Unified Smooth Vector Graphics: Modeling Gradient Meshes and Curve-based Approaches Jointly as Poisson Problem

Xingze Tian and Tobias Günther

Abstract—Research on smooth vector graphics is separated into two independent research threads: one on interpolation-based gradient meshes and the other on diffusion-based curve formulations. With this paper, we propose a mathematical formulation that unifies gradient meshes and curve-based approaches as solution to a Poisson problem. To combine these two well-known representations, we first generate a non-overlapping intermediate patch representation that specifies for each patch a target Laplacian and boundary conditions. Unifying the treatment of boundary conditions adds further artistic degrees of freedoms to the existing formulations, such as Neumann conditions on diffusion curves. To synthesize a raster image for a given output resolution, we then rasterize boundary conditions and Laplacians for the respective patches and compute the final image as solution to a Poisson problem. We evaluate the method on various test scenes containing gradient meshes and curve-based primitives. Since our mathematical formulation works with established smooth vector graphics primitives on the front-end, it is compatible with existing content creation pipelines and with established editing tools. Rather than continuing two separate research paths, we hope that a unification of the formulations will lead to new rasterization and vectorization tools in the future that utilize the strengths of both approaches.

Index Terms—vector graphics, diffusion curve, gradient mesh

I. INTRODUCTION

The field of smooth vector graphics describes image content based on geometric primitives, such as curves or meshes. This is in contrast to raster graphics, which store the color of each pixel explicitly. Smooth vector graphics are a popular tool for the modeling of scale-independent content, such as icons, schematic illustrations, or web content [20], since images can be synthesized on any output resolution. Basic vector graphics primitives are standardized in the SVG format, which includes the most basic shapes and color gradients. In research, two conceptually different approaches have been developed to extend the artistic control in smoothly-shaded regions. On the one hand, there are gradient meshes [27], [50], which interpolate colors from tensor product surfaces efficiently. And on the other hand, there are curve-based approaches such as diffusion curves [24], [38] and Poisson curves [19] that model the final image as solution to a diffusion problem from user-defined boundary conditions, e.g., curve colors. Both approaches remained incompatible, and research on editing, rasterization, and vectorization, has continued separately on the two research threads [55]. With this paper,

Xingze Tian and Tobias Günther are with the Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany.

we propose a mathematical formulation that allows combining gradient meshes and curve-based approaches consistently for the first time. Fig. 1 gives an example, where gradient meshes are combined with diffusion curves and Poisson curves. To maximize the compatibility with existing methods we take a set of gradient meshes, diffusion curves, and Poisson curves as input, which are well-known primitives that the user interacts with on the front-end. To unify the modeling, we convert those representations first into an intermediate patch representation that specifies for each patch a target Laplacian function and boundary conditions. This intermediate representation is not visible on the user interface and is only used internally to assemble the Poisson problems. We propose to model the pixel color within gradient meshes as solution to a partial differential equation (PDE) in order to match their mathematical treatment with that of curve-based methods. In the conversion process to our patch representation, geometric intersections of gradient meshes and curves get resolved automatically to yield a non-overlapping domain decomposition. In our new formulation, additional types of boundary conditions are added to the existing primitives, which extends the artistic freedom. For example, Neumann conditions can be specified on diffusion curves, as in Bang et al. [2]. To synthesize a raster image for a given output resolution, we rasterize boundary conditions and Laplacians for the respective patches and compute the final image as solution to a Poisson problem. We evaluate the method on various test scenes containing gradient meshes and curve-based primitives. In summary, our contributions are:

- We propose a unified mathematical formulation of gradient meshes and curve-based methods as a Poisson problem.
- We unify the treatment of boundary conditions, which adds (optional) degrees of freedom for artistic control.
- We introduce an automatic conversion algorithm that divides the input primitives into non-overlapping patches.
- We describe a rasterization algorithm for the novel patch formulation that synthesizes images for a given resolution.
- We provide an open source implementation that reads the scenes, constructs the patches, and renders them.
- We compare the unified scenes with vectorizations using gradient meshes only, and using diffusion curves only.

For the first time, our approach enables the unified mathematical treatment of gradient meshes and curve-based methods, which have been developed independently for over a decade. Since we take existing gradient mesh and curve formulations as input, our approach achieves high compatibility with existing

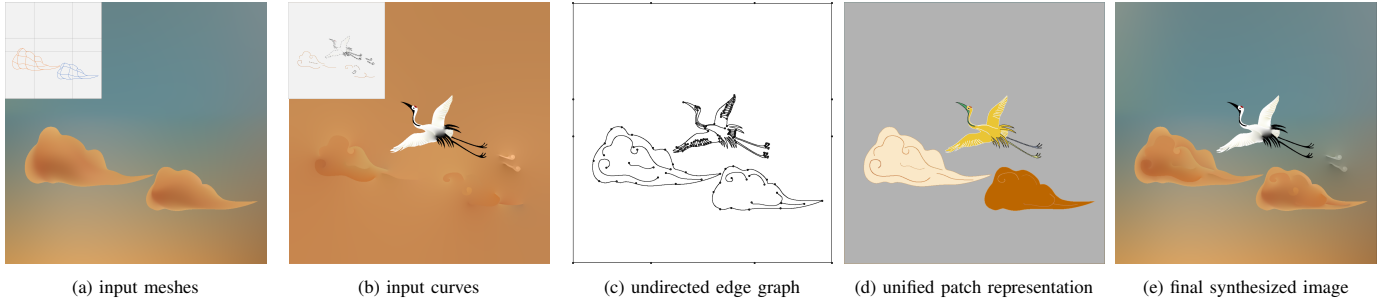| (a) input meshes | (b) input curves | (c) undirected edge graph | (d) unified patch representation | (e) final synthesized image |

Fig. 1. In this paper, we present an algorithm that unifies the image synthesis of smooth vector graphics containing, gradient meshes, diffusion curves, and Poisson curves in the same scene. Given a set of gradient meshes (a) and diffusion and Poisson curves (b), we first resolve the geometric intersections by forming an undirected edge graph (c), from which a unified patch representation is constructed that divides the domain into separate regions with well-defined Dirichlet and/or Neumann boundary conditions (d). Lastly, the image can be computed with an off-the-shelf Poisson solver (e).

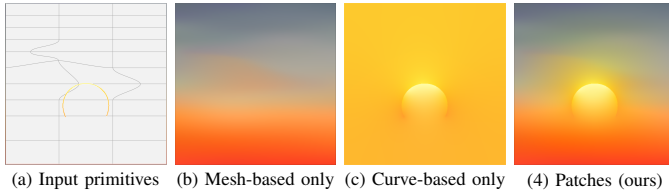| (a) Input primitives | (b) Mesh-based only | (c) Curve-based only | (4) Patches (ours) |

Fig. 2. This figure shows the benefits of combining gradient meshes with diffusion curves. Gradient meshes are useful for controlling large multi-hue color gradients (the sky), while curve-based methods can add details (the Sun). From left to right, we show the input primitives (gradient meshes and diffusion curves) (a), the result of only rasterizing the gradient meshes (b), the result of only diffusing diffusion curves (c), and lastly our combined representation that contains both (d). Note that the sun diffuses into the sky, mimicking scattering.

content creation pipelines. Further, the final algorithm leads to a standard Poisson problem, for which highly optimized numerical solvers readily exist. The benefits of combining gradient meshes and curve-based formulations in the same scene are shown in Fig. 2. While curve-based approaches are useful to create intricate details, it is easier to create larger color gradients with mesh-based approaches [55]. By combining both methods, we are able to generate expressive results that are difficult to obtain with either representation on its own.

In the following section, we first introduce into the necessary background on curve modeling before covering related work on gradient meshes and curve-based formulations. Afterwards, we introduce our unified mathematical description, which is followed by our introduction of the automatic conversion algorithm from standard primitives into our patch formulation. Then, we describe the image synthesis process, user interactions, and lastly we evaluate the approach on numerous scenes, containing both gradient meshes and curve representations.

II. BACKGROUND

In this section, we formally introduce the notation used to describe mesh-based and curve-based vector graphics primitives. For a detailed coverage, we refer to a recent survey [55].

A. Bézier Curves

In this paper, we denote Bézier curves of degree n as parametric curves $\gamma(t) : [0, 1] \to \mathbb{R}^m$ in m-dimensional space:

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

where $B_i^n(t)$ are the Bernstein basis functions, and $\mathbf{b}_i \in \mathbb{R}^m$ are the Bézier control points. The curve tangent is denoted by $\dot{\gamma}(t) = \frac{d\gamma(t)}{dt}$. A cubic Bézier curve ($n = 3$) in a two-dimensional plane ($m = 2$) has four control points $\mathbf{b}_0$, $\mathbf{b}_1$, $\mathbf{b}_2$, and $\mathbf{b}_3$, and has by convention a right-oriented curve normal $\gamma_n(t) : [0, 1] \to \mathbb{R}^2$:

$$\gamma_n(t) = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \frac{\dot{\gamma}(t)}{\|\dot{\gamma}(t)\|}. \quad (2)$$

Thus, given the control points $\mathbf{b}_0$, $\mathbf{b}_1$, $\mathbf{b}_2$, $\mathbf{b}_3$, the curve point $\gamma(t)$, tangent $\dot{\gamma}(t)$, and normal $\gamma_n(t)$ can be evaluated for each t. Multiple Bézier curves can be concatenated with suitable continuity conditions to assemble a Bézier-spline curve [12].

B. Mesh-based Methods

Mesh-based vector graphics primitives specify color gradients inside connected regions by using interpolation. Depending on the mesh topology, they are categorized into triangular, rectangular, and irregular meshes. Our work focuses on rectangular meshes.

a) Rectangular Mesh: By generalization of the Bézier curves in Eq. (1), a Bézier tensor-product surface $f(u, v) : [0, 1]^2 \to \mathbb{R}^m$ is

$$f(u, v) = \sum_{i=0}^{n_1} \sum_{j=0}^{n_2} B_i^{n_1}(u) \cdot B_j^{n_2}(v) \cdot f_{i,j}. \quad (3)$$

Later, we use the symbol $\partial_u f_{i,j} := \frac{\partial f(u,v)}{\partial u}\big|_{u=u_i, v=v_j}$ to abbreviate the evaluation of a derivative at a discrete location. Each tensor-product surface is bound by four Bézier curves $f(u, 0)$, $f(u, 1)$, $f(0, v)$ and $f(1, v)$ with $u, v \in [0, 1]$.

Price and Barret [41] used bi-cubic surfaces, which have a degree of $n_1 = n_2 = 3$. The direct editing of its 16 control points $f_{0,0}, \ldots, f_{3,3}$ is cumbersome. Fortunately, a bi-cubic tensor-product surface can be expressed as a Ferguson patch, cf. Sun et al. [50]:

$$f(u, v) = \mathbf{t}(u)^{\mathrm{T}} \cdot \mathbf{C}^{\mathrm{T}} \cdot \mathbf{Q} \cdot \mathbf{C} \cdot \mathbf{t}(v) \quad (4)$$

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

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

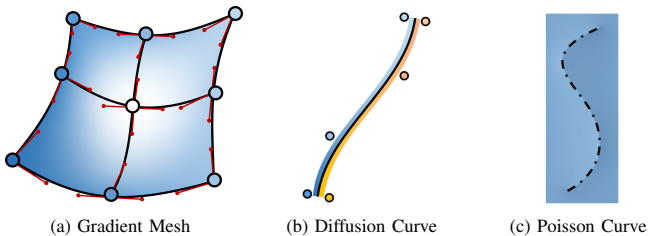(a) Gradient Mesh   (b) Diffusion Curve   (c) Poisson Curve

Fig. 3. Overview of the three mesh and curve primitives that are used in our framework. Gradient meshes (a) are assembled from Ferguson patches, which contain smooth color gradients and are controlled by colors at the corners and tangent handles. Diffusion curves (b) define a color on the left and right side, typically by piecewise linear interpolation among color stops. Poisson curves (c) allow for the modeling of cusps and highlights by locally adjusting the Laplacian of the underlying color field.

