Computers & Graphics (2019)

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Scribble-based Colorization for Creating Smooth-shaded Vector Graphics

Bin Bao[a], Hongbo Fu[b]

[a]*School of Data and Computer Science, Shadong Women's University*
[b]*School of Creative Media, City University of Hong Kong*

ARTICLE INFO

Article history:
Received April 14, 2019

Keywords: Colorization, Vector Graphics, Diffusion Curves, Scribble-based UI

ABSTRACT

This paper proposes a novel colorizaiton tool for intuitively creating smooth-shaded vector graphics. Our technique takes advantage of Diffusion Curves, a powerful vector graphics representation. Rather than specifying colors along curves in previous works, we allow a user to intuitively paint on regions of arbitrary line drawings. Once the user scribbles on drawings, our algorithm automatically estimates the colors along the curves of drawings, resulting in smooth color regions as close as possible to the user specification. Compared with the previous color estimation techniques for image vectorization, we propose a new diffusion curve colorization algorithm for fitting sparse colors of input scribbles. Our approach is fast, and provides instant feedback to the user. We have tested our system on a variety of line drawings with varying shape complexity, and shown that our technique can produce visually pleasing smooth-shaded images intuitively and effectively.

1. Introduction

Vector graphics images remain invaluable for a broad range of 2D applications because of their resolution independence, compactness of representation, and powerful editability. Visually pleasing vector graphics heavily relies on color gradients to achieve soft shadows, diffuse shading, and various material effects. Existing vector-based drawing tools, such as Adobe Illustrator and CorelDraw, offer only limited support for representing complex color gradients. Researchers have proposed more powerful tools for handling complex gradients, including *gradient mesh* [1], gradient layers [2], *diffusion curves* [3] and the extended curve-based vector graphics [4] [5] [6].

Creating vector graphics is not an easy work. Both gradient mesh and gradient layers have not provided a natural way for free hand creation of vector graphics from scratch. The concept of drawing shapes by first sketching and then developing colors seems to be very natural and intuitive. However, existing curve-based vector graphics has focused on how to represent complex gradients with a preliminary drawings, but has not provided an efficient approach to specify curve colors, which still require much efforts and high drawing skills.

As illustrated in Figure 1, we adopt a scribble-based user interface, which has been used in various interactive applications (see detailed discussion in Section 2). Our new interface allows users to intuitively paint color scribbles in regions of line drawings (expressed as predefined curves). The system automatically computes color points along input curves, and then creates smooth gradients on both side of the curves by a diffusion process. Our approach provides instant feedback to users and produces better visual effects, compared with both an existing color estimation algorithm proposed by Jeschke et al. [7] and LazyBrush, a pixel-level hand-drawn coloring technique [8] (see the detailed discussion in Section 5).

The main contribution of this work are therefore:

- A novel tool for interactively creating vector graphics of complex gradients, which allows painting on regions in an intuitive way.

- An automatic diffusion curve coloring algorithm for fitting sparse colors of input scribbles.

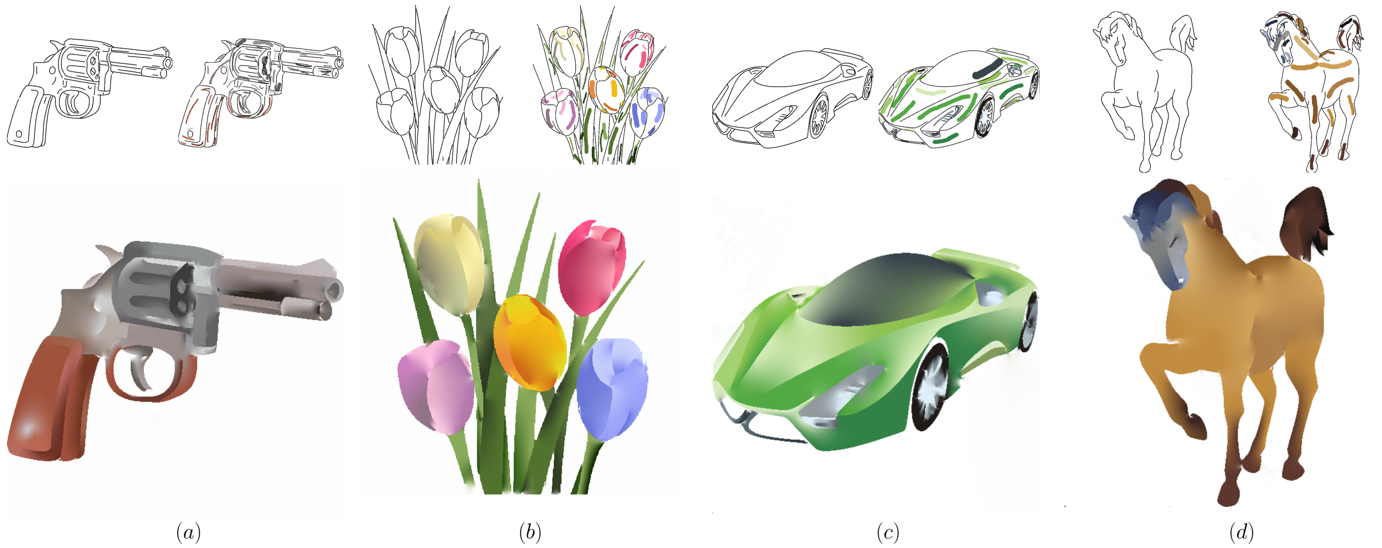

Fig. 1. Representative results generated by our proposed scribble-based colorizaiton system. The user-specified scribbles (colored lines in the respective top right images) on the input line drawings (top left images), result in the images in the bottom row.

2. Related Work

Diffusion Curves. Diffusion curve [3] was proposed as a new vector-based primitive, which allows arbitrarily shaped curves to shade an image through a diffusion process. A most recent work [5] generalizes the classic expression of diffusion curves, to allow more color control away from the diffusion curves. To create or edit color gradients, a user is required to specify color points on diffusion curves and thus make indirect changes to the shading of images. To address this issue, we present a scribble-based diffusion coloring user interface. For simplicity we focus on the origin diffusion curves, but our approach could be extended to diffusion curve variants. A diffusion curve image is rendered by solving a Laplace equation (Equation 1) , which is computationally expensive. Many existing methods [9] [10] have been studied to facilitate interactive applications, and we adopt [9] for rendering in our system implementation, which is discussed in detail in Section 5. Bowers et al. [7] proposed a ray tracing approach to evaluate the color contributions of each diffusion curve for a given pixel. Later prevost et al. [11] take the benefits of such a vectorial representation to facilitate their ray tracing approach. Our algorithm is motivated by their ray tracing method (see detailed discussion in Section 3.2).

Reverse Problem of Diffusion Curve Images. Although efficient rendering of diffusion curve images has been well explored, its inverse problem of creating diffusion curves from raster images remains challenging. Since Orzan et al. [3] provided a simple approach based on canny edge detection, several approaches for curve extraction or color estimation for diffusion curves have been proposed. A recent work [12] focused on the problem of directly optimizing curve geometry, and [13] [14] largely focused on optimizing curve coloring with curve geometry predetermined. These works were proposed for natural image vectorization, and not suitable for sparse scribble inputs of our problem.

