, complex color variations are hard to represent directly by a single primitive or through simple artistic interaction, hence, the vectorization approach often leads to a richer set of primitives, and the results are therefore more difficult to manipulate directly. The two directions are not completely orthogonal. The advances in the formulation of artist-friendly primitives may inspire vectorization methods, while the vectorized results can act as a basis for further editing and manipulation. Until now, both directions have been studied intensively within their respective vector representations.

Currently, there are two major extensions to simple shapes and gradients: *mesh-based* and *curve-based* representations, as explained by Barla and Bousseau [6] in their artist-oriented introduction to gradient art. Mesh-based representations divide the image domain into 2D patches, where color and other attributes are stored at the vertices or in the patch interiors. The research on mesh-based methods can be divided roughly into two directions, aligning with the two aforementioned directions in vector graphics. One path focuses on finding better topological representations of the color variations, such as grouping smaller patches [7], [8], [9], using subdivision-based techniques [84], [144], or adopting irregular shapes like Bézigons [139]. These methods mainly concentrate on automatic extraction of a vector-based description from a given image. The other path adds expressiveness by defining additional attributes on the patches, and hence provides more artistic control. For

• *X. Tian and T. Günther are with the Department of Computer Science, Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany. E-mail: xingze.tian@fau.de, tobias.guenther@fau.de*

Manuscript received April 19, 2005; revised August 26, 2015.

example, the gradient mesh formulation [74], [118] allows users to specify tangents at the patch vertices. In this way, color gradients can be controlled directly. Despite this, mesh-based representations can still be less intuitive as users may need to decompose the domain by themselves. In contrast, the creation process of curve-based methods is closer to free-hand drawing. Curve-based methods use various curves with different attributes like color and normal derivatives as the fundamental primitives. Users can draw the outline of their design and add colors afterwards. The rasterization process of curve-based representations like diffusion curves [101] often requires solving a partial differential equation (PDE) on the entire domain, which inhibits local control. In addition, the solving step may suffer from discretization problems, and an additional blurring step may be required to generate smoother results. Hence, most research in curve-based methods either focused on establishing sound formulations to enhance local control [41], [61], [64], [68], or put effort in developing better rendering techniques to avoid the intrinsic discretization issues [69], [120], [121]. As curve-based representations often describe color discontinuities in an image, it can still be difficult to represent either extremely smooth or very detailed regions.

With ongoing research, advances have been made on both fronts, adding new representations, creation tools, rasterization methods, and vectorization algorithms to both mesh-based and curve-based methods. Synergies are possible, since the edges of patches are often aligned with the curvilinear features of an image. However, in general, the research threads within both approaches evolved independently with their own mathematical formulations, making it challenging to transfer ideas from one branch to the other. Our goal is not only to present a comprehensive overview of the advances in smooth vector graphics, but also to convey a solid understanding of the underlying mathematical formulations by phrasing them consistently in the same notation. Further, we provide an in-depth discussion of existing algorithms, such that a generalized and unified representation may be developed in the future to unite the advantages of previous methods.

1.1 Methodology

Research over the past decades led to a wide range of vector graphics representations, which we categorize into two types:

1) **Mesh-based representations** decompose an image into 2D patches, where color is interpolated inside each patch. In this survey, we consider this formulation as generalization of simple shapes and gradients.
2) **Curve-based representations** mimic the traditional artistic habits of drawing on paper. The curves usually represent extrema in the color gradient, and the resulting image is often computed by solving a partial differential equation (PDE).

For each type of primitive, we classify the research efforts into the following parts of the content creation pipeline:

Representation. Research on this topic introduces novel vector graphics representations, which consist of two ingredients: (1) the definition what kind of information is stored on the geometric primitives, such as colors or other features, and (2) the underlying mathematical model to fill in void areas between primitives for which appropriate solvers can be chosen or developed.

Creation. Tools for the creation of vector art are often tailored to a particular vector graphics representation. Methods that ease the creation and editing process or that provide more flexibility and richness in the final results are classified into this category.

Rasterization. Given a vector graphics representation, the rasterization computes a rasterized image. Methods in this category seek to improve either the rendering quality or the rendering speed on a desired output device. Currently, solvers are tightly linked to a particular representation, i.e., there is no unified solver for all existing formulations.

Vectorization. For any input image or video, vectorization methods output a vector graphics representation, which resembles the input image or video as closely as possible. When additional information is provided, e.g., when the curve geometry is known [68], [71], the vectorization process may only solve for a part of the model.

As illustrated in Figure 1, the research topics are internally connected to each other. The creation and editing outputs a vector graphics representation, which is then rendered to a raster image or a video (i.e., an image sequence). On the other hand, vectorization acts as an inverse process that converts rasterized image(s) into a vector graphics representation.

1.2 Structure of the Survey

The survey is structured with respect to the categorization proposed in Section 1.1. For each vector graphics primitive, we describe state-of-art methods in representation, creation, rasterization, and vectorization. In Section 2, we introduce the mathematical notation used in this paper, and we discuss related mathematical foundations. We then concentrate on mesh-based representations in Section 3, including simple shapes and gradients. Subsequently, we thoroughly analyze curve-based vector graphics representations in Section 4. Lastly, we conclude the survey in Section 5, where we discuss the connections between the formulations and cover possible avenues for future work.

2 Mathematical Preliminaries

2.1 Notation

In order to unify the mathematical formulations in this paper, we adopt the same notations for all equations. Scalars and scalar-valued functions are expressed with italic letters, such as s or $s(t)$. Scalar-valued basis functions are denoted with capital italic letters, for example $B(t)$. Vectors and vector-valued functions are expressed with bold lowercase letters, like $\mathbf{v}$ or $\mathbf{v}(t)$. All vectors are laid out as column-vectors. Matrices are denoted by capitalized bold letters, such as $\mathbf{M}$. To make it clear whether a symbol denotes a constant or a function, we always list functions with their arguments, for example $\mathbf{x}(t)$. When needed, discrete values are indexed with lower scripts, such as $\mathbf{x}_i$. Note that user parameters are treated as constants in the expressions and are therefore not explicitly listed in the function arguments. If a derivative is a user-defined constant, we use $\partial_u f_{i,j}$ to abbreviate the derivative at a discrete location $\frac{\partial f(u,v)}{\partial u}_{|u=u_i,v=v_j}$.

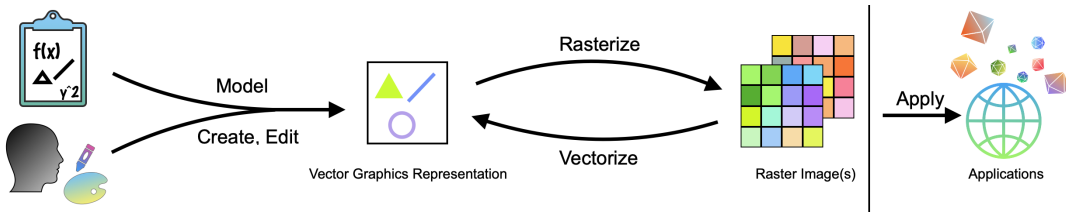


Fig. 1: The connections between the main research topics in vector graphics: a vector graphics representation is defined by its underlying mathematical model, which can be created or edited accordingly. The rasterization process generates raster image(s) from a vector graphics representation, while vectorization describes the inverse problem.

2.2 Bézier Curves

Bézier curves are widely used as the fundamental representation of curve-based methods, as well as patch boundaries in mesh-based methods. A Bézier curve $\mathbf{x}(t) : [0,1] \to \mathbb{R}^2$ is:

$$\mathbf{x}(t) = \sum_{i=0}^{n} B_i^n(t) \cdot \mathbf{b}_i \quad \text{with} \quad B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i} \tag{1}$$

where $B_i^n(t)$ are Bernstein basis functions and $\mathbf{b}_i$ are Bézier control points. The first and last control point are always interpolated, i.e., $\mathbf{x}(0) = \mathbf{b}_0$ and $\mathbf{x}(1) = \mathbf{b}_n$. A cubic Bézier curve has a polynomial degree of $n = 3$ and is specified by 4 control points $\mathbf{b}_0, \ldots, \mathbf{b}_3 \in \mathbb{R}^2$. From the tangent $\dot{\mathbf{x}}(t) = \frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t}$, the curve normal $\mathbf{n}(t) : [0,1] \to \mathbb{R}^2$ is defined as:

$$\mathbf{n}(t) = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \frac{\dot{\mathbf{x}}(t)}{\|\dot{\mathbf{x}}(t)\|} = \begin{pmatrix} n_x(t) \\ n_y(t) \end{pmatrix} \tag{2}$$

When interpolating multiple points, it is numerically more suitable to join multiple curves, as described next.

2.3 Spline Curves

Two (or more) curves $\mathbf{x}_1(t) : [t_0, t_1] \to \mathbb{R}^2$ and $\mathbf{x}_2(t) : [t_1, t_2] \to \mathbb{R}^2$ can be joined at the end points with suitable continuity constraints to form a Bézier spline curve $\mathbf{x}(t) : [t_0, t_2] \to \mathbb{R}^2$. For example, C^0-continuity requires matching end points $\mathbf{x}_1(t_1) = \mathbf{x}_2(t_1)$, whereas C^1-continuity requires C^0 and matching tangent vectors $\dot{\mathbf{x}}_1(t_1) = \dot{\mathbf{x}}_2(t_1)$. For a cubic curve, the hardest constraint is C^2-continuity, i.e., C^1 and matching acceleration vectors $\ddot{\mathbf{x}}_1(t_1) = \ddot{\mathbf{x}}_2(t_1)$. Finding the control points of a cubic Bézier spline that interpolates given points $\mathbf{x}(t_0) = \mathbf{p}_0$, $\mathbf{x}(t_1) = \mathbf{p}_1$ and $\mathbf{x}(t_2) = \mathbf{p}_2$ for a given parameterization (t_0, t_1, t_2) requires the solution of a linear problem with boundary conditions to find a unique solution. When selecting natural end conditions $\ddot{\mathbf{x}}_1(t_0) = \ddot{\mathbf{x}}_2(t_2) = \mathbf{0}$, the resulting curve minimizes the approximate bending energy $\int_{t_0}^{t_2} \|\ddot{\mathbf{x}}(t)\|^2 \, \mathrm{d}t$. In analogy to the extension of this minimizer to surfaces, this curve is sometimes also referred to as *thin plate spline*. The solution of a global linear problem can be avoided if a continuity constraint is given up. For example, Catmull-Rom and Hermite splines are only C^1 continuous cubic curves that can be computed locally. Switching from Bernstein basis functions $B_i^n(t)$ to B-spline basis functions leads to B-splines, which are equivalent to Bézier splines in terms of the function space. The control points of B-splines are called de Boor points. Rational B-splines are an extension that includes a weighting term for each de Boor point, making it possible to also represent conic sections. We refer to Farin [36] for a comprehensive introduction to curve design.

3 Mesh-based Methods

We begin our overview of vector graphics representations with mesh-based methods. An overview of existing methods is presented in Table 1, which follows the categorization from Section 1.1. We start with the different mesh representations.

3.1 Representation

Mesh-based methods divide the image domain into non-overlapping 2D patches across which colors are interpolated. The representation is mainly concerned with the placement and connectivity of patches and determines how color is interpolated. The patch shape can be triangular, rectangular, or even irregular, while color and other attributes are stored at vertices or in the patch interior. Fig. 2 provides an overview. Next, we categorize the methods based on the domain decomposition.

3.1.1 *Triangular Meshes*

Triangle-based methods interpolate a function $f(u,v,w)$ across each triangle with $u,v,w \in \mathbb{R}$ being barycentric coordinates with $u+v+w = 1$. Given discrete control values $f_{i,j,k}$ and associated basis functions $F_{i,j,k}^n(u,v,w)$ of degree n with $i,j,k \in \{0,\ldots,n\}$ and $i+j+k = n$, the continuous function $f(u,v,w)$ is:

$$f(u,v,w) = \sum_{i+j+k=n} F_{i,j,k}^n(u,v,w) \cdot f_{i,j,k} \tag{3}$$

Using Bernstein polynomials $B_{i,j,k}^n(u,v,w) = \frac{n!}{i!\,j!\,k!} u^i v^j w^k$ as basis functions $F_{i,j,k}^n(u,v,w)$ gives rise to Bézier triangles and the special case of $n = 1$ corresponds to linear barycentric interpolation.

The usage of triangulations or triangular patches to represent images has a long history [27], [29], [116], especially in image vectorization. An overview of basic triangle primitives is shown in Fig. 3. Basic triangulations with piecewise constant colors as in Fig. 3(a) require too many triangles to depict complex color variations [132]. Further, C^0 continuity across patch boundaries may become noticeable at higher resolution for smooth regions [84]. One way to better align the features is to use curved edges around the triangular patches. Xia et al. [132] proposed a representation using non-overlapping triangular patches with curved edges, see Fig. 3(b). Each patch is modeled as a 2D Bézier triangle as specified in Eq. (3), such that every boundary is a 2D cubic Bézier curve as in Eq. (1). The color of interior Bézier control points is fitted by thin plate splines. The boundary Bézier curves are aligned with color discontinuities for a given image, which enables larger color variation at the boundaries

TABLE 1: An overview of existing mesh-based methods.

Type	Method	Representation	Creation	Rendering	Vectorization
Triangular	[132]	Cubic Bézier triangle	-	Thin plate spline	Triangulation, lift to 3D, project to 2D
	[84]	Subdivision surfaces	Multi-level editing, local shape and color editing	Surface rendering	Feature detection, subdivision, mesh simplification, color optimization
	[144]		Multi-level editing, sharpness factor	Surface rendering	Feature detection, triangulation, color optimization,
	[135]	Quadratic Bézier triangle	-	Surface rendering	Feature detection and simplification, triangulation, mesh refinement
	[146]	TCB-spline surface	Multi-level editing, local selection and editing	TCB-spline interpolation	Feature detection, mesh parameterization, knot mesh generation, geometry and color optimization
Rectangular	[105]	Bicubic Bézier patch	Hole filling, Multi-level editing, object editing, user path for animation	Bicubic interpolation	Image segmentation, grid generation patch fitting
	[118]	Ferguson patch	-	Bicubic interpolation	Manual initialization, parameter optimization
	[74]		Quality control attribute	Bicubic interpolation	Image segmentation, triangulation, reparameterization, local refinement
	[136]		Example-based color transfer	Bicubic interpolation	-
	[137]		Optimization-based color transfer	Bicubic interpolation	-
	[127]		Scribble-based interface	Bicubic interpolation	-
	[5]		Local refinement, branching, sharp color	Hermite interpolation	-
	[55]		Noise textures	Bicubic interpolation	-
	[130]		Local control, image mosaic, image embedding	Bessel interpolation	Cross field generation quadrangulation
	[3]		Mesh color textures	Hermite interpolation	-
Irregular	[81]	Arbitrary polygon	-	Cubic mean coordinates	Iterative node removal
	[139]	Bézigon	-	Specific rasterization function	Segmentation, boundary fitting, optimization
	[86]	Subdivision surfaces	Multi-level editing	Surface rendering	-
	[122]		Local refinement, multi-level editing	Surface rendering	-
	[126]		Local refinement, multi-level editing, sharp color transition	Surface rendering	-
	[145]		Local refinement, multi-level editing	Surface rendering	-

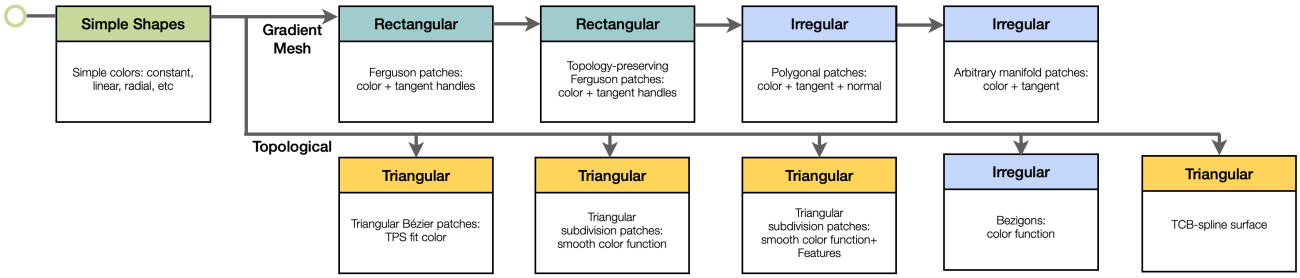


Fig. 2: Evolution of mesh-based representations, including gradient-meshes and topology-based representations.