To simplify the modeling, the mixed partial derivatives are commonly set to zero, i.e., $\partial_u \partial_v f_{i,j} = 0$, and thus, a function $f(u,v)$ can be controlled by the user by specifying the values $f_{i,j}$ and the tangent handles $\partial_u f_{i,j}$ and $\partial_v f_{i,j}$ at the four corners $i, j \in \{0, 3\}$, see Fig. 3a. A *gradient mesh* consists of multiple Ferguson patches that are stitched together with suitable continuity condition. The functions $f(u,v)$ that are modeled by the gradient mesh are the position $\mathbf{x}(u,v)$ and the color $\mathbf{c}(u,v)$. Thus, the degrees of freedom are $\mathbf{x}_{i,j}$, $\mathbf{c}_{i,j}$ and the tangent handles $\partial_u \mathbf{x}_{i,j}$, $\partial_u \mathbf{c}_{i,j}$ and $\partial_v \mathbf{x}_{i,j}$, $\partial_v \mathbf{c}_{i,j}$ at the four corners $(i,j) \in \{(0,0), (0,3), (3,0), (3,3)\}$. To model holes in surfaces, Lai et al. [27] proposed to split gradient meshes along discontinuities. Barendrecht et al. [3] enabled local refinement of cubic rectangular patches. Baksteen et al. [1] fitted mesh colors [66] for colorization of gradient meshes. In line with many other papers [56], [58], [61], [62], we likewise base our description of gradient meshes on Sun et al. [50], see Eq. (4).

b) Triangular and Irregular Meshes: As an alternative to rectangular meshes, triangle-based patches have been explored, where the patch boundaries are straight [28] or curved edges [57], [59], [60]. Control over the continuity across adjacent patches is desirable [31], [68]. To this end, quadratic triangle configuration B-splines (TCBs) [7], [45] have been used by Zhu et al. [69] to model color variations while staying C^1 continuous. An example for irregular patches are the polygonal patches of Swaminarayan and Prasad [53], who used constant colors in the interior. With the introduction of Bézigons [63] closed regions have been bounded by Bézier curves. For color interpolation in the interior, several options are available [17], [18], including cubic mean value coordinates [29]. Since the methods above use differentiable interpolants, they are compatible with our approach.

C. Curve-based Methods

With curve-based methods, color gradients are the result of a diffusion process from colors given at the domain boundaries.

a) Diffusion Curves: Originally, diffusion curves [38] are Bézier curves $\gamma(t)$ with a piecewise linear color on the left and right side, i.e., $\mathbf{c}_l(t)$ and $\mathbf{c}_r(t)$. The colors serve as Dirichlet boundary conditions in a diffusion process that produces smooth color gradients, see Fig. 3b. The reformulation of Jeschke et al. [25] treats diffusion curves as domain boundaries $\partial\Omega$ of a smoothly shaded region Ω. The color field of the shaded region $\mathbf{c}(\mathbf{x})$ is as smooth as possible and meets the boundary color $\mathbf{g}(\mathbf{x})$, as specified by the colors on the left and right hand side of the diffusion curve.

$$\mathbf{c}(\mathbf{x}) = \begin{cases} \Delta\mathbf{c}(\mathbf{x}) = 0, & \mathbf{x} \in \Omega\backslash\partial\Omega \\ \mathbf{c}(\mathbf{x}) = \mathbf{g}(\mathbf{x}), & \mathbf{x} \in \partial\Omega \end{cases} \quad (5)$$

Follow-up research on rendering [2], [39], [51], vectorization [33], [67] and editing [26], [32] utilized this formulation. To provide more control over the color gradients, Bezerra et al. [4] controlled the strength and direction of the diffusion. Alternatively, Sun et al. [52] introduced textures in diffusion curves. Several renderers have been proposed by now that are able to rasterize diffusion curves, including ray tracing approaches [5], [40], Monte Carlo methods [9], [30], [34], [36], [43], [44], [48], [49], and boundary element methods [2], [8], [21]. Lately, Monte Carlo methods have been phrased in a differentiable manner, which is useful for inverse problems such as vectorizations [35], [64], [65].

b) Poisson Curves: By introducing *Poisson vector graphics*, Hou et al. [19] edited the gradient in the domain by introducing a desired (piecewise-constant) Laplacian $\mathbf{f}(\mathbf{x})$:

$$\mathbf{c}(\mathbf{x}) = \begin{cases} \Delta\mathbf{c}(\mathbf{x}) = \mathbf{f}(\mathbf{x}), & \mathbf{x} \in \Omega\backslash\partial\Omega \\ \mathbf{c}(\mathbf{x}) = \mathbf{g}(\mathbf{x}), & \mathbf{x} \in \partial\Omega \end{cases} \quad (6)$$

Thus, rather than looking for the smoothest possible color gradient (via $\mathbf{f}(\mathbf{x}) = 0$), bumps and discontinuities can be introduced in the color gradient, which are controlled locally by the user by drawing curve and region primitives, see Fig. 3c. Non-constant Laplacians in Poisson regions have been described by raster images, and were utilized for color transfer between images [14] and for image vectorization [15]. Alternatively, Finch et al. [13] proposed to constrain high-order derivatives, seeking solutions that are harmonic in their Laplacian domain, i.e., $\Delta^2\mathbf{c}(\mathbf{x}) = \mathbf{0}$. With this formulation, extrapolation may create new extrema at unconstrained positions [6], [13], which requires manual adjustment of control points [13] or a non-linear optimization [23]. Further, the diffused color range is tied to the scene geometry, which may lead to oversaturation [24] when scaling objects. In our work, we concentrate on the Poisson formulation in Eq. (6), which includes the diffusion curves in Eq. (5) as special case. Poisson vector graphics [19] have led to a new line of work on the control over color gradients within smooth vector graphics. In this paper, we seek to utilize the well-established gradient meshes [50], which so far have not been incorporated into this framework.

III. METHOD OVERVIEW

Our goal is to synthesize raster images for vector graphics scenes that contain mesh-based and curve-based primitives at the same time. Usually, mesh-based and curve-based methods are rasterized in fundamentally different ways. While mesh-based approaches use *interpolation*, curve-based methods solve a *Poisson problem*. To model both approaches consistently, we describe the mesh-based approaches as solution to a Poisson problem that is identical to interpolation in the absence of any curve-based primitives. A user of our system may decide

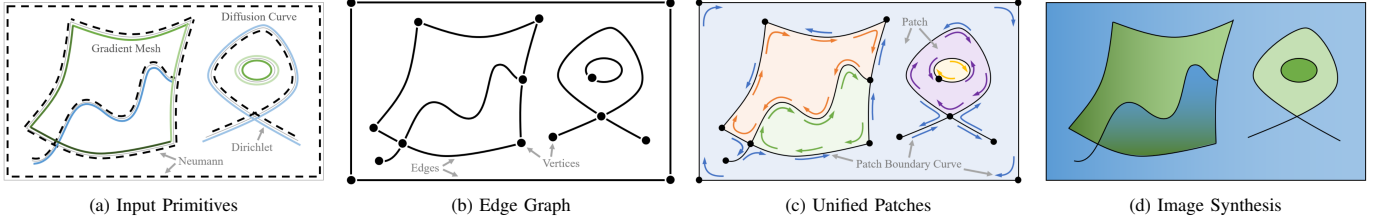| (a) Input Primitives | (b) Edge Graph | (c) Unified Patches | (d) Image Synthesis |

Fig. 4. Schematic overview of our generalized smooth vector graphics pipeline. (a) Input to our method is a collection of (extended) gradient meshes, (extended) diffusion curves, and Poisson curves. (b) The input primitives are converted and inserted into an edge graph that resolves the intersection of input primitives. (c) From the edge graph a unified patch data structure is formed that defines patches as closed regions with boundary conditions according to the underlying input primitives. (d) The unified patch data structure specifies the ingredients for a Poisson problem that is solved to synthesize the final image.

to draw curve-based primitives that intersect each other or intersect with mesh-based primitives. To formulate the Poisson problem, the scene is decomposed into non-overlapping regions for which boundary conditions are defined and for which a Laplacian is determined that controls the color gradient in the interior. We formulate the rasterization of smooth vector graphics as a pipeline consisting of multiple steps, see Fig. 4.

1) *Input Primitives.* Input to our method are standard smooth vector graphics primitives, such as gradient meshes, diffusion curves, and Poisson curves, which are created and edited by the user. We extend gradient meshes and diffusion curves by allowing the user to specify Dirichlet conditions and homogeneous Neumann conditions on either side.

2) *Edge Graph.* Both the extended gradient meshes and the extended diffusion curves are inserted into an undirected edge graph data structure [16], which records and resolves all intersections of boundary curves.

3) *Unified Patches.* By traversing the edge graph, we define closed, non-overlapping regions that have consistent boundary conditions. The interior of each patch is equipped with a Laplacian function that is either homogeneous, or sampled from a gradient mesh and/or multiple Poisson curves.

4) *Image Synthesis.* Given the patches with their boundary conditions, we solve the Poisson problem for each patch with an off-the-shelf Poisson solver.

A C++ reference implementation of our method can be found at [54]. In the following, the four steps are explained in more detail.

IV. INPUT PRIMITIVES

Input to our approach is a collection of extended gradient meshes $\mathcal{G}$, a collection of extended diffusion curves $\mathcal{D}$ and a collection of Poisson curves $\mathcal{P}$. Those three types of input primitives are drawn by the user and can be edited and controlled by manipulation of control points, tangent handles, and color stops, identical to the editing processes of conventional diffusion curves, gradient meshes, and Poisson curves [55]. In the following, we explain how the primitives are defined and how they are extended in our framework.

A. Definition

First, we introduce an *input boundary curve*, which is a triplet $(\mathbf{x}(t), \mathcal{B}_l, \mathcal{B}_r)$, as shown in Fig. 5. It consists of:
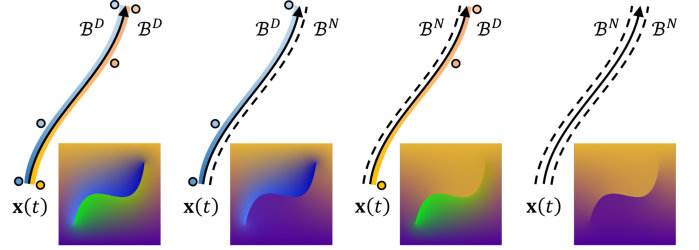

Fig. 5. Input boundary curves consist of a spatial curve $\mathbf{x}(t)$ and boundary conditions on the left and right side, which may either be a Dirichlet boundary condition $\mathcal{B}^D$ (shown as colored line with control points) or a homogeneous Neumann boundary condition $\mathcal{B}^N$ (shown as a dashed line).

- the spatial curve $\mathbf{x}(t)$,
- a left boundary condition $\mathcal{B}_l$,
- and a right boundary condition $\mathcal{B}_r$.