Sketch-based Content Generation. Sketches are the most elementary primitives in painting, both digitally and physically. Researchers have long been aware of the gap between sketches and visually appealing artworks, and various ideas have been proposed for converting simple sketches into richer and more expressive images. Sykora et al. [8] proposed a novel fexible painting tool for easily coloring in various drawing styles. However, they have no consider on color gradients. Michal et al. [15] proposed the brush and fill tools for digital image painting, but has mainly focused on texture synthesis. Recently, due to the effectiveness of deep learning, several learning-based methods have been proposed for sketch-based image generation. Isola et al. [16] and Su et al. [17] respectively generate textures and shades on sketches. PaintsChainer [18], Sangkloy et al. [19] and Liu et al. [20] are techniques for sketch colorizaiton. The main difference from our work is that they generate natural images, while we create vector graphics. A quick comparison with PaintsChainer will be discussed in Section 5.

Scribble-based User Interface. Scribble-based UI has been adopted for various interactive applications such as colorization of images [21] [22], and has also been extensively used for sketches, including for sketch segmentation [23], for data-driven segmentation [24] and for lazy selection [25]. In our approach, we make use of a similar user interface, but aim for turning an input line drawing into a smooth-shaded vector graphics with the diffusion curve representation.

3. Overview

3.1. User Interface

Our system provides a scribble-based user interface for interactive coloring on line drawings. The system also supports progressive coloring, which gives instant feedback to users and allows them to adjust effects by a casual way. Note that our current implementation uses mouse inputs, but it naturally support pen inputs as well. Typically, a user starts with a small number of color scribbles, and then fine-tunes the colorization results by adding more scribbles. See Figure 2 (a)-(b)-(c)-(d). A user

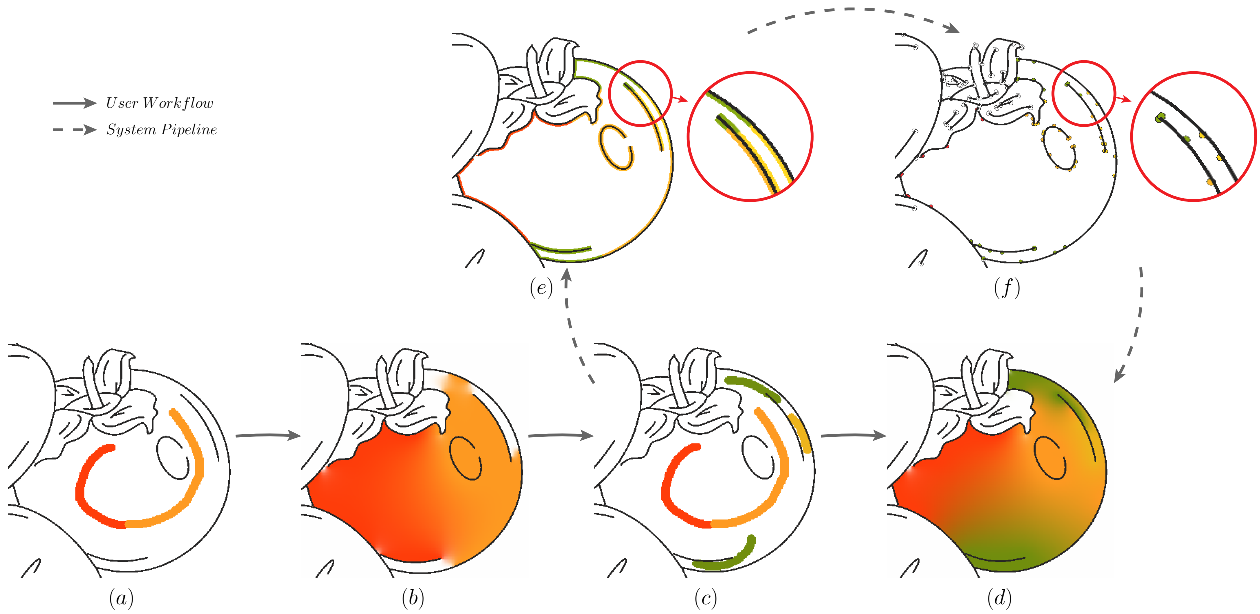

Fig. 2. Overview. (a)-(b)-(c)-(d) shows the user workflow of progressive coloring. (a) and (c) show user input scribbles. (b) and (f) show corresponding feedback of our system. (c)-(e)-(f)-(d) illustrates the computation pipeline of our system. (e) illustrates the core computation of assigning each curve segments with a corresponding input color. The colored curve segments are then transferred to color points (f), and be diffused to generate the finnal image (d).

progressively marks color scribbles on the canvas by dragging the mouse cursor while holding a button (Figure 2 (a)). The coloring process is triggered once each marking scribble is drawn (Figure 2 (b)). The user then inspects the colorization result and decides if more scribbles need to be added. It is therefore critical that our system generates color points on the curves with very little delay. We present a novel color estimation algorithm to achieve real-time feedback.

3.2. Methodology

Background. We start by briefly revisiting the Diffusion Curves, which are curves that diffuse colors on both sides of space that they divides. Orzan et al. [3] formulat the diffusion curve image as the solution to a Laplace equation with a Dirichlet boundary condition (Equation 1).

$$\begin{cases} I(x,y) = C(x,y) & pixel\ (x,y) \in curves \\ \Delta I(x,y) = 0 & otherwise \end{cases}, \qquad (1)$$

where $C(x,y)$ is the boundary condition defined by curve colors.

Jeschke et al. [13] introduce an automatic diffusion curve coloring algorithm. However their approach is designed for fitting dense colors of image vectorization, which would cause serious artifacts when dealing with scribble-based user inputs (see the detailed discussion in Section 5). In contrast, our approach supports fitting the sparse colors of input scribbles.

System Pipeline. Given color scribbles marked by the user and a set of curves in an input drawing, our goal is to define a set of color points and the corresponding color values along the curves so that the resulting image matches the user inpus as closely as possible. The core of our approach is to assign the input colors to corresponding curve segments (Figure 2 (e)). The resulting colored segments are finally transferred to color points (Figure 2 (f)), which are diffused by Equation 1 to generate smoothly shaded vector graphics. The above process is performed every time the user adds a new scribble, and repeated until no more scribbles are added.

We will discuss the algorithm details in Section 4. Starting with a single scribble marked by a user, our method computes a set of curve segments that are influenced by the scribble (Section 4.1). With more scribbles added by the user, a curve segment can be influenced by multiple scribbles. We compute a partition for such curve segment, leading to that each partitioned cruve segments is assoiated with an independent scribble and is assigned with its color (Section 4.2).

4. Methodology

4.1. Single Scribble Influence

Motivated by the ray tracing rendering framework proposed by Bowers et al. [7]), we compute the influence of each pixel covered by the scribble to its surrounding visible curve points.