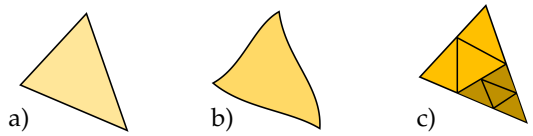


Fig. 3: Different triangular patches. From left to right: basic triangle with straight edges; Bézier triangle with curved edges; subdivision-based triangles.

and smooth colors inside the patches. More recently, Xiao et al. [135] used quadratic Bézier curves as the patch edges. The control points, which are placed on the edges, are computed through an optimization process. However, neither thin plate spline fitting nor energy minimization guarantees C^1 continuity at non-edge patch boundaries [84], [135]. Alternatively, subdivision-based methods can be used to tackle the continuity problem, as illustrated in Fig. 3(c). Liao et al. [84] improved the continuity property within and across the triangular patches, such that the boundary curves are C^2 continuous. Other than regions with discontinuity features (for example across edges that are intended to be sharp), the final color is no less than C^1 continuous [84]. The patches are constructed using a piecewise smooth subdivision scheme. The color attributes are stored at the vertices and are optimized such that the color function over the entire image is C^1 continuous. Zhou et al. [144] also used subdivision-based triangular meshes as their fundamental primitives. They used cubic B-splines to describe the curvilinear features in an image. The triangular meshes are constructed and refined similarly as in Liao et al. [84]. The color attributes are stored at the vertices. To model sharp edges, they used two nearby curves with separate colors. In addition, a sharpness factor is stored at the edges to control edge smoothness. Generally, the subdivision-based approaches may generate an excess amount of primitives, and can be strongly dependent on the feature extraction output [146]. Recently, Zhu et al. [146]

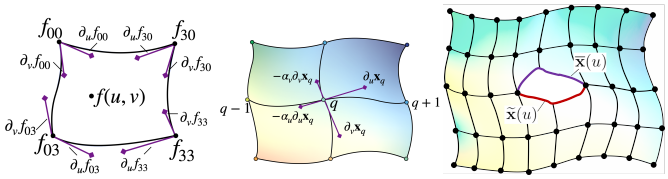
proposed to use quadratic triangle configuration B-splines (TCBs) [19], [112] to faithfully depict complex color variations while remaining C^1 continuous. An image can be treated as a surface in 5D (geometry and color) and can therefore be approximated and refined as a TCB-spline surface.

3.1.2 Rectangular Meshes

Rectangular meshes interpolate functions $f(u,v)$ over a u,v-parameter domain from given control points $f_{i,j}$ with $i \in \{0,\dots,n\}$ and $j \in \{0,\dots,m\}$ by computing tensor products with the basis functions $F_i^n(t)$ and $F_j^m(t)$:

$$f(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} F_i^n(u)\cdot F_j^m(v)\cdot f_{i,j} \tag{4}$$

Price and Barret [105] used bi-cubic tensor product surfaces, also known as Bézier patches, where the basis functions $F_i^n(t)$ are the Bernstein basis functions $B_i^n(t)$ from Eq. (1) and $n = m = 3$. Thus, every patch has four rows and columns of Bézier control points $\mathbf{x}_{i,j}$, each representing a cubic Bézier curve. Colors $\mathbf{c}_{i,j}$ are stored at the control points and are likewise interpolated using Eq. (4).



(a) Ferguson patch (b) Gradient mesh (c) Mesh with holes

Fig. 4: Most common gradient mesh types. Images (a)(b) inspired from [118], (c) inspired from [74].

Although the bi-cubic Bézier patch formulation is simple and straightforward, it still lacks control over the color variations from the editing perspective. One alternative is to use gradient meshes. This representation was first introduced by Adobe Illustrator for manually creating complex color variations, although the actual implementations were proprietary. In 2007, Sun et al. [118] published a formal definition of gradient meshes that are composed of Ferguson patches [40]. A Ferguson patch is a bi-cubic tensor-product surface following Eq. (4), which is specified through four corners $f_{i'j'}$ with $i',j' \in \{0,3\}$ and u,v-tangent handles at each of the four corners $\partial_u f_{i'j'}$ and $\partial_v f_{i'j'}$ see Fig. 4(a). By definition, the second-order mixed partials $\partial_u\partial_v f_{i',j'}$ are set to zero at the corners, which is referred to as zero twist, cf. later Section 3.3.2. A gradient mesh is composed by aligning Ferguson patches along mesh-lines, see Fig. 4(b). Each vertex q of the gradient mesh's mesh-lines stores:

Position: The 2D position $\mathbf{x}_q = (x_q, y_q)^{\mathrm{T}}$ of the vertex.

Tangents: Every corner has four u,v tangents, shared by the four neighboring patches, see Fig. 4(b). The four tangents are constructed by the two directions $\partial_u\mathbf{x}_q$ and $\partial_v\mathbf{x}_q$ while the opposite directions are scaled by factors α_u and α_v.

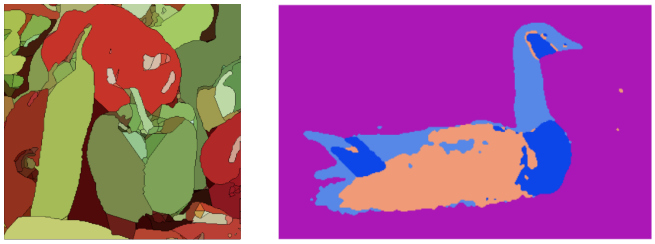
Color: Each vertex has an RGB value $\mathbf{c}_q = (r_q, g_q, b_q)^{\mathrm{T}}$.

Color derivatives: For every vertex q, its color derivatives $\partial_u\mathbf{c}_q$, $\partial_v\mathbf{c}_q$ along the u,v tangents are estimated per color channel with monotonic cubic spline interpolation [131]. For example, in Fig. 4(b), the color derivative in the u direction can be analytically calculated from the cubic splines between vertices $q-1, q$ and $q, q+1$. The color derivative for the v direction can be computed analogously.

One limitation in Sun et al. [118] is that regions with holes are hard to represent. Thus, Lai et al. [74] proposed a topology-preserving gradient mesh representation, which models a cut in a surface $\mathbf{x}(u,v)$ by duplicating consecutive horizontal grid lines, see Fig. 4(c). Let $\mathbf{x}(u) := \mathbf{x}(u, v_c)$ be the horizontal curve with $u \in [u_0, u_1]$ and $v_c = \text{const}$, along which the surface is cut. The two formerly adjacent patches receive distinct boundary curves $\overline{\mathbf{x}}(u)$ and $\widetilde{\mathbf{x}}(u)$, requiring G^0 continuity ($\overline{\mathbf{x}}(u_0) = \widetilde{\mathbf{x}}(u_0)$ and $\overline{\mathbf{x}}(u_1) = \widetilde{\mathbf{x}}(u_1)$), as well as G^1 continuity ($\partial_u\overline{\mathbf{x}}(u_0) = -\partial_u\widetilde{\mathbf{x}}(u_0)$ and $\partial_u\overline{\mathbf{x}}(u_1) = -\partial_u\widetilde{\mathbf{x}}(u_1)$) at the end points to form a closed loop. If there is more than one hole in the mesh, several cuts can be performed accordingly. By splitting a topology-preserving mesh [74] with n holes along the horizontal grid lines, this representation can be transformed into $n+1$ gradient meshes [118]. The two representations were adopted by many following works [127], [130], [136], [137]. Later, Barendrecht et al. [5] used cubic rectangular patches and extended the primitives to support local refinement.

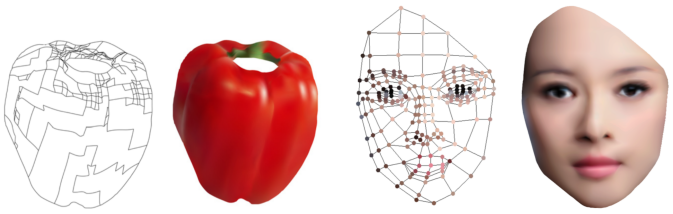
3.1.3 Irregular Meshes

Swaminarayan and Prasa [123] used polygonal patches with constant color, see Fig. 5(a), where the edges of the polygons align with curvilinear features of an input image. Apart from polygonal patches, constant colors and linear gradients have been used in other shapes, such as areas of homogeneous features [43], see Fig. 5(b). More recently, Yang et al. [139] used Bézigons (closed regions bounded by Bézier curves) to depict clipart images. The color in each region is represented by a spatially-varying color function. Later, Hettinga et al. [53] investigated several schemes to interpolate the values and gradients of color from boundary conditions, including the approach for topologically unrestricted gradient meshes by Lieng et al. [86], the cubic mean value coordinates by Li et al. [81], and two newly-proposed interpolation methods based on the Gregory generalized Bézier patch and the Charrot-Gregory corner interpolator. The extension to gradient meshes of arbitrary topology by Lieng et al. [86] is illustrated in Fig. 6, where each patch can have an arbitrary number of vertices and edges. Each control point has an assigned color value and color gradients, and the edges can be set to different discontinuity levels (C^1 smooth, C^0 crease, or C^{-1} discontinuous).



(a) Swaminarayn et al. [123] (b) Froumentin et al. [43]

Fig. 5: Patches of arbitrary shape, filled with constant color.



(a) Li et al. [81] (b) Lieng et al. [86]

Fig. 6: Gradient meshes of arbitrary connectivity.

3.2 Creation

While simple shapes and gradients are easy to control, vector-based representations are more difficult to create and edit when the amount of required detail increases. In the following, we cover manual and data-driven approaches, as well as alternative recoloring and texturing methods, leading to representations as illustrated in Fig. 7.

3.2.1 Manual Methods

Simple shapes are directly specified by shape parameters, e.g., position and radius of a circle. In Adobe Illustrator, gradient meshes are created manually by selecting an object and specifying the number of vertices per row and column. Users then need to choose a color for each grid point. Sun et al. [118] enabled users to create meshes in a similar way. After specifying the objects and dividing each object into 4 components, each sub-object is uniformly split parametrically. Users can click to add or remove mesh lines. The position of a vertex can be dragged or typed in, the four derivatives can be dragged through tangent handles, and the color can be picked from a color palette. Xia et al. [132] allowed users to select patches of interest and to edit colors using thin plate splines. Similarly, Liao et al. [84] let users select and edit either an individual vertex or a feature, or even a local region by defining a radius around a selected vertex or feature. After selecting the center of deformation, the geometry can be refined through distortion minimization [63]. Users can change the color of the chosen vertex or feature by assigning a new color or by blending the new color with the old color. Sverja et al. [122] edited irregular meshes by adding faces to the mesh interactively, and by splitting or collapsing edges of the mesh. Verstraaten and Kosinka [126] further let users add sharp color transitions in this formulation. Barendrecht et al. [5] added branching such that the mesh is not restricted to a rectangular grid. They also extended sharp color transitions such that a control point can be assigned with up to 4 different colors. Subdivision-based methods provide users with multi-resolution editing [147]. Liao et al. [84] and Zhou et al. [144] let users deform the mesh at a desired resolution level. Lieng et al. [86] enabled users to change colors at a higher resolution, while the geometry is manipulated at a coarser level.

3.2.2 Color Transfer, Recoloring, and Textures

Rather than relying on manual color editing, color transfer methods have been proposed to ease color assignment. Xiao et al. [136] proposed an example-based color transfer method for gradient meshes. Given a reference image, the algorithm transforms the colors of a gradient mesh using a linear operator constructed by PCA-based color transfer algorithms [1], [134]. The method assumes Gaussian color distributions in both the reference image and the gradient mesh, but may fail to preserve image details like textures after color transfer [136], [137]. Later, they proposed a refined method using an optimization-based linear operator [137]. The color transfer process is modeled by minimizing the difference in color distribution, which also enables adding image gradients as additional constraints such that fine details can be preserved after color transfer. Instead of using a reference image, Wan et al. [127] developed a scribble-based method to recolor a gradient mesh. Given an input mesh and scribbles by the user, an auxiliary mesh called control net is constructed. The control net structure can be computed for both classic gradient meshes [118] or topology-preserving gradient meshes [74], which are then recolored with extended chrominance blending [140] and are in the end converted back to a gradient mesh representation.

High-frequency image details often lead to a large number of patches when converted to a mesh-based representation, and can therefore be a burden for future editing. Methods have been presented to add texture details in different ways. Hettinga et al. [55] added Perlin, Worley and Gabor noise. Baksteen et al. [3] used mesh colors to add textures to gradient meshes, which is another option for 3D texture mapping in 2D [142]. In this representation, colors are defined at both the vertices, and at uniformly sampled points along the edges and inside the patches to represent complex color variations with a limited number of patches.

3.3 Rasterization

The rasterization process determines how colors are computed within each patch, which may be constant, a linear gradient, a combination of simple gradients, or the result of a fitting or interpolation from boundary colors. The rasterization process of mesh-based representations can be roughly divided into subdivision-based methods [84], [86], [144] that may use adaptive GPU-accelerated tessellation [52], and interpolation-based methods [81]. In this section, we discuss approaches for the various mesh formulations.

3.3.1 Triangular Meshes

Xia et al. [132] rendered triangular Bézier patches in three steps. First, they recursively subdivided each patch into 4 smaller Bézier patches until triangles were smaller than a pixel. Second, they determined the associated patch and its barycentric coordinates for each pixel. Third, the color of the pixel was computed by thin plate spline fitting using the barycentric coordinates. Liao et al. [84] accelerated the rendering by a GPU-based patch subdivision. Zhu et al. [146] used TCB-spline interpolation for image reconstruction.

3.3.2 Rectangular Meshes

Across rectangular patches, colors are interpolated using tensor products. Using Bernstein basis functions $B_i^n(t)$ from Eq. (1) with $n = m = 3$ in Eq. (4) gives rise to bi-cubic Bézier patches. Instead of specifying 16 control points $f_{0,0}$, $f_{1,0}$, ..., $f_{3,3}$ for the color function, the same tensor product surface $f(u, v)$ can be expressed equivalently using only the corner

(a) Verstraaten et al. [126] (b) Wei et al. [130] (c) Baksteen et al. [3] (d) Xiao et al. [135]

Fig. 7: Images created with different mesh-based frameworks.

control points and their first-order and second-order mixed partial derivatives, cf. Sun et al. [118]:

$$f(u,v) = \mathbf{t}(u)^{\mathrm{T}} \cdot \mathbf{C}^{\mathrm{T}} \cdot \mathbf{Q} \cdot \mathbf{C} \cdot \mathbf{t}(v) \tag{5}$$

with the monomial basis functions $\mathbf{t}(t) = (1, t, t^2, t^3)^{\mathrm{T}}$ and

$$\mathbf{C}=\begin{pmatrix} 1 & 0 & -3 & 2 \\ 0 & 0 & 3 & -2 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & -1 & 1 \end{pmatrix}, \mathbf{Q}=\begin{pmatrix} f_{0,0} & f_{0,3} & \partial_v f_{0,0} & \partial_v f_{0,3} \\ f_{3,0} & f_{3,3} & \partial_v f_{3,0} & \partial_v f_{3,3} \\ \partial_u f_{0,0} & \partial_u f_{0,3} & \partial_u\partial_v f_{0,0} & \partial_u\partial_v f_{0,3} \\ \partial_u f_{3,0} & \partial_u f_{3,3} & \partial_u\partial_v f_{3,0} & \partial_u\partial_v f_{3,3} \end{pmatrix} \tag{6}$$

Since the Ferguson patch formulation specifies only the corner control points and its first-order partial derivatives, the mixed second-order partial derivatives in Eq. (6) are usually set to 0, such that $\partial_u\partial_v f_{0,0} = \partial_u\partial_v f_{0,3} = \partial_u\partial_v f_{3,0} = \partial_u\partial_v f_{3,3} = 0$. By substituting positions $\mathbf{x}_q$ and tangents $\partial_u\mathbf{x}_q$, $\partial_v\mathbf{x}_q$ or colors $\mathbf{c}_q$ and tangents $\partial_u\mathbf{c}_q$, $\partial_v\mathbf{c}_q$, respectively, into f_q and its tangents $\partial_u f_q$, $\partial_v f_q$ in Eq. (6) allows for the interpolation of positions and colors. Equivalently, Barendrecht et al. [5] used cubic Hermite boundary curves.