The spatial location of the input boundary curve is given by a cubic Bézier spline $\mathbf{x}(t) : \mathbb{R} \to \mathcal{U}$, where $\mathcal{U} \subset \mathbb{R}^2$ denotes the image domain. The curve parameter range $t \in [t_0, t_1]$ defines the *orientation* of the curve. We require $t_0 < t_1$, where the curve begins at t_0 and ends at t_1. This gives rise to the notion of a *left* side and a *right* side of the curve. Each input boundary curve will later define a boundary condition in the field $\mathbf{c}(\mathbf{x}) : \mathcal{U} \to \mathcal{C}$, where $\mathcal{C} \subset \mathbb{R}^3$ is the color space. The boundary condition is either a Dirichlet boundary condition $\mathcal{B}^D$ with color $\mathbf{c}(t)$ or a homogeneous Neumann boundary condition $\mathcal{B}^N$ with boundary normal $\mathbf{x}_n(t)$, cf. Eq. (2):

$$\mathcal{B}^D = \left\{ \mathbf{c}(\mathbf{x})_{|\mathbf{x}=\mathbf{x}(t)} = \mathbf{c}(t) \qquad \text{Dirichlet with } \mathbf{c}(t) \quad (7)\right.$$

$$\mathcal{B}^N = \left\{ \frac{\partial \mathbf{c}(\mathbf{x})}{\partial \mathbf{x}}_{|\mathbf{x}=\mathbf{x}(t)} \cdot \mathbf{x}_n(t) = 0 \quad \text{homogen. Neumann} \quad (8)\right.$$

For notational convenience, we name the *left* boundary condition $\mathcal{B}_l \in \{\mathcal{B}^D, \mathcal{B}^N\}$ and the *right* boundary condition $\mathcal{B}_r \in \{\mathcal{B}^D, \mathcal{B}^N\}$. Adding inhomogeneous Neumann conditions would be a straightforward extension that would, however, add further complexity to the user interface.

a) Extended Gradient Mesh: A gradient mesh that consists of a single Ferguson patch as in Eq. (4), with position $\mathbf{x}(u, v)$ and color $\mathbf{c}(u, v)$, gives rise to four clock-wise oriented boundary curves with positions $\mathbf{x}(t) \in \{\mathbf{x}(1, 1-t), \mathbf{x}(1-t, 0), \mathbf{x}(0, t), \mathbf{x}(t, 1)\}$. Using the clock-wise orientation of the boundary curves, we set the *right* boundary conditions (i.e., the inside of the gradient mesh) to a Dirichlet condition $\mathcal{B}_r = \mathcal{B}^D$ by using the corresponding colors $\mathbf{c}(t) \in \{\mathbf{c}(1, 1-t), \mathbf{c}(1-$
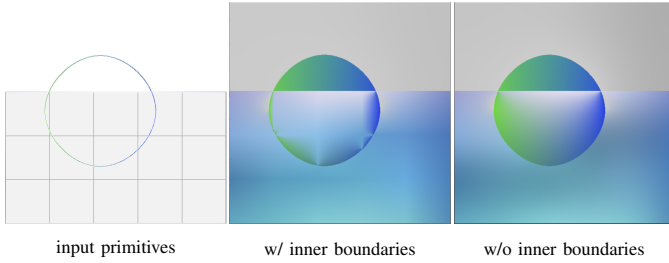
Fig. 6. When constructing input boundary curves, the inner edges of gradient meshes are not added, since they would create diffusion barriers. The inner edges are not needed, since the color gradients are fully determined by the gradient mesh Laplacian, which is considered in the Poisson problem.
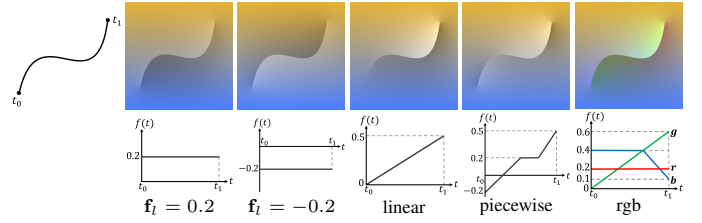

Fig. 7. Overview of supported target Laplacians along Poisson curves, including constant values, linear slopes, piecewise linear functions, as well as the specification of a target Laplacian per color channel.

$t, 0), \mathbf{c}(0, t), \mathbf{c}(t, 1)\}$. Our *extension* compared to a regular gradient mesh is that we add the definition of a boundary condition on the *outside* of the gradient mesh, as well. The *left* boundary condition (i.e., the outside of the gradient mesh) is by default set to a homogeneous Neumann condition $\mathcal{B}_l = \mathcal{B}^N$. The user may change this to a Dirichlet condition if desired. For a gradient mesh that consists of multiple connected Ferguson patches, input boundary curves are only generated for the outermost Ferguson patch boundaries, which coincide with the boundary of the gradient mesh. Adding the interior Ferguson patch boundaries is neither necessary nor useful, since the smooth color gradient in the interior of the gradient mesh will be fully defined by the Laplacian of its Ferguson patches. Fig. 6 demonstrates this at the example of a closed diffusion curve (circle), which intersects a gradient mesh (blue). If interior patch boundaries were added, it would introduce unwanted diffusion barriers that are counter-intuitive.

b) Extended Diffusion Curve: A diffusion curve $\gamma(t)$: $\mathbb{R} \to \mathcal{U}$ with left color $\mathbf{c}_l(t) : \mathbb{R} \to \mathcal{C}$ and right color $\mathbf{c}_r(t)$: $\mathbb{R} \to \mathcal{C}$ translates to an input boundary curve by setting $\mathbf{x}(t) = \gamma(t)$ and by applying Dirichlet conditions on the left and right with $\mathcal{B}_l = \mathcal{B}^D$ and $\mathcal{B}_r = \mathcal{B}^D$. Same as Bang et al. [2], we *extend* diffusion curves by allowing to replace the Dirichlet conditions with homogeneous Neumann conditions.

c) Poisson Curve: Analogous to diffusion curves, we define a Poisson curve as a spatial curve $\mathbf{p}(t) : \mathbb{R} \to \mathcal{U}$ with a target Laplacian function on the left $\mathbf{f}_l(t) : \mathbb{R} \to \mathbb{R}^3$ and on the right $\mathbf{f}_r(t) : \mathbb{R} \to \mathbb{R}^3$. Upon rasterization, the sides cover the pixels $\mathcal{F}_l \subseteq \mathcal{U}$ and $\mathcal{F}_r \subseteq \mathcal{U}$, respectively, with $\mathbf{f}_l(\mathbf{x}) : \mathcal{F}_l \to \mathbb{R}^3$ and $\mathbf{f}_r(\mathbf{x}) : \mathcal{F}_r \to \mathbb{R}^3$ being the target Laplacian at spatial coordinates $\mathbf{x} \in \mathcal{D}$. To obtain shading consistency, Hou et al. [19] required the functions $\mathbf{f}_l(t)$ and $\mathbf{f}_r(t)$ to sum to zero along the curve, i.e., $\forall \mathbf{x} : \mathbf{f}_l(t) = -\mathbf{f}_r(t)$. For notational convenience, we collectively refer to a sampling from the left or right side of the target Laplacian as $\mathbf{f}_{\mathbf{p}(t)}(\mathbf{x}) : \mathcal{D} \to \mathbb{R}^3$:

$$\mathbf{f}_{\mathbf{p}(t)}(\mathbf{x}) = \begin{cases} \mathbf{f}_l(\mathbf{x}) \cdot \gamma_l(\mathbf{x}) & \mathbf{x} \in \mathcal{F}_l \\ \mathbf{f}_r(\mathbf{x}) \cdot \gamma_r(\mathbf{x}) & \mathbf{x} \in \mathcal{F}_r \\ \mathbf{0} & \text{else} \end{cases} \quad (9)$$

where $\gamma_l(\mathbf{x})$ is the number of adjacent pixels that belong to $\mathcal{F}_r$ and $\gamma_r(\mathbf{x})$ is the number of adjacent pixels in $\mathcal{F}_l$, which are used for normalization, cf. Hou et al. [19]. An overview of the supported target Laplacians $\mathbf{f}_l(t)$ is shown in Fig. 7.

V. EDGE GRAPH

Diffusion curves and gradient meshes are placed and edited by the user. In order to formulate the image synthesis as a diffusion process with well-defined boundary conditions, it is necessary to subdivide the image into closed, non-overlapping regions. When one or multiple possibly-closed or nested diffusion curves intersect with a gradient mesh, it is necessary to consistently determine a Laplacian for the interior of each separate region. In the following, we utilize a planar map data structure [16] that we use to resolve intersections. The subsequent section will use this data structure to define connected regions and to determine a Laplacian for each region.

A. Definition

We introduce the undirected edge graph $(\mathcal{V}, \mathcal{E})$, which consists of vertices $\mathbf{v} \in \mathcal{V}$ and edges $\mathbf{e} \in \mathcal{E}$. The set of vertices $\mathcal{V}$ contains all end points of the input boundary curves, as well as their intersections. The set of edges $\mathcal{E}$ expresses how the vertices are connected via the input boundary curves. Thereby, each edge $\mathbf{e} \in \mathcal{E}$ stores a reference to the underlying input boundary curve that it was created for, such that it can access the orientation of the input boundary curve and its boundary conditions on the left side and right side.

B. Construction

An example of the edge graph construction is given in Fig. 8. We begin with an empty graph, i.e., $\mathcal{V} = \varnothing$, $\mathcal{E} = \varnothing$. For each input boundary curve $(\mathbf{x}(t), \mathcal{B}_l, \mathcal{B}_r) \in \mathcal{G} \cup \mathcal{D}$ from all extended gradient meshes $\mathcal{G}$ and extended diffusion curves $\mathcal{D}$, we take the following steps:

1) Allocate two vertices $\mathbf{v}_i = \mathbf{x}(t_0)$, $\mathbf{v}_j = \mathbf{x}(t_1)$ and connect them by a curved edge $\mathbf{e}_{ij}$ using the curve, i.e., $\mathbf{e}_{ij} = \mathbf{x}(t)$.
2) If there is any other existing point $\mathbf{v}_k \in \mathcal{V}$ with $\|\mathbf{v}_i - \mathbf{v}_k\|^2 < \tau$ or $\|\mathbf{v}_j - \mathbf{v}_k\|^2 < \tau$, then merge nearby points.
3) If there is any other edge $\mathbf{e}_{kl} \in \mathcal{E}$ with $\mathbf{e}_{ij} \cap \mathbf{e}_{kl} \neq \varnothing$, then find the intersection and split the connected edges.

An inserted input boundary curve may have multiple intersections with other curves or with itself. The intersections are resolved by inserting vertices until no further intersections can be found. With this, the graph is brought back to a consistent state and the next input boundary curve can be processed. The parameter τ allows for closing gaps that would otherwise cause unwanted color leaking. Fig. 9 illustrates the snapping of an edge endpoint onto an existing curve. For this, the closest point
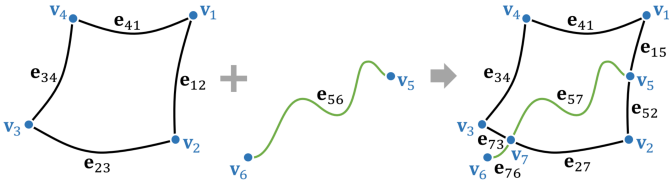
Fig. 8. Given a graph with vertices $\mathcal{V} = \{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4\}$ and edges $\mathcal{E} = \{\mathbf{e}_{12}, \mathbf{e}_{23}, \mathbf{e}_{34}, \mathbf{e}_{41}\}$. Inserting a new edge $\mathbf{e}_{56}$ with end points $\mathbf{v}_5$, $\mathbf{v}_6$ leads to a new node $\mathbf{v}_7$ at the edge intersection and a splitting of edges.