Our technique starts with visibility test, which is akin to seeking visibility on a 2D map that commonly occurs in games. There are various methods to calculate visibility in 2D. We adopt the method presented in [10], which considers visibility only in terms of the curve nodes. The visibility algorithm finds a set of surrounding curve nodes of a particular point (denoted as p) and lists them in a counterclockwise winding order, as illustrated in Figure 3 (a).

For a more complex case as shown in Figure 3 (b), we obtain the visible curve nodes of a whole scribble (which is uniformly sampled and denoted as $\mathcal{P}_{scribble}$), and aim to drop those which are excluded from the influnce scope of the scribble. To achieve this goal, we limit the influence of the whole scribble in a local region. As no explicit closed boundary is defined, we find a sequence of visible curve segments that implicitly define a local region, listed in a counterclockwise winding order, which are called *boundary curve segments* (see green nodes in Figure 3 (b)). We then detect the visible curve nodes inside the implicit region formed by the *boundary curve segments*, which are also influenced by the scribble (see orange nodes in Figure 3 (b)).

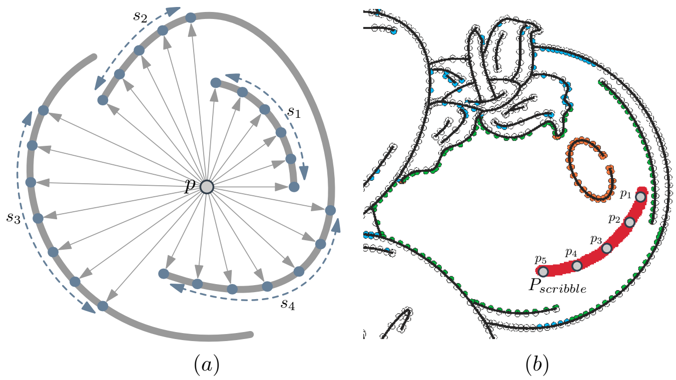

Fig. 3. Visibility test. (a) shows the visibility test from a single point. (b) shows the visible curve nodes to a scribble, which are marked with colors. Our algorithm distinguishes nodes that are on implicit boundary (green), inside the boundary (orange), and those are excluded from the influnce scope of the scribble (blue).

Energy Function. To obtain the boudary curve segments, we formulate an energy minimization problem on the *visibility graph* $\mathcal{G}$. Figure 4 shows how the visibility graph is constructed. For each sampled point $p_i \in \mathcal{P}_{scribble}$, the visibility algorithm [10] generates a sequence of curve nodes in counterclockwise order, from which we create a circuit of visibility curve segments, denoted as $\mathcal{C}_{vis}(p_i)$ (for example $\mathcal{C}_{vis}(p) = s_1 \to s_2 \to s_3 \to s_4 \to s_1$ in Figure 3 (a)). We then construct a visibility graph $\mathcal{G} = (\mathcal{S}, \mathcal{N})$ by combining $\{\mathcal{C}_{vis}(p_i)\}$, where $\mathcal{S}$ is a set of nodes, each corresponding to a curve segment by merging overlapped segments of $\{\mathcal{C}_{vis}(p_i)\}$ (for example, node s_1 is partially visible to each p_i in Figure 4), and $\mathcal{N}$ is a set of edges by uniting the connectivity between curve segments from $\{\mathcal{C}_{vis}(p_i)\}$ (see a detailed illustration in Figure 4). Note that in our implementation, we have omitted the curve segments shorter than a given threshold (3 sampled points) in computation of $\mathcal{C}_{vis}(p_i)$.
Given the visibility graph $\mathcal{G} = (\mathcal{S}, \mathcal{N})$, we consider the boundary curve segments as a circuit in a broad sense. Thus a solution candidate is denoted as $\mathcal{C} = \{s_1, .., s_n\}$, which meets that for each (s_i, s_{i+1}) there is a path on $\mathcal{G}$. To determine an optimal solution $\mathcal{C}^*$, we formulate the energy function E as:

$$E(\mathcal{C}) = \mu \sum_{i=1}^{n} D(s_i) + (1-\mu) \sum_{i=1}^{n} V(s_i, s_{(i+1) \bmod n}) \qquad (2)$$

The terms of E is designed based on the following key observations or rules:

- **Rule 1** Suppose the number of scribble points that this curve segment is visible to is κ. With an increase of κ, the probability that a curve segment is adopted increases.

- **Rule 2** The closer a curve segment is to the scribble, it is more likely that the curve segement is influenced by the scribble.

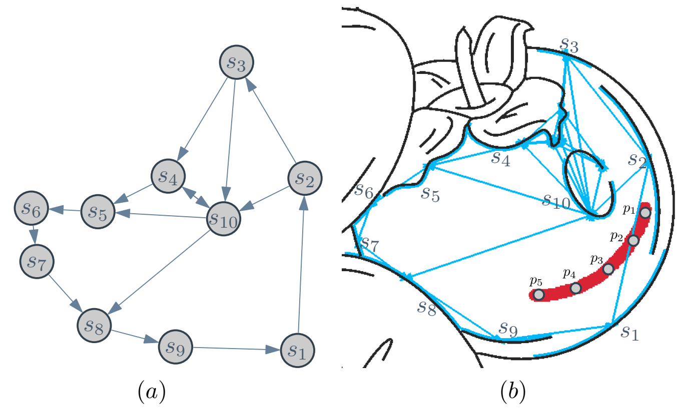

Fig. 4. Visibility graph constructed from the visibility test of Figure 3 (b). In detail, the edge connectivity comes from $\{\mathcal{C}_{vis}(p_i)\}$: $\mathcal{C}_{vis}(p_1)$ leads to $s_1 \to s_2 \to s_3 \to s_4 \to s_{10} \to s_8 \to s_9 \to s_1$; $\mathcal{C}_{vis}(p_2)$ leads to $s_1 \to s_2 \to s_3 \to s_{10} \to s_5 \to s_6 \to s_7 \to s_8 \to s_9 \to s_1$; $\mathcal{C}_{vis}(p_3), \mathcal{C}_{vis}(p_4), \mathcal{C}_{vis}(p_5)$ leads to $s_1 \to s_2 \to s_{10} \to s_4 \to s_5 \to s_6 \to s_7 \to s_8 \to s_9 \to s_1$ respectively.

- **Rule 3** The longer a curve segment is, it is more likely that the curve segement serves as a part of the boundary.

- **Rule 4** For geometric and semantic reasons, the implicit boundary that is formed by curve segments favors continuity between two successive curve segments.