3.3.3 Irregular Meshes

SVGenie [9] and SVGWave [7] both use polygonal mesh representations, for which a constant color or a simple color gradient can be assigned directly. Lecot and Lévy [77] approximate the color in each polygonal region with a linear combination of constant, linear and quadratic basis functions for each color channel. Later, more advanced techniques have been developed to interpolate polygons. Li et al. [81] used cubic mean values to interpolate the colors and gradients at the vertices, which was developed from the mean value property for biharmonic functions. Beatson et al. [10] then extended the domain to piecewise quadratic boundaries, where they proposed the Hermite mean value interpolation technique for polygons. Hettinga et al. [53] further adjusted the interpolation schemes from generalized Bézier patches [125] and Charrot-Gregory corner interpolation patches [22] (both are multi-sided parametric patches) such that the interpolation scheme can be used for 2D gradient meshes. By using the algorithm from Chiyokura and Kimura [25], they achieved at least G^1 continuity for the color. Lieng et al. [86] proposed an interpolation method based on Catmull-Clark subdivisions for rendering gradient meshes with arbitrary topology [21], [98].

3.4 Vectorization

In contrast to the manual creation of meshes, a vectorization process decomposes a given image into 2D patches. Common starting points are image segmentations and triangulations based on edge detections. When complex color models are used, such as gradient meshes with color tangents [118] or Bézigons with parametric edges [139], optimization techniques that minimize a reconstruction error are commonly used. Apart from the generic mesh types discussed in the following, several learning-based techniques have been proposed to vectorize line drawings [33], [49], sketches or manga [35], [117], and floorplans [91].

3.4.1 Triangular Meshes

Triangular patches are often extracted in two steps. The first step determines the mesh geometry, then the second step extracts or optimizes the color attributes at the vertices or within the patches. Xia et al. [132] performed a triangulation at pixel resolution as their first step. In order to keep image features, subpixels are inserted at edges. By treating each color channel as a height field, every pixel is lifted to a vertex in a 3D domain, and every subpixel is transformed into two vertices with the same 2D projection but with distinct color values to avoid blurry edges. Afterwards, the 3D meshes are simplified collectively regarding the quadric error metric [47] and are projected to the 2D image space. Non-overlapping triangular Bézier patches are fit to the triangulated domain through an optimization process. Colors are fit to the Bézier patches using thin plate spline fitting [103]. Liao et al. [84] performed the vectorization in a hierarchical manner. First, a Canny edge detector [18] is used to extract image features. In order to obtain region boundaries, GrabCut [109] is applied for image segmentation. Second, an initial mesh is constructed, where each pixel is initialized with a vertex, and additional vertices are inserted at the extracted features if needed. A retriangulation step is performed locally. Third, the dense mesh is simplified using quadric error metrics [47]. The color at the vertices is globally optimized. Zhou et al. [144] developed the vectorization step similarly for their subdivision-based patches. First, curvilinear features are extracted using contour detection [2] and corner detection [96]. Cubic B-splines are then fit to the detected edges. Second, a constrained Delaunay triangulation is carried out to obtain the initial mesh. Colors are assigned to the subdivision patches through an optimization process similar to Hoppe et al. [58]. The color at each vertex is then optimized through a least squares minimization. For places where the color fitting error exceeds a threshold, new vertices are added, and a constrained Delaunay triangulation is further applied for refinement. More recently, Hettinga et al. [51] fit cubic Bézier triangles to an input image, and

mapped textures as mesh colors through an optimization process. Later, they extended the vectorization framework to enhance user control and feature extraction [54].

3.4.2 Rectangular Meshes

Early methods like Price and Barret [105] vectorize an image in two steps. First, the mesh geometry is constructed. For this, a graph cut [16] is used to extract the object contour, which is guided by the user. Once an object is detected, curvature is used to locate four corner points on the object. Then, axes can be added to the object by conducting a least cost search to find a minimum cost path. Mesh lines are added similarly by subdividing the object and recursively performing the search. Each mesh cell is then fit by a Bézier patch. Second, color attributes are assigned to the Bézier control points by sampling colors from the input image. As gradient meshes have a richer set of attributes, the geometry and the attributes are often optimized together in a single minimization step. Sun et al. [118] required a manual initialization in which users first decompose the input image into multiple sub-objects, and then divided the boundary of each sub-object into 4 segments to which cubic Bézier splines were fitted. Users can interactively subdivide the patches further. The manual result is then used as initialization to a non-linear least-squares minimization to fit Ferguson patches to the input image, which includes the color values and color derivatives. The non-linear optimization is solved with the Levenberg-Marquardt algorithm [79], [99]. Users can guide the optimization further by drawing preferred directions of the mesh lines. Lai et al. [74] extended the representation to topology-preserving gradient meshes, and proposed a fully automatic vectorization method for this representation. They first segmented the image into objects, using a graph-based method for coarse segmentation [39], and refined the boundary with Grab-Cut [109]. The objects are converted into triangular meshes by performing a constrained Delaunay triangulation on the samples from a saliency map with error diffusion [67]. This is computed by applying a compass filter on the image. The triangular mesh is then mapped into a rectangular domain, and is reparameterized for local refinement. Color are sampled from the underlying image and for each control point, color is computed through interpolation. The derivatives of color are calculated using monotonic cubic interpolation [131]. Wei et al. [130] developed a faster method by adopting quadrangulation techniques for 3D meshes that are often used in geometry processing tasks.

3.4.3 Irregular Meshes

The Data Dependent Triangulation method (DDT [32]) in SVGenie [9] and the Wavelet Based Triangulation method (WBT [78]) in SVGWave [7] both decompose the image domain into triangles and group the triangles into polygonal patches based on similarity. SVGenie [9] seeks for a locally optimal triangulation iteratively. Based on Yu et al. [141], each pixel is first divided into two triangles by connecting the lowest cost diagonal. For each convex polygon composed of adjacent triangles, a look-ahead step is performed to swap polygon edges if necessary. The triangulation is further optimized by minimizing a cost function iteratively. SVGWave [7] performs a wavelet transformation hierarchically. In each level, the triangulation can be computed with the wavelet coefficient and is refined iteratively. Swaminarayan and Prasad [123] first used an edge detector to extract edge pixel chains and performed a constrained Delaunay triangulation such that the edges of the triangles align with the extracted edges. Each triangle is assigned a constant color by sampling several points within the triangle, which constitutes a so-called trixel. Neighboring trixels are grouped together to form polygons with respect to their proximity and continuity. Triangulation may also be used as an intermediate step to accelerate vectorization. Lecot and Lévy [77] developed a vectorization technique to compute closed regions bounded by cubic splines. The algorithm partitions the domain into a Voronoi diagram. In every region, the color is estimated with a quadratic polynomial. In a basic setup, the algorithm can be performed at pixel-level. Each pixel grows into a region in a greedy way and the polynomial coefficients are updated iteratively. In order to accelerate the vectorization process, instead of applying the algorithm on each pixel, an intermediate triangulated structure of trixels is constructed. Yang et al. [139] proposed a method to vectorize clipart images into Bézigons. A Bézigon is a closed Bézier curve with a color function defined in the interior. The initial Bézigons can be extracted automatically using image segmentation [39] and a curve fitting method [113], or they can be created by hand. In addition to the reconstruction error, the optimization seeks to minimize failure cases, such as self-intersection and angle-variation. He et al. [50] introduced a perception-aware vectorization for quantized raster images.

4 Curve-based Methods

In their seminal work, Elder et al. [34] used edges with attributes (intensity, blur scale, and gradient direction) to encode images, which has spurred further research as discussed throughout this section. A comprehensive overview based on the taxonomy in Section 1.1 is presented in Table 2.

4.1 Representation

First, we concentrate on the representations that have been developed to describe images by means of curves. An overview of the evolution is provided in Fig. 8.

4.1.1 Basic Formulation

Inspired from Elder et al. [34], Orzan et al. [101] proposed a new vector graphics primitive called diffusion curves, which uses Bézier curves as geometric primitives with colors defined on the left and right sides of the curves. The final image is constructed by diffusing the colors from both sides of the curves, which allows for the modeling of color discontinuities. The diffusion process may be followed by a blurring step to obtain smooth edges. Fig. 9 illustrates the three components of a diffusion curve, as defined below:

A cubic Bézier spline: The geometric curves are specified as splines, formed from cubic Bézier curves $\mathbf{x}(t) : \mathbb{R} \rightarrow \mathbb{R}^2$, see Eq. (1).

Color control points: On each side of the curve $\mathbf{x}(t)$, a color is specified by the user, formally denoted using functions $\mathbf{c}_l(t)$ and $\mathbf{c}_r(t)$. The colors are piecewise linearly interpolated from a set of color control points, placed along

TABLE 2: An overview of the existing curve-based methods.

Type	Method	Representation	Creation	Rendering	Vectorization
Harmonic	[69]	$\nabla^2\mathbf{u}(\mathbf{x})=\mathbf{0}$, $\mathbf{u}(\mathbf{x})\|_{\partial\Omega}=\mathbf{g}(\mathbf{x})$	-	Variable-stencil	-
	[70]		-	Variable-stencil	-
	[102]		-	Mesh rendering with MVC	-
	[119]		Multi-touch interface	Multi-grid	-
	[71]		Texture, stippling, hatching	Variable-stencil	Color estimation
	[121]		-	Green's functions	-
	[26]		-	Random walk	Superpixel boundaries
	[120]		Seamless cloning, diffusion points	Fast multipole	-
	[143]		-	FEM	Shape optimization
	[94]		Object editing	Variable-stencil	Depth enhancement
	[4]		Scribble-based interface	Variable-stencil	-
	[93]		Global deformation	Variable-stencil	-
	[82]		-	Variable-stencil	Edge extraction, propagation in time
	[68]	$\mathbf{u}(\mathbf{x})=w_l(\mathbf{x})\cdot\mathbf{u}_1(\mathbf{x})+(1-w_l(\mathbf{x}))\cdot\mathbf{u}_2(\mathbf{x})$, $\mathbf{u}_1(\mathbf{x})\|_{\partial\Omega}=\mathbf{g_1}(\mathbf{x})$, $\mathbf{u}_2(\mathbf{x})\|_{\partial\Omega}=\mathbf{g_2}(\mathbf{x})$	Click & drag	Variable-stencil	Color and blur estimation
Biharmonic	[41]	$\nabla^4\mathbf{u}(\mathbf{x})=\mathbf{0}$, $\mathbf{u}(\mathbf{x})\|_{\partial\Omega}=\mathbf{g}(\mathbf{x})$	Feature curves and points	Direct and iterative	-
	[15]		Feature curves and points	FEM	
	[64]		Sharp and smooth profiles	BEM	
	[138]		Hierarchical editing	Green's functions	Edge extraction in bilaplacian space
Poisson	[101]	$\nabla^2\mathbf{u}(\mathbf{x})=\mathbf{w}_x(\mathbf{x})+\mathbf{w}_y(\mathbf{x})$, $\mathbf{u}(\mathbf{x})\|_{\partial\Omega}=\mathbf{g}(\mathbf{x})$	Zooming, panning, blur switch	Multi-grid	Edge extraction
	[11]		Diffusion barrier, strength constraints direction constraints	Direct solver	-
	[56]		-	Multi-grid	-
	[89]		Mobile device interface	Multi-grid	-
	[60]	$\nabla^2\mathbf{u}(\mathbf{x})=\mathbf{f}(\mathbf{x})$, $\mathbf{u}(\mathbf{x})\|_{\partial\Omega}=\mathbf{g}(\mathbf{x})$, $\mathbf{f}(\mathbf{x})$ is piecewise constant	Laplacian constraints	Harmonic B-spline	-
	[61]		Seamless cloning	Harmonic B-spline	-
	[46]		-	Harmonic B-spline	-
	[45]		-	Harmonic B-spline	-
	[44]		Contrast & highlight	Harmonic B-spline	-
	[88]	$\frac{\partial\mathbf{u}(\mathbf{x},t)}{\partial t}=\nabla^2\mathbf{u}(\mathbf{x})\cdot c(\mathbf{x})+\nabla\mathbf{u}(\mathbf{x})\cdot\nabla c(\mathbf{x})+\mathbf{g}(\mathbf{x})$, $\mathbf{u}(\mathbf{x},0)=\mathbf{g}(\mathbf{x})$	Diffusion coefficients	Iteration	-
	[83]	$\frac{\partial\phi(\mathbf{x},t)}{\partial t}=c(\mathbf{x})\cdot\nabla^2\phi(\mathbf{x},t)$, $\phi(\mathbf{x},0)=\mathbf{g}(\mathbf{x})$	Different strokes, Diffusion constraints	Fourier transform	-
Ray tracing	[14]	$\mathbf{u}(\mathbf{x})=\int_0^{2\pi}L(\mathbf{x}_i(\mathbf{x},\theta))w(\mathbf{x}_i(\mathbf{x},\theta))d\theta$	Curve-domain shaders, transparency	2D ray tracing on regular grid	-
	[104]		Curve-domain shaders, image-domain shaders, multi-layering, instancing	2D ray tracing on triangular grid	-
	[85]	$\mathbf{u}(\mathbf{x})=\frac{1}{N(\mathbf{x})}\int_0^{2\pi}L(\mathbf{x}_i(\mathbf{x},\theta))w(\mathbf{x}_i(\mathbf{x},\theta))d\theta+\frac{1}{N(\mathbf{x})}\sum_{i=1}^{n}C(\mathbf{p}_i)w(\mathbf{x},\mathbf{p}_i)V(\mathbf{x},\mathbf{p}_i)$	(Local) diffusion points	2D ray tracing on triangular grid	-
Shading curve	[87]	Surface reconstruction	Shading profiles	Mesh generation, surface rendering	-

the curve. Each set contains at least two colors to define the color at the start ($t=0$) and end ($t=1$) of the curve.

Blur control points: As an additional degree of freedom, Orzan et al. [101] introduced a smoothness parameter $\sigma(t)$ along the curve, which controls the filter size of a Gaussian smoothing carried out in a post-process. As with the colors, the function $\sigma(t)$ is piecewise linearly interpolated from a set of blur control points. The final image $\mathbf{u}(\mathbf{x})$ is then defined as the smoothest image that meets both the color gradient $\mathbf{w}(\mathbf{x})$ along the diffusion curves as well as specific colors along the image boundary, formally solved for via:

$$\nabla^2\mathbf{u}(\mathbf{x})=\frac{\partial\mathbf{w}(\mathbf{x})}{\partial x}+\frac{\partial\mathbf{w}(\mathbf{x})}{\partial y},\quad \mathbf{x}\in\Omega\setminus\partial\Omega \tag{7}$$

$$\mathbf{u}(\mathbf{x})|_{\partial\Omega}=\mathbf{g}(\mathbf{x}) \tag{8}$$

Eq. (7) requires the x- and y-partial derivatives of the color field $\frac{\partial\mathbf{w}(\mathbf{x})}{\partial x}:\mathbb{R}^2\to\mathbb{R}^3$ and $\frac{\partial\mathbf{w}(\mathbf{x})}{\partial y}:\mathbb{R}^2\to\mathbb{R}^3$ that are zero everywhere, except along the curves $\mathbf{x}(t)$:

$$\frac{\partial\mathbf{w}(\mathbf{x})}{\partial x}_{|\mathbf{x}=\mathbf{x}(t)}=(\mathbf{c}_l(t)-\mathbf{c}_r(t))\cdot n_x(t) \tag{9}$$

$$\frac{\partial\mathbf{w}(\mathbf{x})}{\partial y}_{|\mathbf{x}=\mathbf{x}(t)}=(\mathbf{c}_l(t)-\mathbf{c}_r(t))\cdot n_y(t) \tag{10}$$

which requires the curve normal components $n_x(t)$ and $n_y(t)$, cf. Eq. (2). Note that in regions away from the diffusion curves, we have $\frac{\partial\mathbf{w}(\mathbf{x})}{\partial x}=\frac{\partial\mathbf{w}(\mathbf{x})}{\partial y}=\mathbf{0}$ to achieve smoothness.