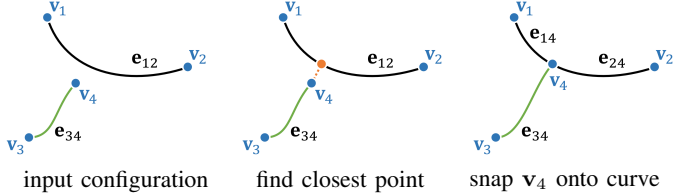input configuration   find closest point   snap $\mathbf{v}_4$ onto curve

Fig. 9. When the endpoint of the next edge ($\mathbf{e}_{34}$, green) is close to an already existing edge ($\mathbf{e}_{12}$, black), then we snap the endpoint of the new edge onto the existing edge.

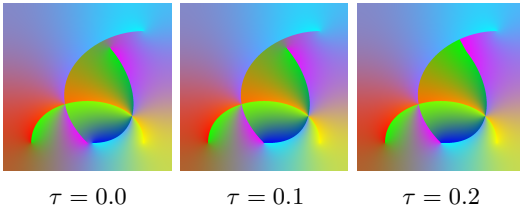$\tau = 0.0$ $\tau = 0.1$ $\tau = 0.2$

Fig. 10. Here, the snapping threshold τ is varied. On the left, the result without clean-up is shown. In the middle, the two endpoints at the bottom merged. On the right, the gap at the top was closed. This simple test scene is defined over the domain $[-0.5, 2.5]^2$.

on the existing edge is determined, which is either one of the end points or a point on the edge. In case of the latter, the existing edge is subdivided, and the vertex of the new edge is snapped onto the curve. Fig. 10 shows renderings for different thresholds τ for a simple test scene.

C. Implementation Detail

During the graph construction process, input boundary curve intersections need to be calculated. To accelerate the intersection tests, we internally discretize the input boundary curves into piece-wise linear polylines in a top-down construction that is based on the Ramer-Douglas-Peucker algorithm [11], [42]. However, instead of starting from a polyline, we start directly with the cubic Bézier curve. In Fig. 11(a), we define a line that connects the first and last vertex of the curve. In (b), we determine the curve point, which has the largest vertical distance from the line. This is done recursively using Bézier clipping [46]. In (c), we insert a new vertex at that point and recursively proceed on the segments left and right. In (d), we see the result after termination, which happens when the distance of the farthest point falls below an error threshold ϵ. In (e), we can see that it is not sufficient to only measure the vertical distance to the line (blue). Instead, we also compute the (signed) horizontal distance from the start and end point (green). For subdivision, the maximum of the three distances
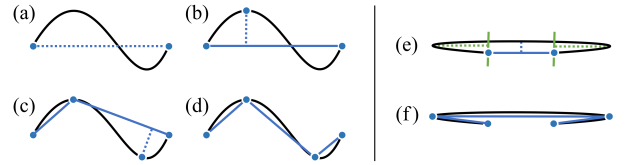

Fig. 11. Illustration of the curve discretization method. In (a)-(d), the recursive subdivision is shown, while (e)-(f) illustrates at an example why measuring vertical distances alone is not sufficient.
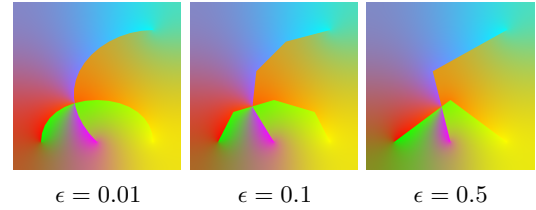

$\epsilon = 0.01$ $\epsilon = 0.1$ $\epsilon = 0.5$

Fig. 12. Here, the results for different curve discretizations are shown, which are obtained when choosing different error thresholds ϵ. This simple test scene is defined over the domain $[-0.5, 2.5]^2$.

is computed. In (g), the result is shown, which successfully approximates the curve. Examples of different discretizations are shown in Fig. 12 for a simple test scene. Finding curve intersections then reduces to a line-line intersection test. When a new curve is added by the user or when the control points of an existing curve are moved, the curve is re-discretized.

VI. UNIFIED PATCHES

Using the edge graph, we subdivide the image domain into non-overlapping patches with well-defined boundary conditions and a Laplacian in the interior. The image synthesis can then be performed for each patch independently. In the following, we explain the patch generation process.

A. Definition

Formally, we denote the image domain as $\mathcal{U} \subset \mathbb{R}^2$. Our goal is to split the image domain into connected regions $\Omega_i \in \Omega$:

$$\bigcup_{\Omega \in \Omega} \Omega = \mathcal{U} \qquad (10)$$

with the requirement that for every $\Omega_i, \Omega_j \in \Omega$ with $\Omega_i \neq \Omega_j$, the intersection of the regions $\Omega_i \cap \Omega_j \subset \mathcal{E}$ is either empty, a 0-simplex (point), or a 1-simplex (line) in the set of graph edges $\mathcal{E}$. In other words, the boundaries of regions are sections of the input boundary curves. For each region Ω, we denote the desired color field as $c(\mathbf{x}) : \Omega \to \mathcal{C}$ for $\mathbf{x} = (x, y) \in \Omega$, and its boundary as $\partial\Omega \subset \mathcal{E}$, which might comprise just one closed loop or multiple closed loops if there are holes inside a region. We refer to the boundary curves of a patch as *patch boundary curves*, which are clockwise-oriented curves with a boundary condition $\mathcal{B}$ on only the *right* side. Further, each patch Ω is equipped with a target Laplacian function $\mathbf{f}(\mathbf{x}) : \Omega \to \mathbb{R}^3$ that controls the smooth color gradient inside the patch.

B. Construction

In the following, we elaborate on how patch boundary curves and patches are created from the edge graph, and how the patch Laplacian function $\mathbf{f}(\mathbf{x})$ is defined for each patch.
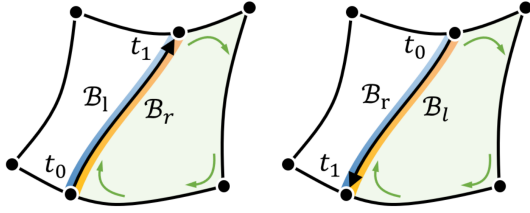
Fig. 13. Each edge is associated with an underlying input boundary curve, which has an orientation (based on the parameter range $[t_0, t_1]$) and a left and right boundary condition ($\mathcal{B}_l$, $\mathcal{B}_r$). When traversing the edge during construction of a patch (here green), the left or right boundary condition is chosen based on the orientation of the underlying input boundary curve.
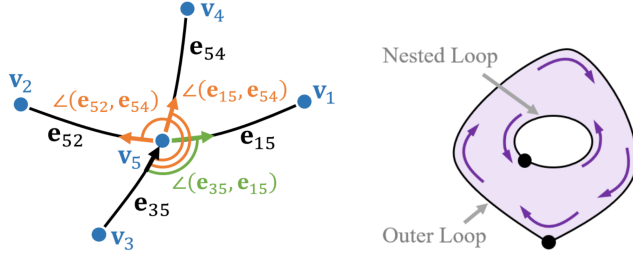

Fig. 14. Left: When traversing from edge $\mathbf{e}_{35}$ to vertex $\mathbf{v}_5$, there are three directions, in which the traversal could continue ($\mathbf{e}_{15}$, $\mathbf{e}_{54}$, $\mathbf{e}_{52}$). Following edges, we always turn *right*, i.e., we select the edge with smallest angle (green). Right: When a loop is completely nested inside another loop, the patch boundaries created from the inner loop are added to the patch.
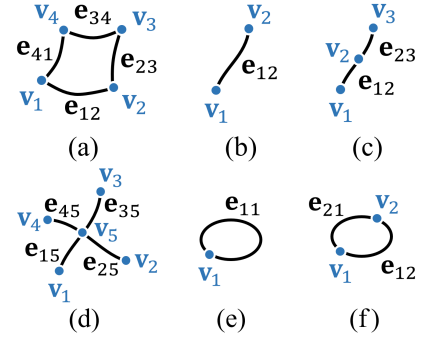

Fig. 15. Exemplary edge graphs to demonstrate the traversal order. Starting from an arbitrary edge in an arbitrary direction, the traversal continues always taking right turns until the loop is closed.

we list the corresponding traversal sequences.

(a) Traverse outer edge of a closed domain: $\mathbf{v}_1 \rightarrow \mathbf{e}_{12} \rightarrow \mathbf{v}_2 \rightarrow \mathbf{e}_{23} \rightarrow \mathbf{v}_3 \rightarrow \mathbf{e}_{34} \rightarrow \mathbf{v}_4 \rightarrow \mathbf{e}_{41} \rightarrow \mathbf{v}_1$

(b) Traverse single edge: $\mathbf{v}_1 \rightarrow \mathbf{e}_{12} \rightarrow \mathbf{v}_2 \rightarrow \mathbf{e}_{12} \rightarrow \mathbf{v}_1$

(c) Traverse sequence of edges: $\mathbf{v}_1 \rightarrow \mathbf{e}_{12} \rightarrow \mathbf{v}_2 \rightarrow \mathbf{e}_{23} \rightarrow \mathbf{v}_3 \rightarrow \mathbf{e}_{23} \rightarrow \mathbf{v}_2 \rightarrow \mathbf{e}_{12} \rightarrow \mathbf{v}_1$

(d) Traverse crossing of edges: $\mathbf{v}_1 \rightarrow \mathbf{e}_{15} \rightarrow \mathbf{v}_5 \rightarrow \mathbf{e}_{25} \rightarrow \mathbf{v}_2 \rightarrow \mathbf{e}_{25} \rightarrow \mathbf{v}_5 \rightarrow \mathbf{e}_{35} \rightarrow \mathbf{v}_3 \rightarrow \mathbf{e}_{35} \rightarrow \mathbf{v}_5 \rightarrow \mathbf{e}_{45} \rightarrow \mathbf{v}_4 \rightarrow \mathbf{e}_{45} \rightarrow \mathbf{v}_5 \rightarrow \mathbf{e}_{15} \rightarrow \mathbf{v}_1$

(e) Traverse closed loop with one vertex: $\mathbf{v}_1 \rightarrow \mathbf{e}_{11} \rightarrow \mathbf{v}_1$

(f) Traverse closed loop with multiple vertices: $\mathbf{v}_1 \rightarrow \mathbf{e}_{12} \rightarrow \mathbf{v}_2 \rightarrow \mathbf{e}_{21} \rightarrow \mathbf{v}_1$

c) Nested patches: A nested edge loop, as shown in Fig. 14 (right), has an interior patch boundary curve sequence, which bounds the patch that is contained by the edge loop, and an exterior patch boundary curve sequence, which denotes a patch boundary of the parent patch that contains the nested edge loop. Due to the convention of always turning *right* during the edge graph traversal, the two sequences are distinguished by their turning number. For all interior curve sequences, the boundary curve tangent performs a clockwise rotation (negative turning number), while the tangent of the exterior curve sequence performs a counter-clockwise rotation (positive turning number) when traversing the edge loop. For an arclength parameterized closed edge sequence $\mathbf{e}(s) : [s_0, s_1] \rightarrow \partial\Omega$, the integer-valued turning number is calculated by integrating the signed curvature $k(s)$ along the closed patch boundary curve:

$$\text{turn}(\mathbf{e}(s)) = \frac{1}{2\pi} \int_{s_0}^{s_1} k(s) \, ds, \quad \text{with} \quad k(s) = \frac{\det(\dot{\mathbf{e}}(s), \ddot{\mathbf{e}}(s))}{\|\dot{\mathbf{e}}(s)\|^3}. \tag{11}$$

d) Patch Laplacian: The patch Laplacian $\mathbf{f}(\mathbf{x})$ defines the color gradient inside a patch, which is controlled by evaluating and adding the Laplacian $\mathbf{f}_{\mathbf{p}(t)}(\mathbf{x})$ of Poisson curves $\mathbf{p}(t) \in \mathcal{P}$ and by sampling the color derivatives of gradient meshes, cf. Section II-B.