Here $D(s)$ captures the properties of an individual curve segment (e.g., length and distance to the scribble etc.), $V(s_i, s_{i+1})$ evaluates the cost of continuity from curve segment s_i to s_{i+1}, and μ is a weight to balance the influence between the two terms (empirically, we found $\mu \in [0.3, 0.5]$ and set to 0.5 by default). The individual curve cost $D(s)$ is measured according to **Rules 1-3** in our implementation. In Equation 3, the first term $e_{vis}(s, \mathcal{P}_{scribble})$ measures the number of visible (see **Rule 1**), the other two terms $e_{dist}(s, \mathcal{P}_{scribble})$ and $e_{len}(s)$ measure the distance to the scribble (**Rule 2**), and the length of the curve segment (**Rule 3**), respectively, and α, β, γ are the weights to balance the three terms ($\alpha + \beta + \gamma = 1$, and each is set to $\frac{1}{3}$ by default).

$$D(s) = \alpha e_{vis}(s, \mathcal{P}_{scribble}) + \beta e_{dist}(s, \mathcal{P}_{scribble}) + \gamma e_{len}(s), \qquad (3)$$

The continuity cost $V(s_i, s_j)$ according to **Rule 4** is expressed as Equation 4. Empirically, we measure collinearity as the distance (denoted as $|gap|$) between the closest points on s_i and s_i, and $(\theta_i + \theta_j)^2$ as the positive angular difference between the endpoint tangents of s_i and s_j, where weight η (in the range [0, 1] and set to 0.4 by default) balances the two terms.

$$V(s_i, s_j) = \eta \|gap\| + (1-\eta)(\theta_i + \theta_j)^2 \qquad (4)$$

Energy Minimization. Our minimization approach starts with enumerating all of the solution candidates. We first find all elementary circuits on the directed graph $\mathcal{G}$, denoted as $\{\mathcal{C}_j\}$. Then for each circuit $\mathcal{C}_j = \{s_{j,1}, s_{j,2}, ..., s_{j,m}\}$, we obtain a subset of circuit elements that minimizing E, denoted as $\mathcal{C}_j^* = \{s_{j,1}^*, s_{j,2}^*, ..., s_{j,m^*}^*\}$. The solution minimize E therefore is $\mathcal{C}^* = \text{argmin}\, E(\mathcal{C}_j^*)$. The number of all solution candidates can reach up to exponential growth, and therefore that simply using an exhaustive search to find the optimal solution might be computational prohibited. To support immediate feedback to users, we adopt approximate algorithms to reduce computational complexity, in both steps of finding circuits and minimizing the energy on each circuit.

The details are discussed in Algorithm 1. To find all circuits candidates, we adopt a branch-and-bound approach, based on the framework of existing approches [26, 27] for finding all elementary circuits of a directed graph. We only allow a limited number of circuits (denoted as σ, empirically $\sigma \leq 50$). The approximate algorithm adopts a heuristic search for finding the circuits with low cost as early as possible. We progressively search by graph connectivity, in order of cost $D(s)$ for seed vertices of the path, and in order of cost $V(s, s_{adj})$ for adjacent vertices.

Starting with $s_{j,k} = \min_{i \in [1,m]} D(s_{j,i})$, we compute the best candidates on a particular circuit by finding the shortest path on $\{s_{j,k}, ..., s_{j,m}, s_{j,1} ..., s_{j,k-1}\}$ from $s_{j,k}$ to $s_{j,k-1}$, with the total energy of a path defined as the sum of vertex weights and edge weights (defined by cost $D(s)$ and $V(s_{j,v}, s_{j,u})$, respectively) along the path. This problem can be effectively solved by using dynamic programming.

Algorithm 1. Energy minimization on the visibility graph

Input: *visibility graph* $\mathcal{G} = (\mathcal{S}, \mathcal{N})$, *upper bound of circuits numbers* σ

Output: $\mathcal{C}^*$ *that minimize* E

function ENERGYMINIMIZE($\mathcal{G} = (\mathcal{S}, \mathcal{N})$)
  local variables: $circuits = \emptyset$
  loop $s \in \mathcal{G}$ *in order of* $D(s)$
    $\mathcal{P} \leftarrow \{s\}$
    $FindCircuit(\mathcal{G}, \mathcal{P})$
    $s^{visit} \leftarrow True$
  loop $\mathcal{C}_j = \{s_{j,1}, s_{j,2}, ..., s_{j,m}\} \in circuits$
    $s_{j,k} = min_{i \in [1,m]} D(s_{j,i})$
    $\mathcal{C}'_j = \{s'_{j,1}, s'_{j,2}, ..., s'_{j,m}\}$
      $= \{s_{j,k}, ..., s_{j,m}, s_{j,1} ..., s_{j,k-1}\}$
    loop $i \in [1, m]$
      loop $t \in [j+1, m]$
        $\mathcal{E}'_j(i,t) = D(s'_{j,t}) + V(s'_{j,i}, s'_{j,t})$
    $\mathcal{C}^*_j = ShortestPath(\mathcal{G}'_j = (\mathcal{C}'_j, \mathcal{E}'_j), s'_{j,1}, s'_{j,m})$
  $\mathcal{C}^* = \text{argmin}\, E(\mathcal{C}^*_j)$
function FINDCIRCUIT($\mathcal{G} = (\mathcal{S}, \mathcal{N}), \mathcal{P}$)
  if $Size(circuits) > \sigma$ **then** $Return$
  $s \leftarrow Front(\mathcal{P})$
  loop $\{s, s_{adj}\} \in \mathcal{N}$ *in order of* $V(s, s_{adj})$
    if $s_{adj} = Front(\mathcal{P})$ **then**
      $Push(circuits, \mathcal{P})$
    if $u \notin \mathcal{P}$ *and* $u^{visit} = False$ **then**
      $Push(\mathcal{P}, s_{adj})$
      $FindCircuit(\mathcal{G}, \mathcal{P})$
      $Pop(\mathcal{P})$

4.2. Multiple Scribble Coloring

In the previous section, we discussed how to find influenced curve segments of a single scribble. Now we focus on the curve segments under the effects of multiple scribbles. Once a new scribble is added, its influenced curve segments would overlap with those which are assoiated with a particular scribble added in previous steps (Figure 5 (a)). We then partition the overlapped segments into many pieces so that each piece is assoiated with either its previous scribble or the new scribble (Figure 5 (b)). Finally each curve segement is assoiated with definitely one scribble before another new scribble is added. Our computation therefore occurs in a pair of scribbles in each step of adding one scribble. In other words, the label of each partition is 1 or 2 (Figure 6).

(a) (b)

Fig. 5. When a blue scribble is added, its influenced curve segments overlap with those are assoiated with red scribbles previously, which are marked with blue in (a). (b) shows that the overlapped segments are partitioned into pieces either colored with red or green.

Energy Function. Given a uniformly sampled curve segment $\mathcal{P}_{curve}$ and two user input scribbles, our target is to label each curve points with 1 or 2. The label results are denoted as $\{l_p\}_{p \in \mathcal{P}_{curve}}$, which could also be constructed as curve segments denoted as $\mathcal{S}_{curve} = \{s_i\}$ (Figure 6).