Eqs. (7)–(8) were solved on a discrete image. To place the colors on the left and right of the diffusion curves, Orzan et al. [101] moved the color sources with an offset of a few pixels away from the curve geometry. It is worth noting that smooth color transitions across diffusion curves are difficult to represent directly. Hence a blurring post-process is essential for this model. Later, several variations have been proposed to enrich the modeling abilities of diffusion curves. Based on how the diffusion process is formulated, we categorize these models into the following types.

4.1.2 Harmonic Equation

Jeschke et al. [69] generalized the formulation of Orzan et al. [101] by treating diffusion curves as domain boundary curves with colors $\mathbf{c}_l(t)$ and $\mathbf{c}_r(t)$, such that there is no need to define the gradient fields $\frac{\partial\mathbf{w}(\mathbf{x})}{\partial x}$ and $\frac{\partial\mathbf{w}(\mathbf{x})}{\partial y}$ as in Eq. (7) anymore. The diffusion process can then be formulated as a Laplacian equation, which is also called a *harmonic equation*:

$$\nabla^2\mathbf{u}(\mathbf{x})=\mathbf{0},\quad \mathbf{x}\in\Omega\setminus\partial\Omega \tag{11}$$

$$\mathbf{u}(\mathbf{x})|_{\partial\Omega}=\mathbf{g}(\mathbf{x}) \tag{12}$$

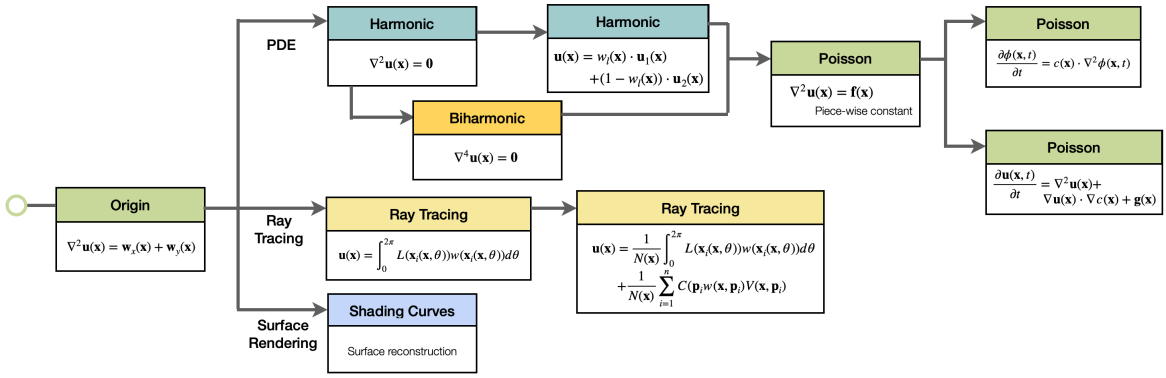
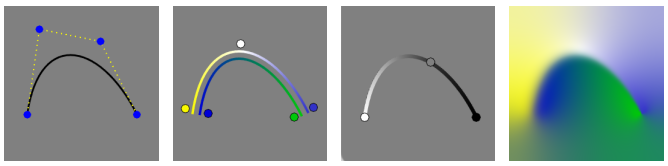


Fig. 8: Evolution of mathematical formulations in curve-based methods.



(a) A cubic Bézier spline (b) Color control points (c) Blur control points (d) Rendered result

Fig. 9: Components of a diffusion curve: (a) A cubic Bézier curve $\mathbf{x}(t)$ defined by 4 control points (marked by blue dots). (b) The left and right color sources $\mathbf{c}_l(t)$ and $\mathbf{c}_r(t)$ defined on both sides of the curve, with color control points indicated by the colored dots. (c) The blur source function $\sigma(t)$ is linearly interpolated from control points. (d) The final image is constructed by diffusion and subsequent blurring.

(a) harmonic formulation (b) biharmonic formulation

Fig. 10: The harmonic equation only enforces smoothness away from the curves' constraints, therefore artifacts may occur near the curves. Images adapted from [6].

In this formulation, the final image is smooth everywhere except at the boundaries, which now includes not only the rectangular image boundaries but also the diffusion curves. As in Eq. (8), Dirichlet boundary conditions are used in Eq. (12). This formulation has become the most popular one for rendering methods [69], [102], [120], vectorization techniques [94], [143] and editing tools [71], [93].

One problem of the harmonic formulation is the lack of control over the color gradients. The formulation currently seeks smoothness inside the domain but not across diffusion curves, as illustrated in Fig. 10. This may generate tent-like artifacts around the curves. Jeschke [68] addressed the problem by using a linear combination of harmonic functions:

$$\mathbf{u}(\mathbf{x}) = w_l(\mathbf{x}) \cdot \mathbf{u}_1(\mathbf{x}) + (1 - w_l(\mathbf{x})) \cdot \mathbf{u}_2(\mathbf{x}) \quad (13)$$

$$\mathbf{u}_1(\mathbf{x})|_{\partial\Omega} = \mathbf{g_1}(\mathbf{x}), \quad \mathbf{u}_2(\mathbf{x})|_{\partial\Omega} = \mathbf{g_2}(\mathbf{x}) \quad (14)$$

where $\mathbf{u}_1(\mathbf{x})$ and $\mathbf{u}_2(\mathbf{x})$ are the solutions to two individual Laplacian equations defined according to Eq. (11) on the same Bézier curve boundary with different color functions $\mathbf{g}_1$ and $\mathbf{g}_2$ for the Dirichlet boundary conditions. Thereby, $w_l(\mathbf{x})$ is an interpolation function that spatially blends between $\mathbf{u}_1(\mathbf{x})$ and $\mathbf{u}_2(\mathbf{x})$. The function $w_l(\mathbf{x})$ is computed by a Poisson equation and a parameter l controls the blending speed between $\mathbf{u}_1(\mathbf{x})$ and $\mathbf{u}_2(\mathbf{x})$. Likewise, the subsequent blurring is adapted and optimized. When $\mathbf{u}_1 = \mathbf{u}_2$, this reduces to Eq. (7). Both the harmonic model and the composition model require a subsequent blurring step, though in cases when the shape geometry and color control points are optimized, this process can sometimes be omitted [143].

4.1.3 Biharmonic Equation

The basic harmonic formulation lacks control over smoothness and direction of color gradients. In addition, with each added curve the derivative continuity is broken [41]. One way to address this problem is to specify constraints on higher-order derivatives. Finch et al. [41] proposed a biharmonic formulation based on thin plate splines, which seeks solutions that are harmonic in their Laplacian domain:

$$\nabla^4\mathbf{u}(\mathbf{x}) = \mathbf{0}, \quad \mathbf{x} \in \Omega \setminus \partial\Omega \quad (15)$$

$$\mathbf{u}(\mathbf{x})|_{\partial\Omega} = \mathbf{g}(\mathbf{x}) \quad (16)$$

Here, $\mathbf{g}(\mathbf{x})$ is a Cauchy boundary condition function, which can be considered as a combination of Dirichlet and Neumann boundary conditions. The boundary curves are constrained in both value and gradient of color. Later, Illbery et al. [64] generalized the formulation of biharmonic equations, such that harmonic diffusion curves can be considered as a special case of the biharmonic formulation.

Since directional color derivatives can be controlled directly for biharmonic diffusion curves, there is no need for a subsequent blurring process. However, the biharmonic formulation may bring other artifacts. One problem is that extrapolation can create new extrema at unconstrained positions, especially at colinear control points [15], [41]. This can be solved by manually adjusting the control point locations [41] or by applying a non-linear optimization [66]. Another problem is that when the curve geometry is scaled, the diffused color range will scale with the geometry, which may result in color oversaturation [68]. Later methods use other formulations such as a composition of Laplacian equations (Eq. 13) [68] or a combination of Laplacian and Poisson equations (Eq. 17) [61] to avoid this artifact.

4.1.4 Poisson Equation

Orzan's representation [101] and the harmonic equation in Eq. (11) can be further generalized to a Poisson equation:

$$\nabla^2 \mathbf{u}(\mathbf{x}) = \mathbf{f}(\mathbf{x}), \quad \mathbf{x} \in \Omega \setminus \partial\Omega \tag{17}$$

$$\mathbf{u}(\mathbf{x})|_{\partial\Omega} = \mathbf{g}(\mathbf{x}) \tag{18}$$

Therefore, when $\mathbf{f}(\mathbf{x}) = \mathbf{0}$, Eq. (17) is reduced to the harmonic model specified in Eq. (11). The initial representation in Eq. (7) can also be considered a special case of Eq. (17) when setting $\mathbf{f}(\mathbf{x}(t)) = (\mathbf{c}_l(t) - \mathbf{c}_r(t)) \cdot (n_x(t) + n_y(t))$ on the curves and $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ elsewhere.

By manipulating the expression of $\mathbf{f}(\mathbf{x})$ and $\mathbf{g}(\mathbf{x})$, more control over the diffusion process can be achieved. Bezerra et al. [11] solved Eq. (17) linearly, adding additional diffusion constraints to the system, such that the strength and direction of the diffusion process could be controlled. Later, Hou et al. [61] set $\mathbf{f}(\mathbf{x})$ as a piecewise constant function in order to control color gradients locally and globally. They used harmonic diffusion curves to depict color variants in an image, and added Poisson curves or Poisson regions controlled by $\mathbf{f}(\mathbf{x})$ for controlling shading details. Same as with the harmonic equations, $\mathbf{g}(\mathbf{x})$ is used to specify Dirichlet boundary conditions for the curve colors.

If $\mathbf{f}(\mathbf{x}) = \alpha \frac{\partial \mathbf{u}(\mathbf{x},t)}{\partial t}$, Eq. (17) is a standard heat equation, with time t and thermal diffusivity α, i.e., the rate of heat transfer. By introducing a time t, the result image is extended to $\mathbf{u}(\mathbf{x}, t) : \mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}^3$. Lin et al. [88] modified the heat equation to control diffusion strength and direction via a scalar-valued diffusion coefficient $c(\mathbf{x}) : \mathbb{R}^2 \to [0, 1]$:

$$\begin{aligned} \frac{\partial \mathbf{u}(\mathbf{x},t)}{\partial t} &= \nabla^2 \mathbf{u}(\mathbf{x}) \cdot c(\mathbf{x}) + \nabla \mathbf{u}(\mathbf{x}) \cdot \nabla c(\mathbf{x}) + \mathbf{g}(\mathbf{x}) \\ &= \left(\frac{\partial^2 \mathbf{u}}{\partial x^2} + \frac{\partial^2 \mathbf{u}}{\partial y^2} \right) \cdot c + \frac{\partial \mathbf{u}}{\partial x} \frac{\partial c}{\partial x} + \frac{\partial \mathbf{u}}{\partial y} \frac{\partial c}{\partial y} + \mathbf{g}(\mathbf{x}) \end{aligned} \tag{19}$$

$$\mathbf{u}(\mathbf{x}, 0) = \mathbf{g}(\mathbf{x}) \tag{20}$$

Eq. (20) is similar to previous methods in that $\mathbf{g}(\mathbf{x})$ is zero everywhere except along the boundary curves where it denotes the Dirichlet boundary condition. At time $t = 0$, the initial state of the image $\mathbf{u}(\mathbf{x}, 0)$ contains the undiffused colored curves. We can reorganize Eq. (19) to a Poisson equation when $c(\mathbf{x}) \neq 0$:

$$\nabla^2 \mathbf{u}(\mathbf{x}, t) = \frac{\frac{\partial \mathbf{u}(\mathbf{x},t)}{\partial t} - \nabla \mathbf{u}(\mathbf{x}) \cdot \nabla c(\mathbf{x}) - \mathbf{g}(\mathbf{x})}{c(\mathbf{x})} \tag{21}$$

In this case, the function $\mathbf{f}(\mathbf{x})$ in Eq. (17) is simply the right hand side of Eq. (21). The multi-layer method contains at least one color layer defined by the boundary curves $\mathbf{g}(\mathbf{x})$. The function $c(\mathbf{x})$ is used to model the strength and direction layers that control the diffusion process.

Instead of diffusing to an equilibrium, Li et al. [83] used the heat equation to only diffuse for a fixed time interval. Here, the scalar-valued opacity $\phi : \mathbb{R}^2 \times \mathbb{R} \to [0, 1]$ is diffused:

$$\frac{\partial \phi(\mathbf{x}, t)}{\partial t} = c(\mathbf{x}) \cdot \nabla^2 \phi(\mathbf{x}, t) \tag{22}$$

$$\phi(\mathbf{x}, 0) = \mathbf{g}(\mathbf{x}) \tag{23}$$

As before, $c(\mathbf{x})$ is the diffusion coefficient and controls the diffusion width. Eq. (22) can be reorganized similarly to Eq. (21) to form a Poisson equation $\mathbf{f}(\mathbf{x}, t) = \frac{\partial \phi(\mathbf{x},t)}{\partial t} \cdot \frac{1}{c(\mathbf{x})}$ when $c(\mathbf{x}) \neq 0$. The scalar-valued opacity ϕ can be further modified to model different stroke effects like watercoloring and oil painting. The opacity distribution is combined with the curve colors to determine the final image $\mathbf{u}(\mathbf{x})$.

By extending the formulation from Laplacian equations to Poisson equations, a significantly larger solution space is introduced, providing more control over the curves [60]. An additional post-processing step is not needed for the Poisson models as it can be added directly in the diffusion process by manipulating the diffusion coefficients [88].

4.1.5 Ray Tracing

The diffusion can be approximated by 2D ray tracing, similar to the final gathering concept in global illumination. The colored curves can be considered as virtual light sources [14]:

$$\mathbf{u}(\mathbf{x}) = \int_0^{2\pi} L(\mathbf{x}_i(\mathbf{x}, \theta)) \cdot w(\mathbf{x}_i(\mathbf{x}, \theta)) \, \mathrm{d}\theta \tag{24}$$

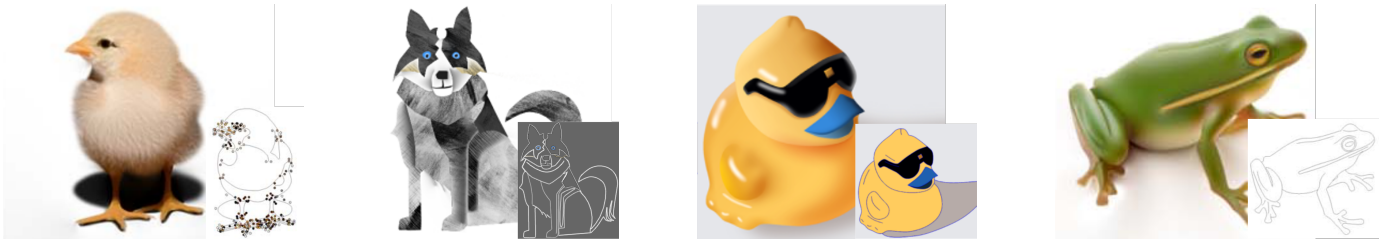
where the pixel color $\mathbf{u}(\mathbf{x})$ is determined by the weighted radiance integral over all directions. For a ray starting from a 2D point $\mathbf{x}$ in direction θ, the closest intersection point of the ray with a diffusion curve is $\mathbf{x}_i(\mathbf{x}, \theta)$, and $L(\mathbf{x}_i(\mathbf{x}, \theta))$ is its radiance. The normalized weight $w = w_c \cdot w_d$ is determined by a curve importance w_c that models diffusion barriers [11], [14], and a distance weighting $w_d(\mathbf{x}_i, \mathbf{x}) = \|\mathbf{x}_i - \mathbf{x}\|^{-p}$. When no blur is taking place and the rays are not occluded, the ray tracing formulation is equivalent to mean value interpolation, which is a solution to the Laplacian equation [14], [42], [72]. This formulation utilizes shaders for flexible color control and avoids the rasterization problems [14] of the original representation [101]. It was later adopted by Prévost et al. [104] and further extended by Lieng [87] to diffusion points.