$$\mathbf{f}(\mathbf{x}) = \underbrace{\sum_{\mathbf{p}(t) \in \mathcal{P}} \mathbf{f}_{\mathbf{p}(t)}(\mathbf{x})}_{\text{Poisson curves}} + \underbrace{\sum_{\mathbf{c}_i(\mathbf{u}) \in \mathcal{G}} \lambda_i \cdot \Delta_{\mathbf{x}} \mathbf{c}_i(\mathbf{u}(\mathbf{x}))}_{\text{Gradient meshes}}. \tag{12}$$

If a patch boundary curve of a patch is part of a gradient mesh interior, we add the gradient mesh to a list. Each Laplacian

a) Patch Boundary Curves: Each edge in the edge graph corresponds to an *input boundary curve*. Further, every edge is adjacent to two patches (except for image domain boundaries, where one side would be outside of the image). Thus, when constructing patches each edge will be visited up to two times when constructing *patch boundary curves* (once for the left side and once for the right side). The *patch boundary condition* $\mathcal{B}$ of a patch boundary curve is chosen from the underlying input boundary curve based on its orientation, as illustrated in Fig. 13, i.e., we choose from the left and right boundary conditions $\mathcal{B}_l$, $\mathcal{B}_r$ of the input boundary curve. Note that the two adjacent patches can be the *same* patch, for example, when a diffusion curve is nested inside a patch and when it is not intersecting with the outer patch boundary.

b) Patches: The valence $|\mathbf{v}_i|$ of vertex $\mathbf{v}_i \in \mathcal{V}$ in the graph, i.e., the number of edges connected to it, tells how often the vertex can be visited when traversing the graph in the search for closed patches, namely $2 \cdot |\mathbf{v}_i|$ (with the exception of image boundary vertices for which it is $2 \cdot |\mathbf{v}_i| - 2$). Starting from the first vertex and its first edge in the graph, we traverse the edges from vertex to vertex tracing out a sequence of patch boundary curves that encloses a patch. A closed patch is formed by always turning *right* when visiting a junction, i.e., a vertex. This is illustrated in Fig. 14 (left). For the required angle measurements, the curve tangents $\dot{\mathbf{x}}(t)$ are evaluated at the end vertex of the edge, i.e., at t_0 or t_1, respectively. The edge traversal terminates when a loop is found, i.e., when the traversal returns to the start vertex. By recording how often a vertex was visited, we repeat the process until all patches are inserted and no vertex is expecting another visit. Fig. 15 shows a number of exemplary edge graphs. In the following,
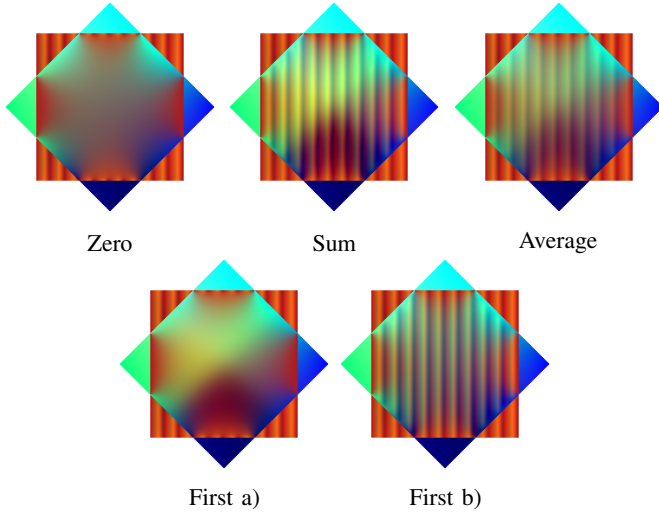
Fig. 16. Different options for handling overlapping gradient meshes. 'Zero' sets the Laplacian to zero, which gives the smoothest possible interpolation of the boundary conditions. 'Sum' sums up the Laplacians, which can lead to high values when many gradient meshes overlap. 'Average' computes the average of the Laplacians, which scales better to larger numbers of gradient meshes. 'First' only takes one of the Laplacians, for which we show both options: choosing the smooth square in a) and the striped square in b).

$\Delta_{\mathbf{x}}\mathbf{c}(\mathbf{u}(\mathbf{x}))$ can be evaluated analytically as explained in the supplemental material (including the coordinate transformation $\mathbf{u}(\mathbf{x})$ that takes image coordinates to a (u,v) coordinate in the gradient mesh). When a patch is formed from one gradient mesh, then the diffused color gradient inside the patch matches exactly the color gradient of the interpolated gradient mesh. In the event that multiple gradient meshes are overlapping at the pixel coordinate $\mathbf{x}$, we use a weighted sum of the gradient mesh Laplacians to compute the source term $\mathbf{f}(\mathbf{x})$. We offer four choices for how the weights λ_i are chosen. In the following, let n be the number of gradient meshes, i.e., $i \in \{1, \ldots, n\}$.

1) All weights are $\lambda_i = 0$, i.e., the Laplacians are ignored.
2) All weights are $\lambda_i = 1$, i.e., all Laplacians are summed.
3) All weights are $\lambda_i = \frac{1}{n}$, i.e., the Laplacian's are averaged.
4) Only one Laplacian is chosen with $\lambda_1 = 1$, and the others are set to zero $\lambda_2 = \cdots = \lambda_n = 0$.

Options 1) – 3) are order-independent, while option 4) requires the specification of an order. We assume that the gradient meshes $\mathcal{G}$ are given in a particular order by the user, i.e., we choose the gradient mesh that is sitting 'on top' of all others. For this, the user interface contains a list of all gradient meshes, which can be reordered by the user. Fig. 16 gives examples.

C. Implementation Detail

The turning number in Eq. (11) is calculated from the polyline discretization, previously described in Section V-C, by summing up the discrete turning angles. A loop is nested inside another loop if one of its vertices is contained in the other loop, since loops are by construction intersection free. To determine containment bounding boxes are used to find candidates and a detailed check is performed by using the winding angle theorem [22], [47], which is a numerically robust way of testing containment.

VII. IMAGE SYNTHESIS

In the following, we discuss how to rasterize a patch Ω into a raster image that covers the image domain $\mathcal{U}$.

A. Definition

The color field $\mathbf{c}(\mathbf{x})$ of a region Ω is determined by solving a partial differential equation, i.e., a well-known Poisson problem:

$$\Delta\mathbf{c}(\mathbf{x}) = \mathbf{f}(\mathbf{x}), \quad s.t. \ \mathcal{B} \in \partial\Omega \qquad (13)$$

with the patch Laplacian $\mathbf{f}(\mathbf{x})$ as given by Eq. (12) and subject to the patch boundary conditions $\mathcal{B}$ that arise from the patch boundary curves on $\partial\Omega$, as introduced in Eqs. (7)–(8). There are many ways how the Poisson problem could be solved, including direct solvers, iterative solvers, or Monte Carlo methods. We refer to a survey [55] for an overview of solvers in the smooth vector graphics literature and also point out several recent solvers, which appeared after the survey [2], [8], [36], [44], [48]. Since the PDE solver is orthogonal to our contributions, we utilized a simple (multi-grid) Jacobi relaxation [25]:

$$\mathbf{c}_{i,j}^{(n+1)} = \frac{\sum_{x \in \{-1,1\}} \sum_{y \in \{-1,1\}} w_{i+x,j+y} \cdot \mathbf{c}_{i+x,j+y}^{(n)} - h^2 \mathbf{f}_{i,j}}{w_{i+1,j} + w_{i-1,j} + w_{i,j+1} + w_{i,j-1}} \qquad (14)$$

which iteratively calculates the pixel color $\mathbf{c}_{ij}$ for a pixel (i,j) from its neighbors such that it meets the target Laplacian $\mathbf{f}_{ij}$ and where h is the grid spacing of pixels in the image domain $\mathcal{U}$. The weights w_{ij} are used to handle boundary conditions, which is explained in the implementation details below.

B. Implementation Detail

We implemented the Jacobi relaxation on the GPU, which requires a texture containing the target Laplacians $\mathbf{f}_{i,j}$, a texture containing the type of the pixel $k_{i,j} \in \{\text{interior}, \text{Dirichlet}, \text{Neumann}\}$, a bit mask on a staggered grid that indicates whether horizontally and vertically adjacent pixels are separated by a patch boundary curve $s_{i,j}^h, s_{i,j}^v \in \{\text{open}, \text{closed}\}$, and a texture that stores the final pixel color $\mathbf{c}_{i,j}$, see Fig. 17. If the pixel (i,j) is a:

1) *interior pixel* ($k_{i,j} = $ interior), then the weights of all adjacent pixels are set to 1, i.e., $w_{i+1,j} = w_{i-1,j} = w_{i,j+1} = w_{i,j-1} = 1$.
2) *boundary pixel with Dirichlet condition* ($k_{i,j} = $ Dirichlet), then Eq. (14) is not executed. Instead, the color of the pixel is determined by the boundary condition.
3) *boundary pixel with Neumann condition* ($k_{i,j} = $ Neumann), then the weight of the adjacent pixel $w_{i+1,j}$ is set to 1 (otherwise 0) if it is non-separated by a patch boundary curve $s_{i,j}^h = $ open (analogous for the other neighbor pixels).

We identify if a pixel is a boundary pixel in a pre-process by determining the pixels that belong to the patch using the winding angle theorem [22], and flag pixel (i,j) as boundary if one of its neighbors $(i+1,j), (i-1,j), (i,j+1), (i,j-1)$ is not in the patch, or if a ray cast between adjacent pixels intersects a patch boundary curve. We determine the closest patch boundary curve and set the pixel type $k_{i,j}$ and the openness $s_n, s_e, s_s,$
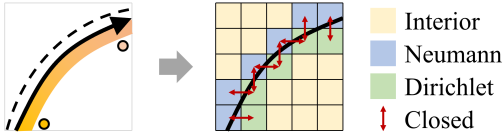
Fig. 17. Left: Given is a diffusion curve with Neumann condition ($\mathcal{B}_l$) and Dirichlet condition ($\mathcal{B}_r$). Right: depiction of the pre-computed masks. We determine for each pixel, whether it is in the interior, at a Neumann boundary, or a Dirichlet boundary, which is stored in $k_{i,j}$. Further, we identify if adjacent pixels are separated (horizontally or vertically) by a patch boundary curve, and store this as binary flag in staggered grids $s_{i,j}^h$, $s_{i,j}^v$.

s_w accordingly. If the pixel has a Dirichlet condition, then we set $\mathbf{c}_{i,j}$ according to the closest hit location on the patch boundary curve. If the Dirichlet condition originated from an input boundary curve that belonged to a gradient mesh, then we interpolate the pixel color in the gradient mesh at the location of the pixel center (i, j), rather than taking the color on its boundary curve. The texture with the target Laplacian $\mathbf{f}_{i,j}$ is set by evaluating the gradient mesh Laplacian $\Delta_{\mathbf{x}}\mathbf{c}(\mathbf{u}(\mathbf{x}))$ (if a gradient mesh was contained, cf. Eq. (12)), and the Laplacians $\mathbf{f}_{\mathbf{p}(t)}$ of Poisson curves $\mathbf{p}(t)$ are additively rasterized on top.