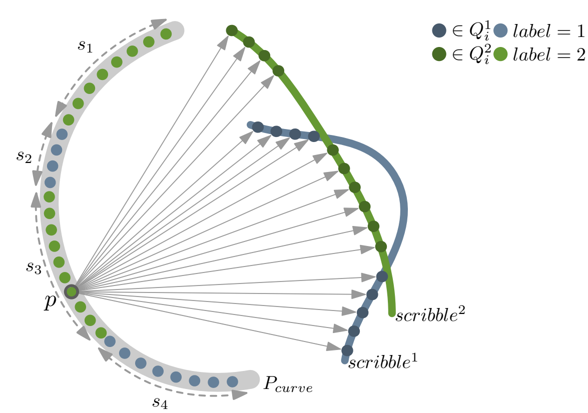

Fig. 6. Illustration for the ray tracing from a particular curve points as well as partitioning results, denoted as s_1, s_2, s_3, s_4.

Our goal is to obtain the best labels combination which satisfies the following three constraints: (1) The distance between the scribble and its corresponding curve points should be as small as possible; (2) The scribble is visible to its corresponding curve points, as much as possible, with considering of the block of the

other scribble; and (3) Not too many curve segments generated by the labelling process are desired.
We first define $L(p)$ as the measure term of each $p \in \mathcal{P}_{curve}$. By minimizing the sum of this term, denoted as $\Sigma_{p\in\mathcal{P}_{curve}} L(p)$, the first two constraints are satisfied. To compute the $L(p)$ (Equation 5), we start with making rays from p in a counterclockwise order by angle δ (empirically, $\delta = 5$), which generates a sequence of nearest intersection points $\mathcal{Q}$ that can be divided into two subsets, denoted as $\mathcal{Q}^1 \in scribble^1$ and $\mathcal{Q}^2 \in scribble^2$ (see Figure 6).

$$L(p) = \frac{n(\mathcal{Q}^{l_p})}{n(\mathcal{Q}^{l_p}) + n(\mathcal{Q}^{3-l_p})} + \frac{d(\mathcal{Q}^{3-l_p}, p)}{d(\mathcal{Q}^{l_p}, p) + d(\mathcal{Q}^{3-l_p}, p)}, \quad (5)$$

where $n(\mathcal{Q}^i)$ computes the number points in $\mathcal{Q}^i$, and $d(\mathcal{Q}^i, p)$ computes the average distance of $\mathcal{Q}^i$ to p.
To satisfie the third constraints, we define an evaluation term $H(x) = \frac{1}{2 - e^{\frac{x}{\tau - x}}}$. $H(x)$ is a function whose value grows slowly at the beginning (from $x = 0$) and quite fast when x grows out of a threshold that is controlled by τ ($\tau = 20$ by default and the threshold is $x = 8$). Setting x as the number of the curve segments constructed from $\{l_p\}$, this term adds a large penalty when $\{l_p\}$ introduces too many pieces (note that $x \geq 2$ in our case).
Combining the two terms together, the final energy minimization problem is defined as:

$$\min_{\{l_p\}_{p\in\mathcal{P}_{curve}}} \Sigma_{p\in\mathcal{P}_{curve}} L(p) + \omega H(counts(\mathcal{S}_{curve})), \quad (6)$$

where ω is a balancing weight (set to 0.1 as default).
Energy Minimization. As discussed above, we formulate a combination optimization problem, which can be simply solved by an exhaustive search by enumerating all possible label combinations. However, such an approach could be quite time consuming as the length of curves goes long.

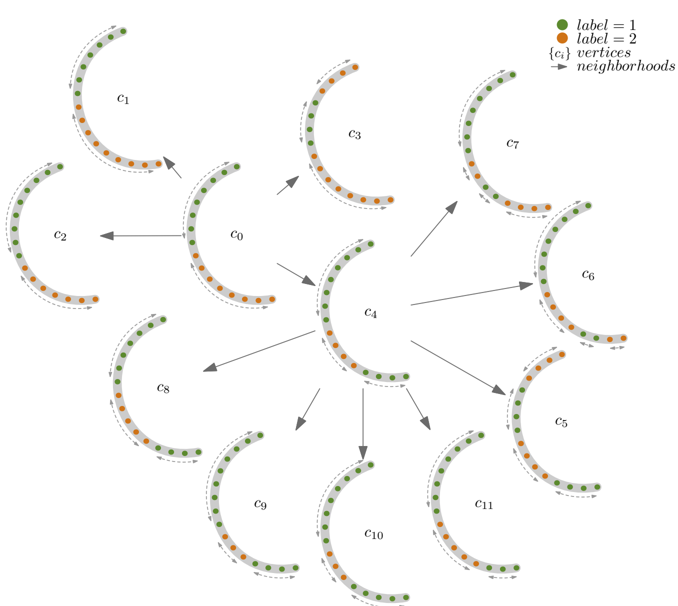

Fig. 7. Illustration of the vertices and neighborhoods of our local search graph, in which the initial state is c_0.

Here we introduce an approximate method by local search, the graph of which is constructed as 7 shows. The local search starts with an average partition as initial state (c_0 in Figure 7). With a current state, we define neighborhoods by two rules: (1) a bit change of points numbers between each two connected pieces, respectively. (eg. c_1, c_2 with current state c_0; c_8, c_9, c_{10} and c_{11} with current state c_4 in Figure 7); (2) adding more pieces by cutting each piece separately. (eg. c_3, c_4 with current state c_0; c_5, c_6, c_7 with current state c_4 in Figure 7) To avoid getting stuck on a local minimum, a simulated annealing algorithm is used in practice.
Note that the input scribbles in different orders might generate visually slightly different results (Figure 8). It is mainly because we take an approximate approach to solve the energy minimization problem.

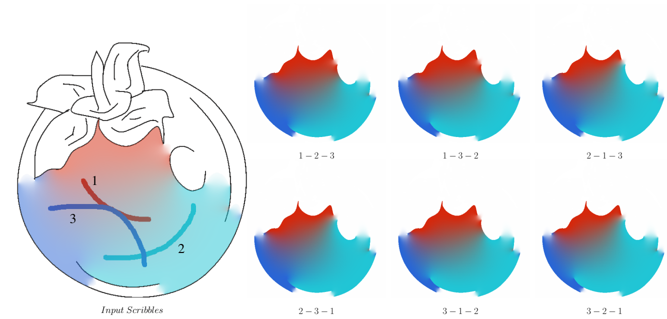

Fig. 8. An example of results generated by inputs scribbles in different orders. Left shows the three input scribbles, denoted as 1, 2 and 3 respectively. Right shows 6 results of all permutation of scribble orders.

5. Results and Discussion

We implemented our method in C++, Python and GLSL (OpenGL4.1) as a real-time drawing application running on the OS X/Linux platform. A notebook (running OS X 10.9) with Intel(R) Core(TM) i7 2.40GHz, 8GB RAM and a graphics card NVIDIA GeForce GT 650M was used as the testing device. The algorithm output depends on a few key parameters: weight μ in Equation 2, α, β, γ in Equation 3, η in Equation 4 and ω in Equation 6. Typically parameter values are selected within their corresponding ranges, and we used fixed parameter setting for generating all results (see Section 4).
We have tested the system on 12 line drawings (A-L), as listed in Figure 9. The set of line drawings covers different types of objects or scenes, including fruits, animals, man-made objects, human portraits etc. Our experiments show that our algorithm is able to achieve real-time performance on this device: for a typical input scribble, it takes $0.01 - 0.1$ seconds for color computing, and 0.6 seconds for rendering (The render process adopts a GPU diffusion solver proposed by [9] , which was implemented in GLSL). Our results are shown in Figure 1, 12, 13 and 16. These visually appealing results can be created in a short time, while it usually takes longer than an hour with tools provided by previous diffusion curve coloring tools and traditional commercial tools such as Illustrator. A typical editing session for the results shown here was in $5 - 20$ minutes, depending on the complexity of drawing shapes and the level of details the user wishes to achieve.