4.1.6 Shading Curves

Shading curves [87] are an alternative for the modeling of shading and lighting effects. A shading curve is composed of a B-spline curve geometry and shading profiles at the left and right sides of the curve. During creation, users manually draw areas with the curves in constant tone, and each area is filled with constant color, resulting in a piecewise flat image. The shading effects are then added using shading profiles attached to the left and right sides of the curves, which are rendered as meshes rather than curves, making this a combination of curve-based and mesh-based methods.

4.2 Creation

Research on methods to enhance artistic control can be roughly divided into two directions. The first stream studies how users can better interact with existing features, such that they can create or edit primitives more easily. The second stream seeks to enrich the feature set of a certain model to create more expressive results. The fundamental functionality in diffusion curve frameworks is the manipulation of curve control points. Users can manipulate the curve geometry, as well as add, delete, duplicate, or change the values of attributes. In this section, we discuss methods and techniques that further improve artistic control on top of the basic editing functionality for different curve-based models, and we additionally group them into contributions to user



(a) Jeschke et al. [71] (b) Bowers et al. [14] (c) Lieng et al. [87] (d) Jeschke et al. [68]

Fig. 11: Images created with different curve-based frameworks, along with visualizations of input curves and control points.

interaction or feature enhancement wherever applicable. Figure 11 displays results of different frameworks.

4.2.1 Harmonic Equation

User Interaction. Sun et al. [119] proposed a multi-touch sketching interface for creating and editing diffusion curves. Jeschke et al. [68], [71] adopted a "click and drag" interaction. Sun et al. [120] enabled seamless cloning, where users can copy-paste control curves within a region. Lu et al. [94] used depth information to group curves semantically into objects, which are edited collectively. Later, they enabled users to add arbitrary handles to diffusion curves, which achieves global shape manipulation through linear blending deformation [93]. Bao and Fu [4] developed a scribble-based interface for editing diffusion curves, where users draw scribbles and colors are automatically assigned to curve points such that the rendered result aligns with the input.

Feature Enhancement. As diffusion always strives for smooth regions, Jeschke et al. [71] proposed to add texture parameters in a procedural way for creating finer details using Gabor noise. The noise parameters were automatically estimated to fit an input image with associated diffusion curves. This led to the application of stippling and hatching processes. Sun et al. [120] used Gaussian radial basis functions as diffusion points in their fast multipole representation of the harmonic equation. This enabled users to add non-harmonic color fields to the image.

4.2.2 Biharmonic Equation

User Interaction. The biharmonic formulation enables users to change not only the color values, but also the color derivatives in the curve normal direction. Finch et al. [41] allowed users to sketch curves freely and let them add color constraints at different points. Curves from vectorization are often too complicated to be edited directly. Xie et al. [138] improved this with their hierarchically extracted curves. Users can view the multi-scale curves in order to manipulate curves in different resolutions.

Feature Enhancement. Finch et al.'s [41] representation enables user to add point values, critical points and five curve types, including value curves, tear and crease curves, and contour and slope curves, which specify different boundary conditions. Boyé et al. [15] generalized the formulation from Finch et al. [41] and support diffusion, barrier, tear, crease, crease-value, value, value-slope, and slope curves, as well as nil curves. Ilbery et al. [64] used sharp-profiles and smooth-profiles to control colors at boundaries and in the interior.

4.2.3 Poisson Equation

User Interaction. Orzan et al. [101] let users create diffusion curves manually by sketching. Zooming and panning is interactive when only diffusing at low resolution in the visible viewport. Lin et al. [89] provided an implementation for mobile devices. In order to make the selection of curves easier, Lin et al. [88] labeled the diffusion curves during the rasterization process, such that when any pixel that belongs to a curve is selected, the entire curve can be immediately chosen. To achieve seamless cloning, Hou et al. [61] added Laplacian constraints to the places where the curves intersect with each other, such that users do not have to manually break diffusion curves into short disjoint segments.

Feature Enhancement. Bezerra et al. [11] added further constraints, including barrier curves that do not emit colors but block colors from other curves, they supported anisotropic diffusion, and smoothly blended between regions via soft constraints. Lin et al. [88] achieved this similarly by solving the diffusion process with diffusion coefficients. Further, they blended multiple layers and added diffusion point sources. Hou et al. [61] introduced Poisson curves and Poisson regions, which enable effects like core shadows, specular highlights, halos, and translucency. They separated hues and tones explicitly so that users could edit hue or tone locally, giving the possibility of editing specular highlights. Li et al. [83] focused on generating stylized images like watercolor or oil paint by introducing different strokes styles.

4.2.4 Ray Tracing

User Interaction. Lieng et al. [85] control how colors are propagated locally around a diffusion point by drawing line constraints for the triangulation near the point.

Feature Enhancement. Using shaders, Bowers et al. [14] added gradient fill and texture to diffusion curve images, and provided parameters for the influence w_c and the distance-weighting w_d in Eq. (24). Prévost et al. [104] have both curve-domain shaders (attached along the curves, similar to Bowers et al. [14]) and image-domain shaders (defined in the whole image domain) to avoid undersampling due to sparse interpolation. A multi-layer scheme is implemented in the framework, such that users can add local curves (for example to create specular highlights) and clone objects in the scene. Similar to Prévost et al. [104], Lieng et al. [85] added local diffusion points in a particular image layer.

4.3 Rasterization

Curve-based vector graphics are frequently formulated as solutions to partial differential equations. Hence, two components are essential in the rasterization process. The *discretization* of the curves and the domain determines the rendering quality. Especially for diffusion curves, a small discretization error may result in large shifts after the diffusion process [69]. The *solving scheme* determines the speed of the rendering result. In this section, we discuss the contributions made to both components.

4.3.1 Harmonic Equation: Discretization

Based on the type of grid that the solvers are applied on, we divide the discretization step into the following types.

Regular grid. Jeschke et al. [69] proposed a robust rasterization scheme. The curves are first discretized into smaller line segments, and a Voronoi diagram is constructed [57]. The color of each pixel is determined by its closest point on the curves, which sets the initial condition. The initial guess and a distance map are used for the diffusion. By diffusing from an entire image (the initial condition) instead of from discrete curves, the discretization error is reduced. Although at pixels near curves, strobing artifacts may still occur [69].

Triangular grid. Pang et al. [102] rendered diffusion curve images on a triangulated domain. They first discretized diffusion curves and the image boundaries into line segments, then they triangulated the domain using a constrained Delaunay triangulation. Instead of solving at each pixel, they solved for every vertex. The triangulated space was also used as intermediate representation in FEM solvers [15]. Boyé et al. [15] applied a constrained Delaunay triangulation, and used quadratic Lagrange polynomials as basis functions. Zhao et al. [143] used a similar triangulation method from the `Triangle` package [115] with second-order basis functions.

Irregular grid. Dai et al. [26] used image segmentation to avoid the discretization of individual curves. Their curves are extracted from a set of superpixels of an input image. The left and right pixels adjacent to a superpixel boundary are considered as the left and right color sources of a curve. Therefore, overlapping of color sources would not occur during the diffusion process.

4.3.2 Harmonic Equation: Solving Scheme

Closed-form. Sun et al. [121] derived a closed-form solution for *closed* diffusion curves using Green's functions:

$$\mathbf{u}(\mathbf{x}) = \oint_{\partial\Omega} \Big(\mathbf{u}(\mathbf{x}')G_n(\mathbf{x},\mathbf{x}') - \frac{\partial \mathbf{u}(\mathbf{x}')}{\partial \mathbf{n}(\mathbf{x}')} G(\mathbf{x},\mathbf{x}')\Big) d\mathbf{x}' \quad (25)$$

with $\mathbf{x}' \in \partial\Omega$ and where

$$G(\mathbf{x},\mathbf{x}') = \frac{1}{2\pi}\ln(\|\mathbf{x}-\mathbf{x}'\|),\ G_n(\mathbf{x},\mathbf{x}') = \frac{\partial G(\mathbf{x},\mathbf{x}')}{\partial \mathbf{n}(\mathbf{x}')} \quad (26)$$

and where the curve normal $\mathbf{n}$ points outwards the domain, $G(\mathbf{x},\mathbf{x}')$ is Green's function, and $G_n(\mathbf{x},\mathbf{x}')$ is its normal derivative. In this formulation, every point in the image can be evaluated directly as boundary integral in Eq. (26). In practice, Eq. (25) is evaluated with boundary element methods (BEM) by sampling points on each diffusion curve. Adaptive sampling methods select a random position on the curve, and iteratively add more samples such that the approximation error is below a threshold. Sun et al. [120] reformulated Eq. (25) to a fast multipole representation. Using a hierarchical lattice on the image domain, they calculated boundary integrals hierarchically only on curves in adjacent lattice cells to speed up the computation.

Iterative scheme. The harmonic equation can be solved iteratively using Jacobi relaxations. In each iteration k, the discrete pixel colors $\mathbf{u}_{i,j}$ of pixel (i,j) are updated:

$$\mathbf{u}_{i,j}^{k+1} = \begin{cases} \mathbf{g}_{i,j} & (i,j) \text{ on curve} \\ \frac{\mathbf{u}_{i-1,j}^k + \mathbf{u}_{i+1,j}^k + \mathbf{u}_{i,j-1}^k + \mathbf{u}_{i,j+1}^k}{4} & \text{otherwise} \end{cases} \quad (27)$$

Solving this for every pixel is slow when the required resolution is large. Hence, Orzan et al. [101] used a multi-grid solver, see later in Section 4.3.5. Jeschke et al. [69] designed a GPU-based variable stencil size solver such that a bigger step size can be used. After rasterization, a closest point map and an initial guess of the final image are generated and fed into the solver. In this setup, the Laplacian is not computed as an average of its 4 neighboring pixels, but can be treated as an average of its surrounding circle according to the mean value theorem [42], [69]. The circle radius can be increased as long as boundary curves are not crossed, which is determined by the closest point map. Alternatively, Dai et al. [26] formulated the diffusion process as a random walk process.

Direct solvers. Pang et al. [102] solved the diffusion process by directly evaluating the colors on a triangulated domain. For curve nodes, colors are defined on the curve. For other triangle vertices, colors are interpolated using mean value coordinates (MVC) [42] from the colored curve nodes. MVC algorithms assume that nodes are surrounded by a closed polygon. Therefore, an additional visibility test is performed to select the nearest curve nodes around each non-curve node before evaluating the MVC. Colors inside each triangle can then be computed simply using barycentric interpolation. Alternatively, Boyé et al. [15] and Zhao et al. [143] used the finite-element method to solve the harmonic equation linearly in the weak formulation.

4.3.3 Biharmonic Equation: Discretization

In addition to curve geometry and colors, other attributes used in biharmonic formulations also need to be discretized. The primitives can be rasterized onto a uniform grid or they can be discretized onto different elements, using for example finite elements (FEM) or boundary elements (BEM).

Regular grid. The formulation of Finch et al. [41] contains a set of critical points and feature curves. They rasterize each critical point with respect to the bilinear weights of its four neighbor pixels. The curves are rasterized by computing the intersections of the curves and the edges of the pixels. Since the feature curves are used to control local gradients, these constraints are added at the intersections.

Triangular grid. Boyé et al. [15] discretized the domain into non-overlapping quadratic triangular patches for both the harmonic and biharmonic formulation. However, two additional conditions need to be fulfilled for biharmonic equations. First, the weak formulation of biharmonic equations requires C^1 continuity across conforming elements.

Therefore, unlike the harmonic formulation, quadratic Lagrange patches with C^0 continuity cannot be directly used as FEM elements. Boyé et al. [15] use a third-order non-conforming element (Fraeijs de Veubeke (FV)) [28] such that C^1 continuity is not needed for convergence. Second, biharmonic equations enable gradient constraints, which can be attached or interpolated at the vertices and edges when using FV as FEM elements.

Irregular grid. Ilbery et al. [64] discretized curves into line segments through recursive subdivision with a threshold on curve straightness. The constraints on color values and color derivatives in normal direction are computed by interpolation for each line segment and are fed into a BEM solver for line by line evaluation.

4.3.4 Biharmonic Equation: Solving Scheme

Closed-form. Weber et al. [129] and Ilbery et al. [64] derived a closed-form solution using Green's functions:

$$\mathbf{u}(\mathbf{x}) = \underbrace{\oint_{\partial\Omega} \mathbf{u}(\mathbf{x}')G_n^H(\mathbf{x},\mathbf{x}')d\mathbf{x}'}_{①} - \underbrace{\oint_{\partial\Omega} \frac{\partial \mathbf{u}(\mathbf{x}')}{\partial \mathbf{n}(\mathbf{x}')} G^H(\mathbf{x},\mathbf{x}')d\mathbf{x}'}_{②}$$
$$+ \underbrace{\oint_{\partial\Omega} \nabla^2\mathbf{u}(\mathbf{x}')G_n^B(\mathbf{x},\mathbf{x}')d\mathbf{x}'}_{③} - \underbrace{\oint_{\partial\Omega} \frac{\partial \nabla^2\mathbf{u}(\mathbf{x}')}{\partial \mathbf{n}(\mathbf{x}')} G^B(\mathbf{x},\mathbf{x}')d\mathbf{x}'}_{④} \tag{28}$$

where $\mathbf{x}' \in \partial\Omega$ is a boundary point. The Green's function G^H and its normal derivative G_n^H are the fundamental solutions to a harmonic equation and are the same as in Eq. (26). The Green's function G^B and G_n^B for biharmonic equations are:

$$\begin{aligned} G^B(\mathbf{x},\mathbf{x}') &= \frac{1}{8\pi}(\|\mathbf{x}-\mathbf{x}'\|)^2\Big(\ln(\|\mathbf{x}-\mathbf{x}'\|)+1\Big) \\ G_n^B(\mathbf{x},\mathbf{x}') &= \frac{\partial G^B(\mathbf{x},\mathbf{x}')}{\partial \mathbf{n}(\mathbf{x}')} \end{aligned} \tag{29}$$

In order to improve computation efficiency, a curve-aware upsampling scheme may be used. As the line segment field becomes smoother when the distance between a point and a line segment increases, its value can be approximated through interpolation. In a multi-level formulation, most computation is done on a coarse grid.

Iterative scheme. Finch et al. [41] solved on a smaller grid and upsampled the result using a hierarchical strategy [13]. When solving on a coarse domain, they adopt a direct solver (see below). This solution is then upsampled to a finer level, and Gauss-Seidel iterations are performed to refine the solution at that level.

Direct solvers. Finch et al. [41] used a direct solver for small images. They structured their biharmonic formulation as a linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$. The final image is solved by calculating the banded-diagonal Cholesky decomposition (CD) of the matrix $\mathbf{A}$. In the cases when not sufficiently many value constraints are added, such that $\mathbf{A}$ is positive definite rather than positive semi-definite, a small regularization weight is added to obtain a unique solution. Boyé et al. [15] used the weak formulation for biharmonic equations from Lascaux and Lessant [75]. The final image can be approximated as a linear system, which is solved directly.