VIII. RESULTS

In the following section, we present our results using scenes that contain gradient meshes, diffusion curves, and Poisson curves. We made all test scenes available online [54]. All images are rendered at a resolution of 1024×1024.

A. Qualitative Results

For all test scenes, we show results of only rasterizing the gradient meshes in Fig. 18(a) and only rasterizing the input curves in (b). This way, we can see which parts of the image are modeled with gradient meshes or diffusion/Poisson curves, respectively. To illustrate the input primitives, we depict the gradient mesh control polygon with solid lines, where different colors are applied for each mesh. Input boundary curves are shown as double-sided curve with an illustration of the respective boundary condition on the left and right. Dirichlet conditions directly show the corresponding color, while Neumann conditions are illustrated with a dash-dot pattern. Poisson curves are visualized with a long dash pattern. In (c), the undirected edge graph is visualized by solid black curves and the vertices are depicted with black circles. Note that Poisson curves are not part of the edge graph, since they are rendered additively onto the target Laplacian function. The unified patch representation in (d) gives all pixels that belong to the same region a common color. The colors are assigned from a custom color table such that the regions are visually distinguishable. The color is not related to the image content. The last column in (e) shows the final result.

a) Crane: Inspired from an ancient Chinese painting (the Auspicious Cranes by Emperor Huizong), we modeled a crane with diffusion curves, which is shown in Fig. 1. The boundary of the crane is largely modeled with Neumann conditions. We modeled the clouds with two 3×4 gradient meshes. In addition, we added diffusion curves and Poisson curves on the gradient

meshes for the layered and detailed effects of clouds. The background is formed from a 3×3 gradient mesh that mimics the similar color transition of the inspirational image.

b) Pepper: Fig. 18 (first row) shows an extension of the *poivron* scene from Orzan et al. [38]. We replaced the background with a 30×30 gradient mesh that was fitted with Adobe Illustrator to a photograph inside a restaurant. The gradient mesh's smoothness creates the impression of defocus blur, putting emphasis on the pepper in the foreground. We took a bite out of the pepper by adding further diffusion curves. The outer boundary of the pepper is modeled with homogeneous Neumann conditions, such that it can be placed in the scene without causing color diffusion into the background. The shadow on the table is modeled with Poisson curves.

c) Sunset: Fig. 18 (second row) shows a colorful sunset. The color gradient in the background contains multiple hues and is modeled with a 10×3 gradient mesh. We also deformed the spatial control points for artistic effects. Since it is interrupted by other curve primitives in-between, the color gradient would have been difficult to model with diffusion curves only. The sun is drawn using a diffusion curve, which allows for the modeling of the faint glow into the background. On the top and bottom of the image, we placed diffusion curves that contrast the nature background with expressive wavy shapes.

d) Bubbles: In Fig. 18 (third row), we modeled two soap bubbles. We used a 1×3 gradient mesh to model smooth color transitions in the sky. The bubbles are created by two 4×4 gradient meshes. The smooth color transitions are difficult to model directly with diffusion curves. This example shows that a vivid scene can be represented by a small number of primitives when combining mesh-based and curve-based methods.

e) Ladybug: The test scene in Fig. 18 (fourth row) is based on the *ladybug* diffusion curve model of Orzan et al. [38]. We made minor adjustments to the original curve endpoints to remove edge crossings that would otherwise cause color leakage. In addition, we set the outer boundary of the ladybug to Neumann condition. We placed the ladybug in a beach scene, which contains an ocean and a blue sky. The background is modeled using a 11×3 gradient mesh. In addition, we modeled a leaf on which the ladybug sits with a 6×2 gradient mesh, which captures the subtle color changes without smoothing. The leaf veins are created with Poisson curves, and the outer boundary is modeled with diffusion curves to add scattering.

f) Portal: The last test scene in Fig. 18 (fifth row) shows a portal. We created the portal with three gradient meshes: a 4×3 background mesh, and two 3×4 gradient meshes for the outer and internal parts of the portal. The rendered portal contains smooth color transitions to represent light and shadows. Inside the portal, We modified the *zephyr* diffusion curve model of Orzan et al. [38]. The diffusion curves diffuse color into the portal. The mesh Laplacian provides richer effects and more potential for artistic control compared to curve-only results.

B. Comparison with Vectorizations

So far, there exists no other method to compare with that supports the image synthesis for scenes that contain both gradient meshes and diffusion curves. However, once an
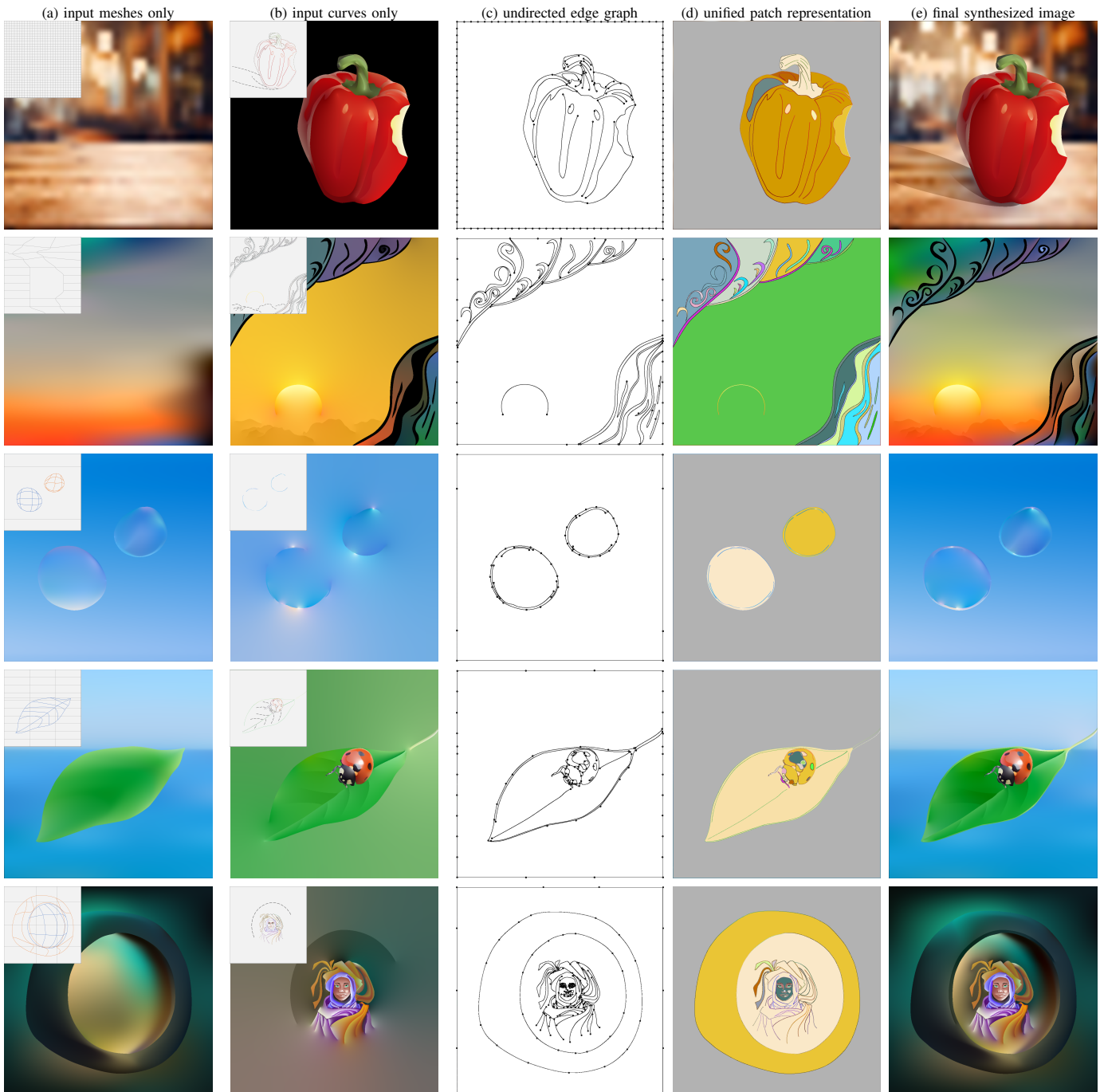
Fig. 18. Qualitative results of the pepper, sunset, bubbles, ladybug, and portal scene. From left to right, we show the input primitives (a) and (b), the undirected edge graphs (c), the unified patch representations (d) that were formed by our algorithm and the final rendering results (e).

automatic vectorization algorithm is developed for our unified formulation, it should be possible to compare how well gradient meshes only, diffusion curves only, and our unified formulation are able to represent the same raster image. For now, we can do a slightly different experiment. In Fig. 19, we take the output from our unified renderer (a) and vectorize this image using gradient meshes (b) and diffusion curves (c)-(d). For each image, the input primitives and a difference image to (a) are shown. The gradient meshes are placed manually using Adobe Illustrator. We can see that gradient meshes have a difficult

time adjusting to local structures since they always subdivide entire rows or columns, see for example the swirls behind the crane. Diffusion curves are useful for modeling sharp edges and details, which can be seen in the foreground objects across all test scenes. In (c), the diffusion curves are vectorized using Canny edge detection [37] on the multi-channel color gradient of Di Zenzo [10], as done by Orzan et al. [38]. In (d), edges are found using isocontours in the residual field as done by [67], although without their subsequent shape optimization. For both (c) and (d), we use the vectorization methods to fill the
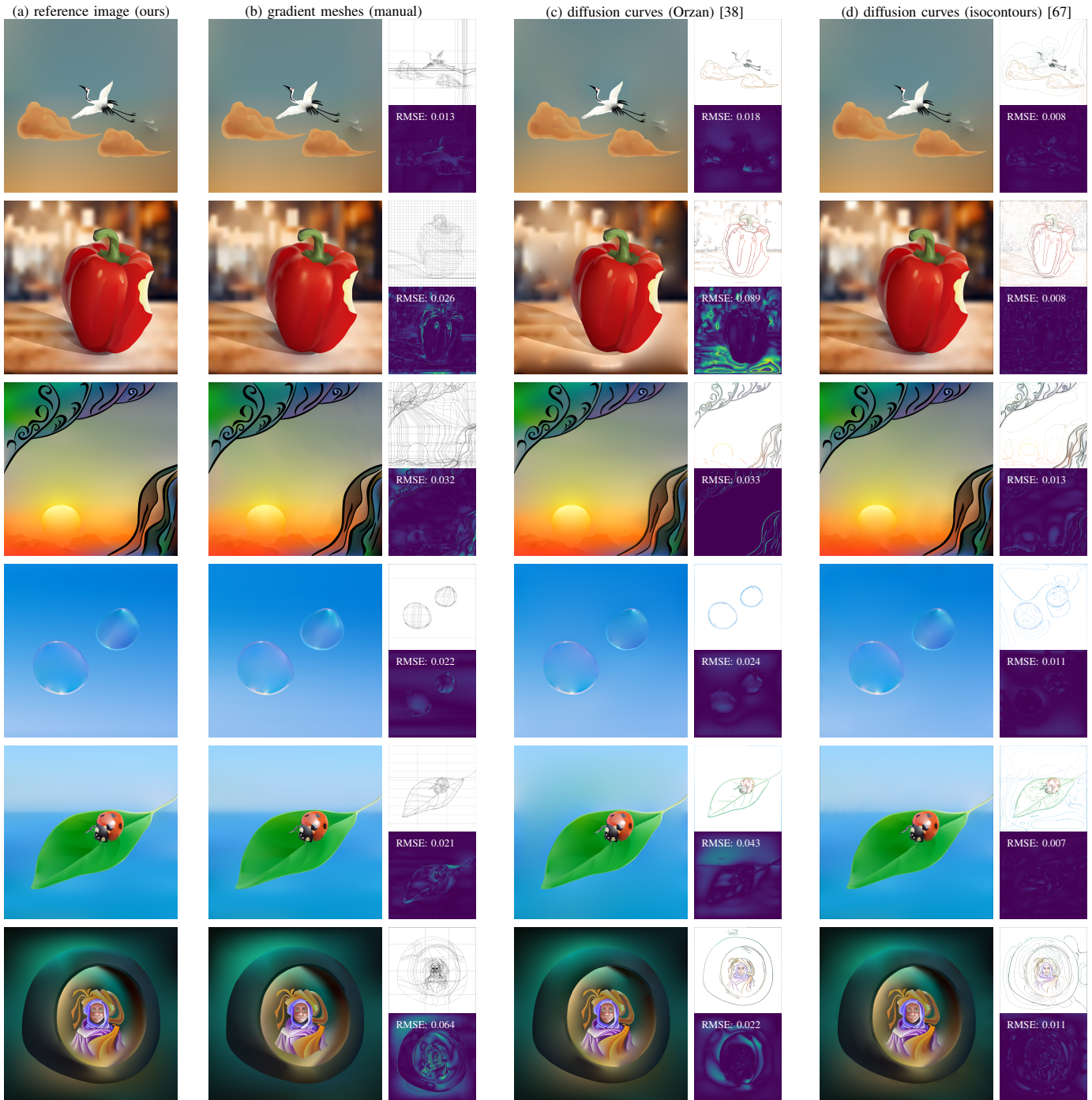
Fig. 19. Taking our result (a) as a reference, we test how well gradient meshes only (b) and diffusion curves only (c)-(d) approximate our result. Difference images show that gradient meshes often better capture inhomogeneous gradients in the background, while diffusion curves capture details in the foreground.