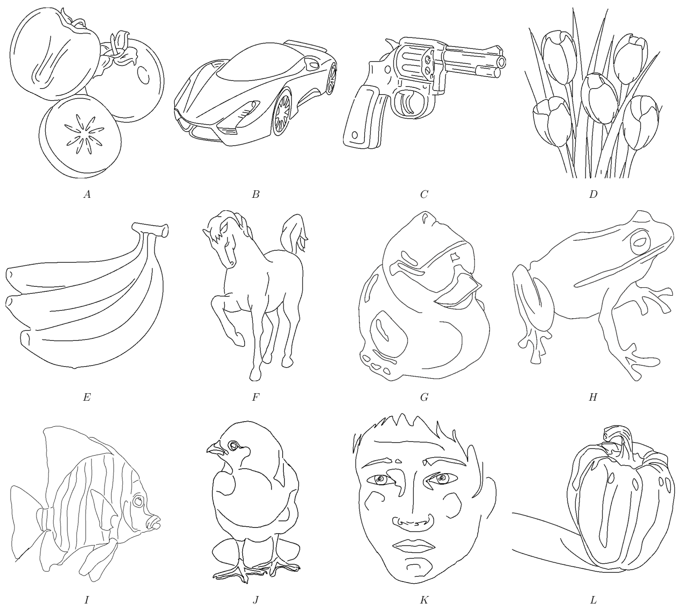

Fig. 9. Line drawings used in our tests. A-H are vectorized hand-drawns. I-J and K-L come from vectorized pixel images that were used in [13] and [9], respectively.

Figure 10 shows the flexibility of our tool. Our technique is largely insensitive to shape curves, and allows the user scribbles cross multiple regions. The scribbles are permitted to overlap each other, which largely decreases the frequency of undos. Figure 11 shows that our tool can produce more color variations by adding more scribbles (top row), while in the other examples richer visual effects can be produced by providing more geometry details with additional curves and scribbles (bottom row). Two additional examples with more complexity are shown in Figure 12.

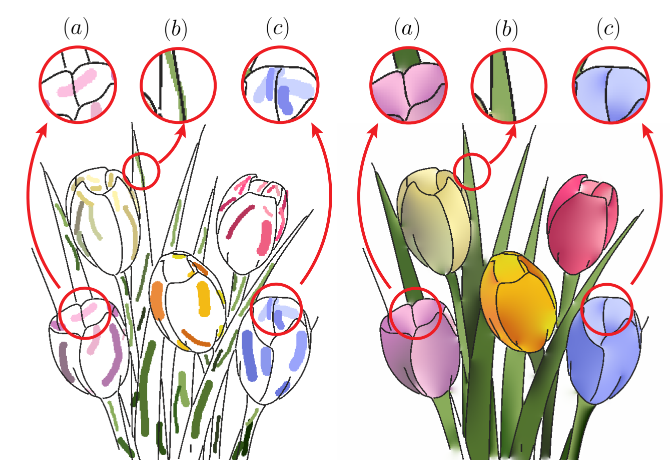

Fig. 10. Our tool permits scribbles to move out of range of boundary (b), cross multiple regions (a), overlap with each other (c).

5.1. Comparison

Under the same scribble configuration, we compared our tool with previous works, including the LazyBrush tool (Sykora et al. [8]), a color estimation tool for diffusion curves (Jeschke et al. [13]). Note that unlike LazyBrush and Jeschke et al. [13], which both model the competition among all the scribble, our tool processes user-specified inputs progressively every time a new scribble is added. The global computation of both LazyBrush and [13] would take a few seconds to a few minutes, which are relatively slow for instant feedback. In our test, the performance of LazyBrush depends on the number of scribbles and resolution of images, and [13] depends on the number of color points of diffusion curves, which would be up to ten minutes for complex gradients. In contrast, our progressive computation can provide instant feedback ($0.01 - 0.1$ seconds) for each scribble. For a typical input line drawing, it usually requires an artist to input $20 - 60$ scribbles for creating an art work.

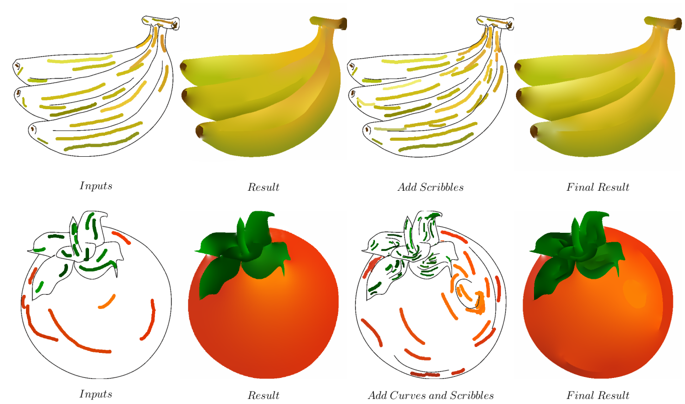

Fig. 11. Adding scribbles (top) or curves (bottom) to produce more color variations.

Fig. 12. Additional results. Left: Input line drawings and user-specified scribbles. Right: results produced by our system.

Figure 13 shows the comparison of visual effects. Jeschke et al. [13] suffers from serious artifacts for local inputs: generating out-of-range colors and incomplete coverage, and sometimes producing undesired colors as shown in Figure 13 and 16. To address these issues, a possible solution could provide a complete image first by applying image segmentation, which is then followed by image vectorization of [13]. LazyBrush is the state-of-the-art technique for pixel-level hand-drawn coloring or segmentation. Howerver it is not always guaranteed that desired labels can be achieved, especially in the open region. We provide the results in comparison to those by a combination of LazyBrush and color estimation approach of [13] in Figure 16.

Figure 14 shows a comparison with a typical deep learning technique for line drawing colorization, PaintsChainer [18]. Due to the implementation constraints, we are able to provide as similar scribbles as input to the two systems. It can be seen from Figure 14 middle and right that PainsChainer is able to generate interesting results even with very few scribbles but suffers from unnatural colors and artifacts, which though might be alleviated by the recent two-stage sketch colorization framework [20]. In contrast, our technique allows more accurate control of results (Figure 14 left).