4.3.5 Poisson Equation: Discretization

Regular grid. The original diffusion curve formulation [101] requires to rasterize the color sources, the color derivatives across the curves, and an additional blur source, see Fig. 12. The color curves are rasterized with an offset in the normal direction to avoid overlapping in the rasterized results. However, overlaps can still occur, especially at high curvatures, thin structures, and intersections [101].

Triangular grid. Zhao et al. [143] used a finite element method to solve the harmonic equation for their diffusion curves. In addition, their method requires solving a Poisson equation during the vectorization process for the cost function. They discretized the domain similarly as described for the harmonic formulation, and the velocity attribute (for vectorization) is discretized along polylines.

Irregular grid. Hou et al. [60] first decomposed the image domain into disjoint sub-regions, such that each sub-domain is bounded by a closed diffusion curve. For each sub-domain, they discretized the primitives and constraints on a quad-tree, which was used for their closed-form solver.

4.3.6 Poisson Equation: Solving Scheme

The solving process of the general Poisson formulation in Eq. (17) varies for different definitions of $\mathbf{f}(\mathbf{x})$ and $\mathbf{g}(\mathbf{x})$.

Closed-form. Like for harmonic and biharmonic functions, Green's identities can be used to derive a closed-form solution to Poisson equations. Hou et al. [60] used Green's third identities to structure the solution to their formulation:

$$\mathbf{u}(\mathbf{x}) = \underbrace{\iint_{\Omega} G(\mathbf{x},\mathbf{x}')\nabla^2\mathbf{u}(\mathbf{x}')d\mathbf{x}'}_{①} + \underbrace{\oint_{\partial\Omega} \mathbf{u}(\mathbf{x}')G_n(\mathbf{x},\mathbf{x}') - \frac{\partial \mathbf{u}(\mathbf{x}')}{\partial \mathbf{n}(\mathbf{x}')} G(\mathbf{x},\mathbf{x}')d\mathbf{x}'}_{②} \tag{30}$$

When $\nabla^2\mathbf{u}(\mathbf{x}') = 0$, integral ① vanishes, and the equation is reduced to Eq. (25): the closed-form solution to a harmonic equation. The definition of G and G_n are the same as in Eq. (26). Intuitively, Eq. (30) can be directly evaluated as was done for the harmonic formulation. However, integral ① is defined on the entire domain, which is expensive to compute. Hou et al. [60] discretized the domain onto a quad-tree and derived an approximation for each cell. Li et al. [83] used a Fourier transform to solve their heat equation formulation in Eqs. (22)–(23). Therefore, for any 2D point $\mathbf{x} = (x, y)$, the scalar-valued opacity ϕ at time t can be evaluated as:

$$\phi(\mathbf{x},t) = \frac{1}{4\pi\ t\cdot c(\mathbf{x})}\int_{-\infty}^{+\infty} \phi(\mathbf{x}',0)\cdot e^{-\frac{(x-x')^2}{4t\cdot c(\mathbf{x})}}\cdot e^{-\frac{(y-y')^2}{4t\cdot c(\mathbf{x})}}\ \mathrm{d}\mathbf{x}' \tag{31}$$

The closed-form solution provides a direct result for any time t in the diffusion process. The calculated result is then rasterized as a whole with respect to the desired resolution.

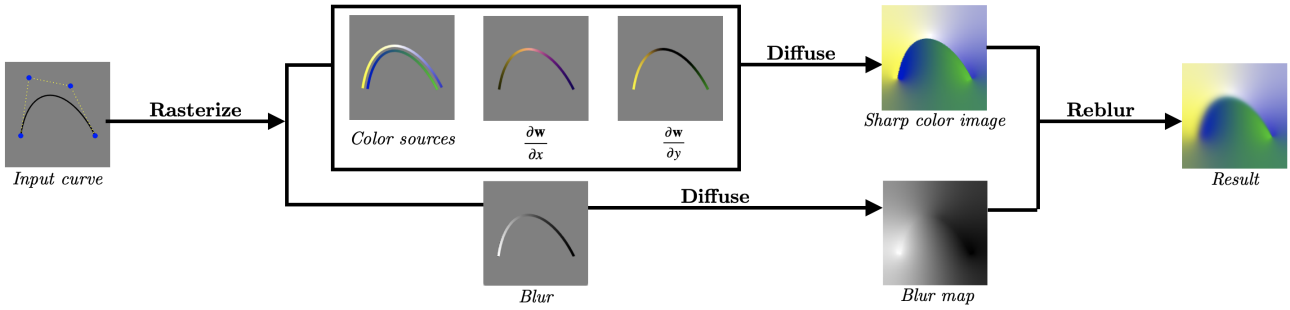


Fig. 12: Rendering pipeline of a diffusion curve image by Orzan et al. [101].

Iterative Scheme. Similar to Eq. (27), the original formulation from Orzan et al. [101] can be solved with Jacobi relaxations where for iteration k, the image $\mathbf{u}^{k+1}$ is:

$$\mathbf{u}_{i,j}^{k+1} = \begin{cases} \mathbf{g}_{i,j} & \text{on curve} \\ \frac{\mathbf{u}_{i-1,j}^{k}+\mathbf{u}_{i+1,j}^{k}+\mathbf{u}_{i,j-1}^{k}+\mathbf{u}_{i,j+1}^{k}-\text{div}(\nabla \mathbf{w})_{i,j}}{4} & \text{otherwise} \end{cases}$$

The divergence of the gradient field $\text{div}(\nabla \mathbf{w})$ is

$$\text{div}(\nabla \mathbf{w})_{i,j} = \frac{\partial_x \mathbf{w}_{i+1,j} - \partial_x \mathbf{w}_{i-1,j}}{2} + \frac{\partial_y \mathbf{w}_{i,j+1} - \partial_y \mathbf{w}_{i,j-1}}{2} \tag{32}$$

To achieve interactivity, they used a multi-grid approach [17], [48] and recursively decreased the domain resolution to solve on coarser domains, and upsampled until the original resolution is reached. Similarly, Lin et al. [88] solved their Poisson formulation in Eq. (21) for each color layer iteratively.

Direct Solvers. Bezerra et al. [11] reformulated the Poisson equation in Eq. (7) as a constrained optimization to add more control over the diffusion. Due to the existence of non-local constraints (e.g., two closed areas can be linked to enable diffusion between them), the multi-grid solver generates unsatisfying results and can be inefficient. They used a global linear solver instead. Alternatively, finite element methods can be used to solve harmonic equations by solving a linear system iteratively or directly, cf. Section 4.3.1. Zhao et al. [143] used a second-order basis to solve the weak form of a Poisson equation linearly.

4.3.7 Ray Tracing: Discretization

Currently, two domains have been studied for ray tracing: a regular grid or a triangulated domain. For regular grids, the ray starts from each point in the domain, while the ray originates from each triangle vertex in a triangulated domain.

Regular grid. Bowers et al. [14] subdivided the curves into a set of line segments for their ray tracing algorithm. They built a uniform acceleration grid with a grid size $2\sqrt{n} \times 2\sqrt{n}$ for n line segments, such that an increase in the number of curves would not cause a drastic decrease in performance.

Triangular grid. Prévost et al. [104] solved the ray tracing process on a triangulated domain. Similarly, they discretized curves into line segments, and triangulated the image domain with a constrained Delaunay triangulation. Lieng [85] used the same concept for the diffusion points extension. For every triangle, evaluation points are sampled to compute the final result. For pixels close to a diffusion point, evaluation points receive the diffusion point color.

4.3.8 Ray Tracing: Solving Scheme

The ray tracing formulation is solved stochastically. Bowers et al. [14] performed parallel ray tracing on the GPU for each pixel. The angles of the rays are determined through stratified sampling. Simple adaptive sampling can also be applied for acceleration. This can be done by first dividing the image into a set of blocks and performing ray tracing with fewer rays. During this process, the shortest intersection distance is stored for each block. If this distance is above a threshold, ray tracing is further applied to avoid undersampling. Prévost et al. [104] performed ray tracing on each triangle vertex. Unlike Pang et al. [102] who used barycentric interpolation, cubic interpolation is performed to avoid mach banding effects. Lieng et al. [85] performed ray tracing similar to Prévost et al. [104]. To add local diffusion points, they render a Catmull-Clark subdivision surface [98]. The final result is composed by blending the local and global results linearly. Recently, novel mesh-less stochastic PDE solvers [110] have been introduced for vector graphics [106] to avoid discretization.

4.3.9 Anti-aliasing

Anti-aliasing techniques have been added in some of the methods to reduce aliasing artifacts. Bowers et al. [14] used a classic technique in their ray tracing formulation, such that the origin of each ray is set as a 2×2 jittered pattern. Pang et al. [102] performed aliasing reduction on curve boundaries by using an accumulation buffer. Sun et al. [121] proposed a random access solver that treats the image plane as a continuous domain. They evaluate the integral over a small rectangular region instead of at each individual point. This formulation, however, may cause a performance drop for the fast multipole formulation [120]. Instead, Sun et al. [120] used supersampling with 9 samples per pixel.

4.4 Vectorization

Current research on the extraction of curve-based representations from images is mainly focused on diffusion curves, especially the original, harmonic and biharmonic formulations. Since the original formulation and the harmonic formulation extract the same attributes, we cover them together.

4.4.1 Basic Formulation and Harmonic Equation

The simplest harmonic formulation comprises curve geometry, color control points, and usually blur control points.

Geometry. Diffusion curves are based on the idea that edges can represent the majority of color variations in an image [73], [97], [101]. Therefore, edge detection often serves

in the initialization. Orzan et al. [101] used a Canny detector in the Gaussian scale space (the collection of Gaussian blurred images) on the input image, which captures local maxima in the gradient domain for different scales. A scale-space analysis [34], [90] is performed to select the best scale for each edge. This approach was used by several follow-up works [82], [94], [143] as starting point, too. However, as discussed by Orzan et al. [101], it is hard to detect edges in smooth regions, and these areas are often extracted as several smaller edges, which necessitates a smoothing post-process to represent smooth regions.

Instead of using a Canny detector, Dai et al. [26] segmented the input image into several superpixels, which were then merged into a single superpixel map with k-labeled regions through bipartite graph partitioning. The curves were later extracted as superpixel boundaries. Smooth regions were segmented into several superpixels, and hence the reblurring process was unnecessary. Zhao et al. [143] used iso-contours of the difference between the reconstructed image and the original image as the initial curves in an optimization step. The curves were iteratively refined through gradient descent until the reconstruction error is small enough. In addition to pixel images, the approach can also be used to generate diffusion curve representations for other color fields such as 3D renderings and gradient meshes, where the initial reconstruction is created using object contours or triangle edges. Lu et al. [94] used an additional depth image to improve the object contour detection. A Canny detector is used on the multi-scale color image, while an enhanced cartoon edge detector [24] is applied on the depth map. The edges of both images are combined to generate diffusion curves, where coarse edges in depth maps represent object contours and edges in the color map represent color details. Li et al. [82] extended the vectorization process from images to videos. In each frame, a Canny detector is used and curves are classified as salient and non-salient curves based on the average gradient along the curves. Non-salient curves have a smaller average gradient and are the ones that cause the flickering artifacts in videos. By assuming that each curve is a rigid body, non-salient curves are propagated from one frame to another frame using optical flow. The curves are eventually transformed to Bézier curves.

Color. Given a curve geometry, the color control points are computed next, e.g., by sampling color points on the left and right sides of the curves in the original image [143]. After uniformly sampling color points, Orzan et al. [101] simplified the piecewise linear color curves using Douglas-Peucker [31] with color differences measured in the perceptually uniform CIE L*a*b color space. Jeschke et al. [71] first computed a dense set of color control points using a geometric heuristic and removed outliers similar to Orzan et al. [101]. A fitting process is then expressed as a linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$, where $\mathbf{x}$ are the color points, $\mathbf{b}$ are the image colors, and $\mathbf{A}$ is the influence of control points onto the input image. The influence matrix $\mathbf{A}$ is computed during a single diffusion process and the optimal color points are calculated using linear least squares. Later, Jeschke et al. [68] further optimized the computation of the influence matrix, leading to an easier implementation and a more accurate reconstruction. The new scheme keeps a fixed set of the most influential color control points for each pixel. To achieve temporal coherence in videos, Li et al. [82] sampled colors on the left and right side of salient curves as an initial guess, and iteratively refined color points by minimizing a color energy.

Blur. Harmonic equations often require a smoothing step to achieve smooth edges, cf. Section. 4.1.2. Thus, blur control points have to be extracted during the inverse process. Orzan et al. [101] computed Gaussian blurred images of the input image and selected the best scale to represent the edges through a scale-space analysis. This ideal scale is used as blur value for the edge pixels. The blur control points are then sampled in a way similar to the color control points. Jeschke et al. [68] first estimated blur values locally at each piecewise linear curve segment. The blur values are then fit to blur control points using a least squares optimization.

Noise. In addition to blur control points, Jeschke et al. [71] estimated a Gaussian distribution of Gabor noise parameters to mimic textures in an input image.

4.4.2 Biharmonic Equation

The inverse problem for biharmonic equations is harder than for harmonic equations, as more constraints are required. So far, only Xie et al. [138] proposed a vectorization method.

Geometry. Xie et al. [138] have shown that color variations in the Laplacian domain cannot be properly described by edges extracted in the gradient domain. To solve this, they applied a Canny detector on the Laplacian and bi-Laplacian domain of the input image hierarchically. This multi-scale representation avoids the smoothing process and extends the application to biharmonic diffusion curves. As derived by Ilbery et al. [64], Laplacian curves can be considered a special case of bi-Laplacian curves. Therefore, a naïve approach is to directly solve for all curves in the bi-Laplacian domain. However, it is faster to solve for Laplacian curves than for bi-Laplacian curves. Therefore, Xie et al. [138] first extracted curves in the Laplacian domain, and used a voting method to determine if a curve is a Laplacian curve. Then, they extracted curves in the bi-Laplacian domain and subtracted the unclassified curves extracted in the Laplacian domain. The subtracted bi-Laplacian curves and the classified Laplacian curves are the output diffusion curves.

Attributes. Since Green's formulation from Sun et al. [121] and Ilbery et al. [64] is adopted, the curve color, the normal derivative of color, the Laplacian along curves, and the normal derivative of the Laplacian are solved with a direct linear system. Xie et al. [138] reduced the size of the system matrix by linearly interpolating most of the weights, using a variable stencil size similar to Jeschke et al. [69], solving for both Laplacian and bi-Laplacian together, and by using Bézier splines to share control points.

5 Conclusion and Outlook

We provided a comprehensive overview of state-of-art methods for the representation, creation, rasterization, and vectorization of mesh-based and curve-based vector graphics. Although mesh- and curve-based methods have conceptual differences in their underlying mathematical representations, there are potential synergies in their rasterization and vectorization processes, as well as in the interaction techniques

that enable artistic control. Next, we point out a number of opportunities for future work.

3D Vector Graphics. This survey concentrated on mesh-based and curve-based methods for smooth 2D vector graphics. It is mentioned in several works [11], [14], [26], [69], [70] that their representations may be extended to 3D (2D animation or 3D image) or even 4D (3D + time). Compared to 3D images, 2D animations or vector videos have been studied more intensively. One popular commercial tool for creating and editing vector graphics animations is Adobe Flash, which enables users to manipulate with simple shapes and gradients. Video vectorization algorithms have also been proposed for both curve-based [82] and mesh-based [128] representations. On the other hand, the inverse processes, such as the creation and rasterization of vector videos, are less researched. Takayama et al. [124] extended diffusion curves in 2D to diffusion surfaces in 3D. Similarly, Lu et al. [65] proposed a tool to create 2D paths that can be joined to form meshes. However, current results are still much less expressive compared to the achievements in 2D images, and so far the concept of 4D vector graphics remains untouched to our best knowledge.