areas of the gradient meshes, i.e., we take our diffusion curves as initial set to start from. We can see in the difference images that many diffusion curves are needed to represent the bi-cubic color gradients of a gradient mesh, which is apparent in scenes with complex backgrounds, such as in the pepper scene. While (d) obtains a low reconstruction error, the high curve density is not necessarily user-friendly. A gradient mesh could have represented the inhomogeneous color gradients more compactly. With our unified approach, both primitives can be combined in the same scene.

C. User Interaction

When individual input primitives are moved in the scene by the user, intersections with other primitives will occur that lead to the creation or removal of patches. To ensure a consistent editing experience, we made sure that all user input is specified on the extended gradient meshes, extended diffusion curves, and Poisson curves alone. On the front end, the user is not accessing the edge graph or the patch representation, which are both computed fully automatically from the input primitives. Fig. 20 shows the rendered output when taking a closed diffusion curve
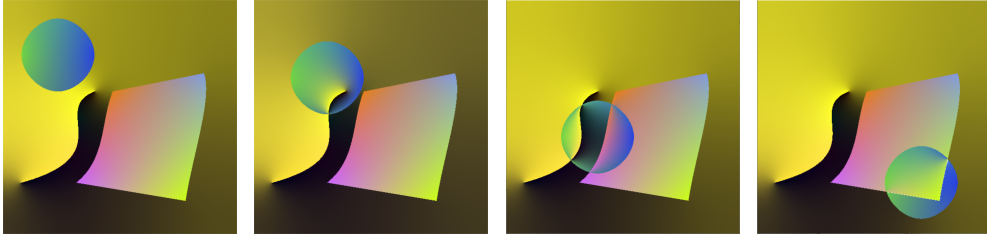
Fig. 20. During user interaction, new patches are formed and destroyed. To achieve a consistent scene editing experience, the user only edits input gradient meshes, diffusion curves, and Poisson curves. The edge graph and the unified patches are not exposed to the user and are created fully automatically.

TABLE I
PERFORMANCE MEASUREMENTS FOR THE VARIOUS TEST SCENES USED IN THE PAPER. THE COLUMNS LIST THE CORRESPONDING FIGURE, THE NUMBER OF INPUT DIFFUSION CURVES (#DC), THE NUMBER OF INPUT POISSON CURVES (#PC), THE NUMBER OF INPUT GRADIENT MESHES (#GM) FOR WHICH WE LIST THE GRID RESOLUTION IN BRACKETS, THE NUMBER OF VERTICES (#V) AND EDGES (#E) IN THE EDGE GRAPH, THE NUMBER OF PATCHES (#P), THE COMPUTATION TIME TO FORM THE EDGE GRAPH, THE COMPUTATION TIME TO TRAVERSE THE EDGE GRAPH, THE TIME NEEDED TO RESOLVE THE NESTING OF PATCHES, AND THE NUMBER OF JACOBI ITERATIONS FOR RENDERING. ALL TIMINGS ARE IN MILLISECONDS.

Scene	Figure	#DCs	#PCs	#GMs	#Vs	#Es	#Ps	Construct graph	Traverse graph	Create patches	It.
Crane	Fig. 1	68	5	3	160	124	8	16	5	13	1 k
Pepper	Fig. 18 (first row)	36	3	1	184	165	8	11	9	12	10 k
Sunset	Fig. 18 (second row)	38	6	1	88	108	37	16	2	7	15 k
Bubble	Fig. 18 (third row)	9	0	3	62	53	3	2	1	1	15 k
Ladybug	Fig. 18 (fourth row)	59	8	2	111	113	20	14	2	16	2 k
Portal	Fig. 18 (fifth row)	87	1	3	211	167	17	20	3	14	15 k

and moving it across a scene. Intersections with other diffusion curves and gradient meshes are created, which causes new patches to form or disappear. Using our data structures, the results remain consistent throughout the editing process.

D. Performance Analysis

We evaluated our method on an Intel Core i9-10980XE with 4.6 Ghz and an Nvidia RTX 2080 TI GPU. The performance measurements are reported in Table I. To give context we report for each scene statistics about the complexity of the input data (number of diffusion curves, number of Poisson curves, number of gradient meshes), statistics on the complexity of the edge graph (number of vertices and edges), and the number of patches in our patch representation. We report the time it takes to compute the edge graph, to traverse the edge graph during the patch construction, and to resolve the nesting relationships.

Across all test scenes, the edge graph construction was at around 12-17 ms, the edge graph traversal at 4-10 ms, and the patch creation at 13-45 ms. In sum, we achieved across all scenes an interactive frame rate (29-67 milliseconds per frame). While we incorporated bounding volume hierarchies to accelerate the intersection and containment tests, we think that further improvements could include partial updates of both the edge graph and the patch representation whenever an object moves. Further, a hierarchical discretization could be used to accelerate the winding angle tests. Despite us rebuilding the graph and patches from scratch, we obtained interactive results.

To assess the scalability, we randomly generated a varying number of diffusion curves in the unit square. The x-coordinates of the control points were set to $(0, 0.3, 0.7, 1.0)$. The y-coordinates are random numbers in $[0, 1]$. This test leads to a large number of edge crossings. In Fig. 21 (left), we plot the total construction time as a function of the number of vertices $|\mathcal{V}|$ in the edge graph, which is a measure of the scene
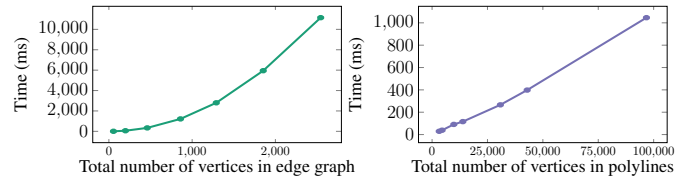

Fig. 21. Scalability experiments (total time in milliseconds) for varying input complexity (left) and discretization (right) on an Intel Core i7-6700K.

complexity. We observe that the runtime scales quadratically in the number of graph vertices, which can be expected since each edge is tested for intersections against each other edge.

Further, we examine the scalability of the algorithm for varying polyline discretizations in the crane scene. In Fig. 21 (right), we varied the threshold $\tau \in \{0.1, 0.05, \ldots, 0.0001\}$, which leads to an increasing number of polyline vertices. We observe that the total construction time scales nearly linear in the number of polyline vertices. This observation holds for the other scenes, as well, but with a different slope.

E. Discussion

The edge graph and our resulting patch construction directly depend on the quality of the input primitives. If the input primitives contain tiny gaps or small crossings, then a corresponding patch representation would carry on those artifacts. In recent years, more exact vector graphics rendering algorithms have been developed and utilized, such as Walk-on-Spheres [43], Walk-on-Boundaries [48], or Walk-on-Stars [44], or Walkin' Robin [36]. These approaches likewise suffer from inaccuracies in the scene description, since they are able to resolve the color transport through narrow gaps. In Section V-C, we described our scene preprocessing, which closes gaps to prevent color leaking. To alleviate the need for such preprocessing, more research towards user interfaces that naturally steer the user

towards more accurate scene descriptions would be helpful for both scene modeling and rendering.

IX. CONCLUSIONS

In this paper, we unified the mathematical modeling of mesh-based vector graphics and curve-based vector graphics, which allows for the first time to include both types of primitives in the same scene. For this, we rephrased the interpolation inside gradient meshes as solution to a Poisson problem by searching for a color field that matches the Laplacian of the gradient mesh. We developed a four-stage pipeline that takes gradient meshes, diffusion curves, and Poisson curves as input. First, we extended gradient meshes and diffusion curves by enabling the specification of Neumann conditions in addition to the usual Dirichlet conditions. Second, our approach handles arbitrary intersections of diffusion curves and gradient meshes, since all intersections are resolved through the construction of an edge graph. Third, from the edge graph, non-overlapping patches with well-defined boundary conditions and a target Laplacian are derived. Fourth, a standard Poisson problem is solved in the interior of each patch to synthesize the output image. The final rasterized images can be computed with any off-the-shelf Poisson solver.

In the past, research on mesh-based and curve-based smooth vector graphics followed independent research threads, which concentrated on the editing, rasterization, or vectorization of the individual primitives. We hope that the unified treatment will spur further research that promotes synergies of the two approaches, for example regarding novel image synthesis or vectorization methods. The next step in the generalization will include the incorporation of bi-harmonic diffusion curves [13], for which harmonic solutions can be searched in the Laplacian domain. Similar to common content creation tools, it could be useful to introduce multiple layers of smooth vector graphics. Further, it would be interesting to develop a vectorization algorithm for the unified patch description. It would be imaginable to divide the input domain into regions. Per region, the algorithm could decide whether it is better to fit a gradient mesh or a set of diffusion curves. We leave the development of a vectorization algorithm to future work.

REFERENCES

[1] S. D. Baksteen, G. Hettinga, J. Echevarria, and J. Kosinka. Mesh colours for gradient meshes. In *Smart Tools and Applications in Graphics 2021*. Eurographics Association, 2021. doi: 10.2312/stag.20211478

[2] S. Bang, K. Serkh, O. Stein, and A. Jacobson. An adaptive fast-multipole-accelerated hybrid boundary integral equation method for accurate diffusion curves. *ACM Transactions on Graphics (TOG)*, 42(6), dec 2023. doi: 10.1145/3618374

[3] 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. doi: 10.1007/s00371-018-1547-1

[4] 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. Association for Computing Machinery, New York, NY, USA, 2010. doi: 10.1145/1809939.1809944

[5] J. C. Bowers, J. Leahey, and R. Wang. A ray tracing approach to diffusion curves. *Computer Graphics Forum*, 30(4):1345–1352, 2011. doi: 10.1111/j.1467-8659.2011.01994.x

[6] 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. doi: 10.1145/2366145.2366192

[7] 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. doi: 10.1016/j.cma.2019.112598

[8] J. Chen, F. Schäfer, and M. Desbrun. Lightning-fast method of fundamental solutions. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 43(4), 2024. doi: 10.1145/3658199

[9] M. Czekanski, B. Faber, M. Fairborn, A. Wright, and D. Bindel. Walking on spheres and talking to neighbors: Variance reduction for Laplace's equation, 2024. https://arxiv.org/abs/2404.17692.