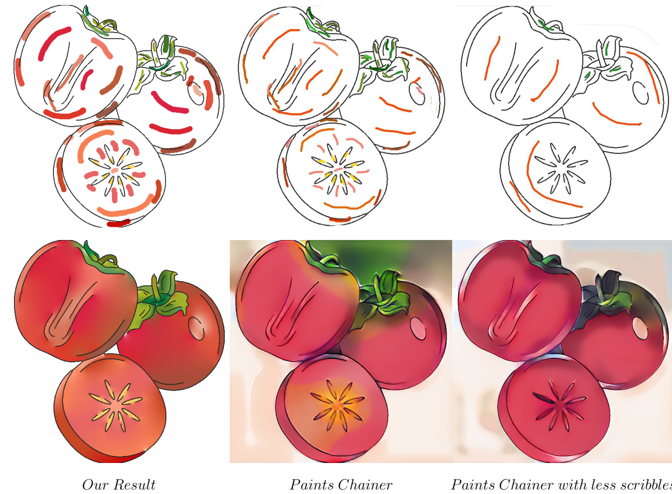

Fig. 14. Comparison with PaintsChainer.

Fig. 13. Comparison with Jeschke et al. [13].

5.2. Pilot Study

To evaluate the usability of our system, we conducted a pilot study to compare our interface with the original interface designed for Diffusion Curves [3]. We invited 8 volunteers (a1 to a8) to participate in the study. Two participants (a1, a5) were with good drawing skills. The other 6 participants had little drawing experience or knowledge.

Design and Procedure. We prepared 2 line drawings (vector graphics) to be colorized by each participant, as listed in Figure 15 (a), and a reference image for each line drawing that could be refered to by the participants as listed in Figure 15 (b). Each participant was first given a short tutorial of two coloring interfaces, by a short practice on an line drawing that was different from the two assigned ones. Next, each participant colorized the assigned 2 line drawings. For each line drawing, a participant painted with one of two interfaces, in the form of free drawing (as close to as but not necessarily exactly the same as the reference image). Then a similar colonization is required to be done with another interface. Figure 15 (c) (d) shows representative results created in the study. Half of the participants with and without good drawing skills (a1-a4) started with our system first. During the study we recorded the time for colorizing each drawing with each interface. Finally, the participants were asked to complete a questionnaire.

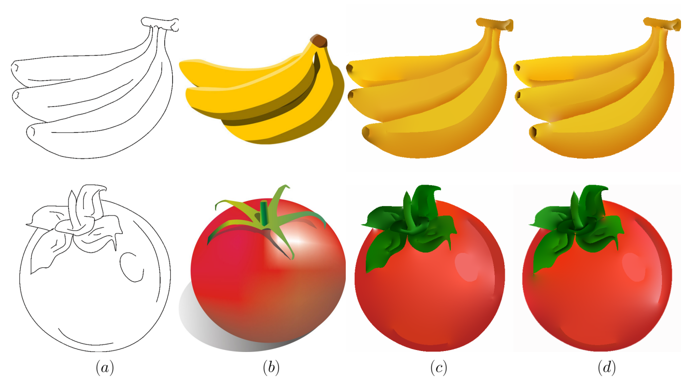

Fig. 15. (a) and (b) are the input line drawings and reference images used in the study, respectively. (c) and (d) are the representative results created with our interface and the original interface designed for Diffusion Curves, by the same participants.

Results. Paired t-test found that our system significantly outperformed the original interface of Diffusion Curves ($t = -8.88, p = 0.00004$ and $t = -10.32, p = 0.00002$ for banana and tomato examples respectively). The drawing time of our system was a bit slower than expected (average: 7min and 13min for banana and tomato examples respectively), but was still significantly faster than the original interface (average: 12min and 33min respectively). We observed that with our interface the participants could achieve a coarse result in a short time as expected, however they put more efforts in producing detailed color variation. We suspect the time for creating richer

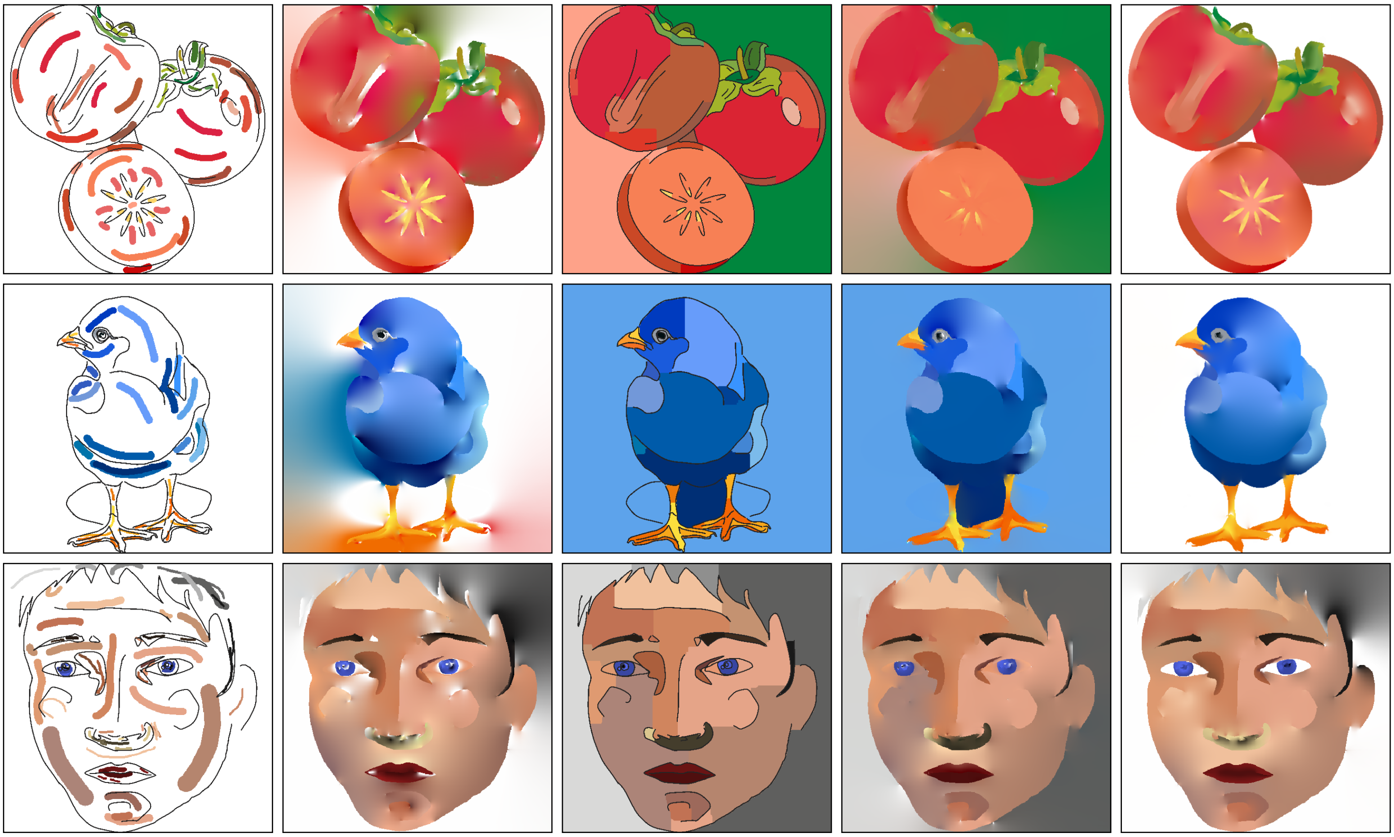

Fig. 16. More results in comparison with Jeschke et al. [13] and its combination with LazyBrush.

visual effects can be shortened once the users are more familiar with our tool.