Vectorization. The vectorization process is a popular research topic in mesh-based and curve-based methods. Both representations use similar steps. For many mesh-based and all the curve-based representations, current algorithms follow a two-step scheme. The first step extracts the geometry from an input raster image, where edge and corner detection methods are usually adopted to obtain the image features. Mesh-based methods will then perform a patch fitting or triangulation process to locate the patch vertices, while curve-based methods usually fit a parametric curve or spline to the detected features. It is also common that the geometry may be further optimized through a remeshing or shape optimization process to reduce the construction error [84], [144]. The second step is then to extract the attributes of the geometry. Both representations sample colors from the input image. At present, there is only one method designed for biharmonic curves, while all the other methods focused on the original or harmonic representations. Further, there is no vectorization method for the Poisson representation yet. Contrarily, to solve for more attributes, the gradient mesh representation computes both the geometry and the attributes in one optimization step.

Artistic Degrees of Freedom. At present, users may freely edit and control color and shape through control points, which is a rather fine-grained user control. In raster graphics, plenty of research went into the artistic editing of appearance, lighting and materials [111]. It is well imaginable that similarly more such high-level editing tools will be developed for vector art. Especially in stylized images, where the shading might not be physically plausible, relighting and appearance editing are challenging tasks.

Sketch-based Systems. Currently, mainly curve-based methods support free-hand drawing of the curve geometry. The input and adjustment of attributes are more restricted to a select-and-change format for both curve- and mesh-based representations. It would be interesting to extend sketch-based input for all formulations, such that attributes can be added in a more intuitive way. In addition, the ideas from sketch-based systems and applications may also be transferred to vector graphics [62], [133].

Data-driven Methods. So far, data-driven methods have been intensively studied for generating new vector-based images or vectorizing a given image, mainly for simple mesh-based representations, especially in the SVG format. For example, DeepSVG [20] presented a hierarchical transformer-based network to generate a set of drawing commands that can be rendered into SVG images. SVG-VAE [92] generated similar drawing instructions through a variational autoencoder. Later, Reddy et al. [107] proposed a generative model that does not require direct supervision through a differentiable rasterizer [80]. So far, the training data are often restricted to stylized images [107], [108]. Although this has been extended to photorealistic images [95] recently, the more complex vector representations like diffusion curves and gradient meshes are still not covered. One reason is the lack of data, as these representations are less available to artists and automatic vectorization results often lead to too many primitives. It is then not easy to generate sufficient ground truth data that has a good balance between image expressiveness and low number of primitives. Another reason might be the richness of the attributes in these representations, as more parameters need to be properly learned to generate convincing results, which also adds complexity in the training model.

Standardization. The SVG format has become industry standard for the representation of basic vector graphics primitives. For the many curve-based and mesh-based formulations, no such default format is available yet. Standardization will be an important step on the road to a wider adoption, because content creation tools and rendering APIs will need a common basis for communication.

Synergies. Curve-based and mesh-based methods have both their individual strengths and weaknesses. While editing tools are more advanced for curve-based methods, vectorization is more matured for mesh-based techniques. First hybrid techniques have already been proposed. Lieung et al. [87] introduced the idea of shading curves with which users can draw lines on flat images, while the shading effects are rendered as Catmull-Clark surfaces along the curves. Chen et al. [23] embedded image features into parametric patches and solved for the color through thin plate spline interpolation to enable real-time vectorization. In the future, it will be rewarding to develop new mathematical models that unify curve-based and mesh-based methods to leverage the benefits of both formulations.

Applications. Vector graphics have been adapted in other areas in visual computing, such as for textures [70], [121] or height fields [56]. The compactness of diffusion curve representations can also be used as an intermediate representation for image compression in data-driven methods. For example Poisson vector graphics has been used in the color transfer for faces and portraits [44], [45]. With more advances in the representation, creation, rasterization, and image vectorization in the future, we look forward to more exciting applications of vector graphics in other domains.

REFERENCES

[1] A. Abadpour and S. Kasaei. An efficient PCA-based color transfer method. *Journal of Visual Communication and Image Representation*, 18(1):15–34, 2007.

[2] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(5):898–916, 2010.

[3] S. D. Baksteen, G. J. Hettinga, J. Echevarria, and J. Kosinka. Mesh colours for gradient meshes. *STAG: Smart Tools and Applications in Graphics*, 2021.

[4] B. Bao and H. Fu. Scribble-based colorization for creating smooth-shaded vector graphics. *Computers & Graphics*, 81:73–81, 2019.

[5] P. J. Barendrecht, M. Luinstra, J. Hogervorst, and J. Kosinka. Locally refinable gradient meshes supporting branching and sharp colour transitions. *The Visual Computer*, 34(6):949–960, 2018.

[6] P. Barla and A. Bousseau. Gradient art: Creation and vectorization. In *Image and Video-Based Artistic Stylisation*, pp. 149–166. Springer, 2013.

[7] S. Battiato, G. Barbera, G. Di Blasi, G. Gallo, and G. Messina. Advanced SVG triangulation/polygonalization of digital images. In *Internet Imaging VI*, vol. 5670, pp. 1–11. SPIE, 2005.

[8] S. Battiato, G. Blasi, G. Gallo, and E. Messina. Firemark: a Java tool for SVG watermarking. In *Proc. Fourth Ann. Conf. Scalable Vector Graphics (SVGOpen'05)*, 2005.

[9] S. Battiato, G. Gallo, and G. Messina. SVG rendering of real images using data dependent triangulation. In *Proceedings of the 20th Spring Conference on Computer Graphics*, SCCG '04, p. 185–192. Association for Computing Machinery, New York, NY, USA, 2004.

[10] R. Beatson, M. S. Floater, and C. E. Kåshagen. Hermite mean value interpolation on polygons. *Computer Aided Geometric Design*, 60:18–27, 2018.

[11] H. Bezerra, E. Eisemann, D. DeCarlo, and J. Thollot. Diffusion constraints for vector graphics. In *Proc. International Symposium on Non-photorealistic Animation and Rendering*, pp. 35–42, 2010.

[12] A. K. Bhunia, P. N. Chowdhury, Y. Yang, T. M. Hospedales, T. Xiang, and Y.-Z. Song. Vectorization and rasterization: Self-supervised learning for sketch and handwriting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5672–5681, 2021.

[13] M. Botsch, D. Bommes, and L. Kobbelt. Efficient linear system solvers for mesh processing. In *Mathematics of Surfaces XI*, pp. 62–83. Springer, 2005.

[14] J. C. Bowers, J. Leahey, and R. Wang. A ray tracing approach to diffusion curves. In *Computer Graphics Forum*, vol. 30, pp. 1345–1352. Wiley Online Library, 2011.

[15] S. Boyé, P. Barla, and G. Guennebaud. A vectorial solver for free-form vector gradients. *ACM Transactions on Graphics (TOG)*, 31(6):1–9, 2012.

[16] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9):1124–1137, 2004.

[17] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A multigrid tutorial*. SIAM, 2000.

[18] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (6):679–698, 1986.

[19] J. Cao, Z. Chen, X. Wei, and Y. J. Zhang. A finite element framework based on bivariate simplex splines on triangle configurations. *Computer Methods in Applied Mechanics and Engineering*, 357:112598, 2019.

[20] A. Carlier, M. Danelljan, A. Alahi, and R. Timofte. DeepSVG: A hierarchical generative network for vector graphics animation. *Advances in Neural Information Processing Systems*, 33:16351–16361, 2020.

[21] E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 10(6):350–355, 1978.

[22] P. Charrot and J. A. Gregory. A pentagonal surface patch for computer aided geometric design. *Computer Aided Geometric Design*, 1(1):87–94, 1984.

[23] K.-W. Chen, Y.-S. Luo, Y.-C. Lai, Y.-L. Chen, C.-Y. Yao, H.-K. Chu, and T.-Y. Lee. Image vectorization with real-time thin-plate spline. *IEEE Transactions on Multimedia*, 22(1):15–29, 2019.

[24] M.-M. Cheng. Curve structure extraction for cartoon images. In *Proceedings of the 5th Joint Conference on Harmonious Human Machine Environment*, pp. 13–25, 2009.

[25] H. Chiyokura and F. Kimura. Design of solids with free-form surfaces. In *Proceedings of the 10th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 289–298, 1983.

[26] W. Dai, T. Luo, and J. Shen. Automatic image vectorization using superpixels and random walkers. In *International Congress on Image and Signal Processing (CISP)*, vol. 2, pp. 922–926. IEEE, 2013.

[27] F. Davoine, M. Antonini, J.-M. Chassery, and M. Barlaud. Fractal image compression based on Delaunay triangulation and vector quantization. *IEEE Trans. Image Processing*, 5(2):338–346, 1996.

[28] B. F. De Veubeke. Variational principles and the patch test. *International Journal for Numerical Methods in Engineering*, 8(4):783–801, 1974.

[29] L. Demaret, N. Dyn, and A. Iske. Image compression by linear splines over adaptive triangulations. *Signal Processing*, 86(7):1604–1616, 2006.

[30] E. A. Dominici, N. Schertler, J. Griffin, S. Hoshyari, L. Sigal, and A. Sheffer. PolyFit: Perception-aligned vectorization of raster clip-art via intermediate polygonal fitting. *ACM Transactions on Graphics (TOG)*, 39(4):77–1, 2020.

[31] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: the international journal for geographic information and geovisualization*, 10(2):112–122, 1973.

[32] N. Dyn, D. Levin, and S. Rippa. Data dependent triangulations for piecewise linear interpolation. *IMA Journal of Numerical Analysis*, 10(1):137–154, 1990.

[33] V. Egiazarian, O. Voynov, A. Artemov, D. Volkhonskiy, A. Safin, M. Taktasheva, D. Zorin, and E. Burnaev. Deep vectorization of technical drawings. In *European Conference on Computer Vision*, pp. 582–598. Springer, 2020.

[34] J. H. Elder. Are edges incomplete? *International Journal of Computer Vision*, 34(2-3):97–122, 1999.

[35] K. Ellis, D. Ritchie, A. Solar-Lezama, and J. Tenenbaum. Learning to infer graphics programs from hand-drawn images. *Advances in Neural Information Processing Systems*, 31, 2018.

[36] G. E. Farin and G. Farin. *Curves and surfaces for CAGD: a practical guide*. Morgan Kaufmann, 2002.

[37] J.-D. Favreau, F. Lafarge, and A. Bousseau. Fidelity vs. simplicity: a global approach to line drawing vectorization. *ACM Transactions on Graphics (TOG)*, 35(4):1–10, 2016.

[38] J.-D. Favreau, F. Lafarge, and A. Bousseau. Photo2ClipArt: Image abstraction and vectorization using layered linear gradients. *ACM Transactions on Graphics (TOG)*, 36(6):1–11, 2017.

[39] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167–181, 2004.

[40] J. Ferguson. Multivariable curve interpolation. *J. ACM*, 11(2):221–228, apr 1964.

[41] M. Finch, J. Snyder, and H. Hoppe. Freeform vector graphics with controlled thin-plate splines. *ACM Transactions on Graphics (TOG)*, 30(6):1–10, 2011.

[42] M. S. Floater. Mean value coordinates. *Computer Aided Geometric Design*, 20(1):19–27, 2003.

[43] M. Froumentin, F. Labrosse, and P. Willis. A vector-based representation for image warping. In *Computer Graphics Forum*, vol. 19, pp. 419–425. Wiley Online Library, 2000.

[44] Q. Fu, Y. He, F. Hou, Q. Sun, A. Zeng, Z. Huang, J. Zhang, and Y.-J. Liu. Poisson vector graphics (PVG)-guided face color transfer in videos. *IEEE Computer Graphics and Applications*, 2020.

[45] Q. Fu, Y. He, F. Hou, J. Zhang, A. Zeng, and Y.-J. Liu. Vectorization based color transfer for portrait images. *Computer-Aided Design*, 115:111–121, 2019.

[46] Q. Fu, F. Hou, Q. Sun, S.-Q. Xin, Y.-J. Liu, W. Wang, H. Qin, and Y. He. Decorating 3D models with poisson vector graphics. *Computer-Aided Design*, 102:1–11, 2018.

[47] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 209–216, 1997.

[48] N. Goodnight, C. Woolley, G. Lewin, D. Luebke, and G. Humphreys. A multigrid solver for boundary value problems using programmable graphics hardware. In *ACM SIGGRAPH 2005 Courses*, pp. 193–es. 2005.

[49] Y. Guo, Z. Zhang, C. Han, W. Hu, C. Li, and T.-T. Wong. Deep line drawing vectorization via line subdivision and topology reconstruction. In *Computer Graphics Forum*, vol. 38, pp. 81–90. Wiley Online Library, 2019.

[50] Y. He, S. H. Kang, and J.-M. Morel. Topology and perception aware image vectorization. *Research Square Preprint*, 2022. doi: 10.21203/rs.3.rs-1817017/v1.

[51] G. Hettinga, J. Echevarria, and J. Kosinka. Efficient image vectorisation using mesh colours. In *Italian Chapter Conference 2021: Smart Tools and Apps in Graphics*. Eurographics Association, 2021.

[52] G. J. Hettinga, P. J. Barendrecht, and J. Kosinka. A comparison of gpu tessellation strategies for multisided patches. In *Eurographics (Short Papers)*, pp. 45–48, 2018.

[53] G. J. Hettinga, R. Brals, and J. Kosinka. Colour interpolants for polygonal gradient meshes. *Computer Aided Geometric Design*, 74:101769, 2019.

[54] G. J. Hettinga, J. Echevarria, and J. Kosinka. Adaptive image vectorisation and brushing using mesh colours. *Computers & Graphics*, 2022.

[55] G. J. Hettinga, R. van Beckhoven, and J. Kosinka. Noisy gradient meshes: Augmenting gradient meshes with procedural noise. *Graphical Models*, 103:101024, 2019.

[56] H. Hnaidi, E. Guérin, S. Akkouche, A. Peytavie, and E. Galin. Feature based terrain generation using diffusion equation. In *Computer Graphics Forum*, vol. 29, pp. 2179–2186. Wiley Online Library, 2010.

[57] K. E. Hoff III, J. Keyser, M. Lin, D. Manocha, and T. Culver. Fast computation of generalized Voronoi diagrams using graphics hardware. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 277–286, 1999.

[58] H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, and W. Stuetzle. Piecewise smooth surface reconstruction. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, pp. 295–302, 1994.

[59] S. Hoshyari, E. A. Dominici, A. Sheffer, N. Carr, Z. Wang, D. Ceylan, and I.-C. Shen. Perception-driven semi-structured boundary vectorization. *ACM Transactions on Graphics (TOG)*, 37(4):1–14, 2018.

[60] F. Hou, Q. Sun, Z. Fang, Y.-J. Liu, S.-M. Hu, H. Qin, A. Hao, and Y. He. Poisson vector graphics (PVG) and its closed-form solver. *arXiv preprint arXiv:1701.04303*, 2017.

[61] F. Hou, Q. Sun, Z. Fang, Y.-J. Liu, S.-M. Hu, H. Qin, A. Hao, and Y. He. Poisson vector graphics (PVG). *IEEE Transactions on Visualization and Computer Graphics*, 26(2):1361–1371, 2018.

[62] Z. Hu, H. Xie, T. Fukusato, T. Sato, and T. Igarashi. Sketch2VF: Sketch-based flow design with conditional generative adversarial network. *Computer Animation and Virtual Worlds*, 30(3-4):e1889, 2019.

[63] T. Igarashi, T. Moscovich, and J. F. Hughes. As-rigid-as-possible shape manipulation. *ACM Transactions on Graphics (TOG)*, 24(3):1134–1141, 2005.

[64] P. Ilbery, L. Kendall, C. Concolato, and M. McCosker. Biharmonic diffusion curve images from boundary elements. *ACM Transactions on Graphics (TOG)*, 32(6):1–12, 2013.

[65] M.-Y. Iu. Diffusionmesh. In *Proceedings of Graphics Interface 2019*. Canadian Information Processing Society, 2019.

[66] A. Jacobson, T. Weinkauf, and O. Sorkine. Smooth shape-aware functions with controlled extrema. In *Computer Graphics Forum*, vol. 31, pp. 1577–1586. Wiley Online Library, 2012.