[10] S. Di Zenzo. A note on the gradient of a multi-image. *Computer Vision, Graphics, and Image Processing*, 33(1):116–125, 1986. doi: 10.1016/0734-189X(86)90223-9

[11] 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. doi: 10.3138/FM57-6770-U75U-7727

[12] G. Farin. *Curves and surfaces for CAGD: a practical guide*. Elsevier Science, Amsterdam, Netherlands, 2002. doi: 10.1016/B978-1-55860-737-8.X5000-5

[13] 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. doi: 10.1145/2070781.2024200

[14] 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. doi: 10.1016/j.cad.2019.05.005

[15] Q. Fu, L. Liu, F. Hou, and Y. He. Hierarchical vectorization for facial images. *Computational Visual Media*, 10(1):97–118, 2024. doi: 10.1007/s41095-022-0314-4

[16] M. Gangnet, J.-C. Hervé, T. Pudet, and J.-M. van Thong. Incremental computation of planar maps. In *Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '89, p. 345–354. Association for Computing Machinery, New York, NY, USA, 1989. doi: 10.1145/74333.74369

[17] G. J. Hettinga, R. Brals, and J. Kosinka. Colour interpolants for polygonal gradient meshes. *Computer Aided Geometric Design*, 74:101769, 2019. doi: 10.1016/j.cagd.2019.101769

[18] G. J. Hettinga, J. Echevarria, and J. Kosinka. Efficient image vectorisation using mesh colours. In *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference*. The Eurographics Association, 2021. doi: 10.2312/stag.20211484

[19] 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. doi: 10.1109/TVCG.2018.2867478

[20] K.-W. Hsiao, Y.-L. Yang, Y.-C. Chiu, M.-C. Hu, C.-Y. Yao, and H.-K. Chu. Img2Logo: Generating golden ratio logos from images. *Computer Graphics Forum (Proc. Eurographics)*, 42(2):37–49, 2023. doi: 10.1111/cgf.14742

[21] 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. doi: 10.1145/2508363.2508426

[22] A. Jacobson, L. Kavan, and O. Sorkine-Hornung. Robust inside-outside segmentation using generalized winding numbers. *ACM Transactions on Graphics (TOG)*, 32(4), jul 2013. doi: 10.1145/2461912.2461916

[23] A. Jacobson, T. Weinkauf, and O. Sorkine. Smooth shape-aware functions with controlled extrema. *Computer Graphics Forum*, 31(5):1577–1586, 2012. doi: 10.1111/j.1467-8659.2012.03163.x

[24] S. Jeschke. Generalized diffusion curves: An improved vector representation for smooth-shaded images. *Computer Graphics Forum*, 35(2):71–79, 2016. doi: 10.1111/cgf.12812

[25] S. Jeschke, D. Cline, and P. Wonka. A GPU Laplacian solver for diffusion curves and Poisson image editing. *ACM Transactions on Graphics*, 28(5):1–8, 2009. doi: 10.1145/1618452.1618462

[26] S. Jeschke, D. Cline, and P. Wonka. Estimating color and texture parameters for vector graphics. *Computer Graphics Forum*, 30(2):523–532, 2011. doi: 10.1111/j.1467-8659.2011.01877.x

[27] 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. doi: 10.1145/1531326.1531391

[28] K. Lawonn and T. Günther. Stylized image triangulation. *Computer Graphics Forum*, 38(1):221–234, 2019. doi: 10.1111/cgf.13526

[29] X.-Y. Li, T. Ju, and S.-M. Hu. Cubic mean value coordinates. *ACM Transactions on Graphics (TOG)*, 32(4):126–1, 2013. doi: 10.1145/2461912.2461917

[30] Z. Li, G. Yang, X. Deng, C. De Sa, B. Hariharan, and S. Marschner. Neural caches for Monte Carlo partial differential equation solvers. In *SIGGRAPH Asia Conference*. ACM, New York, NY, USA, 2023. doi: 10.1145/3610548.3618141

[31] 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. doi: 10.1109/TVCG.2012.76

[32] S. Lu, X. Ding, F. Gao, and J. Chen. Shape manipulation of diffusion curves images. *IEEE Access*, 8:57158–57167, 2020. doi: 10.1109/ACCESS.2020.2982457

[33] 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. doi: 10.1007/s00371-019-01671-0

[34] B. Miller, R. Sawhney, K. Crane, and I. Gkioulekas. Boundary value caching for walk on spheres. *ACM Transactions on Graphics*, 42(4), jul 2023. doi: 10.1145/3592400

[35] B. Miller, R. Sawhney, K. Crane, and I. Gkioulekas. Differential walk on spheres. *ACM Transactions on Graphics*, 43(6), Nov. 2024. doi: 10.1145/3687913

[36] B. Miller, R. Sawhney, K. Crane, and I. Gkioulekas. Walkin' Robin: Walk on stars with Robin boundary conditions. *ACM Transactions on Graphics*, 43(4), jul 2024. doi: 10.1145/3658153

[37] A. Orzan, A. Bousseau, P. Barla, and J. Thollot. Structure-preserving manipulation of photographs. In *Proceedings of the 5th International Symposium on Non-Photorealistic Animation and Rendering*, NPAR '07, p. 103–110. Association for Computing Machinery, New York, NY, USA, 2007. doi: 10.1145/1274871.1274888

[38] A. Orzan, A. Bousseau, H. Winnemöller, P. Barla, J. Thollot, and D. Salesin. Diffusion curves: a vector representation for smooth-shaded images. *ACM Transactions on Graphics (TOG)*, 27(3):1–8, 2008. doi: 10.1145/2483852.2483873

[39] 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. doi: 10.1109/MCG.2011.86

[40] R. Prévost, W. Jarosz, and O. Sorkine-Hornung. A vectorial framework for ray traced diffusion curves. *Computer Graphics Forum*, 34(1):253–264, 2015. doi: 10.1111/cgf.12510

[41] B. Price and W. Barrett. Object-based vectorization for interactive image editing. *The Visual Computer*, 22(9):661–670, 2006. doi: 10.1007/s00371-006-0051-1

[42] U. Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer graphics and image processing*, 1(3):244–256, 1972. doi: 10.1016/S0146-664X(72)80017-0

[43] 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. doi: 10.1145/3386569.3392374

[44] R. Sawhney, B. Miller, I. Gkioulekas, and K. Crane. Walk on stars: A grid-free Monte Carlo method for pdes with neumann boundary conditions. *ACM Transactions on Graphics*, 42(4), jul 2023. doi: 10.1145/3592398

[45] D. Schmitt. Bivariate b-splines from convex pseudo-circle configurations. In *Fundamentals of Computation Theory*, p. 335–349. Springer-Verlag, Berlin, Heidelberg, 2019. doi: 10.1007/978-3-030-25027-0_23

[46] T. Sederberg and T. Nishita. Curve intersection using Bézier clipping. *Computer-Aided Design*, 22(9):538–549, 1990. doi: 10.1016/0010-4485(90)90039-F

[47] J. Spainhour, D. Gunderman, and K. Weiss. Robust containment queries over collections of rational parametric curves via generalized winding numbers. *ACM Transactions on Graphics*, 43(4), jul 2024. doi: 10.1145/3658228

[48] R. Sugimoto, T. Chen, Y. Jiang, C. Batty, and T. Hachisuka. A practical walk-on-boundary method for boundary value problems. *ACM Transactions on Graphics*, 42(4), jul 2023. doi: 10.1145/3592109

[49] R. Sugimoto, N. King, T. Hachisuka, and C. Batty. Projected walk on spheres: A Monte Carlo closest point method for surface PDEs. In *SIGGRAPH Asia 2024 Conference Papers*, SA '24, p. 1–10. ACM, Dec. 2024. doi: 10.1145/3680528.3687599

[50] 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. doi: 10.1145/1276377.1276391

[51] 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. doi: 10.1145/2601097.2601187

[52] 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 Transactions on Graphics*, 31(4):1–9, 2012. doi: 10.1145/2185520.2185570

[53] S. Swaminarayan and L. Prasad. Rapid automated polygonal image decomposition. In *Applied Imagery and Pattern Recognition (AIPR'06)*, pp. 28–28. IEEE, USA, 2006. doi: 10.1109/AIPR.2006.30

[54] X. Tian and T. Günther. Unified smooth vector graphics: Modeling gradient meshes and curve-based approaches jointly as Poisson problem - reference implementation, 2025. doi: 10.5281/zenodo.15095999

[55] X. Tian and T. Günther. A survey of smooth vector graphics: Recent advances in representation, creation, rasterization, and image vectorization. *IEEE Transactions on Visualization and Computer Graphics*, 30(3):1652–1671, 2024. doi: 10.1109/TVCG.2022.3220575

[56] 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. doi: 10.1007/s11042-017-4987-0

[57] W. Wang, Z. Chen, L. Fang, and J. Cao. Curved image triangulation based on differentiable rendering. *Computer Graphics Forum*, 43(7):e15232, 2024. doi: 10.1111/cgf.15232

[58] G. Wei, Y. Zhou, X. Gao, Q. Ma, S. Xin, and Y. He. Field-aligned quadrangulation for image vectorization. *Computer Graphics Forum*, 38(7):171–180, 2019. doi: 10.1111/cgf.13826

[59] 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. doi: 10.1145/1661412.1618461

[60] Y. Xiao, J. Cao, and Z. Chen. Image representation on curved optimal triangulation. *Computer Graphics Forum*, 41(6):23–36, 2022. doi: 10.1111/cgf.14495

[61] 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. doi: 10.1109/TMM.2012.2233725

[62] Y. Xiao, L. Wan, C. S. Leung, Y.-K. Lai, and T.-T. Wong. Optimization-based gradient mesh colour transfer. *Computer Graphics Forum*, 34(6):123–134, 2015. doi: 10.1111/cgf.12524

[63] 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. doi: 10.1109/TVCG.2015.2440273

[64] E. F. Yilmazer, D. Vicini, and W. Jakob. Solving inverse PDE problems using grid-free Monte Carlo estimators. *ACM Transactions on Graphics*, 43(6), Nov. 2024. doi: 10.1145/3687990

[65] Z. Yu, L. Wu, Z. Zhou, and S. Zhao. A differential Monte Carlo solver for the poisson equation. In *ACM SIGGRAPH 2024 Conference Papers*, SIGGRAPH '24. Association for Computing Machinery, New York, NY, USA, 2024. doi: 10.1145/3641519.3657460

[66] C. Yuksel, J. Keyser, and D. H. House. Mesh colors. *ACM Transactions on Graphics*, 29(2), Apr. 2010. doi: 10.1145/1731047.1731053

[67] 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. doi: 10.1109/TVCG.2017.2721400

[68] 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. doi: 10.1109/TIP.2014.2327807

[69] H. Zhu, J. Cao, Y. Xiao, Z. Chen, Z. Zhong, and Y. J. Zhang. TCB-spline-based image vectorization. *ACM Transactions on Graphics*, 41(3), 2022. doi: 10.1145/3513132

Xingze Tian is currently working toward the PhD degree with the Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany. Her research interests include visualization and computer graphics. Contact her at xingze.tian@fau.de.

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. Contact him at tobias.guenther@fau.de.