After completing the coloring tasks, in the questionnaire each participant was asked to rate the coloring interfaces, in terms of ease of use and ease of learning on a discrete scale from 1 (poorest) to 5 (best). For the ease of learning, only one participant (a7) had no preference. The remaining the participants gave a higher rating to our system. Paired t-test confirmed that the ratings of our system were significantly higher ($t = 7.0, p = 0.0002$). All the participants found our interface were easier to use ($t = 9.0, p = 0.00004$).

5.3. Limitations and Future Work

As shown in Figure 17, our technique might not produce satisfactory results. It is possible that desired areas are not fully covered by our approach, and the user needs more efforts to refine them with more input scribbles. In the future work, we speculate that such limitations can be addressed by using multi-bounce ray cast, to achieve user desired scope as much as possible.

Fig. 17. A limitation. (b) shows coloring result of a failure case when a user scribble is across multiple open regions (a). (c) (d) shows that the user needs more efforts for generating the desired results, by adding more scribbles.

Acknowledgement

We would like to thank the anonymous reviewers for their valuable comments and the user study participants for their time. This work was partially supported by grants from the City University of Hong Kong (Project No. 7005176) and the Centre for Applied Computing and Interactive Media (ACIM) of School of Creative Media, CityU.

References

[1] Lai, YK, Hu, SM, Martin, RR. Automatic and topology-preserving gradient mesh generation for image vectorization. In: ACM Transactions on Graphics (TOG); vol. 28. ACM; 2009, p. 85.

[2] Richardt, C, Lopez-Moreno, J, Bousseau, A, Agrawala, M, Drettakis, G. Vectorising bitmaps into semi-transparent gradient layers. Computer Graphics Forum (Proceedings of EGSR) 2014;33(4).

[3] Orzan, A, Bousseau, A, Winnemöller, H, Barla, P, Thollot, J, Salesin, D. Diffusion curves: A vector representation for smooth-shaded images. ACM Transactions on Graphics 2008;27(3):1.

[4] Finch, M, Snyder, J, Hoppe, H. Freeform vector graphics with controlled thin-plate splines. In: ACM Transactions on Graphics (TOG); vol. 30. ACM; 2011, p. 166.

[5] Jeschke, S. Generalized diffusion curves: An improved vector representation for smooth-shaded images. In: Computer Graphics Forum; vol. 35. Wiley Online Library; 2016, p. 71–79.
[6] Hou, F, Sun, Q, Fang, Z, Liu, YJ, Hu, SM, Qin, H, et al. Poisson vector graphics. IEEE Transactions on Visualization and Computer Graphics 2018;.
[7] Bowers, JC, Leahey, J, Wang, R. A ray tracing approach to diffusion curves. In: Computer Graphics Forum; vol. 30. Wiley Online Library; 2011, p. 1345–1352.
[8] Sykora, D, Dingliana, J, Collins, S. Lazybrush: Flexible painting tool for hand-drawn cartoons. In: Computer Graphics Forum; vol. 28. Wiley Online Library; 2009, p. 599–608.
[9] Jeschke, S, Cline, D, Wonka, P. A gpu laplacian solver for diffusion curves and poisson image editing. In: ACM Transactions on Graphics (TOG); vol. 28. ACM; 2009, p. 116.
[10] Pang, WM, Qin, J, Cohen, M, Heng, PA, Choi, KS. Fast rendering of diffusion curves with triangles. IEEE Computer Graphics and Applications 2012;32(4):68–78.
[11] Prévost, R, Jarosz, W, Sorkine-Hornung, O. A vectorial framework for ray traced diffusion curves. In: Computer Graphics Forum; vol. 34. Wiley Online Library; 2015, p. 253–264.
[12] Zhao, S, Durand, F, Zheng, C. Inverse diffusion curves using shape optimization. arXiv preprint arXiv:161002769 2016;.
[13] Jeschke, S, Cline, D, Wonka, P. Estimating color and texture parameters for vector graphics. In: Computer Graphics Forum; vol. 30. Wiley Online Library; 2011, p. 523–532.
[14] Xie, G, Sun, X, Tong, X, Nowrouzezahrai, D. Hierarchical diffusion curves for accurate automatic image vectorization. ACM Transactions on Graphics (TOG) 2014;33(6):230.
[15] Lukáč, M, Fišer, J, Bazin, JC, Jamriška, O, Sorkine-Hornung, A, Sýkora, D. Painting by feature: Texture boundaries for example-based image creation. ACM Transaction on Graphics 2013;32(4).
[16] Isola, P, Zhu, JY, Zhou, T, Efros, AA. Image-to-image translation with conditional adversarial networks. arxiv 2016;.
[17] Su, W, Du, D, Yang, X, Zhou, S, Fu, H. Interactive sketch-based normal map generation with deep neural networks. Proceedings of the ACM on Computer Graphics and Interactive Techniques 2018;1(1):1–17.
[18] TaiZan, . Paintschainer. PreferredNetwork 2016;.
[19] Sangkloy, P, Lu, J, Fang, C, Yu, F, Hays, J. Scribbler: Controlling deep image synthesis with sketch and color. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2017, p. nil.
[20] Zhang, L, Li, C, Wong, TT, Ji, Y, Liu, C. Two-stage sketch colorization. ACM Transactions on Graphics (SIGGRAPH Asia 2018 issue) 2018;37(6):261:1–261:14.
[21] Levin, A, Lischinski, D, Weiss, Y. Colorization using optimization. In: ACM Transactions on Graphics (TOG); vol. 23. ACM; 2004, p. 689–694.
[22] Qu, Y, Wong, TT, Heng, PA. Manga colorization. In: ACM Transactions on Graphics (TOG); vol. 25. ACM; 2006, p. 1214–1220.
[23] Noris, G, Sỳkora, D, Shamir, A, Coros, S, Whited, B, Simmons, M, et al. Smart scribbles for sketch segmentation. In: Computer Graphics Forum; vol. 31. Wiley Online Library; 2012, p. 2516–2527.
[24] Huang, Z, Fu, H, Lau, RW. Data-driven segmentation and labeling of freehand sketches. ACM Transactions on Graphics (TOG) 2014;33(6):175.
[25] Xu, P, Fu, H, Au, OKC, Tai, CL. Lazy selection: a scribble-based tool for smart shape elements selection. ACM Transactions on Graphics (TOG) 2012;31(6):142.
[26] Tiernan, JC. An efficient search algorithm to find the elementary circuits of a graph. Communications of the ACM 1970;13(12):722–726.
[27] Johnson, DB. Finding all the elementary circuits of a directed graph. SIAM Journal on Computing 1975;4(1):77–84.