[67] J. F. Jarvis, C. N. Judice, and W. H. Ninke. A survey of techniques for the display of continuous tone pictures on bilevel displays. *Computer Graphics and Image Processing*, 5(1):13–40, 1976.

[68] S. Jeschke. Generalized diffusion curves: An improved vector representation for smooth-shaded images. In *Computer Graphics Forum*, vol. 35, pp. 71–79. Wiley Online Library, 2016.

[69] S. Jeschke, D. Cline, and P. Wonka. A GPU Laplacian solver for diffusion curves and Poisson image editing. In *ACM SIGGRAPH Asia 2009 papers*, pp. 1–8. 2009.

[70] S. Jeschke, D. Cline, and P. Wonka. Rendering surface details with diffusion curves. In *ACM SIGGRAPH Asia papers*, pp. 1–8. 2009.

[71] S. Jeschke, D. Cline, and P. Wonka. Estimating color and texture parameters for vector graphics. In *Computer Graphics Forum*, vol. 30, pp. 523–532. Wiley Online Library, 2011.

[72] T. Ju, S. Schaefer, and J. Warren. Mean value coordinates for closed triangular meshes. In *ACM Siggraph 2005 Papers*, pp. 561–566. 2005.

[73] J. J. Koenderink and A. J. Van Doorn. The internal representation of solid shape with respect to vision. *Biological cybernetics*, 32(4):211–216, 1979.

[74] Y.-K. Lai, S.-M. Hu, and R. R. Martin. Automatic and topology-preserving gradient mesh generation for image vectorization. *ACM Transactions on Graphics (TOG)*, 28(3):1–8, 2009.

[75] P. Lascaux and P. Lesaint. Some nonconforming finite elements for the plate bending problem. *Revue française d'automatique, informatique, recherche opérationnelle. Analyse numérique*, 9(R1):9–53, 1975.

[76] K. Lawonn and T. Günther. Stylized image triangulation. *Computer Graphics Forum*, 38(1):221–234, 2019.

[77] G. Lecot and B. Levy. Ardeco: Automatic region detection and conversion. 2006.

[78] S.-M. Lee. *Wavelet-Based Multiresolution Surface Approximation from Height Fields*. PhD thesis, Virginia Polytechnic Institute and State University, 2002.

[79] K. Levenberg. A method for the solution of certain problems in least squares. quart. appl math. 1944.

[80] T.-M. Li, M. Lukáč, M. Gharbi, and J. Ragan-Kelley. Differentiable vector graphics rasterization for editing and learning. *ACM Transactions on Graphics (TOG)*, 39(6):1–15, 2020.

[81] X.-Y. Li, T. Ju, and S.-M. Hu. Cubic mean value coordinates. *ACM Transactions on Graphics (TOG)*, 32(4):126–1, 2013.

[82] Y. Li, C. Wang, J. Hong, J. Zhu, J. Guo, J. Wang, Y. Guo, and W. Wang. Video vectorization via bipartite diffusion curves propagation and optimization. *IEEE Transactions on Visualization and Computer Graphics*, 2021.

[83] Y. Li, X. Zhai, F. Hou, Y. Liu, A. Hao, and H. Qin. Vectorized painting with temporal diffusion curves. *IEEE Transactions on Visualization and Computer Graphics*, 27(1):228–240, 2019.

[84] Z. Liao, H. Hoppe, D. Forsyth, and Y. Yu. A subdivision-based representation for vector image editing. *IEEE Transactions on Visualization and Computer Graphics*, 18(11):1858–1867, 2012.

[85] H. Lieng. Ray-traced diffusion points. In *ACM SIGGRAPH 2016 Posters*, pp. 1–2. 2016.

[86] H. Lieng, J. Kosinka, J. Shen, and N. A. Dodgson. A colour interpolation scheme for topologically unrestricted gradient meshes. In *Computer Graphics Forum*, vol. 36, pp. 112–121. Wiley Online Library, 2017.

[87] H. Lieng, F. Tasse, J. Kosinka, and N. A. Dodgson. Shading curves: Vector-based drawing with explicit gradient control. In *Computer Graphics Forum*, vol. 34, pp. 228–239. Wiley Online Library, 2015.

[88] H. Lin, J. Zhang, and C. Xu. Diffusion curves with diffusion coefficients. *Computational Visual Media*, 4(2):7, 2018.

[89] J. Lin, G. Zhang, L. Wang, X. Gao, M. Liao, and Y. Zhang. Diffusion based vector graphics on mobile devices. In *2015 Ninth International Conference on Frontier of Computer Science and Technology*, pp. 153–156. IEEE, 2015.

[90] T. Lindeberg. Edge detection and ridge detection with automatic scale selection. *International Journal of Computer Vision*, 30(2):117–156, 1998.

[91] C. Liu, J. Wu, P. Kohli, and Y. Furukawa. Raster-to-Vector: Revisiting floorplan transformation. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2195–2203, 2017.

[92] R. G. Lopes, D. Ha, D. Eck, and J. Shlens. A learned representation for scalable vector graphics. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 7930–7939, 2019.

[93] S. Lu, X. Ding, F. Gao, and J. Chen. Shape manipulation of diffusion curves images. *IEEE Access*, 8:57158–57167, 2020.

[94] S. Lu, W. Jiang, X. Ding, C. S. Kaplan, X. Jin, F. Gao, and J. Chen. Depth-aware image vectorization and editing. *The Visual Computer*, 35(6):1027–1039, 2019.

[95] X. Ma, Y. Zhou, X. Xu, B. Sun, V. Filev, N. Orlov, Y. Fu, and H. Shi. Towards layer-wise image vectorization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16314–16323, June 2022.

[96] M. Maire, P. Arbeláez, C. Fowlkes, and J. Malik. Using contours to detect and localize junctions in natural images. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8. IEEE, 2008.

[97] D. Marr and E. Hildreth. Theory of edge detection. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 207(1167):187–217, 1980.

[98] M. Nießner, C. Loop, M. Meyer, and T. Derose. Feature-adaptive GPU rendering of Catmull-Clark subdivision surfaces. *ACM Transactions on Graphics (TOG)*, 31(1):1–11, 2012.

[99] J. Nocedal and S. J. Wright. *Numerical optimization*. Springer, 1999.

[100] G. Noris, A. Hornung, R. W. Sumner, M. Simmons, and M. Gross. Topology-driven vectorization of clean line drawings. *ACM Transactions on Graphics (TOG)*, 32(1):1–11, 2013.

[101] A. Orzan, A. Bousseau, H. Winnemöller, P. Barla, J. Thollot, and D. Salesin. Diffusion curves: a vector representation for smooth-shaded images. *ACM Trans. Graph. (TOG)*, 27(3):1–8, 2008.

[102] W.-M. Pang, J. Qin, M. Cohen, P.-A. Heng, and K.-S. Choi. Fast rendering of diffusion curves with triangles. *IEEE Computer Graphics and Applications*, 32(4):68–78, 2011.

[103] M. Powell. A thin plate spline method for mapping curves into curves in two dimensions. *Computational Techniques and Applications: CTAC95*, pp. 43–57, 1996.

[104] R. Prévost, W. Jarosz, and O. Sorkine-Hornung. A vectorial framework for ray traced diffusion curves. In *Computer Graphics Forum*, vol. 34, pp. 253–264. Wiley Online Library, 2015.

[105] B. Price and W. Barrett. Object-based vectorization for interactive image editing. *The Visual Computer*, 22(9):661–670, 2006.

[106] Y. Qi, D. Seyb, B. Bitterli, and W. Jarosz. A bidirectional formulation for walk on spheres. *Computer Graphics Forum*, 2022.

[107] P. Reddy, M. Gharbi, M. Lukac, and N. J. Mitra. Im2Vec: Synthesizing vector graphics without vector supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7342–7351, 2021.

[108] L. S. F. Ribeiro, T. Bui, J. Collomosse, and M. Ponti. Sketchformer: Transformer-based representation for sketched structure. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14153–14162, 2020.

[109] C. Rother, V. Kolmogorov, and A. Blake. "GrabCut" – interactive foreground extraction using iterated graph cuts. *ACM Transactions on Graphics (TOG)*, 23(3):309–314, 2004.

[110] R. Sawhney and K. Crane. Monte Carlo geometry processing: A grid-free approach to PDE-based methods on volumetric domains. *ACM Trans. Graph.*, 39(4), jul 2020.

[111] T.-W. Schmidt, F. Pellacini, D. Nowrouzezahrai, W. Jarosz, and C. Dachsbacher. State of the art in artistic editing of appearance, lighting and material. *Computer Graphics Forum*, 35(1):216–233, 2016.

[112] D. Schmitt. Bivariate B-splines from convex pseudo-circle configurations. In *International Symposium on Fundamentals of Computation Theory*, pp. 335–349. Springer, 2019.

[113] P. J. Schneider. An algorithm for automatically fitting digitized curves. *Graphics Gems*, 1:612–626, 1990.

[114] I.-C. Shen and B.-Y. Chen. ClipGen: A deep generative model for clipart vectorization and synthesis. *IEEE Transactions on Visualization and Computer Graphics*, 2021.

[115] J. R. Shewchuk. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In *Workshop on Applied Computational Geometry*, pp. 203–222. Springer, 1996.

[116] D. Su and P. Willis. Image interpolation by pixel-level data-dependent triangulation. In *Computer Graphics Forum*, vol. 23, pp. 189–201. Wiley Online Library, 2004.

[117] H. Su, J. Niu, X. Liu, J. Cui, and J. Wan. Vectorization of raster manga by deep reinforcement learning. *arXiv preprint arXiv:2110.04830*, 2021.

[118] J. Sun, L. Liang, F. Wen, and H.-Y. Shum. Image vectorization using optimized gradient meshes. *ACM Transactions on Graphics (TOG)*, 26(3):11–es, 2007.

[119] Q. Sun, C.-W. Fu, and Y. He. An interactive multi-touch sketching interface for diffusion curves. In *Proc. SIGCHI Conference on Human Factors in Computing Systems*, pp. 1611–1614, 2011.

[120] T. Sun, P. Thamjaroenporn, and C. Zheng. Fast multipole representation of diffusion curves and points. *ACM Transactions on Graphics (TOG)*, 33(4):53–1, 2014.

[121] X. Sun, G. Xie, Y. Dong, S. Lin, W. Xu, W. Wang, X. Tong, and B. Guo. Diffusion curve textures for resolution independent texture mapping. *ACM Trans. Graph. (TOG)*, 31(4):1–9, 2012.

[122] J. K. Svergja and H. Lieng. A gradient mesh tool for non-rectangular gradient meshes. In *ACM SIGGRAPH 2017 Posters*, pp. 1–2. 2017.

[123] S. Swaminarayan and L. Prasad. Rapid automated polygonal image decomposition. In *35th IEEE Applied Imagery and Pattern Recognition Workshop (AIPR'06)*, pp. 28–28. IEEE, 2006.

[124] K. Takayama, O. Sorkine, A. Nealen, and T. Igarashi. Volumetric modeling with diffusion surfaces. In *ACM SIGGRAPH Asia 2010 papers*, pp. 1–8. 2010.

[125] T. Várady, P. Salvi, and G. Karikó. A multi-sided Bézier patch with a simple control structure. In *Computer Graphics Forum*, vol. 35, pp. 307–317. Wiley Online Library, 2016.

[126] T. W. Verstraaten and J. Kosinka. Local and hierarchical refinement for subdivision gradient meshes. In *Computer Graphics Forum*, vol. 37, pp. 373–383. Wiley Online Library, 2018.

[127] L. Wan, Y. Xiao, N. Dou, C.-S. Leung, and Y.-K. Lai. Scribble-based gradient mesh recoloring. *Multimedia Tools and Applications*, 77(11):13753–13771, 2018.

[128] C. Wang, J. Zhu, Y. Guo, and W. Wang. Video vectorization via tetrahedral remeshing. *IEEE Transactions on Image Processing*, 26(4):1833–1844, 2017.

[129] O. Weber, R. Poranne, and C. Gotsman. Biharmonic coordinates. In *Computer Graphics Forum*, vol. 31, pp. 2409–2422. Wiley Online Library, 2012.

[130] G. Wei, Y. Zhou, X. Gao, Q. Ma, S. Xin, and Y. He. Field-aligned quadrangulation for image vectorization. In *Computer Graphics Forum*, vol. 38, pp. 171–180. Wiley Online Library, 2019.

[131] G. Wolberg and I. Alfy. Monotonic cubic spline interpolation. In *Computer Graphics International*, pp. 188–195, 1999.

[132] T. Xia, B. Liao, and Y. Yu. Patch-based image vectorization with automatic curvilinear feature alignment. *ACM Transactions on Graphics (TOG)*, 28(5):1–10, 2009.

[133] C. Xiao, D. Yu, X. Han, Y. Zheng, and H. Fu. SketchHairSalon: Deep sketch-based hair image synthesis. *ACM Transactions on Graphics (TOG)*, 40(6):1–16, 2021.

[134] X. Xiao and L. Ma. Color transfer in correlated color space. In *Proceedings of the 2006 ACM International Conference on Virtual Reality Continuum and its Applications*, pp. 305–309, 2006.

[135] Y. Xiao, J. Cao, and Z. Chen. Image representation on curved optimal triangulation. In *Computer Graphics Forum*. Wiley Online Library, 2022.

[136] Y. Xiao, L. Wan, C.-S. Leung, Y.-K. Lai, and T.-T. Wong. Example-based color transfer for gradient meshes. *IEEE Transactions on Multimedia*, 15(3):549–560, 2012.

[137] Y. Xiao, L. Wan, C. S. Leung, Y.-K. Lai, and T.-T. Wong. Optimization-based gradient mesh colour transfer. In *Computer Graphics Forum*, vol. 34, pp. 123–134. Wiley Online Library, 2015.

[138] G. Xie, X. Sun, X. Tong, and D. Nowrouzezahrai. Hierarchical diffusion curves for accurate automatic image vectorization. *ACM Transactions on Graphics (TOG)*, 33(6):1–11, 2014.

[139] M. Yang, H. Chao, C. Zhang, J. Guo, L. Yuan, and J. Sun. Effective clipart image vectorization through direct optimization of Bézigons. *IEEE Transactions on Visualization and Computer Graphics*, 22(2):1063–1075, 2015.

[140] L. Yatziv and G. Sapiro. Fast image and video colorization using chrominance blending. *IEEE Transactions on Image Processing*, 15(5):1120–1129, 2006.

[141] X. Yu, B. Bryan, and T. W. Sederberg. Image reconstruction using data-dependent triangulation. *IEEE Computer Graphics and Applications*, 21(3):62–68, 2001.

[142] C. Yuksel, J. Keyser, and D. H. House. Mesh colors. *ACM Transactions on Graphics (TOG)*, 29(2):1–11, 2010.

[143] S. Zhao, F. Durand, and C. Zheng. Inverse diffusion curves using shape optimization. *IEEE Transactions on Visualization and Computer Graphics*, 24(7):2153–2166, 2017.

[144] H. Zhou, J. Zheng, and L. Wei. Representing images using curvilinear feature driven subdivision surfaces. *IEEE Transactions on Image Processing*, 23(8):3268–3280, 2014.

[145] J. Zhou, G. J. Hettinga, S. Houwink, and J. Kosinka. Feature-adaptive and hierarchical subdivision gradient meshes. In *Computer Graphics Forum*, vol. 41, pp. 389–401. Wiley Online Library, 2022.

[146] H. Zhu, J. Cao, Y. Xiao, Z. Chen, Z. Zhong, and Y. J. Zhang. TCB-spline-based image vectorization. *ACM Trans. Graph.*, 41(3), 2022.

[147] D. Zorin. Modeling with multiresolution subdivision surfaces. In *ACM SIGGRAPH 2006 Courses*, pp. 30–50. 2006.

Xingze Tian is a PhD student at the Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany. Her research interests include visualization and computer graphics.

Tobias Günther is a professor of computer science at the Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany. His research interests include visualization and computer graphics. He received his PhD from the University of Magdeburg in 2016.